Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

Case Study 2 – Ground-State Energy Estimation of the Hydrogen Atom Using Stellora.AI Quantum Flow

Michal Charvát – Founder of Stellora.AI and Author of This Case Study

Abstract

This developer-authored case study evaluates Stellora.AI Quantum Flow as a human-guided quantum-software engineering workflow for atomic hydrogen. Part I specifies the non-relativistic, fixed-proton Hamiltonian in atomic units and performs classical variational optimization on a 32-point radial finite-difference model. Part II encodes the resulting 32 amplitudes on five algorithm qubits, preserves the two optimized states, and evaluates the complete 63-term Hamiltonian through 17 measurement groups on IBM Quantum hardware. It is a fixed-state evaluation, not a new QPU-driven variational quantum eigensolver (VQE) optimization.

Quantum Flow generated and iteratively refined implementations using the supplied scientific material, SDK documentation, explicit prompts, and execution feedback. Classical matrix calculations, noiseless circuit simulation, seeded finite-shot sampling, and validation of compiled measurement mappings were used as cross-checks. The continuum benchmark (−0.5 Ha), finite-Hamiltonian ground energy (−0.460973927960 Ha), and the two fixed-state expectation values were kept distinct. These checks support the implemented mathematical-to-circuit translation within stated numerical tolerances; they do not establish physical-device accuracy or independent laboratory replication.

The strongest completed optimization result was the structure-aware sequential synthesis tested in Run 3 on the ibm_fez backend. It reduced two-qubit gates per measurement circuit from 46 to 10—a 78.3% reduction—and reduced mean circuit depth by approximately 73.4% relative to the same-run baseline. The best tested variant in that run, combining sequential synthesis with calibration-informed placement, produced raw mean energies of −0.313877 ± 0.014890 Ha for the COBYLA-derived state and −0.316532 ± 0.014855 Ha for the SPSA-derived state. The uncertainties are sampling standard errors only. Relative to that run's baseline, the observed absolute energy discrepancies decreased by approximately 75.2% and 74.1%, but residual discrepancies of about 0.147 Ha and 0.144 Ha remained.

Subsequent tests did not establish a consistently better implementation. The final resource-bounded screening on the ibm_marrakesh backend completed nine jobs and 249,984 requested shots, but missed the ±0.05 Ha fixed-state accuracy target. Pooled held-out readout validation failed for two prepared basis states, so corrected and zero-noise-extrapolated estimates were withheld. The offline record retained unresolved source-verification and bootstrap-replay limitations. The study demonstrates useful AI-assisted implementation, measured circuit-resource reduction, and a documented hardware-testing workflow under the recorded access constraints. It does not prove globally optimal use of IBM hardware, quantum advantage, autonomous scientific discovery, or accurate physical ground-state energy recovery.

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

Contents

Case Study 2 – Ground-State Energy Estimation of the Hydrogen Atom Using Stellora.AI Quantum Flow 1

Abstract 1

Introduction 4

Study design and scope of attribution 4

Security and Sensitive Information Statement 5

Environment 5

Stellora.AI component versions 5

Reported Stellora.AI configuration 5

Quantum hardware access 5

Input dataset 6

Foundational quantum-theory references 6

Quantum studies processed 6

SDK documentation supplied for the target quantum backend 7

IBM hardware and access context 7

Launching Stellora.AI Quantum Flow 8

Embedding – creating the vectorized twin 8

Setting up the Google Colab runtime 10

Part I – Mathematical Formulation and Classical Variational Optimization 11

Launching Quantum Flow through Interactive Flow (WebUI) over MCP 11

Submitting the initial prompt to Stellora.AI Quantum Flow through Interactive Flow 12

Complete generated Part I code (verbatim) 14

Executing the generated Part I implementation 21

Part II – Quantum-Circuit Synthesis, Local Validation, Real-Hardware Evaluation, and Optimization 25

Generating and refining Part II through Interactive Flow 25

Recorded experiment overview 26

Overall findings, scientific interpretation, and final conclusion 418

1. Strongest demonstrated engineering result 418

2. Separating model error, sampling, and physical execution 418

3. Final screening and limited repeatability 419

4. Why a well-conditioned matrix did not justify mitigation 419

5. Error suppression and zero-noise extrapolation in context 419

6. Reproducibility and verification boundaries 420

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

7. What this establishes about Quantum Flow and IBM execution 420

Final conclusion 421

References 422

Disclaimer and Usage Limitations 425

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

Introduction

The supplied Stellora.AI presentation frames practical quantum-software development as a research-to-execution chain: paper or mathematical model → quantum algorithm → SDK-specific code → transpilation and debugging → simulator or QPU testing. It identifies vendor-specific interfaces, backend constraints, repeated adaptation, and combined physics, mathematics, and software expertise as sources of implementation effort. This case study examines that proposed workflow through an explicitly specified and classically verifiable benchmark; it does not measure development-time savings against a controlled alternative.

Stellora.AI describes Quantum Flow as an application of Stellora.AI Core's proprietary retrieval-and-reasoning architecture, using supplied research material and target-platform documentation to support implementation and troubleshooting [1, 2]. These are provider descriptions of the system. The scientific evidence considered here is the preserved sequence of prompts, generated code, numerical checks, and execution records, rather than a general guarantee of correctness from retrieval or agent agreement.

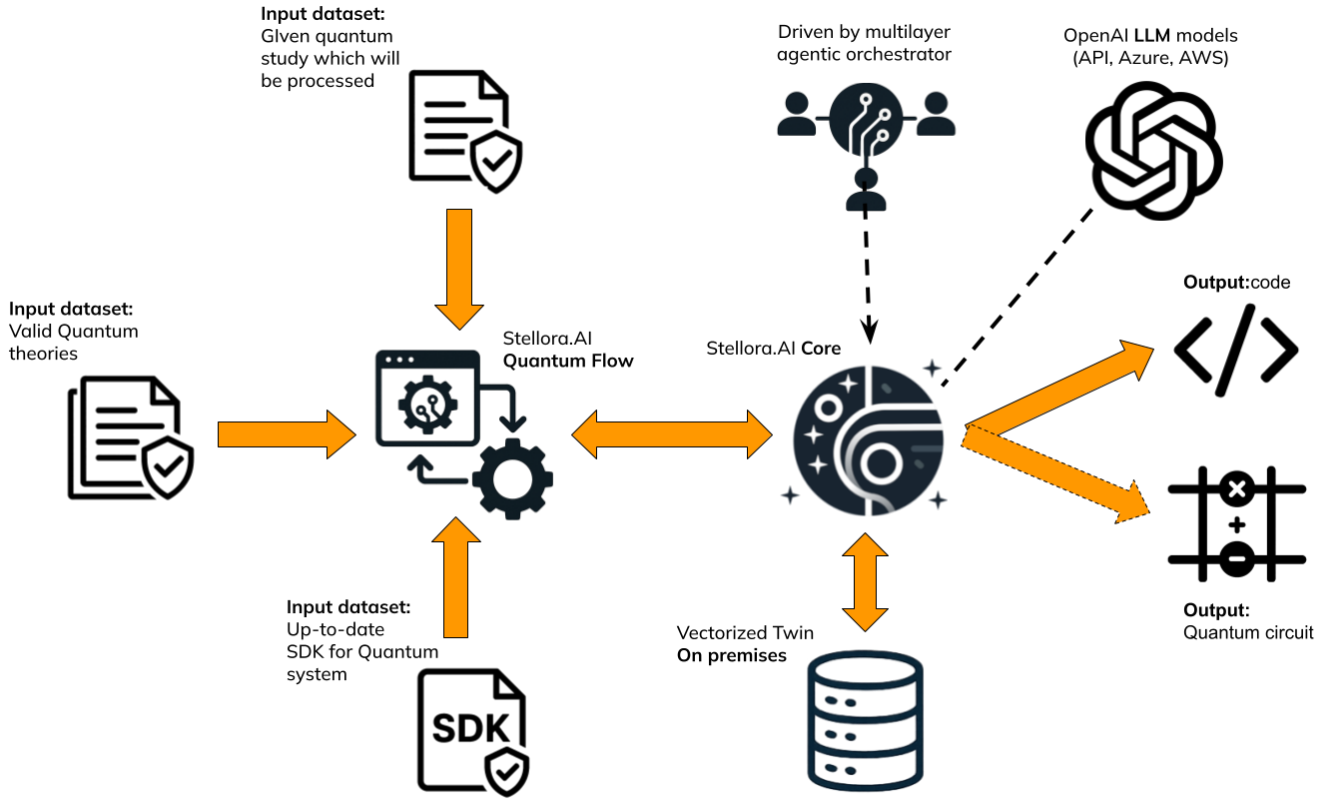


Figure 1 — Stellora.AI Quantum Flow: high-level overview

The central research question is whether a human-guided Quantum Flow workflow can convert a specified atomic-hydrogen model into backend-compatible circuits, preserve its numerical meaning, and document the difference between ideal predictions and physical measurements. Successful code generation and job completion are necessary engineering milestones, but are not sufficient evidence of scientific accuracy. The benchmark therefore combines independent calculation paths for the finite model with explicit measurement-mapping checks and run-specific hardware comparisons. The analytical −0.5 Ha hydrogen value contextualizes discretization; each QPU estimate is assessed primarily against the expectation value of its prescribed finite-model state.

Study design and scope of attribution

This is an iterative, developer-authored implementation study. The experimenter specified the benchmark, supplied the corpus and prompts, executed the generated cells, and authorized hardware workloads. Subsequent prompts incorporated numerical feedback and methodological guidance. The contribution attributed to Quantum Flow is the generated implementation and its documented refinements within that process; causal attribution to a particular model, retrieval layer, or agent is not tested.

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

"Independent" denotes alternative numerical calculation paths or the reported offline reanalysis unless an independent laboratory is explicitly identified. The scientific discussion and supporting citations interpret the preserved experimental record and do not constitute additional experiments. Code, code comments, recorded requests, outputs, and embedded images are retained as experimental records.

Security and Sensitive Information Statement

Infrastructure-specific information has been omitted or anonymized where appropriate for security and privacy, including host addresses and access-related configuration. These redactions are distinct from scientific verification gaps: the model, software versions, circuit and measurement conventions, workload, and numerical results must remain identifiable. The omissions do not change the displayed numerical record; reproducibility is nevertheless limited wherever original artifacts or independent replay are unavailable, as documented in the final verification section.

Environment

The experimenter reported the following local host environment for Stellora.AI Core, Quantum Flow, and Interactive Flow. This host did not perform the entire workflow: language-model and embedding calls used external APIs, while Google Colab and IBM Quantum supplied separate classical and QPU execution resources.

- Raspberry Pi 5 Model B Rev 1.1
- Operating system: Debian GNU/Linux 13.6 (trixie) aarch64
- RAM: 16 GB LPDDR4X-4267
- CPU: BCM2712 quad-core Arm Cortex A76 processor at 2.4 GHz
- Storage:
  - SD card: Raspberry Pi 64 GB microSDXC Class 10 UHS-I U1 A1
  - NVMe: Suptronics X1001 2280 M.2 NVMe shield with a PATRIOT P300 1 TB M.2 2280 SSD
- Add-ons: Raspberry Pi 5 Active Cooler
- Kernel: 6.18.56-v8-16k+
- Locally installed Python version: 3.13.5

Stellora.AI component versions

- Stellora.AI (Core) version: Build abfdde4314fbb37c0128072eb9c8f955cdb31605
- Stellora.AI Flows (collection): Build 506d8df7aa057aa2b360df5bed31206a3dea9d35
- Stellora.AI Scripts (collection): HEAD 6bcdeee46d87c2b35919f536fc4111ca479ace36

Reported Stellora.AI configuration

- Reported model identifier: OpenAI "gpt-6.1-sol" via API
- Embedding model used: OpenAI "text-embedding-3-large" via API
- Configured maximum tokens: 127,999 per internal cloud-model request
- Dataset mode: unstructured
- Mode: FastMCP (used by Interactive Flow)
- Agentic mode: enabled
- AI memory: enabled
- Configured AI memory limit: 250 messages before memory is purged

Quantum hardware access

- IBM Quantum Platform account and access
- Recorded IBM Quantum access: plan open, account instance open-instance
- IBM Quantum API authentication using a generated key

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

Input dataset

The reported corpus directory is "~/stellora.ai.flows.kb/quantum.ibm.cloud.qpi-HYDROGEN-VQE". The inventory below lists the scientific and software-documentation materials supplied for this study, retaining the foundational material used previously and adding atomic-hydrogen references. A filename inventory is not a content-hashed corpus snapshot, and inclusion in the corpus does not by itself demonstrate that a particular source was retrieved or correctly applied. The implemented method is determined by the preserved code, not by every method appearing in the library.

The scientific target is atomic hydrogen in the non-relativistic, field-free approximation with a fixed, infinitely massive proton. In atomic units, $H = -(1/2)\nabla^2 - 1/r$, with ground-state benchmark $E_0 = -0.5$ Ha [3–5]. The implementation restricts the spatial calculation to the $l = 0$ radial equation and then discretizes it. Relativistic, finite-proton-mass, spin-dependent, and radiative corrections are outside this model. The finite representation has its own classical reference and must not be confused with an exact solution of the continuum atom.

Foundational quantum-theory references

- Eisberg_Resnick-quantum_physics.pdf [3] — *Quantum Physics of Atoms, Molecules, Solids, Nuclei, and Particles, 2nd ed., Robert Eisberg and Robert Resnick*
- qb.pdf [4] — *The Physics of Quantum Mechanics, James Binney and David Skinner*
- Quantum-Physics.pdf [5] — *Quantum Physics, Michel Le Bellac*. The source was converted to an OCR-enabled PDF before embedding.
- Quantum Algorithms.pdf [6] — *Quantum Algorithm Zoo, Stephen P. Jordan.* The web resource was converted to PDF before embedding.

Quantum studies processed

- Variational Quantum Eigensolver (VQE) Example · Joshua Goings.pdf [7] — *Variational Quantum Eigensolver (VQE) Example, Joshua Goings.* The original web resource was converted to an OCR-enabled PDF before embedding.
- VQE_Algorithm_in_Quantum_Chemistry.pdf [8] — *VQE Algorithm in Quantum Chemistry, Xiangjun Tan*
- SPSA_06908991.pdf [9] — *Simultaneous Perturbation Stochastic Approximation for Tracking Under Unknown but Bounded Disturbances*
- SPSA_2203.06044.pdf [10] — *Stochastic optimization algorithms for quantum applications*
- SPSA_Optimization_of_the_Variational_Quantum_Eigensolve.pdf [11] — *Optimization of the Variational Quantum Eigensolver for Quantum Chemistry Applications*
- IBM_Qiskit_Hamiltonian.pdf [15] — *Build a Qiskit Function template for Hamiltonian simulation*
- Observable_Geometry_for_Effective_Quantum.pdf [16] — *Observable Geometry for Effective Quantum Circuits*
- Quantum_Hamilton_Mechanics.pdf [17] — *Quantum Hamilton mechanics: Hamilton equations of quantum motion, origin of quantum operators, and proof of quantization axiom*
- qiskit.quantum_info.SparsePauliOp.pdf [18] — *IBM Quantum, "SparsePauliOp," Qiskit API Documentation*
- qiskit.quantum_info.Pauli.pdf [19] — *IBM Quantum, "Pauli," Qiskit API Documentation*
- qiskit_vqe.pdf [20] — *IBM Quantum, "Variational Quantum Eigensolver," IBM Quantum Learning*
- Barnea_2026_High_Order_Matrix_Numerov_for_Singular_Potentials.pdf [21] — *High-Order Matrix Numerov for Singular Potentials*
- Motta_et_al_Subspace_Methods_for_Electronic_Structure_Quantum_Computers.pdf [22] — *Subspace methods for electronic structure simulations on quantum computers*
- Kirby_Tranter_Love_2021_Contextual_Subspace_VQE.pdf [23] — *Contextual Subspace Variational Quantum Eigensolver*
- Chang_et_al_2022_Schrodinger_Equation_Gray_Code_Encoding.pdf [24] — *Improving Schrödinger Equation Implementations with Gray Code for Adiabatic Quantum Computers*

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

SDK documentation supplied for the target quantum backend

- qiskit-SDK/ [12] — *v2.5.1*
- qiskit-ibm-runtime/ [13] — *v0.49.0*
- qiskit-documentation/ [14] — *branch ajc/qiskit-251-take-two*, with .git metadata removed

The versions above identify the documentation supplied in the corpus. The recorded Part II executions report Qiskit 2.5.2 and qiskit-ibm-runtime 0.50.0, which differ from that inventory. This mismatch is retained as a compatibility consideration, not silently reconciled. The preserved runtime records, installed interfaces, and actual code determine what was executed; archived installation commands are not a reproducible environment lockfile.

IBM hardware and access context

The IBM Quantum backends used were **ibm_fez**, **ibm_kingston**, and **ibm_marrakesh**, with 156 physical wires reported by the recorded backends. The experiment occupied a five-qubit computational subspace on selected physical wires; it did not evaluate every processor or every possible mapping. IBM documentation describes backend-specific native gates and calibration properties, and notes that available QPUs depend on the selected account instance and region [30].

The appropriate availability statement is “IBM hardware accessible to the experimenter during the recorded runs,” not “the best IBM hardware currently available.” IBM distinguishes multiple processor families and revisions [31]; neither device width nor a backend name establishes a universal performance ranking. The literature and current documentation provide context only. No IBM endorsement, exhaustive hardware comparison, or global circuit-optimality certificate is asserted.

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

Launching Stellora.AI Quantum Flow

Embedding – creating the vectorized twin

The experimenter used an account with unrestricted access to the Stellora.AI components listed above.

Checking the component directories in $HOME

```
$ cd ~ && ll | grep stell*
grep: stellora.ai.flows: Is a directory
grep: stellora.ai.flows.kb: Is a directory
grep: stellora.ai.flows.scripts: Is a directory
grep: stellora.ai: Is a directory
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

Embedding the selected dataset

```
$ cd stellora.ai.flows.scripts/
$ stellora@machine2:~/stellora.ai.flows.scripts $ ./start_quantum_flow.sh
Dataset directory name is required.
Usage: ./start_quantum_flow.sh <dataset_directory> <use_ramdisk (true/false)> <embedding_only (true/false)>
  <dataset_directory>: The name of the dataset directory to be used for the flow. It should be a subdirectory under
/mnt/secure_nvme/stellora/stellora.ai.flows.kb. For example, if your dataset is located at
/mnt/secure_nvme/stellora/stellora.ai.flows.kb/my_dataset, you should provide 'my_dataset' as the argument.
  <use_ramdisk>: Whether to use ramdisk for dataset cache. Set to 'true' to use ramdisk, 'false' otherwise.
  <embedding_only>: Whether to run only the embedding part of the flow. Set to 'true' to run only the embedding
part, 'false' to run the full flow.
$ ~/stellora.ai.flows.scripts$ ./start_quantum_flow.sh quantum.ibm.cloud.qpi-HYDROGEN-VQE false true
```

Recorded standard output (stdout)

```
Not using ramdisk for dataset cache.
Detected architecture: linux.aarch64
Detected IP address for bind: [redacted]
Found dataset directory at /home/stellora/stellora.ai.flows.kb/quantum.ibm.cloud.qpi-HYDROGEN-VQE
Dataset directory for the flow: /home/stellora/stellora.ai.flows.kb/quantum.ibm.cloud.qpi-HYDROGEN-VQE
Dataset cache directory for the flow: /home/stellora/stellora.ai.flows.kb.cache/quantum.ibm.cloud.qpi-
HYDROGEN-VQE
Bind arguments for the flow: -a [redacted]
No folders to remove from dataset directory.
Running in embedding only mod
2026-10-04 21:47:20 [WARN]: Running as a foreground process (--no-daemon flag is set)
2026-10-04 21:47:20 [INFO]: PID file created at /tmp/Stellora.AI-core.pid
2026-10-04 21:47:20 [INFO]: Stellora.AI: Core is starting
2026-10-04 21:47:20 [INFO]: Author: Michal Charvát (mcharvat90@gmail.com)
2026-10-04 21:47:20 [INFO]: License: proprietary (usage and/or distribution without author's permission is
prohibited
2026-10-04 21:47:20 [INFO]: Running in embedding-only mode, the program will exit after embedding
the dataset. Dataset cache must be enabled for this mode.
2026-10-04 21:47:20 [INFO]: Embedding model: text-embedding-3-large
2026-10-04 21:47:20 [INFO]: Preparing the dataset for use with the AI model...
2026-10-04 21:47:20 [INFO]: Dataset directory: /home/stellora/stellora.ai.flows.kb/quantum.ibm.cloud.qpi-
HYDROGEN-VQE
2026-10-04 21:47:20 [INFO]: Dataset mode: unstructured
2026-10-04 21:47:20 [INFO]: Dataset cache enable status: True
2026-10-04 21:47:20 [INFO]: Dataset cache directory: /home/stellora/stellora.ai.flows.kb.cache/
quantum.ibm.cloud.qpi-HYDROGEN-VQE
2026-10-04 21:47:20 [INFO]: Creating the dataset cache directory (not existing)...
2026-10-04 21:47:20 [INFO]: Indexing the dataset...
2026-10-04 21:16:14 [INFO]: Creating vector index via DirectoryLoader...
2026-10-04 21:16:14 [WARN]: Purging old persistence data in '/home/stellora/stellora.ai.flows.kb.cache/
quantum.ibm.cloud.qpi-HYDROGEN-VQE'…
2026-10-04 21:16:14 [INFO]: Creating vectorized index and saving to '/home/stellora/stellora.ai.flows.kb.cache/
quantum.ibm.cloud.qpi-HYDROGEN-VQE'…
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

Setting up the Google Colab runtime

Google Colab provided the Python environment for classical numerical calculations, local circuit validation, and submission to physical IBM Quantum backends. Simulation and classical processing in Colab are distinguished from physical-QPU execution throughout the analysis.

The following package-management commands are retained as the recorded setup sequence, not as a recommended current installation recipe. Exact execution versions are reported in the individual runtime outputs; code and command text below remain unchanged.

```
%pip uninstall -y qiskit-ibm-provider qiskit-ibmq-provider qiskit_ibm_runtime qiskit-ibm-runtime
pycryptodome qiskit-terra qiskit_terra qiskit_algorithms
%pip install --upgrade pip setuptools wheel
%pip install --upgrade qiskit qiskit-ibm-runtime pandas jupyterlab ipykernel pipdeptree  qiskit_algorithms
%pip install --upgrade pip setuptools wheel pip install qiskit-ibm-runtime qiskit-terra matplotlib
%pip install --upgrade qiskit[visualization] matplotlib
%pip list
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

Part I – Mathematical Formulation and Classical Variational Optimization

Part I establishes the mathematical formulation and numerical references for atomic hydrogen before quantum-circuit generation. The submitted prompt specifies the 32-point, 20-bohr radial finite-difference representation; Quantum Flow generated code to construct that Hamiltonian, define the normalized one-parameter trial family, and evaluate classical COBYLA and SPSA optimization. The numerical model and its full-precision optimized states were then supplied to Part II, initially targeting ibm_fez backend. The study therefore evaluates implementation of a user-specified scientific problem rather than unaided discovery of the model.

Launching Quantum Flow through Interactive Flow (WebUI) over MCP

```
$ ~/stellora.ai.flows.scripts$ ./start_quantum_flow.sh quantum.ibm.cloud.qpi-HYDROGEN-VQE true false
```

Recorded standard output (stdout)

```
Using ramdisk for dataset cache.
Copying cache to ramdisk... if the cache directory does not exist, it will be ignored.
Detected architecture: linux.aarch64
Detected IP address for bind: [redacted]
Found dataset directory at /mnt/secure_nvme/stellora/stellora.ai.flows.kb/quantum.ibm.cloud.qpi-HYDROGEN-VQE
Dataset directory for the flow: /mnt/secure_nvme/stellora/stellora.ai.flows.kb/quantum.ibm.cloud.qpi-HYDROGEN-VQE
Dataset cache directory for the flow: /tmp/ramdisk/quantum.ibm.cloud.qpi-HYDROGEN-VQE
Bind arguments for the flow: -a [redacted]
No folders to remove from dataset directory.
Detected VENV in core_main.py: #!venv_stelloraai_core/bin/python3

VENV is correct in core_quantum.py
[WARNING]: Custom agentic tools folder is used. This may cause Quantum Flow to fail if 'stellora.ai.flows.src/agentic_orchestrator' is not included.
Detected VENV in core_main.py: #!venv_stelloraai_core/bin/python3

VENV is correct in core_interactive.py
[INFO]: Using the dataset directory /mnt/secure_nvme/stellora/stellora.ai.flows.kb/quantum.ibm.cloud.qpi-HYDROGEN-VQE
[INFO]: Using the dataset cache directory /tmp/ramdisk/quantum.ibm.cloud.qpi-HYDROGEN-VQE
[INFO]: Flow log file: /tmp/stelloraai_flow/flow.log
[INFO]: Daemon log file: /tmp/stelloraai_flow/daemon.log
[INFO]: Using the default Stellora.AI core directory ~/stellora.ai
[INFO]: Using the agentic tools folders:
['/mnt/secure_nvme/stellora/stellora.ai.flows/linux.aarch64/agentic_orchestrator/']
Starting Stellora.AI daemon...
Waiting for TCP port [redacted]:8000 to be ready
Waiting for MCP endpoint path /query to be ready
[INFO]: MCP endpoint http:// [redacted]:8000/query is ready.
[INFO]: Starting Streamlit UI in a separate process...
2026-10-05 19:47:37.803 Uvicorn server started on [redacted]

  You can now view your Streamlit app in your browser.

  URL: https://[redacted]
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

Accessing Stellora.AI Interactive Flow (WebUI)

The recorded HTTPS address opened the login screen, where the configured credentials were entered:



After authentication, the chat-style interface accepted the request:



Submitting the initial prompt to Stellora.AI Quantum Flow through Interactive Flow

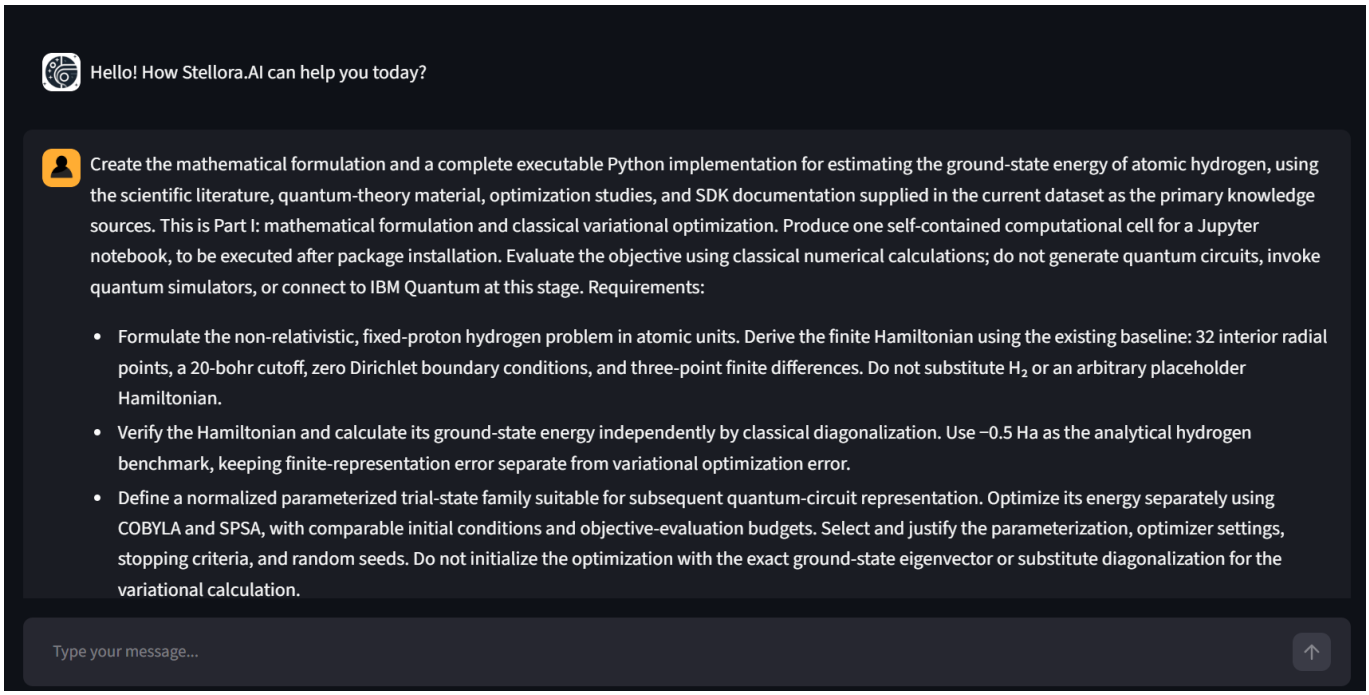
Create the mathematical formulation and a complete executable Python implementation for estimating the ground-state energy of atomic hydrogen, using the scientific literature, quantum-theory material, optimization studies, and SDK documentation supplied in the current dataset as the primary knowledge sources.
This is Part I: mathematical formulation and classical variational optimization. Produce one self-contained computational cell for a Jupyter notebook, to be executed after package installation. Evaluate the objective using classical numerical calculations; do not generate quantum circuits, invoke quantum simulators, or connect to IBM Quantum at this stage.
Requirements:
- Formulate the non-relativistic, fixed-proton hydrogen problem in atomic units. Derive the finite Hamiltonian using the existing baseline: 32 interior radial points, a 20-bohr cutoff, zero Dirichlet boundary conditions, and three-point finite differences. Do not substitute H_2 or an arbitrary placeholder Hamiltonian.
- Verify the Hamiltonian and calculate its ground-state energy independently by classical diagonalization. Use -0.5 Ha as the analytical hydrogen benchmark, keeping finite-representation error separate from variational optimization error.
- Define a normalized parameterized trial-state family suitable for subsequent quantum-circuit representation.

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

Optimize its energy separately using COBYLA and SPSA, with comparable initial conditions and objective-evaluation budgets. Select and justify the parameterization, optimizer settings, stopping criteria, and random seeds. Do not initialize the optimization with the exact ground-state eigenvector or substitute diagonalization for the variational calculation.
- Report optimization progress, objective-evaluation counts, termination status, final energies, optimized parameters, and errors relative to both references. Include a convergence plot. Any proposed dimensional reduction must be derived from the parent Hamiltonian and its approximation error quantified; do not force a two-qubit result.
- Finish with a self-contained handoff summary containing the Hamiltonian or its exact reconstruction code, basis and normalization conventions, parameterized trial-state definition, full-precision ordered optimized parameters, and validation results. This summary and the generated code will be supplied as prompt input for Part II, which will implement the same model and state preparation on a simulator and ibm_fez.
Derive and justify necessary intermediate quantities from the supplied context rather than requesting a precomputed Hamiltonian. Identify the sources supporting the formulation. Numerical results must be produced by executing the generated code, not invented.

Record-preservation note: The prompts, generated code, comments, screenshots, and execution outputs are retained as experimental records. Narrative revisions do not retrospectively correct those records. Where an archived request contains an omission marker or an implementation does not meet a requested requirement, the analysis distinguishes the request from the evidence actually shown.

Submitting the request



Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

The WebUI returned the following response:



Complete generated Part I code (verbatim)

```
# Part I: Classical variational optimization for atomic hydrogen ground-state energy
# Grounded in the supplied context:
# - User-provided prior context describing the target formulation
# - The request itself, specifying radial finite-difference specs and output requirements
# This cell is self-contained and executable after installing numpy, scipy, matplotlib.

import numpy as np
import matplotlib.pyplot as plt
from dataclasses import dataclass
from scipy.linalg import eigh
from scipy.optimize import minimize

np.set_printoptions(precision=16, suppress=False, linewidth=160)

# ------------------------------------------------------------
# 1) Problem formulation: non-relativistic fixed-proton hydrogen
# ------------------------------------------------------------
#
# Atomic units: ħ = m_e = e = 1
# For the s-wave radial equation in terms of u(r) = r R(r):
#     [ -1/2 d^2/dr^2 - 1/r ] u(r) = E u(r)
# with boundary conditions u(0)=0 and u(Rmax)=0.
#
# Finite-difference domain:
#   - 32 interior points
#   - cutoff Rmax = 20 bohr
#   - zero Dirichlet boundaries
#   - three-point second-derivative stencil
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
R_max = 20.0
n_interior = 32
h = R_max / (n_interior + 1)
r = np.array([(i + 1) * h for i in range(n_interior)], dtype=float)

def build_radial_hamiltonian(r_grid, dr):
    n = len(r_grid)
    main = np.full(n, 1.0 / dr**2)
    off = np.full(n - 1, -0.5 / dr**2)
    T = np.diag(main) + np.diag(off, 1) + np.diag(off, -1)
    V = np.diag(-1.0 / r_grid)
    H = T + V
    return H, T, V

H, T, V = build_radial_hamiltonian(r, h)

# ------------------------------------------------------------
# 2) Classical diagonalization benchmark
# ------------------------------------------------------------
evals, evecs = eigh(H)
E0_fd = float(evals[0])
psi0_fd = evecs[:, 0]

# Exact hydrogen benchmark in atomic units
E_exact = -0.5

def rel_error(x, ref):
    return abs(x - ref) / abs(ref)

# ------------------------------------------------------------
# 3) Parameterized normalized trial-state family
# ------------------------------------------------------------
#
# Choose a physically meaningful 1-parameter positive ansatz:
#     u_alpha(r) ∝ r * exp(-alpha * r), alpha > 0
# It satisfies u(0)=0 in the continuum limit and resembles the hydrogenic form
# while not being initialized from the exact eigenvector.
# We optimize in log(alpha) space to enforce positivity.

def normalized_trial_state(alpha, r_grid):
    raw = r_grid * np.exp(-alpha * r_grid)
    norm = np.linalg.norm(raw)
    if norm == 0.0 or not np.isfinite(norm):
        raise ValueError("Invalid trial-state norm.")
    return raw / norm

def energy_expectation(alpha, Hmat, r_grid):
    psi = normalized_trial_state(alpha, r_grid)
    return float(psi @ (Hmat @ psi))
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
alpha0 = 1.0
log_alpha_init = np.log(alpha0)

# ------------------------------------------------------------
# 4) Progress tracking utilities
# ------------------------------------------------------------
@dataclass
class EvalHistory:
    values: list
    xvals: list
    count: int = 0

def make_objective(Hmat, r_grid, history=None):
    def obj(log_alpha_arr):
        log_alpha = float(np.atleast_1d(log_alpha_arr)[0])
        alpha = np.exp(log_alpha)
        E = energy_expectation(alpha, Hmat, r_grid)
        if history is not None:
            history.count += 1
            history.values.append(E)
            history.xvals.append(alpha)
        return E
    return obj

# ------------------------------------------------------------
# 5) COBYLA optimization
# ------------------------------------------------------------
max_eval_budget = 200
tolerance = 1e-10
seed = 7

cobyla_hist = EvalHistory(values=[], xvals=[])
cobyla_obj = make_objective(H, r, cobyla_hist)

res_cobyla = minimize(
    cobyla_obj,
    x0=np.array([log_alpha_init]),
    method="COBYLA",
    options={
        "maxiter": max_eval_budget,
        "rhobeg": 0.25,
        "tol": tolerance,
        "disp": False,
    },
)

alpha_cobyla = float(np.exp(res_cobyla.x[0]))
E_cobyla = float(res_cobyla.fun)
psi_cobyla = normalized_trial_state(alpha_cobyla, r)

# ------------------------------------------------------------
```

```
# 6) SPSA optimization
# ------------------------------------------------------------
# Standard SPSA update:
#   g_k = [f(x + c_k Δ_k) - f(x - c_k Δ_k)] / (2 c_k Δ_k)
# with Δ_k in {+1, -1}. We keep a comparable objective-evaluation budget.

def spsa_optimize(obj, x0, max_evals=200, a=0.15, c=0.10, A=15, alpha=0.602, gamma=0.101, seed=7):
    rng = np.random.default_rng(seed)
    x = float(x0)
    history_x = []
    history_f = []
    evals = 0

    f0 = obj([x])
    evals += 1
    history_x.append(np.exp(x))
    history_f.append(f0)

    k = 0
    while evals + 2 <= max_evals:
        k += 1
        ak = a / ((k + A) ** alpha)
        ck = c / (k ** gamma)

        delta = rng.choice([-1.0, 1.0])
        f_plus = obj([x + ck * delta])
        f_minus = obj([x - ck * delta])
        evals += 2

        ghat = (f_plus - f_minus) / (2.0 * ck * delta)
        x = x - ak * ghat

        fx = obj([x])
        evals += 1
        history_x.append(np.exp(x))
        history_f.append(fx)

        if len(history_f) >= 5:
            recent = np.array(history_f[-5:])
            if np.max(np.abs(np.diff(recent))) < 1e-10:
                break

    return {
        "x": x,
        "alpha": float(np.exp(x)),
        "fun": float(history_f[-1]),
        "nfev": evals,
        "history_x": np.array(history_x),
        "history_f": np.array(history_f),
    }
```

```
spsa_hist = EvalHistory(values=[], xvals=[])
def spsa_obj(log_alpha_arr):
    log_alpha = float(np.atleast_1d(log_alpha_arr)[0])
    alpha = np.exp(log_alpha)
    E = energy_expectation(alpha, H, r)
    spsa_hist.count += 1
    spsa_hist.values.append(E)
    spsa_hist.xvals.append(alpha)
    return E

spsa_res = spsa_optimize(
    spsa_obj,
    x0=log_alpha_init,
    max_evals=max_eval_budget,
    a=0.15,
    c=0.10,
    A=15,
    alpha=0.602,
    gamma=0.101,
    seed=seed,
)

alpha_spsa = float(spsa_res["alpha"])
E_spsa = float(spsa_res["fun"])

# ------------------------------------------------------------
# 7) Validation, errors, and report
# ------------------------------------------------------------
E_fd_err_vs_exact = E0_fd - E_exact
E_cobyla_err_vs_fd = E_cobyla - E0_fd
E_spsa_err_vs_fd = E_spsa - E0_fd

hermitian_check = np.linalg.norm(H - H.T)
expected_shape = H.shape

print("=== Hydrogen radial finite-difference Hamiltonian ===")
print(f"Grid: {n_interior} interior points, cutoff Rmax = {R_max} bohr, spacing h = {h:.12f}")
print(f"Hamiltonian shape: {expected_shape}")
print(f"Hermitian check ||H - H^T||_F = {hermitian_check:.3e}")
print()

print("=== Classical diagonalization benchmark ===")
print(f"Lowest finite-difference eigenvalue E0_fd = {E0_fd:.12f} Ha")
print(f"Analytical hydrogen benchmark E_exact   = {E_exact:.12f} Ha")
print(f"Finite-representation error             = {E_fd_err_vs_exact:.12f} Ha")
print(f"Relative error vs exact                 = {rel_error(E0_fd, E_exact):.6e}")
print()

print("=== Variational optimization ===")
print("Trial family: u_alpha(r) = normalized[r * exp(-alpha * r)] on the discrete radial grid")
print(f"Initial alpha = {alpha0:.6f}")
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
print()

print("--- COBYLA ---")
print(f"status              : {res_cobyla.status}")
print(f"message             : {res_cobyla.message}")
print(f"nfev                : {res_cobyla.nfev}")
print(f"optimized alpha     : {alpha_cobyla:.12f}")
print(f"final energy        : {E_cobyla:.12f} Ha")
print(f"error vs exact      : {E_cobyla - E_exact:.12f} Ha")
print(f"error vs FD ground  : {E_cobyla_err_vs_fd:.12f} Ha")
print()

print("--- SPSA ---")
print(f"nfev                : {spsa_res['nfev']}")
print(f"optimized alpha     : {alpha_spsa:.12f}")
print(f"final energy        : {E_spsa:.12f} Ha")
print(f"error vs exact      : {E_spsa - E_exact:.12f} Ha")
print(f"error vs FD ground  : {E_spsa_err_vs_fd:.12f} Ha")
print()

# ------------------------------------------------------------
# 8) Convergence plot
# ------------------------------------------------------------
plt.figure(figsize=(8, 5))
plt.plot(cobyla_hist.values, label="COBYLA evaluations", linewidth=2)
plt.plot(spsa_hist.values, label="SPSA evaluations", linewidth=2, alpha=0.85)
plt.axhline(E_exact, color="k", linestyle="--", linewidth=1.5, label="Exact hydrogen benchmark (-0.5 Ha)")
plt.axhline(E0_fd, color="gray", linestyle=":", linewidth=1.8, label="Finite-difference ground energy")
plt.xlabel("Objective evaluation count")
plt.ylabel("Energy (Ha)")
plt.title("Variational optimization of atomic hydrogen ground state")
plt.legend()
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

# ------------------------------------------------------------
# 9) Handoff summary for Part II
# ------------------------------------------------------------
handoff = {
   "hamiltonian_construction": {
      "model": "Non-relativistic fixed-proton hydrogen atom in atomic units",
      "operator": "H = -1/2 d^2/dr^2 - 1/r for the l=0 radial equation in u(r)",
      "grid": {
         "interior_points": n_interior,
         "cutoff_bohr": R_max,
         "spacing": h,
         "boundary_conditions": "u(0)=u(Rmax)=0",
         "stencil": "three-point finite difference for second derivative",
      },
      "reconstruction_code": "H, T, V = build_radial_hamiltonian(r, h)",
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    },
    "basis_normalization": {
        "basis": "discrete radial interior grid basis",
        "normalization": "Euclidean normalization of the interior-grid amplitude vector",
    },
    "trial_state": {
        "family": "u_alpha(r) ∝ r * exp(-alpha * r)",
        "parameter": "alpha > 0, optimized via log(alpha)",
        "initial_alpha": alpha0,
    },
    "optimized_parameters": {
        "COBYLA_alpha": alpha_cobyla,
        "COBYLA_log_alpha": float(res_cobyla.x[0]),
        "SPSA_alpha": alpha_spsa,
        "SPSA_log_alpha": float(spsa_res["x"]),
    },
    "validation": {
        "E_exact": E_exact,
        "E_fd": E0_fd,
        "E_cobyla": E_cobyla,
        "E_spsa": E_spsa,
        "fd_error_vs_exact": E_fd_err_vs_exact,
        "cobyla_error_vs_exact": E_cobyla - E_exact,
        "spsa_error_vs_exact": E_spsa - E_exact,
        "hermitian_norm": hermitian_check,
    }
}

print("=== Handoff summary ===")
for k, v in handoff.items():
    print(f"{k}: {v}")
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

Executing the generated Part I implementation

The generated Part I implementation was supplied as a formatted code block and executed in Google Colab.

Generated code in the Google Colab notebook

```python
# Part I: Classical variational optimization for atomic hydrogen ground-state energy
# Grounded in the supplied context:
# - User-provided prior context describing the target formulation
# - The request itself, specifying radial finite-difference specs and output requirements
# This cell is self-contained and executable after installing numpy, scipy, matplotlib.

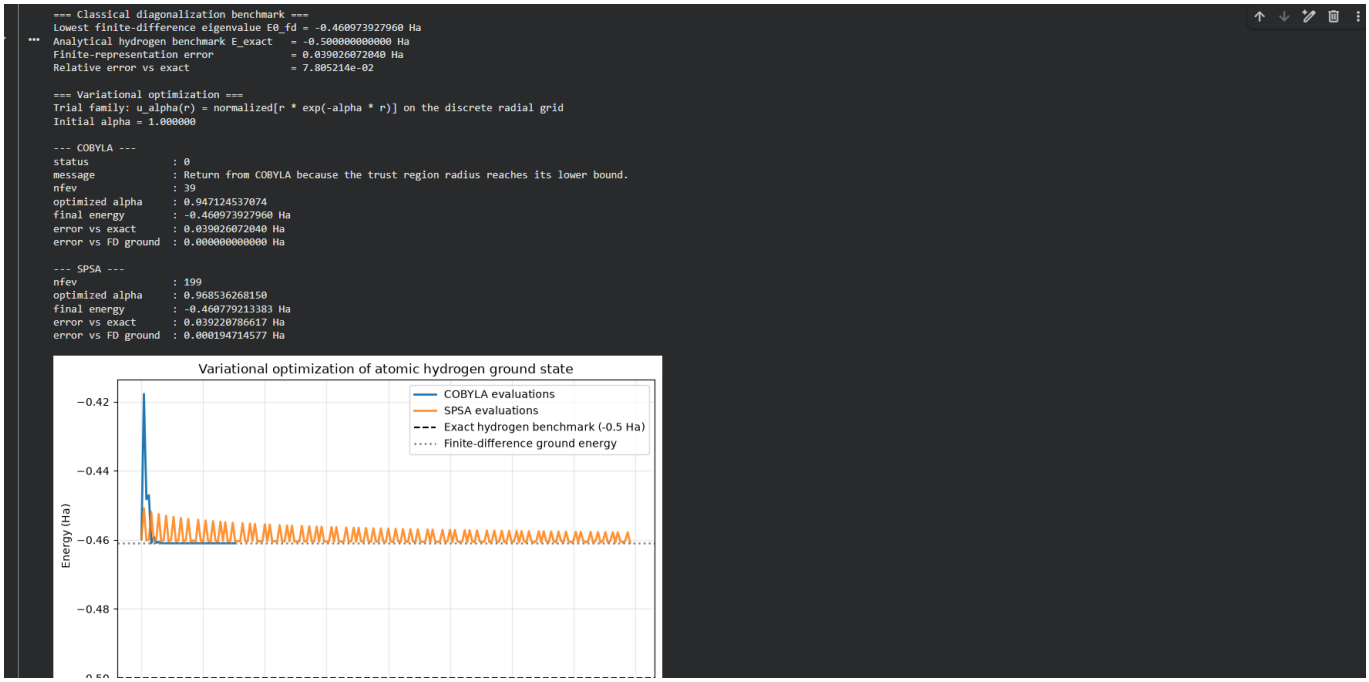
import numpy as np
import matplotlib.pyplot as plt
from dataclasses import dataclass
from scipy.linalg import eigh
from scipy.optimize import minimize

np.set_printoptions(precision=16, suppress=False, linewidth=160)

# ------------------------------------------------------------
# 1) Problem formulation: non-relativistic fixed-proton hydrogen
# ------------------------------------------------------------
#
# Atomic units: ħ = m_e = e = 1
# For the s-wave radial equation in terms of u(r) = r R(r):
#     [ -1/2 d^2/dr^2 - 1/r ] u(r) = E u(r)
# with boundary conditions u(0)=0 and u(Rmax)=0.
#
# Finite-difference domain:
#   - 32 interior points
#   - cutoff Rmax = 20 bohr
#   - zero Dirichlet boundaries
#   - three-point second-derivative stencil

R_max = 20.0
n_interior = 32
```

Recorded Google Colab results



Runtime results (verbatim)

```
=== Hydrogen radial finite-difference Hamiltonian ===
Grid: 32 interior points, cutoff Rmax = 20.0 bohr, spacing h = 0.606060606061
Hamiltonian shape: (32, 32)
Hermitian check ||H - H^T||_F = 0.000e+00

=== Classical diagonalization benchmark ===
Lowest finite-difference eigenvalue E0_fd = -0.460973927960 Ha
```

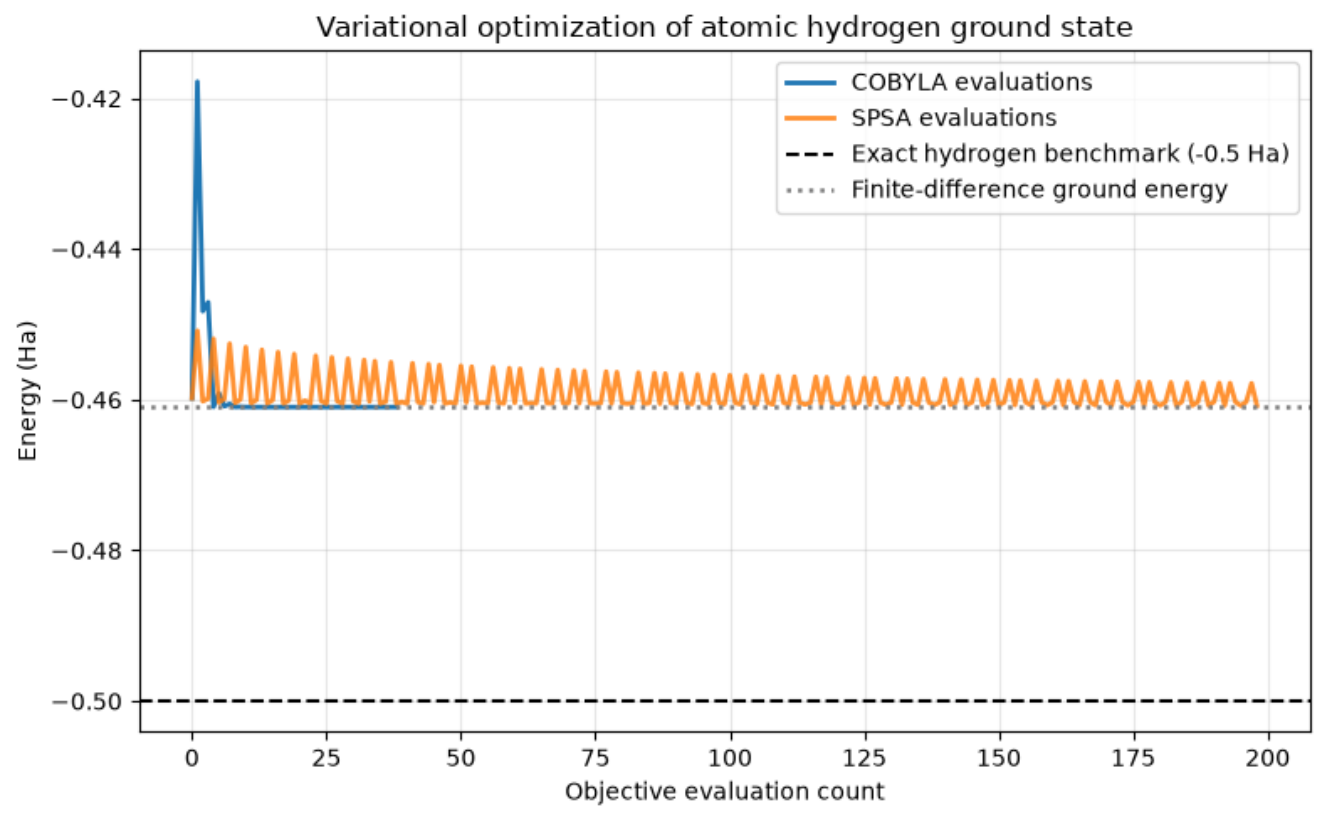
Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

Analytical hydrogen benchmark E_exact = -0.500000000000 Ha
Finite-representation error = 0.039026072040 Ha
Relative error vs exact = 7.805214e-02

=== Variational optimization ===
Trial family: u_alpha(r) = normalized[r * exp(-alpha * r)] on the discrete radial grid
Initial alpha = 1.000000

--- COBYLA ---
status : 0
message : Return from COBYLA because the trust region radius reaches its lower bound.
nfev : 39
optimized alpha : 0.947124537074
final energy : -0.460973927960 Ha
error vs exact : 0.039026072040 Ha
error vs FD ground : 0.000000000000 Ha

--- SPSA ---
nfev : 199
optimized alpha : 0.968536268150
final energy : -0.460779213383 Ha
error vs exact : 0.039220786617 Ha
error vs FD ground : 0.000194714577 Ha



=== Handoff summary ===
hamiltonian_construction: {'model': 'Non-relativistic fixed-proton hydrogen atom in atomic units', 'operator': 'H = -1/2 d^2/dr^2 - 1/r for the l=0 radial equation in u(r)', 'grid': {'interior_points': 32, 'cutoff_bohr': 20.0, 'spacing': 0.6060606060606061, 'boundary_conditions': 'u(0)=u(Rmax)=0', 'stencil': 'three-point finite difference for second derivative'}, 'reconstruction_code': 'H, T, V = build_radial_hamiltonian(r, h)'}
basis_normalization: {'basis': 'discrete radial interior grid basis', 'normalization': 'Euclidean normalization of the

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

interior-grid amplitude vector'}
trial_state: {'family': 'u_alpha(r) ∝ r * exp(-alpha * r)', 'parameter': 'alpha > 0, optimized via log(alpha)', 'initial_alpha': 1.0}
optimized_parameters: {'COBYLA_alpha': 0.9471245370743948, 'COBYLA_log_alpha': -0.05432468750000001, 'SPSA_alpha': 0.9685362681503275, 'SPSA_log_alpha': -0.031969349081638884}
validation: {'E_exact': -0.5, 'E_fd': -0.46097392795978287, 'E_cobyla': -0.4609739279597632, 'E_spsa': -0.46077921338321903, 'fd_error_vs_exact': 0.03902607204021713, 'cobyla_error_vs_exact': 0.03902607204023678, 'spsa_error_vs_exact': 0.03922078661678097, 'hermitian_norm': np.float64(0.0)}

Part I conclusion

Part I established the finite numerical benchmark before circuit synthesis or physical-QPU execution. The generated implementation constructed a real symmetric 32 × 32 radial Hamiltonian using 32 interior points, a 20-bohr cutoff, and h = 20/33 ≈ 0.606060606061 bohr. The reported Frobenius norm of $H - H^T$ was zero. This verifies the matrix-symmetry property in that calculation; correctness of the physical discretization also depends on the independently specified stencil, potential, boundary conditions, and normalization convention.

Classical diagonalization gave $E_{0,fd}$ = −0.46097392795978287 Ha. Its difference from the continuum fixed-proton benchmark is +0.03902607204021713 Ha, approximately 7.805% of |−0.5 Ha|. This is a representation difference for the chosen grid and finite domain, not an error that subsequent QPU measurement or further optimization of the same matrix can remove. The finite-matrix variational principle and Hamiltonian averaging provide the relevant framework for the state comparisons [25].

The trial amplitudes are $\psi_j(\alpha) = r_j \exp(-\alpha r_j)/\sqrt{\Sigma_k r_k^2 \exp(-2\alpha r_k)}$, evaluated on the 32 interior points and normalized in the Euclidean norm. Optimization uses log(α) to enforce α > 0. The boundary values outside this interior vector are fixed to zero by the finite-difference operator. The sampled exponential is not asserted to be an exact continuum wavefunction satisfying the finite-radius endpoint condition. This definition fixes the state to be reproduced by the five-qubit circuit.

COBYLA returned α = 0.9471245370743948 and energy −0.4609739279597632 Ha after 39 reported objective evaluations. Its approximately 1.97×10^{-14} Ha residual relative to the diagonalized finite matrix is numerical agreement to floating-point precision, not a proof of exact equality. The termination message records the trust-region radius reaching its lower bound. The result supports successful classical minimization for this finite instance and trial family; it does not establish comparable performance on arbitrary Hamiltonians.

SPSA returned α = 0.9685362681503275 and energy −0.46077921338321903 Ha after 199 reported objective evaluations, leaving a finite-model residual of approximately 0.194715 mHa. COBYLA therefore used fewer evaluations and obtained the smaller residual in these particular runs. This is not a general optimizer ranking. In this deterministic one-parameter objective, the SPSA ±1 perturbation merely swaps the two centered-difference evaluations, so its gradient estimate does not probe the high-dimensional or measurement-noise setting for which stochastic optimization is often used. The printed SPSA result lacks an explicit termination status; it is reported as the final obtained approximation, not as separately verified convergence.

The residual relative to continuum hydrogen is dominated by the representation difference. No grid- or cutoff-convergence study is reported, and agreement with the finite-matrix minimum does not certify continuum accuracy. The plotted SPSA trace includes objective evaluations at perturbed parameter values as well as updated iterates, so its oscillations must not be interpreted as oscillations of the final accepted state energy alone. The two optimized finite-state energies differ by only about 0.000194715 Ha, far below the hardware sampling uncertainties in the subsequent runs.

These results are classical variational optimization and numerical-reference checks, not quantum-circuit simulation or QPU measurements. They establish the authoritative Hamiltonian, state normalization, basis ordering, optimized parameters, and expectation-value targets used in Part II. The diagonalization is an alternative calculation of the finite-matrix reference, not independent experimental replication of the entire workflow.

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

Part I achieved the intended handoff: a specified finite hydrogen model and two numerically evaluated trial states. Part II tests whether the generated circuits preserve that specification under ideal compilation and whether their physical measurements reproduce the corresponding expectation values. Success in the first test and success in the second are reported separately.

Mathematical handoff and interpretation

For $j = 1,\ldots,32$, $r_j = jh$ and $h = 20/33$. The implemented matrix has $H_{jj} = 1/h^2 - 1/r_j$ and $H_{j,j+1} = H_{j+1,j} = -1/(2h^2)$, with zero endpoint values. The index j maps to computational-basis index $k = j - 1$. This direct spatial amplitude encoding is not a molecular orbital or many-electron second-quantized calculation; atomic hydrogen is not H_2.

For each normalized fixed vector ψ, $E_{ref} = \psi^\dagger H \psi$ is the noiseless target, while $E_{0,fd}$ is the minimum eigenvalue of the finite matrix. The difference $E_{ref} - E_{0,fd}$ is a variational residual. Numerical diagonalization and exact statevector calculations here mean evaluation without Monte Carlo shot noise, subject to floating-point precision. The general VQE framework combines quantum expectation measurements with classical parameter updates [25, 36]; this study deliberately freezes those updates before QPU execution.

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

Part II – Quantum-Circuit Synthesis, Local Validation, Real-Hardware Evaluation, and Optimization

Part II uses the complete Part I implementation and its recorded results as the scientific specification. The 32 amplitudes are mapped directly to five computational-basis qubits, with the Hamiltonian, ordering, normalization, and full-precision optimized parameters unchanged. Here, “logical qubits” means algorithm-level wires before transpilation; no error-corrected logical-qubit encoding is used. The 156-wire backend representation is a device-width convention, not a 156-qubit hydrogen problem.

The recorded experiments use **ibm_fez**, **ibm_kingston**, and **ibm_marrakesh** backends, identified separately in each execution record. Their compiled operations and logical-to-physical-to-classical mappings are checked before submission. Part II evaluates the fixed Part I states rather than driving a new VQE optimizer with quantum measurements. Hardware results are compared with the corresponding numerically evaluated fixed-state energies. Backend changes, calibrations, and run conditions limit cross-run comparisons; ideal simulation agreement does not establish physical accuracy.

Simulation scope: Statevector calculations and StatevectorSampler provide noiseless state/expectation checks and ideal finite-shot sampling. Run 1 additionally uses a custom matrix-product-state contraction for compiled-circuit validation; later versions use compact active-wire statevectors. The larger Run 5 implementation also introduces an approximate noisy density-matrix model for local candidate selection. Run 6 instead uses a calibration/exposure scoring proxy and explicitly does not claim a density-matrix prediction. These local computations are not measured QPU outputs, and the later scoring models are not independently validated replicas of the devices.

Generating and refining Part II through Interactive Flow

The Part II request included the generated Part I code, its recorded output, and full-precision handoff values. Those materials supplied the numerical specification; the corpus supplied methodological context. The transcript also records substantial experimenter constraints and follow-up guidance, including the request for structure-aware sequential preparation. Quantum Flow’s contribution is therefore assessed as assisted generation and refinement within that human-guided process.

The handoff entry H, T, V = build_radial_hamiltonian(r, h) is a function call rather than a definition. Including the function, grid construction, and normalized trial-state calculation makes the interface reconstructable. Subsequent validation recalculates expectation values from those inputs; rounded printed reference constants are not substitutes for the calculations.

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

Recorded experiment overview

Table 1 summarizes the run-specific record. Counts labelled "circuit executions" in archived output are treated here as circuit instances or PUB entries, excluding shot repetition; the total requested shots count the individual repeated measurements. No cross-run pooling or double-counting of repeated offline closure reports is performed.

Run	Recorded backend	Jobs / circuit instances	Requested shots	Outcome within the recorded scope
1	ibm_fez	4 / 68	278,528	Generic preparation; validated ideal baseline; large hardware discrepancy.
2	ibm_kingston	12 / 204	835,584	Three variants; 46 two-qubit gates retained; mixed hardware results.
3	ibm_fez	16 / 272	1,114,112	Sequential synthesis: 46 → 10 gates; best observed within-run energy improvement.
4	ibm_marrakesh	24 / 528	753,664	Four blocks; small compiler-cost reduction; modest readout correction.
5	ibm_marrakesh	115 / 6,858 planned	1,113,600 planned	Cancelled; seven saved-batch entries (0–6) in the preserved output; no final estimator.
6	ibm_marrakesh	9 / 1,014	249,984	Compact screening completed; accuracy target not met; corrected/ZNE inference rejected.

Table 1. Recorded workloads and scope. For Run 5 the full totals are planned, not completed. This table is a narrative synthesis of the corresponding verbatim run records, not a replacement for them. Job count, shots, wall-clock queue time, and QPU usage are different metrics: IBM defines usage by the time the QPU is locked for the workload, with per-job/sub-job overhead and repetition delays [33].

The acceptance hierarchy is: correct finite-model specification; ideal state and measurement equivalence; executable native circuits; reproducible estimates from actual counts; and, separately, accepted hardware accuracy. A valid result may be a failed accuracy target or a rejected correction. A passing validation gate is evidence for that gate's specific property, not proof that every subsequent inference is correct.

Part II generation prompt sent to Quantum Flow (verbatim)

```
Create Part II – Quantum-Circuit Synthesis, Simulator Validation, and Physical-QPU Evaluation for the atomic-hydrogen experiment defined by Part I below.

Generate Part II from scratch as one complete executable Jupyter cell that runs immediately after Part I.

Do not assume that any previous Part II code, quantum circuits, observables, ansätze, helper functions, Sampler objects, Estimator objects, backends, services, or quantum-specific variables exist.

Treat the supplied Part I implementation, its runtime results, and its handoff summary as the authoritative scientific specification.

Target environment:
- Qiskit 2.5.2
- qiskit-ibm-runtime 0.50.0
- IBM Quantum
- backend: ibm_fez
- channel: ibm_quantum_platform
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

PART I → PART II INTERFACE

Part II may use only these existing Part I definitions:

- H
- r
- h
- n_interior
- R_max
- E_exact
- E0_fd
- alpha_cobyla
- alpha_spsa
- normalized_trial_state

Everything else required for Part II must be defined inside the new Part II cell.

At the beginning of Part II, verify that these required Part I definitions exist and that:
- H has shape 32 × 32;
- r contains 32 elements;
- H is Hermitian within a tight numerical tolerance;
- alpha_cobyla and alpha_spsa are finite and positive.

Then perform the following workflow.

1. RECONSTRUCT THE TWO OPTIMIZED STATES

Using the Part I trial-state function, reconstruct:

psi_cobyla = normalized_trial_state(alpha_cobyla, r)
psi_spsa = normalized_trial_state(alpha_spsa, r)

Verify that both states:
- contain exactly 32 amplitudes;
- contain finite values;
- are normalized.

Independently calculate their classical expectation values directly from H.

The expected Part I validation references are approximately:

COBYLA:
-0.4609739279597632 Ha

SPSA:
-0.4607792133832190 Ha

These values are validation references only. Recalculate them from the supplied Part I objects; do not hard-code them as outputs.

2. CREATE THE FIVE-QUBIT QUANTUM REPRESENTATION

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

Because the finite Hamiltonian has dimension 32 = 2^5, represent the problem using exactly five logical qubits.

Map the 32-component Part I state vectors directly to the five-qubit computational basis while preserving the Part I basis ordering.

Construct the required state-preparation circuits inside Part II.

Do not introduce another variational ansatz such as EfficientSU2, TwoLocal, or an unrelated parameterized circuit.

The optimization has already been performed in Part I. Part II must faithfully represent and evaluate the resulting fixed optimized states.

Independently verify that the prepared quantum states reproduce the original 32-component Part I vectors up to physically irrelevant global phase.

3. DERIVE THE PAULI HAMILTONIAN

Construct the complete five-qubit Pauli representation directly from H.

Do not depend on any pre-existing observable or Hamiltonian object.

Validate the representation by reconstructing its dense 32 × 32 matrix and comparing it with H.

Require a tight numerical tolerance.

Report:
- Pauli-term count;
- Hamiltonian reconstruction error.

Evaluate the complete Pauli Hamiltonian on both prepared states and verify that the resulting energies reproduce the independently calculated Part I classical energies.

4. BUILD THE MEASUREMENT WORKFLOW

Use one consistent counts-based Sampler workflow.

Do not use Estimator for Part II.

Partition the Pauli Hamiltonian into valid qubit-wise commuting measurement groups suitable for local Pauli-basis measurements.

Create all groups, basis labels, measurement rotations, measurements, and result-decoding functions inside Part II.

For Qiskit Pauli labels, use the documented Qiskit convention that the rightmost Pauli character corresponds to qubit 0.

For each logical qubit:
- X measurement: apply H before measurement;

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

- Y measurement: apply Sdg followed by H before measurement;
- Z measurement: no basis rotation.

Explicitly measure logical qubit q[j] into classical bit c[j].

Maintain one consistent convention across:
- Pauli labels;
- logical qubits;
- computational-basis indices;
- Statevector probabilities;
- classical registers;
- returned count strings.

Do not blindly reverse state amplitudes, Pauli labels, probability vectors, or returned bitstrings.

5. INDEPENDENT DETERMINISTIC VALIDATION

Before any IBM Quantum connection or hardware submission, independently validate the complete measurement implementation.

For every measurement group calculate:

A. the group expectation directly from the intended group SparsePauliOp acting on the unrotated circuit-prepared Statevector;

and independently:

B. the same group expectation reconstructed from the exact output probabilities of the corresponding rotated measurement circuit using the same measurement semantics intended for hardware execution.

Require A and B to agree within 1e-10 Ha.

Verify that:
- every original Pauli term occurs exactly once across the measurement groups;
- no term is duplicated;
- no term is missing;
- every coefficient is preserved;
- the identity contribution is counted exactly once.

Then sum all exact measurement-group contributions.

Require:

sum(groups)_COBYLA == direct Pauli energy COBYLA within 1e-10 Ha

sum(groups)_SPSA == direct Pauli energy SPSA within 1e-10 Ha

The resulting values should reproduce approximately:

COBYLA:
-0.4609739279597632 Ha

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

SPSA:
-0.4607792133832190 Ha

These are validation references, not hard-coded calculated results.

If any deterministic validation fails, stop execution before IBM Quantum access.

Do not bypass failures using fallback logic or relaxed tolerances.

6. LOCAL FINITE-SHOT SIMULATION

Using exactly the same measurement groups, basis rotations, and outcome-decoding implementation intended for hardware execution, perform seeded local finite-shot sampling.

Use:
4096 shots per measurement circuit.

For COBYLA and SPSA report:
- classical Part I energy;
- direct exact Pauli energy;
- exact measurement-reconstructed energy;
- locally sampled energy;
- sampling standard error.

Calculate sampling uncertainty from the complete weighted outcomes of each measurement group so covariance between jointly measured Pauli terms is retained.

Do not require finite-shot values to equal exact values exactly. Evaluate them statistically.

7. CONNECT TO IBM QUANTUM ONLY AFTER LOCAL VALIDATION PASSES

After all deterministic and local validation succeeds:

Connect to IBM Quantum using:
- channel="ibm_quantum_platform"
- backend="ibm_fez"

Request the IBM Quantum API token securely through hidden runtime input if required.

Never hard-code, print, log, or save the token.

Do not silently choose another backend if ibm_fez is unavailable.

8. COMPILE THE ACTUAL HARDWARE MEASUREMENT CIRCUITS

For every state and measurement group, construct the complete circuit:

state preparation
→ required basis rotation
→ explicit measurement.

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

Transpile these complete circuits for ibm_fez using a supported Qiskit workflow.

The logical problem uses five qubits, while the compiled circuit may report the full physical width of ibm_fez. Do not treat that width difference as an error by itself.

Preserve the actual:
logical qubit
→ physical qubit
→ classical bit
mapping.

Validate the same compiled circuits that will later be submitted.

Use a tractable validation method that does not allocate a dense 156-qubit Statevector.

Compare the ideal measurement probabilities of each logical measurement circuit with the corresponding compiled circuit after correctly accounting for physical placement and classical-bit mapping.

Report the maximum compiled probability discrepancy separately for COBYLA and SPSA.

Require a tight numerical tolerance.

If compiled validation fails, stop before hardware submission.

Do not compile a second unchecked copy of the circuits for actual submission.

9. HARDWARE WORKLOAD

Use a documented counts-based Sampler job-mode implementation compatible with the installed qiskit-ibm-runtime 0.50.0 environment.

Resolve the correct supported Sampler import/API from the supplied SDK documentation.

Do not use Estimator.

Do not use Session.

Use:
- backend: ibm_fez
- shots per measurement circuit: 4096
- complete repetitions per state: 2

For each complete repetition, execute all measurement groups for that state.

Batch compatible circuits into one Sampler job where supported.

The intended experiment should therefore normally consist of:
- 2 complete COBYLA repetitions;
- 2 complete SPSA repetitions;
- 4 Sampler jobs total when one state/repetition is batched per job.

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

If the final documented API requires a materially different job structure, report the reason before submission.

Before submitting anything, print:
- backend;
- logical-qubit count;
- physical backend width;
- Pauli-term count;
- measurement-group count;
- complete repetitions;
- Sampler job count;
- total circuit executions;
- shots per circuit;
- total requested shots.

Require the user to type YES before submitting the hardware experiment.

10. HARDWARE RESULT RECONSTRUCTION

Retrieve the actual returned measurement data using the documented result schema for the selected Sampler implementation.

Use the actual classical register created by the submitted circuits.

For every complete repetition independently:
- reconstruct every measurement-group contribution from its returned outcomes;
- preserve covariance between jointly measured terms;
- sum all groups into one complete energy estimate.

Do not sum complete repetitions together as though they were Hamiltonian terms.

For each state, calculate:
- individual repetition energies;
- mean hardware energy across the two complete repetitions;
- sampling standard error of the mean;
- between-repetition standard deviation.

Do not report unavailable uncertainty as zero.

Sampling uncertainty must be explicitly identified as sampling uncertainty and must not be described as including systematic hardware error.

11. FINAL COMPARISON

For COBYLA and SPSA report separately:

- analytical hydrogen benchmark E_exact;
- finite-Hamiltonian reference E0_fd;
- Part I classical variational energy;
- exact Pauli/circuit energy;
- exact measurement-reconstructed energy;

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

- local finite-shot simulated energy and sampling uncertainty;
- each individual hardware repetition;
- mean hardware energy;
- hardware sampling standard error;
- between-repetition standard deviation.

Also report separately:

finite-representation difference:
E0_fd - E_exact

classical variational residual:
E_classical - E0_fd

quantum implementation difference:
E_exact_measurement - E_classical

hardware difference:
E_hardware_mean - E_exact_measurement

Do not combine these scientifically different quantities into one error.

12. SCIENTIFIC SCOPE

Part I performed the classical variational optimization.

Part II evaluates those same optimized states through:
- quantum-state representation;
- exact circuit simulation;
- finite-shot simulation;
- physical-QPU measurement.

Part II is NOT a new QPU-driven VQE optimizer loop.

Do not perform additional parameter optimization.

Do not create another ansatz.

Do not alter the optimized parameters to improve hardware agreement.

Do not force the physical-QPU result toward the simulator/reference value through offsets, hidden postselection, relabelling, or parameter modification.

Report the measured hardware result honestly, including any hardware-to-simulator discrepancy.

13. REPRODUCIBILITY

Report:
- NumPy version;
- SciPy version;
- Qiskit version;

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

- qiskit-ibm-runtime version;
- backend;
- logical-qubit count;
- physical backend width;
- Pauli-term count;
- measurement-group count;
- shots per circuit;
- repetitions;
- job IDs;
- exact simulator results;
- finite-shot simulator results;
- hardware results;
- uncertainty estimates.

Where practical, save:
- submitted compiled circuits;
- returned counts;
- job IDs;
- execution configuration;
- final numerical summary

to a timestamped evidence directory.

Do not save credentials.

14. STRICT PART I DEPENDENCY SWEEP

Immediately before returning the generated Part II code, inspect every external variable referenced by the code.

The ONLY Part I variables that Part II is allowed to reference are:

H
r
h
n_interior
R_max
E_exact
E0_fd
alpha_cobyla
alpha_spsa
normalized_trial_state

Every other symbol used by Part II must either:
- be imported inside Part II; or
- be defined earlier inside the same Part II cell.

Specifically verify that the generated code contains NO dependency on undefined variables such as:

ansatz
ansatz1
ansatz2

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

ansatz3
observable
observable1
observable2
H_pauli
hamiltonian
qc
circuit
sampler
estimator
backend
service
psi_cobyla
psi_spsa

unless that variable is first explicitly created inside Part II itself.

If any external dependency other than the ten permitted Part I symbols remains, correct the code before returning it.

15. OUTPUT

Return only one complete executable Jupyter-compatible Python cell for Part II.

Do not return:
- pseudocode;
- a template with placeholders;
- an audit script;
- a diagnostic-only script;
- disconnected patches;
- another request for previous Part II code.

Part II is being generated from scratch.

Use the exact marker:

Initial code produced by Stellora.AI Quantum Flow

at the beginning of the generated cell.

===== BEGIN AUTHORITATIVE PART I IMPLEMENTATION =====

```
# Part I: Classical variational optimization for atomic hydrogen ground-state energy
# Grounded in the supplied context:
# - User-provided prior context describing the target formulation
# - The request itself, specifying radial finite-difference specs and output requirements
# This cell is self-contained and executable after installing numpy, scipy, matplotlib.

import numpy as np
import matplotlib.pyplot as plt
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
from dataclasses import dataclass
from scipy.linalg import eigh
from scipy.optimize import minimize

np.set_printoptions(precision=16, suppress=False, linewidth=160)

# ------------------------------------------------------------
# 1) Problem formulation: non-relativistic fixed-proton hydrogen
# ------------------------------------------------------------
#
# Atomic units: ħ = m_e = e = 1
# For the s-wave radial equation in terms of u(r) = r R(r):
#     [ -1/2 d^2/dr^2 - 1/r ] u(r) = E u(r)
# with boundary conditions u(0)=0 and u(Rmax)=0.
#
# Finite-difference domain:
#   - 32 interior points
#   - cutoff Rmax = 20 bohr
#   - zero Dirichlet boundaries
#   - three-point second-derivative stencil

R_max = 20.0
n_interior = 32
h = R_max / (n_interior + 1)
r = np.array([(i + 1) * h for i in range(n_interior)], dtype=float)

def build_radial_hamiltonian(r_grid, dr):
    n = len(r_grid)
    main = np.full(n, 1.0 / dr**2)
    off = np.full(n - 1, -0.5 / dr**2)
    T = np.diag(main) + np.diag(off, 1) + np.diag(off, -1)
    V = np.diag(-1.0 / r_grid)
    H = T + V
    return H, T, V

H, T, V = build_radial_hamiltonian(r, h)

# ------------------------------------------------------------
# 2) Classical diagonalization benchmark
# ------------------------------------------------------------
evals, evecs = eigh(H)
E0_fd = float(evals[0])
psi0_fd = evecs[:, 0]

# Exact hydrogen benchmark in atomic units
E_exact = -0.5

def rel_error(x, ref):
    return abs(x - ref) / abs(ref)

# ------------------------------------------------------------
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
# 3) Parameterized normalized trial-state family
# ------------------------------------------------------------
#
# Choose a physically meaningful 1-parameter positive ansatz:
#     u_alpha(r) ∝ r * exp(-alpha * r), alpha > 0
# It satisfies u(0)=0 in the continuum limit and resembles the hydrogenic form
# while not being initialized from the exact eigenvector.
# We optimize in log(alpha) space to enforce positivity.

def normalized_trial_state(alpha, r_grid):
    raw = r_grid * np.exp(-alpha * r_grid)
    norm = np.linalg.norm(raw)
    if norm == 0.0 or not np.isfinite(norm):
        raise ValueError("Invalid trial-state norm.")
    return raw / norm

def energy_expectation(alpha, Hmat, r_grid):
    psi = normalized_trial_state(alpha, r_grid)
    return float(psi @ (Hmat @ psi))

alpha0 = 1.0
log_alpha_init = np.log(alpha0)

# ------------------------------------------------------------
# 4) Progress tracking utilities
# ------------------------------------------------------------
@dataclass
class EvalHistory:
    values: list
    xvals: list
    count: int = 0

def make_objective(Hmat, r_grid, history=None):
    def obj(log_alpha_arr):
        log_alpha = float(np.atleast_1d(log_alpha_arr)[0])
        alpha = np.exp(log_alpha)
        E = energy_expectation(alpha, Hmat, r_grid)
        if history is not None:
            history.count += 1
            history.values.append(E)
            history.xvals.append(alpha)
        return E
    return obj

# ------------------------------------------------------------
# 5) COBYLA optimization
# ------------------------------------------------------------
max_eval_budget = 200
tolerance = 1e-10
seed = 7
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
cobyla_hist = EvalHistory(values=[], xvals=[])
cobyla_obj = make_objective(H, r, cobyla_hist)

res_cobyla = minimize(
    cobyla_obj,
    x0=np.array([log_alpha_init]),
    method="COBYLA",
    options={
        "maxiter": max_eval_budget,
        "rhobeg": 0.25,
        "tol": tolerance,
        "disp": False,
    },
)

alpha_cobyla = float(np.exp(res_cobyla.x[0]))
E_cobyla = float(res_cobyla.fun)
psi_cobyla = normalized_trial_state(alpha_cobyla, r)

# ------------------------------------------------------------
# 6) SPSA optimization
# ------------------------------------------------------------
# Standard SPSA update:
#   g_k = [f(x + c_k Δ_k) - f(x - c_k Δ_k)] / (2 c_k Δ_k)
# with Δ_k in {+1, -1}. We keep a comparable objective-evaluation budget.

def spsa_optimize(obj, x0, max_evals=200, a=0.15, c=0.10, A=15, alpha=0.602, gamma=0.101, seed=7):
    rng = np.random.default_rng(seed)
    x = float(x0)
    history_x = []
    history_f = []
    evals = 0

    f0 = obj([x])
    evals += 1
    history_x.append(np.exp(x))
    history_f.append(f0)

    k = 0
    while evals + 2 <= max_evals:
        k += 1
        ak = a / ((k + A) ** alpha)
        ck = c / (k ** gamma)

        delta = rng.choice([-1.0, 1.0])
        f_plus = obj([x + ck * delta])
        f_minus = obj([x - ck * delta])
        evals += 2

        ghat = (f_plus - f_minus) / (2.0 * ck * delta)
        x = x - ak * ghat
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        fx = obj([x])
        evals += 1
        history_x.append(np.exp(x))
        history_f.append(fx)

        if len(history_f) >= 5:
            recent = np.array(history_f[-5:])
            if np.max(np.abs(np.diff(recent))) < 1e-10:
                break

    return {
        "x": x,
        "alpha": float(np.exp(x)),
        "fun": float(history_f[-1]),
        "nfev": evals,
        "history_x": np.array(history_x),
        "history_f": np.array(history_f),
    }

spsa_hist = EvalHistory(values=[], xvals=[])
def spsa_obj(log_alpha_arr):
    log_alpha = float(np.atleast_1d(log_alpha_arr)[0])
    alpha = np.exp(log_alpha)
    E = energy_expectation(alpha, H, r)
    spsa_hist.count += 1
    spsa_hist.values.append(E)
    spsa_hist.xvals.append(alpha)
    return E

spsa_res = spsa_optimize(
    spsa_obj,
    x0=log_alpha_init,
    max_evals=max_eval_budget,
    a=0.15,
    c=0.10,
    A=15,
    alpha=0.602,
    gamma=0.101,
    seed=seed,
)

alpha_spsa = float(spsa_res["alpha"])
E_spsa = float(spsa_res["fun"])

# ------------------------------------------------------------
# 7) Validation, errors, and report
# ------------------------------------------------------------
E_fd_err_vs_exact = E0_fd - E_exact
E_cobyla_err_vs_fd = E_cobyla - E0_fd
E_spsa_err_vs_fd = E_spsa - E0_fd
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
hermitian_check = np.linalg.norm(H - H.T)
expected_shape = H.shape

print("=== Hydrogen radial finite-difference Hamiltonian ===")
print(f"Grid: {n_interior} interior points, cutoff Rmax = {R_max} bohr, spacing h = {h:.12f}")
print(f"Hamiltonian shape: {expected_shape}")
print(f"Hermitian check ||H - H^T||_F = {hermitian_check:.3e}")
print()

print("=== Classical diagonalization benchmark ===")
print(f"Lowest finite-difference eigenvalue E0_fd = {E0_fd:.12f} Ha")
print(f"Analytical hydrogen benchmark E_exact   = {E_exact:.12f} Ha")
print(f"Finite-representation error             = {E_fd_err_vs_exact:.12f} Ha")
print(f"Relative error vs exact                 = {rel_error(E0_fd, E_exact):.6e}")
print()

print("=== Variational optimization ===")
print("Trial family: u_alpha(r) = normalized[r * exp(-alpha * r)] on the discrete radial grid")
print(f"Initial alpha = {alpha0:.6f}")
print()

print("--- COBYLA ---")
print(f"status             : {res_cobyla.status}")
print(f"message            : {res_cobyla.message}")
print(f"nfev               : {res_cobyla.nfev}")
print(f"optimized alpha    : {alpha_cobyla:.12f}")
print(f"final energy       : {E_cobyla:.12f} Ha")
print(f"error vs exact     : {E_cobyla - E_exact:.12f} Ha")
print(f"error vs FD ground : {E_cobyla_err_vs_fd:.12f} Ha")
print()

print("--- SPSA ---")
print(f"nfev               : {spsa_res['nfev']}")
print(f"optimized alpha    : {alpha_spsa:.12f}")
print(f"final energy       : {E_spsa:.12f} Ha")
print(f"error vs exact     : {E_spsa - E_exact:.12f} Ha")
print(f"error vs FD ground : {E_spsa_err_vs_fd:.12f} Ha")
print()

# ------------------------------------------------------------
# 8) Convergence plot
# ------------------------------------------------------------
plt.figure(figsize=(8, 5))
plt.plot(cobyla_hist.values, label="COBYLA evaluations", linewidth=2)
plt.plot(spsa_hist.values, label="SPSA evaluations", linewidth=2, alpha=0.85)
plt.axhline(E_exact, color="k", linestyle="--", linewidth=1.5, label="Exact hydrogen benchmark (-0.5 Ha)")
plt.axhline(E0_fd, color="gray", linestyle=":", linewidth=1.8, label="Finite-difference ground energy")
plt.xlabel("Objective evaluation count")
plt.ylabel("Energy (Ha)")
plt.title("Variational optimization of atomic hydrogen ground state")
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
plt.legend()
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

# ------------------------------------------------------------
# 9) Handoff summary for Part II
# ------------------------------------------------------------
handoff = {
    "hamiltonian_construction": {
        "model": "Non-relativistic fixed-proton hydrogen atom in atomic units",
        "operator": "H = -1/2 d^2/dr^2 - 1/r for the l=0 radial equation in u(r)",
        "grid": {
            "interior_points": n_interior,
            "cutoff_bohr": R_max,
            "spacing": h,
            "boundary_conditions": "u(0)=u(Rmax)=0",
            "stencil": "three-point finite difference for second derivative",
        },
        "reconstruction_code": "H, T, V = build_radial_hamiltonian(r, h)",
    },
    "basis_normalization": {
        "basis": "discrete radial interior grid basis",
        "normalization": "Euclidean normalization of the interior-grid amplitude vector",
    },
    "trial_state": {
        "family": "u_alpha(r) ∝ r * exp(-alpha * r)",
        "parameter": "alpha > 0, optimized via log(alpha)",
        "initial_alpha": alpha0,
    },
    "optimized_parameters": {
        "COBYLA_alpha": alpha_cobyla,
        "COBYLA_log_alpha": float(res_cobyla.x[0]),
        "SPSA_alpha": alpha_spsa,
        "SPSA_log_alpha": float(spsa_res["x"]),
    },
    "validation": {
        "E_exact": E_exact,
        "E_fd": E0_fd,
        "E_cobyla": E_cobyla,
        "E_spsa": E_spsa,
        "fd_error_vs_exact": E_fd_err_vs_exact,
        "cobyla_error_vs_exact": E_cobyla - E_exact,
        "spsa_error_vs_exact": E_spsa - E_exact,
        "hermitian_norm": hermitian_check,
    }
}

print("=== Handoff summary ===")
for k, v in handoff.items():
    print(f"{k}: {v}")
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```

===== END AUTHORITATIVE PART I IMPLEMENTATION =====


===== BEGIN AUTHORITATIVE PART I RUNTIME OUTPUT =====

```
 === Hydrogen radial finite-difference Hamiltonian ===
Grid: 32 interior points, cutoff Rmax = 20.0 bohr, spacing h = 0.606060606061
Hamiltonian shape: (32, 32)
Hermitian check ||H - H^T||_F = 0.000e+00

=== Classical diagonalization benchmark ===
Lowest finite-difference eigenvalue E0_fd = -0.460973927960 Ha
Analytical hydrogen benchmark E_exact   = -0.500000000000 Ha
Finite-representation error   = 0.039026072040 Ha
Relative error vs exact   = 7.805214e-02

=== Variational optimization ===
Trial family: u_alpha(r) = normalized[r * exp(-alpha * r)] on the discrete radial grid
Initial alpha = 1.000000

--- COBYLA ---
status   : 0
message   : Return from COBYLA because the trust region radius reaches its lower bound.
nfev   : 39
optimized alpha   : 0.947124537074
final energy   : -0.460973927960 Ha
error vs exact   : 0.039026072040 Ha
error vs FD ground : 0.000000000000 Ha

--- SPSA ---
nfev   : 199
optimized alpha   : 0.968536268150
final energy   : -0.460779213383 Ha
error vs exact   : 0.039220786617 Ha
error vs FD ground : 0.000194714577 Ha

=== Handoff summary ===
hamiltonian_construction: {'model': 'Non-relativistic fixed-proton hydrogen atom in atomic units', 'operator': 'H
= -1/2 d^2/dr^2 - 1/r for the l=0 radial equation in u(r)', 'grid': {'interior_points': 32, 'cutoff_bohr': 20.0, 'spacing':
0.6060606060606061, 'boundary_conditions': 'u(0)=u(Rmax)=0', 'stencil': 'three-point finite difference for
second derivative'}, 'reconstruction_code': 'H, T, V = build_radial_hamiltonian(r, h)'}
basis_normalization: {'basis': 'discrete radial interior grid basis', 'normalization': 'Euclidean normalization of the
interior-grid amplitude vector'}
trial_state: {'family': 'u_alpha(r) ∝ r * exp(-alpha * r)', 'parameter': 'alpha > 0, optimized via log(alpha)',
'initial_alpha': 1.0}
optimized_parameters: {'COBYLA_alpha': 0.9471245370743948, 'COBYLA_log_alpha': -
0.05432468750000001, 'SPSA_alpha': 0.9685362681503275, 'SPSA_log_alpha': -0.031969349081638884}
validation: {'E_exact': -0.5, 'E_fd': -0.46097392795978287, 'E_cobyla': -0.4609739279597632, 'E_spsa': -
```
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

0.46077921338321903, 'fd_error_vs_exact': 0.03902607204021713, 'cobyla_error_vs_exact': 0.03902607204023678, 'spsa_error_vs_exact': 0.03922078661678097, 'hermitian_norm': np.float64(0.0)}

```

====== END AUTHORITATIVE PART I RUNTIME OUTPUT ======
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

Generated Part II code (verbatim)

```
# Initial code produced by Stellora.AI Quantum Flow

_part_ii_required = (
    "H", "r", "h", "n_interior", "R_max", "E_exact", "E0_fd",
    "alpha_cobyla", "alpha_spsa", "normalized_trial_state",
)
_part_ii_missing = [name for name in _part_ii_required if name not in globals()]
if _part_ii_missing:
    raise NameError("Missing Part I definitions: " + ", ".join(_part_ii_missing))


def _run_hydrogen_part_ii(
    H, r, h, n_interior, R_max, E_exact, E0_fd,
    alpha_cobyla, alpha_spsa, normalized_trial_state,
):
    import getpass
    import inspect
    import json
    import math
    from collections import Counter
    from datetime import datetime, timezone
    from importlib.metadata import version
    from pathlib import Path

    import numpy as np
    import scipy
    import qiskit
    from qiskit import ClassicalRegister, QuantumCircuit, QuantumRegister, qpy
    from qiskit.circuit.library import StatePreparation
    from qiskit.primitives import StatevectorSampler
    from qiskit.quantum_info import Operator, SparsePauliOp, Statevector
    from qiskit.transpiler.preset_passmanagers import generate_preset_pass_manager
    from qiskit_ibm_runtime import QiskitRuntimeService, SamplerV2
    from qiskit_ibm_runtime.accounts.exceptions import AccountNotFoundError

    NQ = 5
    SHOTS = 4096
    REPETITIONS = 2
    REGISTER = "hydrogen"
    BACKEND_NAME = "ibm_fez"
    CHANNEL = "ibm_quantum_platform"
    LOCAL_SEED = 20261012
    TRANSPILE_SEED = 731
    MATRIX_TOL = 1e-12
    ENERGY_TOL = 1e-10
    PROBABILITY_TOL = 1e-10

    def require(condition, message):
        if not bool(condition):
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        raise RuntimeError(message)

def real_scalar(value, description, tolerance=MATRIX_TOL):
    value = complex(value)
    require(
        np.isfinite(value.real) and np.isfinite(value.imag),
        f"{description} is nonfinite.",
    )
    require(
        abs(value.imag) <= tolerance,
        f"{description} has an unexpected imaginary component: {value}.",
    )
    return float(value.real)

# Validate the authoritative interface; no other Part I symbols are read.
hmat = np.asarray(H, dtype=complex)
grid = np.asarray(r, dtype=float)
require(hmat.shape == (32, 32), "H must have shape (32, 32).")
require(grid.ndim == 1 and grid.size == 32, "r must contain 32 elements.")
require(np.all(np.isfinite(hmat)), "H contains nonfinite values.")
require(np.all(np.isfinite(grid)), "r contains nonfinite values.")
hermitian_error = float(np.max(np.abs(hmat - hmat.conj().T)))
require(hermitian_error <= MATRIX_TOL, "H is not tightly Hermitian.")
require(int(n_interior) == 32, "n_interior must be 32.")
require(np.isfinite(h) and float(h) > 0, "h must be finite and positive.")
require(np.isfinite(R_max) and float(R_max) > 0, "R_max must be positive.")
require(callable(normalized_trial_state), "Trial-state function is not callable.")

for name, alpha in (("COBYLA", alpha_cobyla), ("SPSA", alpha_spsa)):
    require(
        np.ndim(alpha) == 0 and np.isfinite(alpha) and float(alpha) > 0,
        f"{name} alpha must be a finite positive scalar.",
    )

analytical_energy = real_scalar(E_exact, "E_exact")
fd_energy = real_scalar(E0_fd, "E0_fd")

versions = {
    "numpy": np.__version__,
    "scipy": scipy.__version__,
    "qiskit": qiskit.__version__,
    "qiskit_ibm_runtime": version("qiskit-ibm-runtime"),
}
require(versions["qiskit"] == "2.5.2", "This cell requires Qiskit 2.5.2.")
require(
    versions["qiskit_ibm_runtime"] == "0.50.0",
    "This cell requires qiskit-ibm-runtime 0.50.0.",
)

# Inspect and preserve the installed SDK's public signatures/docstrings.
# The reference URL is an identifier, not a claim of a fetched web page.
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
constructor_signature = inspect.signature(SamplerV2)
run_signature = inspect.signature(SamplerV2.run)
require(
    "mode" in constructor_signature.parameters,
    "Installed SamplerV2 lacks its required public mode argument.",
)
require(
    "pubs" in run_signature.parameters and "shots" in run_signature.parameters,
    "Installed SamplerV2.run lacks the required pubs/shots interface.",
)
api_evidence = {
    "import": "from qiskit_ibm_runtime import SamplerV2",
    "execution": (
        "SamplerV2(mode=backend, options=...).run(circuits, shots=4096)"
    ),
    "constructor_signature": str(constructor_signature),
    "run_signature": str(run_signature),
    "class_docstring": inspect.getdoc(SamplerV2),
    "run_docstring": inspect.getdoc(SamplerV2.run),
    "result_schema": (
        "PrimitiveResult -> SamplerPubResult.data.hydrogen "
        "-> BitArray.get_counts()"
    ),
    "reference_url": (
        "https://quantum.cloud.ibm.com/docs/en/api/"
        "qiskit-ibm-runtime/sampler-v2"
    ),
    "evidence_note": (
        "Installed SDK signatures and docstrings inspected; URL not fetched."
    ),
}

print("=== Part II: fixed optimized-state evaluation; no new VQE loop ===")
print("Versions:", versions)
print("Sampler:", SamplerV2.__module__ + "." + SamplerV2.__name__)
print("Constructor:", constructor_signature)
print("Run interface:", run_signature)

psi_cobyla = np.asarray(
    normalized_trial_state(alpha_cobyla, grid), dtype=complex
)
psi_spsa = np.asarray(
    normalized_trial_state(alpha_spsa, grid), dtype=complex
)
states = {"COBYLA": psi_cobyla, "SPSA": psi_spsa}
alphas = {"COBYLA": float(alpha_cobyla), "SPSA": float(alpha_spsa)}
reference_energies = {
    "COBYLA": -0.4609739279597632,
    "SPSA": -0.4607792133832190,
}
preparations = {}
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
statevectors = {}
results = {}

for name, amplitudes in states.items():
    require(
        amplitudes.ndim == 1 and amplitudes.size == 32,
        f"{name}: expected exactly 32 amplitudes.",
    )
    require(
        np.all(np.isfinite(amplitudes)),
        f"{name}: nonfinite amplitudes.",
    )
    require(
        abs(real_scalar(np.vdot(amplitudes, amplitudes), "State norm") - 1)
        <= MATRIX_TOL,
        f"{name}: state is not normalized.",
    )
    classical_energy = real_scalar(
        np.vdot(amplitudes, hmat @ amplitudes),
        f"{name} classical energy",
    )

    preparation = QuantumCircuit(
        NQ, name=f"{name.lower()}_preparation"
    )
    # Amplitude index k maps directly to |k>; no reversal is performed.
    preparation.append(StatePreparation(amplitudes), range(NQ))
    prepared = Statevector.from_instruction(preparation)
    overlap = np.vdot(amplitudes, prepared.data)
    require(
        abs(overlap) > 0,
        f"{name}: zero state-preparation overlap.",
    )
    global_phase = overlap / abs(overlap)
    preparation_error = float(
        np.linalg.norm(prepared.data - global_phase * amplitudes)
    )
    require(
        preparation_error <= MATRIX_TOL,
        f"{name}: state preparation does not reproduce the Part I state.",
    )
    preparations[name] = preparation
    statevectors[name] = prepared
    results[name] = {
        "alpha": alphas[name],
        "classical_energy": classical_energy,
        "validation_reference_only": reference_energies[name],
        "difference_from_validation_reference": (
            classical_energy - reference_energies[name]
        ),
        "state_preparation_error_up_to_global_phase": preparation_error,
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    }

# Full expansion, with no intentional coefficient thresholding.
pauli_hamiltonian = SparsePauliOp.from_operator(
    Operator(hmat), atol=0.0, rtol=0.0
)
original_terms = pauli_hamiltonian.to_list()
labels = [label for label, coefficient in original_terms]
require(
    len(labels) == len(set(labels)),
    "Duplicate original Pauli labels.",
)
for label, coefficient in original_terms:
    real_scalar(coefficient, f"Coefficient {label}")

reconstruction_error = float(
    np.max(np.abs(pauli_hamiltonian.to_matrix() - hmat))
)
require(
    reconstruction_error <= MATRIX_TOL,
    "Dense Pauli reconstruction does not reproduce H.",
)

groups = pauli_hamiltonian.group_commuting(qubit_wise=True)
grouped_terms = [term for group in groups for term in group.to_list()]
require(
    Counter(label for label, coefficient in grouped_terms)
    == Counter(labels),
    "Measurement groups contain missing or duplicated terms.",
)
original_coefficients = dict(original_terms)
for label, coefficient in grouped_terms:
    require(
        coefficient == original_coefficients[label],
        f"Grouping changed the coefficient of {label}.",
    )
require(
    labels.count("IIIII") == 1
    and sum(
        label == "IIIII" for label, coefficient in grouped_terms
    ) == 1,
    "Identity must occur exactly once.",
)

# In a count string c[4]...c[0], int(bitstring, 2) has bit j = c[j].
# Qiskit Pauli labels have their rightmost character on logical qubit 0.
outcomes = np.arange(1 << NQ, dtype=np.uint64)
group_metadata = []

for group_index, group in enumerate(groups):
    terms = group.to_list()
```

```
    axes = ["I"] * NQ

    for label, coefficient in terms:
        require(len(label) == NQ, "Incorrect Pauli-label width.")
        for j in range(NQ):
            axis = label[NQ - 1 - j]
            if axis != "I":
                require(
                    axes[j] in ("I", axis),
                    f"Group {group_index} is not qubit-wise commuting.",
                )
                axes[j] = axis

    axes = ["Z" if axis == "I" else axis for axis in axes]
    weights = np.zeros(1 << NQ, dtype=float)

    for label, coefficient in terms:
        parity = np.zeros(1 << NQ, dtype=np.uint64)
        for j in range(NQ):
            if label[NQ - 1 - j] != "I":
                parity ^= (outcomes >> j) & 1
        weights += real_scalar(coefficient, label) * (
            1.0 - 2.0 * parity.astype(float)
        )

    group_metadata.append({
        "index": group_index,
        "basis_label": "".join(reversed(axes)),
        "basis_by_logical_qubit": axes,
        "terms": [
            {
                "label": label,
                "coefficient": real_scalar(coefficient, label),
            }
            for label, coefficient in terms
        ],
        "weighted_outcomes": weights,
    })

def make_measurement_circuit(name, group_index):
    qreg = QuantumRegister(NQ, "q")
    creg = ClassicalRegister(NQ, REGISTER)
    measured = QuantumCircuit(
        qreg, creg, name=f"{name.lower()}_group_{group_index:03d}"
    )
    measured.compose(
        preparations[name],
        qubits=list(qreg),
        inplace=True,
    )
    for j, axis in enumerate(
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        group_metadata[group_index]["basis_by_logical_qubit"]
    ):
        if axis == "X":
            measured.h(qreg[j])
        elif axis == "Y":
            measured.sdg(qreg[j])
            measured.h(qreg[j])
        else:
            require(axis == "Z", "Invalid measurement axis.")

    for j in range(NQ):
        measured.measure(qreg[j], creg[j])

    return measured

def extract_counts(pub_result):
    require(
        hasattr(pub_result.data, REGISTER),
        f"Returned PUB lacks classical register {REGISTER!r}.",
    )
    bit_array = getattr(pub_result.data, REGISTER)
    require(
        bit_array.num_bits == NQ,
        "Returned register width is not five.",
    )
    return bit_array.get_counts()

def decode_counts(counts, group_index, expected_shots):
    require(isinstance(counts, dict) and bool(counts), "Missing counts.")
    weights = group_metadata[group_index]["weighted_outcomes"]
    entries = []
    total = 0

    for bitstring, count in counts.items():
        require(
            isinstance(bitstring, str)
            and len(bitstring) == NQ
            and set(bitstring).issubset({"0", "1"}),
            f"Unexpected count key: {bitstring!r}.",
        )
        require(
            isinstance(count, (int, np.integer)) and count >= 0,
            "Invalid count multiplicity.",
        )
        # No bitstring reversal.
        value = float(weights[int(bitstring, 2)])
        entries.append((int(count), value))
        total += int(count)

    require(
        total == expected_shots,
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        "Returned shot count differs from request.",
    )
    require(
        total > 1,
        "At least two shots are needed for sampling uncertainty.",
    )

    mean = math.fsum(
        count * value for count, value in entries
    ) / total

    # Each value is the COMPLETE weighted group outcome, retaining
    # covariance between jointly measured terms.
    centered_sum = math.fsum(
        count * (value - mean) ** 2 for count, value in entries
    )
    variance_of_mean = centered_sum / (total * (total - 1))

    return {
        "energy": float(mean),
        "sampling_variance": float(variance_of_mean),
        "sampling_standard_error": math.sqrt(variance_of_mean),
        "shots": total,
    }

logical_circuits = {}
ideal_probabilities = {}

# Independent deterministic checks: all precede IBM Quantum access.
for name in states:
    direct_energy = real_scalar(
        statevectors[name].expectation_value(pauli_hamiltonian),
        f"{name} direct Pauli energy",
    )
    require(
        abs(direct_energy - results[name]["classical_energy"])
        <= ENERGY_TOL,
        f"{name}: direct Pauli/classical energy mismatch.",
    )

    logical_circuits[name] = []
    ideal_probabilities[name] = []
    exact_contributions = []
    largest_discrepancy = 0.0

    for group_index, group in enumerate(groups):
        measured = make_measurement_circuit(name, group_index)
        logical_circuits[name].append(measured)
        rotated = measured.remove_final_measurements(inplace=False)
        probabilities = Statevector.from_instruction(
            rotated
```

```
        ).probabilities()

        require(
            abs(float(np.sum(probabilities)) - 1) <= MATRIX_TOL,
            "Invalid logical probability normalization.",
        )
        ideal_probabilities[name].append(probabilities)

        direct_group = real_scalar(
            statevectors[name].expectation_value(group),
            f"{name} group {group_index} direct expectation",
        )
        reconstructed_group = float(np.dot(
            probabilities,
            group_metadata[group_index]["weighted_outcomes"],
        ))
        discrepancy = abs(direct_group - reconstructed_group)
        largest_discrepancy = max(largest_discrepancy, discrepancy)

        require(
            discrepancy <= ENERGY_TOL,
            f"{name} group {group_index}: "
            "independent measurement check failed.",
        )
        exact_contributions.append(reconstructed_group)

    exact_energy = float(math.fsum(exact_contributions))
    require(
        abs(exact_energy - direct_energy) <= ENERGY_TOL
        and abs(exact_energy - results[name]["classical_energy"])
        <= ENERGY_TOL,
        f"{name}: exact measurement-group sum failed validation.",
    )
    results[name].update({
        "direct_pauli_energy": direct_energy,
        "exact_measurement_energy": exact_energy,
        "exact_group_contributions": exact_contributions,
        "maximum_exact_group_discrepancy": largest_discrepancy,
    })

print("\nPauli terms:", len(original_terms))
print("Qubit-wise commuting groups:", len(groups))
print(
    f"Hamiltonian reconstruction error: {reconstruction_error:.3e}"
)
print("Term coverage, coefficient preservation, identity accounting: PASS")

for name in states:
    item = results[name]
    print(
        f"{name}: classical={item['classical_energy']:.16f}; "
```

```
        f"Pauli={item['direct_pauli_energy']:.16f}; "
        f"exact measurement={item['exact_measurement_energy']:.16f} Ha; "
        f"max group discrepancy="
        f"{item['maximum_exact_group_discrepancy']:.3e}"
    )
    print(
        "  Difference from approximate Part I validation reference: "
        f"{item['difference_from_validation_reference']:+.3e} Ha"
    )

print("Independent deterministic validation: PASS")

local_order = [
    (name, group_index)
    for name in states
    for group_index in range(len(groups))
]
local_sampler = StatevectorSampler(
    default_shots=SHOTS, seed=LOCAL_SEED
)
local_job = local_sampler.run(
    [
        logical_circuits[name][index]
        for name, index in local_order
    ],
    shots=SHOTS,
)
local_result = local_job.result()
require(
    len(local_result) == len(local_order),
    "Local PUB count mismatch.",
)

local_counts = {name: [] for name in states}
local_statistics = {name: [] for name in states}

for pub_result, (name, group_index) in zip(local_result, local_order):
    counts = extract_counts(pub_result)
    local_counts[name].append(counts)
    local_statistics[name].append(
        decode_counts(counts, group_index, SHOTS)
    )

print("\n=== Seeded local finite-shot sampling ===")

for name in states:
    energy = float(math.fsum(
        x["energy"] for x in local_statistics[name]
    ))
    variance = float(math.fsum(
        x["sampling_variance"] for x in local_statistics[name]
```

```
    ))
    se = math.sqrt(variance)
    difference = energy - results[name]["exact_measurement_energy"]
    z_score = difference / se if se > 0 else None
    consistent = (
        abs(z_score) <= 6.0
        if se > 0
        else abs(difference) <= ENERGY_TOL
    )

    results[name].update({
        "local_sampled_energy": energy,
        "local_sampling_standard_error": se,
        "local_sampling_z_score": z_score,
        "local_group_statistics": local_statistics[name],
    })
    item = results[name]

    print(
        f"{name}: classical={item['classical_energy']:.12f}; "
        f"Pauli={item['direct_pauli_energy']:.12f}; "
        f"exact measurement={item['exact_measurement_energy']:.12f}; "
        f"sampled={energy:.12f} +/- {se:.6f} Ha "
        f"(sampling SE); z={z_score}"
    )
    require(
        consistent,
        f"{name}: seeded local sampling failed the six-SE "
        "statistical check.",
    )

print("Local validation: PASS. IBM Quantum connection may now proceed.")

timestamp = datetime.now(timezone.utc).strftime(
    "%Y%m%dT%H%M%S_%fZ"
)
evidence_directory = (
    Path.cwd() / f"hydrogen_part_ii_evidence_{timestamp}"
)
evidence_directory.mkdir(parents=True, exist_ok=False)

def save_json(filename, content):
    with (evidence_directory / filename).open(
        "w", encoding="utf-8"
    ) as handle:
        json.dump(content, handle, indent=2, allow_nan=False)

configuration = {
    "timestamp_utc": timestamp,
    "versions": versions,
    "backend": BACKEND_NAME,
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    "channel": CHANNEL,
    "logical_qubits": NQ,
    "physical_backend_width": None,
    "pauli_term_count": len(original_terms),
    "measurement_group_count": len(groups),
    "shots_per_circuit": SHOTS,
    "complete_repetitions_per_state": REPETITIONS,
    "intended_sampler_job_count": len(states) * REPETITIONS,
    "total_circuit_executions": (
        len(states) * REPETITIONS * len(groups)
    ),
    "total_requested_shots": (
        len(states) * REPETITIONS * len(groups) * SHOTS
    ),
    "local_seed": LOCAL_SEED,
    "transpiler_seed": TRANSPILE_SEED,
    "optimization_level": 2,
    "energy_tolerance": ENERGY_TOL,
    "matrix_tolerance": MATRIX_TOL,
    "compiled_probability_tolerance": PROBABILITY_TOL,
    "grid": {
        "n_interior": int(n_interior),
        "h": float(h),
        "R_max": float(R_max),
    },
    "processing": (
        "Raw counts; no postselection, offsets, mitigation, "
        "or optimization."
    ),
    "uncertainty_scope": (
        "Sampling uncertainty only; systematic hardware error "
        "is not included."
    ),
    "sampler_job_structure": (
        "One batched job per state and complete repetition."
    ),
}

save_json("configuration.json", configuration)
save_json("sdk_api_evidence.json", api_evidence)
save_json("local_counts.json", local_counts)
save_json("local_numerical_results.json", results)
save_json(
    "measurement_groups.json",
    [
        {
            key: value
            for key, value in item.items()
            if key != "weighted_outcomes"
        }
        for item in group_metadata
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    ],
)

np.savez_compressed(
    evidence_directory
    / "scientific_inputs_and_exact_probabilities.npz",
    H=hmat,
    r=grid,
    psi_cobyla=psi_cobyla,
    psi_spsa=psi_spsa,
    cobyla_probabilities=np.asarray(
        ideal_probabilities["COBYLA"]
    ),
    spsa_probabilities=np.asarray(
        ideal_probabilities["SPSA"]
    ),
    group_weighted_outcomes=np.asarray([
        item["weighted_outcomes"] for item in group_metadata
    ]),
)

with (
    evidence_directory / "logical_measurement_circuits.qpy"
).open("wb") as handle:
    qpy.dump(
        [
            logical_circuits[name][index]
            for name, index in local_order
        ],
        handle,
    )

# Account lookup only; no credentials are saved by this cell.
try:
    runtime_service = QiskitRuntimeService(channel=CHANNEL)
except AccountNotFoundError:
    api_token = getpass.getpass(
        "IBM Quantum API token (hidden; not saved): "
    )
    try:
        require(bool(api_token.strip()), "Empty IBM Quantum API token.")
        runtime_service = QiskitRuntimeService(
            channel=CHANNEL, token=api_token
        )
    finally:
        del api_token

hardware_backend = runtime_service.backend(BACKEND_NAME)
require(
    hardware_backend.name == BACKEND_NAME,
    "Unexpected backend returned.",
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
)
require(
    hardware_backend.status().operational,
    f"{BACKEND_NAME} is not operational.",
)

physical_width = int(hardware_backend.num_qubits)
configuration["physical_backend_width"] = physical_width
save_json("configuration.json", configuration)

coupling = hardware_backend.coupling_map
require(
    coupling is not None,
    "Backend does not expose its coupling map.",
)

adjacency = {
    index: set() for index in range(physical_width)
}
for first, second in coupling.get_edges():
    adjacency[first].add(second)
    adjacency[second].add(first)

def find_connected_five_site_path():
    def extend(path):
        if len(path) == NQ:
            return path
        for neighbor in sorted(adjacency[path[-1]]):
            if neighbor not in path:
                found = extend(path + [neighbor])
                if found is not None:
                    return found
        return None

    for start in sorted(
        adjacency, key=lambda j: (-len(adjacency[j]), j)
    ):
        found = extend([start])
        if found is not None:
            return found

    raise RuntimeError(
        "No connected five-site path exists on ibm_fez."
    )

initial_layout = find_connected_five_site_path()
configuration["requested_initial_layout"] = initial_layout

pass_manager = generate_preset_pass_manager(
    backend=hardware_backend,
    optimization_level=2,
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    initial_layout=initial_layout,
    seed_transpiler=TRANSPILE_SEED,
)

flat_compiled = pass_manager.run([
    logical_circuits[name][index]
    for name, index in local_order
])
require(
    len(flat_compiled) == len(local_order),
    "Compiled circuit count mismatch.",
)

compiled_circuits = {name: [] for name in states}
for compiled, (name, group_index) in zip(
    flat_compiled, local_order
):
    compiled_circuits[name].append(compiled)

def compiled_measurement_probabilities(compiled):
    """
    Simulate the actual compiled gates with a matrix-product state (MPS).

    Idle backend wires are omitted. Active routing/ancilla wires are
    retained and traced out, not postselected. No
    2**physical_width vector is allocated.

    SVD compression removes only numerical-null singular values
    <= 1e-15. Canonical environments make the accumulated discarded
    norms a bound on compression-induced state error. Floating-point
    roundoff is checked independently by normalization and
    logical-probability comparisons.
    """
    require(
        compiled.num_clbits == NQ,
        "Compiled classical width changed.",
    )
    require(
        len(compiled.cregs) == 1
        and compiled.cregs[0].name == REGISTER
        and len(compiled.cregs[0]) == NQ,
        "Compiled circuit changed its classical register.",
    )

    register = compiled.cregs[0]
    measurement_map = {}
    measured_physical = set()
    active_physical = set()
    operations = []

    for instruction in compiled.data:
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        operation = instruction.operation
        physical_args = [
            compiled.find_bit(qubit).index
            for qubit in instruction.qubits
        ]

        if operation.name == "barrier":
            continue

        if operation.name == "measure":
            require(
                len(physical_args) == 1
                and len(instruction.clbits) == 1,
                "Invalid compiled measurement.",
            )
            classical_bit = instruction.clbits[0]
            require(
                classical_bit in register,
                "Measurement uses a foreign register.",
            )
            classical_index = list(register).index(classical_bit)
            require(
                classical_index not in measurement_map
                and physical_args[0] not in measured_physical,
                "Repeated compiled measurement.",
            )
            measurement_map[classical_index] = physical_args[0]
            measured_physical.add(physical_args[0])
            active_physical.update(physical_args)
            continue

        require(
            not instruction.clbits
            and getattr(operation, "condition", None) is None,
            "Dynamic classical operations are not permitted "
            "in validation.",
        )
        require(
            not measured_physical.intersection(physical_args),
            "An operation follows measurement on the same physical wire.",
        )

        if operation.name == "delay":
            continue  # Ideal, noiseless delay is identity.

        require(
            len(physical_args) in (1, 2),
            f"Unsupported compiled operation width: {operation.name}.",
        )
        matrix = np.asarray(
            Operator(operation).data, dtype=complex
```

```
        )
        require(
            np.max(np.abs(
                matrix.conj().T @ matrix - np.eye(len(matrix))
            )) <= MATRIX_TOL,
            f"Compiled operation is nonunitary: {operation.name}.",
        )
        operations.append((matrix, physical_args))
        active_physical.update(physical_args)

    require(
        set(measurement_map) == set(range(NQ)),
        "Missing compiled measurements.",
    )
    require(
        compiled.layout is not None,
        "Compiled transpilation layout is missing.",
    )

    final_positions = [
        int(position)
        for position in compiled.layout.final_index_layout(
            filter_ancillas=True
        )
    ]
    require(
        len(final_positions) == NQ,
        "Unexpected final logical layout width.",
    )

    for j in range(NQ):
        require(
            measurement_map[j] == final_positions[j],
            "Logical/physical/classical mapping is inconsistent.",
        )

    # Start measured wires adjacent in tensor order; physical labels
    # remain explicit and follow each tensor-order permutation.
    wire_order = [measurement_map[j] for j in range(NQ)]
    wire_order += sorted(
        active_physical.difference(wire_order)
    )
    tensors = [
        np.array([1.0, 0.0], dtype=complex).reshape(1, 2, 1)
        for _ in wire_order
    ]
    discarded_norm_bound = 0.0
    maximum_bond_dimension = 1
    swap_matrix = np.eye(4, dtype=complex)[[0, 2, 1, 3], :]

    def canonicalize_around_pair(left):
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        for index in range(left):
            tensor = tensors[index]
            dl, _, dr = tensor.shape
            q_matrix, r_matrix = np.linalg.qr(
                tensor.reshape(dl * 2, dr), mode="reduced"
            )
            tensors[index] = q_matrix.reshape(
                dl, 2, q_matrix.shape[1]
            )
            tensors[index + 1] = np.einsum(
                "ab,bsc->asc",
                r_matrix,
                tensors[index + 1],
                optimize=True,
            )

        for index in range(
            len(tensors) - 1, left + 1, -1
        ):
            tensor = tensors[index]
            dl, _, dr = tensor.shape
            q_matrix, r_matrix = np.linalg.qr(
                tensor.reshape(dl, 2 * dr).T,
                mode="reduced",
            )
            tensors[index] = q_matrix.T.reshape(
                q_matrix.shape[1], 2, dr
            )
            tensors[index - 1] = np.einsum(
                "asb,bc->asc",
                tensors[index - 1],
                r_matrix.T,
                optimize=True,
            )

    def apply_adjacent(matrix, left):
        nonlocal discarded_norm_bound, maximum_bond_dimension

        canonicalize_around_pair(left)
        first, second = tensors[left], tensors[left + 1]
        dl, dr = first.shape[0], second.shape[2]
        pair = np.einsum(
            "asx,xtb->astb", first, second, optimize=True
        )

        # Qiskit gate-matrix convention:
        # the first qarg is the low-order bit.
        gate_tensor = matrix.reshape(
            2, 2, 2, 2
        ).transpose(1, 0, 3, 2)
        pair = np.einsum(
```

```
            "uvst,astb->auvb",
            gate_tensor,
            pair,
            optimize=True,
        )
        u_matrix, singular_values, vh_matrix = np.linalg.svd(
            pair.reshape(dl * 2, 2 * dr),
            full_matrices=False,
        )

        retained = max(
            1,
            int(np.count_nonzero(singular_values > 1e-15)),
        )
        discarded_norm_bound += float(
            np.linalg.norm(singular_values[retained:])
        )
        require(
            discarded_norm_bound <= 1e-12,
            "MPS numerical-rank compression exceeded "
            "its error budget.",
        )

        tensors[left] = u_matrix[:, :retained].reshape(
            dl, 2, retained
        )
        tensors[left + 1] = (
            singular_values[:retained, None]
            * vh_matrix[:retained, :]
        ).reshape(retained, 2, dr)

        maximum_bond_dimension = max(
            maximum_bond_dimension, retained
        )

    for matrix, physical_args in operations:
        if len(physical_args) == 1:
            position = wire_order.index(physical_args[0])
            tensors[position] = np.einsum(
                "ts,asb->atb",
                matrix,
                tensors[position],
                optimize=True,
            )
        else:
            first_position = wire_order.index(physical_args[0])
            second_position = wire_order.index(physical_args[1])
            low, high = sorted(
                (first_position, second_position)
            )
```

```
        # These SWAPs only change tensor ordering; they do not
        # insert physical operations into the submitted circuit.
        for position in range(high - 1, low, -1):
            apply_adjacent(swap_matrix, position)
            wire_order[position], wire_order[position + 1] = (
                wire_order[position + 1],
                wire_order[position],
            )

        left = min(
            wire_order.index(physical_args[0]),
            wire_order.index(physical_args[1]),
        )
        adjacent_matrix = (
            matrix
            if wire_order[left] == physical_args[0]
            else swap_matrix @ matrix @ swap_matrix
        )
        apply_adjacent(adjacent_matrix, left)

    physical_to_classical = {
        physical: classical
        for classical, physical in measurement_map.items()
    }
    probabilities = np.zeros(1 << NQ, dtype=float)

    for outcome in range(1 << NQ):
        environment = np.ones((1, 1), dtype=complex)
        for physical, tensor in zip(wire_order, tensors):
            if physical in physical_to_classical:
                bit = (
                    outcome >> physical_to_classical[physical]
                ) & 1
                selected = tensor[:, bit:bit + 1, :]
            else:
                # Trace over unmeasured physical wires.
                selected = tensor

            environment = np.einsum(
                "ab,asi,bsj->ij",
                environment,
                selected,
                selected.conj(),
                optimize=True,
            )

        probabilities[outcome] = real_scalar(
            environment[0, 0], "Compiled marginal probability"
        )

    require(
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        np.min(probabilities) >= -MATRIX_TOL,
        "Compiled simulation produced negative probabilities.",
    )
    require(
        abs(float(np.sum(probabilities)) - 1.0)
        <= PROBABILITY_TOL,
        "Compiled probability normalization failed.",
    )

    # No normalization adjustment, clipping, or postselection.
    return probabilities, {
        "compiled_width": compiled.num_qubits,
        "active_physical_width": len(active_physical),
        "final_logical_to_physical": final_positions,
        "logical_physical_classical_mapping": [
            {
                "logical_qubit": j,
                "physical_qubit": measurement_map[j],
                "classical_register": REGISTER,
                "classical_bit": j,
            }
            for j in range(NQ)
        ],
        "maximum_mps_bond_dimension": maximum_bond_dimension,
        "numerical_discarded_norm_bound": discarded_norm_bound,
    }

# Check the custom MPS contraction/endianness independently before
# applying it to the actual hardware-compiled circuits.
validator_qreg = QuantumRegister(NQ, "q")
validator_creg = ClassicalRegister(NQ, REGISTER)
validator_test = QuantumCircuit(
    validator_qreg, validator_creg
)
validator_test.ry(0.37, 0)
validator_test.h(2)
validator_test.cx(0, 4)
validator_test.cx(3, 1)
validator_test.rz(-0.29, 4)
validator_test.cx(4, 2)
validator_test.sdg(2)
for j in range(NQ):
    validator_test.measure(j, j)

validator_compiled = pass_manager.run(validator_test)
validator_reference = Statevector.from_instruction(
    validator_test.remove_final_measurements(inplace=False)
).probabilities()
validator_probabilities, validator_mapping = (
    compiled_measurement_probabilities(validator_compiled)
)
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
validator_discrepancy = float(np.max(np.abs(
    validator_probabilities - validator_reference
)))
require(
    validator_discrepancy <= PROBABILITY_TOL,
    "Independent compiled MPS validator self-test failed.",
)
validator_mapping["maximum_probability_discrepancy"] = (
    validator_discrepancy
)
save_json(
    "compiled_validator_self_test.json",
    validator_mapping,
)

print(
    "\n=== Validation of actual compiled submission circuits ==="
)
compiled_validation = {name: [] for name in states}

for name in states:
    maximum_discrepancy = 0.0

    for group_index, compiled in enumerate(
        compiled_circuits[name]
    ):
        probabilities, mapping = (
            compiled_measurement_probabilities(compiled)
        )
        discrepancy = float(np.max(np.abs(
            probabilities
            - ideal_probabilities[name][group_index]
        )))
        require(
            discrepancy <= PROBABILITY_TOL,
            f"{name} group {group_index}: "
            f"compiled probability mismatch {discrepancy:.3e}; "
            "no hardware job submitted.",
        )
        mapping["group_index"] = group_index
        mapping["maximum_probability_discrepancy"] = discrepancy
        compiled_validation[name].append(mapping)
        maximum_discrepancy = max(
            maximum_discrepancy, discrepancy
        )

    results[name][
        "maximum_compiled_probability_discrepancy"
    ] = maximum_discrepancy
    print(
        f"{name}: maximum compiled probability discrepancy "
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        f"= {maximum_discrepancy:.3e}"
    )

print(
    "Compiled validation: PASS. "
    "The checked circuit objects will be submitted."
)

with (
    evidence_directory / "submitted_compiled_circuits.qpy"
).open("wb") as handle:
    qpy.dump(flat_compiled, handle)

save_json(
    "compiled_validation_and_mappings.json",
    compiled_validation,
)
save_json("configuration.json", configuration)
save_json("validated_pre_hardware_results.json", results)

# Backend mode is job mode: no Session is created or used.
sampler_options = {
    "default_shots": SHOTS,
    "dynamical_decoupling": {"enable": False},
    "twirling": {
        "enable_gates": False,
        "enable_measure": False,
    },
}
hardware_sampler = SamplerV2(
    mode=hardware_backend,
    options=sampler_options,
)
configuration["sampler_options"] = sampler_options
save_json("configuration.json", configuration)

job_records = []
hardware_counts = {name: [] for name in states}
hardware_statistics = {name: [] for name in states}

def report_final(hardware_available):
    print("\n=== Final scientific comparison: energies in Ha ===")
    print(
        "Sampling uncertainties do NOT include systematic "
        "hardware error."
    )

    for name in states:
        item = results[name]
        item.update({
            "analytical_hydrogen_benchmark": analytical_energy,
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        "finite_hamiltonian_reference": fd_energy,
        "finite_representation_difference": (
            fd_energy - analytical_energy
        ),
        "classical_variational_residual": (
            item["classical_energy"] - fd_energy
        ),
        "quantum_implementation_difference": (
            item["exact_measurement_energy"]
            - item["classical_energy"]
        ),
    })

print(f"\n{name}")
print(
    "  Analytical hydrogen benchmark:       "
    f"{analytical_energy:.12f}"
)
print(
    "  Finite-Hamiltonian reference:        "
    f"{fd_energy:.12f}"
)
print(
    "  Part I classical variational energy:  "
    f"{item['classical_energy']:.12f}"
)
print(
    "  Exact Pauli/circuit energy:          "
    f"{item['direct_pauli_energy']:.12f}"
)
print(
    "  Exact measurement energy:            "
    f"{item['exact_measurement_energy']:.12f}"
)
print(
    "  Local finite-shot energy:            "
    f"{item['local_sampled_energy']:.12f}"
    f" +/- {item['local_sampling_standard_error']:.6f}"
    " (sampling SE)"
)
print(
    "  Finite-representation difference:    "
    f"{item['finite_representation_difference']:+.12f}"
)
print(
    "  Classical variational residual:      "
    f"{item['classical_variational_residual']:+.12f}"
)
print(
    "  Quantum implementation difference:   "
    f"{item['quantum_implementation_difference']:+.3e}"
```

```
        )

        if hardware_available:
            for repetition, energy in enumerate(
                item["hardware_repetition_energies"],
                start=1,
            ):
                se = item[
                    "hardware_repetition_sampling_standard_errors"
                ][repetition - 1]
                print(
                    f"  Hardware repetition {repetition}:"
                    f"               {energy:.12f}"
                    f" +/- {se:.6f} (sampling SE)"
                )

            print(
                "  Mean hardware energy:                "
                f"{item['hardware_mean_energy']:.12f}"
            )
            print(
                "  Hardware sampling SE of mean:        "
                f"{item['hardware_sampling_standard_error_of_mean']:.6f}"
            )
            print(
                "  Between-repetition standard deviation:"
                f" {item['between_repetition_standard_deviation']:.6f}"
            )
            print(
                "  Hardware difference:                 "
                f"{item['hardware_difference']:+.12f}"
            )
        else:
            item.update({
                "hardware_repetition_energies": None,
                "hardware_repetition_sampling_standard_errors": None,
                "hardware_mean_energy": None,
                "hardware_sampling_standard_error_of_mean": None,
                "between_repetition_standard_deviation": None,
                "hardware_difference": None,
            })
            print(
                "  Hardware results/uncertainties: unavailable; "
                "submission declined."
            )

    summary = {
        "configuration": configuration,
        "hermitian_error": hermitian_error,
        "hamiltonian_reconstruction_error": reconstruction_error,
        "hardware_available": hardware_available,
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        "job_records": job_records,
        "states": results,
        "evidence_directory": str(evidence_directory.resolve()),
    }
    print(
        "\nJob IDs:",
        [record["job_id"] for record in job_records],
    )
    print(
        "Evidence directory:",
        evidence_directory.resolve(),
    )
    save_json("final_numerical_summary.json", summary)
    return summary

print("\n=== Hardware workload: no hardware jobs submitted ===")
print(f"Backend:                    {BACKEND_NAME}")
print(f"Logical-qubit count:        {NQ}")
print(f"Physical backend width:     {physical_width}")
print(f"Pauli-term count:           {len(original_terms)}")
print(f"Measurement-group count:    {len(groups)}")
print(f"Complete repetitions per state: {REPETITIONS}")
print(
    "Sampler job count:          "
    f"{configuration['intended_sampler_job_count']}"
)
print(
    "Total circuit executions:   "
    f"{configuration['total_circuit_executions']}"
)
print(f"Shots per circuit:          {SHOTS}")
print(
    "Total requested shots:      "
    f"{configuration['total_requested_shots']}"
)
print(
    "Job structure: one batched job per state/repetition; "
    "four jobs total."
)
print(
    "No Session, Estimator, additional optimization, "
    "or result adjustment."
)

confirmation = input(
    "Type YES to submit this physical-QPU experiment: "
).strip()
if confirmation != "YES":
    print("Submission declined. No hardware jobs were submitted.")
    return report_final(False)
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
submitted_jobs = []

for name in states:
    for repetition in range(REPETITIONS):
        # Precisely the circuit objects validated above.
        job = hardware_sampler.run(
            compiled_circuits[name],
            shots=SHOTS,
        )
        job_id = job.job_id()
        submitted_jobs.append((name, repetition, job))
        job_records.append({
            "state": name,
            "repetition": repetition + 1,
            "job_id": job_id,
            "pub_count": len(groups),
            "shots_per_pub": SHOTS,
        })
        save_json("job_ids.json", job_records)
        print(
            f"Submitted {name} repetition "
            f"{repetition + 1}: {job_id}"
        )

for name, repetition, job in submitted_jobs:
    primitive_result = job.result()
    require(
        len(primitive_result) == len(groups),
        f"{name} repetition {repetition + 1}: "
        "returned PUB count mismatch.",
    )

    repetition_counts = []
    repetition_statistics = []

    for group_index, pub_result in enumerate(primitive_result):
        counts = extract_counts(pub_result)
        repetition_counts.append(counts)

        # Save raw returned counts before decoding/further validation.
        save_json(
            f"hardware_counts_{name.lower()}_"
            f"repetition_{repetition + 1}.json",
            {
                "job_id": job.job_id(),
                "state": name,
                "repetition": repetition + 1,
                "classical_register": REGISTER,
                "retrieved_group_count": len(repetition_counts),
                "group_counts": repetition_counts,
            },
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
            )
            repetition_statistics.append(
                decode_counts(counts, group_index, SHOTS)
            )

        hardware_counts[name].append(repetition_counts)
        hardware_statistics[name].append(repetition_statistics)
        save_json(
            f"hardware_statistics_{name.lower()}_"
            f"repetition_{repetition + 1}.json",
            repetition_statistics,
        )
        print(
            f"Retrieved {name} repetition "
            f"{repetition + 1}: {job.job_id()}"
        )

    for name in states:
        require(
            len(hardware_statistics[name]) == REPETITIONS,
            f"{name}: incomplete hardware repetitions.",
        )

        # Each repetition measures the complete Hamiltonian.
        # Repetitions are averaged, not added as Hamiltonian terms.
        repetition_energies = [
            float(math.fsum(
                group["energy"] for group in repetition
            ))
            for repetition in hardware_statistics[name]
        ]
        repetition_variances = [
            float(math.fsum(
                group["sampling_variance"]
                for group in repetition
            ))
            for repetition in hardware_statistics[name]
        ]

        mean_energy = float(np.mean(repetition_energies))
        sampling_se_of_mean = math.sqrt(
            math.fsum(repetition_variances) / REPETITIONS**2
        )

        results[name].update({
            "hardware_repetition_energies": repetition_energies,
            "hardware_repetition_sampling_standard_errors": [
                math.sqrt(variance)
                for variance in repetition_variances
            ],
            "hardware_mean_energy": mean_energy,
```

```
            "hardware_sampling_standard_error_of_mean": (
                sampling_se_of_mean
            ),
            "between_repetition_standard_deviation": float(
                np.std(repetition_energies, ddof=1)
            ),
            "hardware_difference": (
                mean_energy - results[name]["exact_measurement_energy"]
            ),
            "hardware_group_statistics": hardware_statistics[name],
        })

    save_json("all_hardware_counts.json", hardware_counts)
    return report_final(True)


# Dependency sweep:
# The function imports/defines all working symbols locally.
# Its only externally supplied scientific inputs are the ten permitted
# Part I symbols. No pre-existing quantum objects or other Part I helpers
# are accessed.
part_ii_summary = _run_hydrogen_part_ii(
    **{name: globals()[name] for name in _part_ii_required}
)
```

First-run results (verbatim)

```
=== Part II: fixed optimized-state evaluation; no new VQE loop ===
Versions: {'numpy': '2.1.3', 'scipy': '1.16.3', 'qiskit': '2.5.2', 'qiskit_ibm_runtime': '0.50.0'}
Sampler: qiskit_ibm_runtime.sampler.SamplerV2
Constructor: (mode: 'BackendV2 | Session | Batch | None' = None, options: 'dict | options.SamplerOptions |
None' = None)
Run interface: (self, pubs: 'Iterable[SamplerPubLike]', *, shots: 'int | None' = None, dry_run: 'bool' = False) ->
'RuntimeJobV2'

Pauli terms: 63
Qubit-wise commuting groups: 17
Hamiltonian reconstruction error: 1.776e-15
Term coverage, coefficient preservation, identity accounting: PASS
COBYLA: classical=-0.4609739279597634; Pauli=-0.4609739279597611; exact measurement=-
0.4609739279597616 Ha; max group discrepancy=2.665e-15
  Difference from approximate Part I validation reference: -2.220e-16 Ha
SPSA: classical=-0.4607792133832190; Pauli=-0.4607792133832159; exact measurement=-
0.4607792133832168 Ha; max group discrepancy=2.665e-15
  Difference from approximate Part I validation reference: -5.551e-17 Ha
Independent deterministic validation: PASS

=== Seeded local finite-shot sampling ===
COBYLA: classical=-0.460973927960; Pauli=-0.460973927960; exact measurement=-0.460973927960;
sampled=-0.482836305527 +/- 0.019723 Ha (sampling SE); z=-1.1084557188476123
SPSA: classical=-0.460779213383; Pauli=-0.460779213383; exact measurement=-0.460779213383; sampled=-
0.469549796695 +/- 0.020027 Ha (sampling SE); z=-0.4379482327523593
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
Local validation: PASS. IBM Quantum connection may now proceed.
qiskit_runtime_service._discover_account:WARNING:2026-10-12 23:12:44,535: Loading account with the given token. A saved account will not be used.
qiskit_runtime_service.__init__:WARNING:2026-10-12 23:12:50,298: Instance was not set at service instantiation. Free and trial plan instances will be prioritized. Based on the following filters: (tags: None, region: us-east, eu-de), and available plans: (pay-as-you-go, open), the available account instances are: open-instance, test. If you need a specific instance set it explicitly either by using a saved account with a saved default instance or passing it in directly to QiskitRuntimeService(). Alternatively, pass instance='auto' or save it to your account for auto-selection without warning.
qiskit_runtime_service.backends:WARNING:2026-10-12 23:12:50,299: Using instance: open-instance, plan: open
=== Validation of actual compiled submission circuits ===
COBYLA: maximum compiled probability discrepancy = 7.036e-15
SPSA: maximum compiled probability discrepancy = 6.162e-15
Compiled validation: PASS. The checked circuit objects will be submitted.

=== Hardware workload: no hardware jobs submitted ===
Backend:                    ibm_fez
Logical-qubit count:        5
Physical backend width:     156
Pauli-term count:           63
Measurement-group count:    17
Complete repetitions per state: 2
Sampler job count:          4
Total circuit executions:   68
Shots per circuit:          4096
Total requested shots:      278528
Job structure: one batched job per state/repetition; four jobs total.
No Session, Estimator, additional optimization, or result adjustment.
/tmp/ipykernel_985/3391451079.py:1150: DeprecationWarning: The SamplerV2 class is deprecated as of qiskit-ibm-runtime 0.50.0 and will be removed no sooner than 3 months after the release date.
`qiskit_ibm_runtime.executor_sampler.Sampler` provides a new Sampler implementation that runs all the pre- and post-processing in the client side. The top-level `qiskit_ibm_runtime.Sampler` import will be updated to point to the client-side sampler in an upcoming release, with the legacy Sampler being available until the deprecation is completed.
  hardware_sampler = SamplerV2(
Submitted COBYLA repetition 1: db6mivg4qg6s73c513h0
Submitted COBYLA repetition 2: db6mivo4qg6s73c513i0
Submitted SPSA repetition 1: db6mj0cvf2bc73d1f1n0
Submitted SPSA repetition 2: db6mj0klf4us73c5jejg
Retrieved COBYLA repetition 1: db6mivg4qg6s73c513h0
Retrieved COBYLA repetition 2: db6mivo4qg6s73c513i0
Retrieved SPSA repetition 1: db6mj0cvf2bc73d1f1n0
Retrieved SPSA repetition 2: db6mj0klf4us73c5jejg

=== Final scientific comparison: energies in Ha ===
Sampling uncertainties do NOT include systematic hardware error.

COBYLA
  Analytical hydrogen benchmark:      -0.500000000000
  Finite-Hamiltonian reference:       -0.460973927960
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
  Part I classical variational energy:  -0.460973927960
  Exact Pauli/circuit energy:           -0.460973927960
  Exact measurement energy:             -0.460973927960
  Local finite-shot energy:             -0.482836305527 +/- 0.019723 (sampling SE)
  Finite-representation difference:     +0.039026072040
  Classical variational residual:       +0.000000000000
  Quantum implementation difference:    +1.832e-15
  Hardware repetition 1:                0.047763109747 +/- 0.023655 (sampling SE)
  Hardware repetition 2:                0.035935469869 +/- 0.023530 (sampling SE)
  Mean hardware energy:                 0.041849289808
  Hardware sampling SE of mean:         0.016682
  Between-repetition standard deviation: 0.008363
  Hardware difference:                  +0.502823217768

SPSA
  Analytical hydrogen benchmark:        -0.500000000000
  Finite-Hamiltonian reference:         -0.460973927960
  Part I classical variational energy:  -0.460779213383
  Exact Pauli/circuit energy:           -0.460779213383
  Exact measurement energy:             -0.460779213383
  Local finite-shot energy:             -0.469549796695 +/- 0.020027 (sampling SE)
  Finite-representation difference:     +0.039026072040
  Classical variational residual:       +0.000194714577
  Quantum implementation difference:    +2.220e-15
  Hardware repetition 1:                0.106480443817 +/- 0.023977 (sampling SE)
  Hardware repetition 2:                0.082569352757 +/- 0.024093 (sampling SE)
  Mean hardware energy:                 0.094524898287
  Hardware sampling SE of mean:         0.016996
  Between-repetition standard deviation: 0.016908
  Hardware difference:                  +0.555304111670

Job IDs: ['db6mivg4qg6s73c513h0', 'db6mivo4qg6s73c513i0', 'db6mj0cvf2bc73d1f1n0',
'db6mj0klf4us73c5jejg']
Evidence directory: /content/hydrogen_part_ii_evidence_20261012T231224_545232Z
```

First-run conclusion

Run 1 established a working fixed-state measurement baseline on the ibm_fez backend: four jobs and 278,528 requested shots. The reported logical and compiled noiseless checks agreed to maximum probability discrepancies below 7.1×10^{-15}, and the validated circuits were reused. Raw means were +0.041849 ± 0.016682 Ha for the COBYLA-derived state and +0.094525 ± 0.016996 Ha for the SPSA-derived state; the quoted uncertainties are conditional sampling standard errors only. The approximately +0.503 Ha and +0.555 Ha deviations from the intended state energies show that ideal implementation checks and successful submission did not yield accurate physical energy estimates. The cause was not isolated, and no hardware-accuracy improvement is claimed for this baseline.

Prompt sent to Quantum Flow to refine the first code version

```
Keep this run as the unmitigated baseline and preserve Part I, both fixed states, and all working validation,
uncertainty, and evidence-saving safeguards. Add corrupted-circuit negative controls and independent
reproducible local sampling streams. Analyze per-group hardware residuals and report compiled depth,
entangling-gate counts, layouts, and relevant calibration data. Benchmark exactly equivalent state-preparation
alternatives and calibration-informed placement/routing against the baseline; demonstrate improvements rather
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

than assuming them. Preserve raw results and distinguish any mitigation explicitly. Return one complete runnable Part II cell, with computed circuit comparisons and the full workload printed before explicit QPU confirmation. Do not change the Hamiltonian, optimized parameters, or reference energies.

Second code version (verbatim)

Backend-change note: The experimenter reported changing the target to **ibm_kingston** because of utilization on the **ibm_fez** backend. The execution log identifies the actual backend; it does not independently quantify the queue conditions that motivated the change. This is a separate benchmark, so differences from Run 1 cannot be attributed solely to software revisions.

```
# Initial code produced by Stellora.AI Quantum Flow

def _hydrogen_part_ii_benchmark(
    H, r, h, n_interior, R_max, E_exact, E0_fd,
    alpha_cobyla, alpha_spsa, normalized_trial_state,
):
    import os
    import json
    import math
    import pickle
    import hashlib
    import inspect
    import getpass
    import uuid
    from pathlib import Path
    from datetime import datetime, timezone
    from collections import Counter
    from importlib.metadata import version

    import numpy as np
    import scipy
    import qiskit
    from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister, qpy
    from qiskit.circuit.library import StatePreparation
    from qiskit.quantum_info import Statevector, Operator, SparsePauliOp
    from qiskit.primitives import StatevectorSampler
    from qiskit.transpiler.preset_passmanagers import generate_preset_pass_manager
    from qiskit_ibm_runtime import QiskitRuntimeService, SamplerV2
    from qiskit_ibm_runtime.accounts.exceptions import AccountNotFoundError

    NQ, DIM, SHOTS, REPS = 5, 32, 4096, 2
    REGISTER = "hydrogen"
    BACKEND_NAME, CHANNEL = "ibm_kingston", "ibm_quantum_platform"
    TRANSPILE_SEED, LOCAL_SEED = 731, 20261012
    MATRIX_TOL, ENERGY_TOL, PROB_TOL = 1e-12, 1e-10, 1e-10
    MAX_ACTIVE_QUBITS = 20

    def require(condition, message):
        if not bool(condition):
            raise RuntimeError(message)
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
def real(value, label):
    value = complex(value)
    require(
        np.isfinite(value.real) and np.isfinite(value.imag)
        and abs(value.imag) <= MATRIX_TOL,
        f"{label}: nonfinite or unexpectedly complex value.",
    )
    return float(value.real)

def utc():
    return datetime.now(timezone.utc).isoformat()

def serializable(value):
    if isinstance(value, np.ndarray):
        return serializable(value.tolist())
    if isinstance(value, np.generic):
        return serializable(value.item())
    if isinstance(value, complex):
        return {"real": value.real, "imag": value.imag}
    if isinstance(value, datetime):
        return value.isoformat()
    if isinstance(value, dict):
        return {str(k): serializable(v) for k, v in value.items()}
    if isinstance(value, (list, tuple)):
        return [serializable(v) for v in value]
    if isinstance(value, float) and not math.isfinite(value):
        return None
    if value is None or isinstance(value, (str, int, float, bool)):
        return value
    return repr(value)

# The baseline is the actual preceding Part II cell, not the optional
# initialize/level-1 baseline: StatePreparation, complete QWC groups,
# level 2, seed 731, deterministic five-site path, no mitigation.
versions = {
    "numpy": np.__version__,
    "scipy": scipy.__version__,
    "qiskit": qiskit.__version__,
    "qiskit-ibm-runtime": version("qiskit-ibm-runtime"),
}
require(versions["qiskit"] == "2.5.2", "Requires Qiskit 2.5.2.")
require(
    versions["qiskit-ibm-runtime"] == "0.50.0",
    "Requires qiskit-ibm-runtime 0.50.0.",
)
matrix = np.array(H, dtype=complex, copy=True)
grid = np.array(r, dtype=float, copy=True)
require(matrix.shape == (DIM, DIM), "H must be 32 x 32.")
require(grid.shape == (DIM,), "r must be a 32-element vector.")
require(np.all(np.isfinite(matrix)), "H contains nonfinite entries.")
require(np.all(np.isfinite(grid)), "r contains nonfinite entries.")
```

```
    require(
        np.max(np.abs(matrix - matrix.conj().T)) <= MATRIX_TOL,
        "H is not Hermitian within tolerance.",
    )
    require(int(n_interior) == DIM, "n_interior must be 32.")
    require(np.isfinite(h) and h > 0, "Invalid h.")
    require(np.isfinite(R_max) and R_max > 0, "Invalid R_max.")
    benchmark = real(E_exact, "E_exact")
    fd_reference = real(E0_fd, "E0_fd")
    parameters = {"COBYLA": alpha_cobyla, "SPSA": alpha_spsa}
    states, classical = {}, {}

    for name, parameter in parameters.items():
        require(
            np.ndim(parameter) == 0 and np.isfinite(parameter) and parameter > 0,
            f"{name}: alpha must be finite and positive.",
        )
        vector = np.array(
            normalized_trial_state(parameter, grid.copy()),
            dtype=complex, copy=True,
        )
        require(vector.shape == (DIM,), f"{name}: invalid state dimension.")
        require(np.all(np.isfinite(vector)), f"{name}: nonfinite state.")
        require(
            abs(real(np.vdot(vector, vector), "State norm") - 1) <= MATRIX_TOL,
            f"{name}: state is not normalized.",
        )
        states[name] = vector
        classical[name] = real(np.vdot(vector, matrix @ vector), name)

    variants = [
        {
            "name": "baseline",
            "preparation": "state_preparation",
            "level": 2,
            "placement": "baseline_path",
        },
        {
            "name": "initialize_matched",
            "preparation": "initialize",
            "level": 2,
            "placement": "matched_baseline",
        },
        {
            "name": "stateprep_calibrated",
            "preparation": "state_preparation",
            "level": 3,
            "placement": "calibration_path",
        },
    ]
    names = list(states)
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
variant_names = [v["name"] for v in variants]

stamp = datetime.now(timezone.utc).strftime("%Y%m%dT%H%M%S_%fZ")
directory = Path.cwd() / f"hydrogen_part_II_{stamp}_{uuid.uuid4().hex[:8]}"
directory.mkdir(mode=0o700, exist_ok=False)

def save_json(filename, value):
    temporary = directory / f".{filename}.{uuid.uuid4().hex}.tmp"
    with temporary.open("x", encoding="utf-8") as handle:
        os.chmod(temporary, 0o600)
        json.dump(serializable(value), handle, indent=2, allow_nan=False)
        handle.flush()
        os.fsync(handle.fileno())
    os.replace(temporary, directory / filename)

def save_pickle(filename, value):
    with (directory / filename).open("xb") as handle:
        os.chmod(directory / filename, 0o600)
        pickle.dump(value, handle, protocol=pickle.HIGHEST_PROTOCOL)
        handle.flush()
        os.fsync(handle.fileno())

def save_qpy(filename, circuits):
    with (directory / filename).open("xb") as handle:
        os.chmod(directory / filename, 0o600)
        qpy.dump(circuits, handle)

manifest = {
    "created_utc": utc(),
    "versions": versions,
    "backend": BACKEND_NAME,
    "channel": CHANNEL,
    "logical_qubits": NQ,
    "shots_per_circuit": SHOTS,
    "repetitions_per_state_per_variant": REPS,
    "transpiler_seed": TRANSPILE_SEED,
    "local_seed": LOCAL_SEED,
    "tolerances": {
        "matrix": MATRIX_TOL, "energy": ENERGY_TOL,
        "probability": PROB_TOL,
    },
    "baseline": (
        "Preceding actual Part II: StatePreparation, full unpruned QWC "
        "Hamiltonian including identity, optimization level 2, seed 731, "
        "first deterministic five-site path ordered by (-degree, index)."
    ),
    "variants": variants,
    "mitigation": "NONE for every variant; raw counts, no postselection.",
    "uncertainty_scope": (
        "Sampling uncertainty retains within-group covariance; it excludes "
        "systematic device error and calibration drift."
```

```
    ),
    "reference_energies": {"E_exact": benchmark, "E0_fd": fd_reference},
    "optimized_parameters": {k: float(v) for k, v in parameters.items()},
    "classical_energies": classical,
    "grid": {
        "h": float(h), "R_max": float(R_max), "n_interior": int(n_interior)
    },
    "hamiltonian_sha256": hashlib.sha256(matrix.tobytes()).hexdigest(),
    "state_sha256": {
        k: hashlib.sha256(v.tobytes()).hexdigest() for k, v in states.items()
    },
    "jobs": [],
    "status": "local_validation",
}
save_json("manifest.json", manifest)
np.savez_compressed(
    directory / "part_I_inputs.npz",
    H=matrix, r=grid, COBYLA=states["COBYLA"], SPSA=states["SPSA"],
)
os.chmod(directory / "part_I_inputs.npz", 0o600)

paulis = SparsePauliOp.from_operator(Operator(matrix), atol=0, rtol=0)
original = paulis.to_list()
require(
    len(original) == len(set(label for label, coefficient in original)),
    "Duplicate original terms.",
)
for label, coefficient in original:
    real(coefficient, label)
reconstruction_error = float(np.max(np.abs(paulis.to_matrix() - matrix)))
require(reconstruction_error <= MATRIX_TOL, "Pauli reconstruction failed.")
operators = paulis.group_commuting(qubit_wise=True)
grouped = [term for operator in operators for term in operator.to_list()]
require(
    Counter(label for label, coefficient in grouped)
    == Counter(label for label, coefficient in original),
    "Missing or duplicated grouped terms.",
)
coefficients = dict(original)
require(
    all(coefficient == coefficients[label] for label, coefficient in grouped),
    "Grouping changed coefficients.",
)
require(
    sum(label == "IIIII" for label, coefficient in original) == 1
    and sum(label == "IIIII" for label, coefficient in grouped) == 1,
    "Identity must be counted exactly once.",
)

groups = []
for index, operator in enumerate(operators):
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    axes = ["I"] * NQ
    score = np.zeros(DIM)
    for label, coefficient in operator.to_list():
        mask = 0
        for q in range(NQ):
            axis = label[NQ - 1 - q]
            if axis != "I":
                require(axes[q] in ("I", axis), "Invalid QWC group.")
                axes[q] = axis
                mask |= 1 << q
        score += real(coefficient, label) * np.array([
            -1.0 if (outcome & mask).bit_count() % 2 else 1.0
            for outcome in range(DIM)
        ])
    axes = ["Z" if axis == "I" else axis for axis in axes]
    groups.append({
        "index": index,
        "basis": "".join(reversed(axes)),
        "axes": axes,
        "terms": operator.to_list(),
        "score": score,
    })

def preparation(vector, method):
    circuit = QuantumCircuit(NQ)
    if method == "state_preparation":
        circuit.append(StatePreparation(vector), range(NQ))
    elif method == "initialize":
        circuit.initialize(vector, range(NQ))
    else:
        raise RuntimeError("Unknown state preparation.")
    return circuit

def measurement(vector, method, group, override_axes=None):
    qr = QuantumRegister(NQ, "q")
    cr = ClassicalRegister(NQ, REGISTER)
    circuit = QuantumCircuit(qr, cr)
    circuit.compose(preparation(vector, method), inplace=True)
    axes = group["axes"] if override_axes is None else override_axes
    for q, axis in enumerate(axes):
        if axis == "X":
            circuit.h(q)
        elif axis == "Y":
            circuit.sdg(q)
            circuit.h(q)
        else:
            require(axis == "Z", "Invalid axis.")
    for q in range(NQ):
        circuit.measure(qr[q], cr[q])
    return circuit
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
def assert_probability(actual, expected, label):
    actual = np.asarray(actual)
    require(
        actual.shape == (DIM,) and np.all(np.isfinite(actual))
        and np.min(actual) >= -MATRIX_TOL,
        f"{label}: invalid probabilities.",
    )
    require(abs(float(actual.sum()) - 1) <= PROB_TOL, f"{label}: invalid norm.")
    error = float(np.max(np.abs(actual - expected)))
    require(error <= PROB_TOL, f"{label}: probability discrepancy {error:.3e}.")
    return error

def state_check(actual, expected):
    overlap = np.vdot(expected, actual)
    require(abs(overlap) > 0, "Zero state overlap.")
    error = float(np.linalg.norm(actual - overlap / abs(overlap) * expected))
    require(error <= MATRIX_TOL, "Prepared state differs beyond global phase.")
    return error

ideal, exact, logical, deterministic = {}, {}, {}, []
for name, vector in states.items():
    intended = Statevector.from_instruction(preparation(vector, "state_preparation"))
    state_check(intended.data, vector)
    direct = real(intended.expectation_value(paulis), "Direct Pauli energy")
    require(abs(direct - classical[name]) <= ENERGY_TOL, "Classical/Pauli mismatch.")
    ideal[name] = []
    contributions = []
    for gi, group in enumerate(groups):
        circuit = measurement(vector, "state_preparation", group)
        probabilities = Statevector.from_instruction(
            circuit.remove_final_measurements(inplace=False)
        ).probabilities()
        group_direct = real(
            intended.expectation_value(operators[gi]), "Direct group energy"
        )
        group_measured = float(probabilities @ group["score"])
        require(
            abs(group_direct - group_measured) <= ENERGY_TOL,
            f"{name}/group-{gi}: independent deterministic check failed.",
        )
        ideal[name].append(probabilities)
        contributions.append(group_measured)
    exact[name] = {
        "direct_pauli": direct,
        "measurement": math.fsum(contributions),
        "group_contributions": contributions,
    }
    require(
        abs(exact[name]["measurement"] - direct) <= ENERGY_TOL,
        "Exact group sum failed.",
    )
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    for variant in variants:
        method = variant["preparation"]
        state_check(Statevector.from_instruction(preparation(vector, method)).data, vector)
        logical[(variant["name"], name)] = []
        for gi, group in enumerate(groups):
            circuit = measurement(vector, method, group)
            error = assert_probability(
                Statevector.from_instruction(
                    circuit.remove_final_measurements(inplace=False)
                ).probabilities(),
                ideal[name][gi],
                f"{variant['name']}/{name}/logical/{gi}",
            )
            logical[(variant["name"], name)].append(circuit)
            deterministic.append({
                "variant": variant["name"], "state": name,
                "group": gi, "probability_error": error,
            })

negative_controls = []

def expect_failure(function, label):
    try:
        function()
    except RuntimeError:
        negative_controls.append({"control": label, "detected": True})
    else:
        raise RuntimeError(f"Negative control was not detected: {label}")

vector = states["COBYLA"]
corrupted_candidates = []
for q in range(NQ):
    for gate in ("x", "z"):
        circuit = preparation(vector, "state_preparation")
        getattr(circuit, gate)(q)
        corrupted = Statevector.from_instruction(circuit).data
        corrupted_candidates.append((
            1 - abs(np.vdot(vector, corrupted)) ** 2, q, gate, corrupted
        ))
distance, q, gate, corrupted = max(corrupted_candidates, key=lambda x: x[0])
require(distance > ENERGY_TOL, "No meaningful preparation corruption found.")
expect_failure(
    lambda: state_check(corrupted, vector), f"corrupted-preparation/{gate}/q{q}"
)

candidates = []
for gi, group in enumerate(groups):
    for q in range(NQ):
        for axis in ("X", "Y", "Z"):
            if axis == group["axes"][q]:
                continue
```

```
            bad_axes = list(group["axes"])
            bad_axes[q] = axis
            bad = Statevector.from_instruction(
                measurement(vector, "state_preparation", group, bad_axes)
                .remove_final_measurements(inplace=False)
            ).probabilities()
            candidates.append((
                float(np.max(np.abs(bad - ideal["COBYLA"][gi]))), gi, q, axis, bad
            ))
distance, bad_gi, q, axis, bad = max(candidates, key=lambda x: x[0])
require(distance > PROB_TOL, "No meaningful basis corruption found.")
expect_failure(
    lambda: assert_probability(bad, ideal["COBYLA"][bad_gi], "Wrong basis"),
    f"corrupted-basis/group{bad_gi}/q{q}/{axis}",
)

def counts_from_publication(publication):
    require(hasattr(publication.data, REGISTER), "Missing returned register.")
    bits = getattr(publication.data, REGISTER)
    require(bits.num_bits == NQ, "Returned register has incorrect width.")
    return bits.get_counts()

def group_statistics(counts, gi, expected):
    frequencies = np.zeros(DIM, dtype=np.int64)
    for bits, count in counts.items():
        require(
            isinstance(bits, str) and len(bits) == NQ
            and set(bits).issubset({"0", "1"}),
            "Unexpected returned bitstring.",
        )
        require(
            isinstance(count, (int, np.integer)) and count >= 0,
            "Invalid count multiplicity.",
        )
        # Rightmost bit is c[0]; no reversal or relabelling.
        frequencies[int(bits, 2)] += int(count)
    total = int(frequencies.sum())
    require(total == SHOTS, "Returned shots differ from request.")
    score = groups[gi]["score"]
    mean = float(frequencies @ score / total)
    variance = float(frequencies @ ((score - mean) ** 2) / (total * (total - 1)))
    intended = float(expected @ score)
    ideal_variance = float(expected @ ((score - intended) ** 2) / total)
    residual = mean - intended
    return {
        "energy": mean,
        "sampling_variance": variance,
        "sampling_se": math.sqrt(variance),
        "ideal_energy": intended,
        "residual": residual,
        "ideal_sampling_se": math.sqrt(ideal_variance),
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        "z_against_ideal_sampling": (
            residual / math.sqrt(ideal_variance) if ideal_variance > 0 else None
        ),
        "zero_ideal_variance_nonzero_residual": (
            ideal_variance == 0 and abs(residual) > ENERGY_TOL
        ),
        "probability_total_variation": float(
            0.5 * np.abs(frequencies / total - expected).sum()
        ),
        "shots": total,
    }

def complete_energy(stats):
    variance = math.fsum(s["sampling_variance"] for s in stats)
    return {
        "energy": math.fsum(s["energy"] for s in stats),
        "sampling_variance": variance,
        "sampling_se": math.sqrt(variance),
    }

# Each local circuit gets an independent, reproducible sampling stream.
# Local and QPU data use the identical register and group-score decoder.
local_records, local_energies = [], []
for vi, variant in enumerate(variants):
    for si, name in enumerate(names):
        for rep in range(REPS):
            stats = []
            for gi in range(len(groups)):
                entropy = [LOCAL_SEED, vi, si, rep, gi]
                seed = int(np.random.SeedSequence(entropy).generate_state(1)[0])
                local_sampler = StatevectorSampler(default_shots=SHOTS, seed=seed)
                publication = local_sampler.run(
                    [logical[(variant["name"], name)][gi]], shots=SHOTS
                ).result()[0]
                counts = counts_from_publication(publication)
                statistic = group_statistics(counts, gi, ideal[name][gi])
                stats.append(statistic)
                local_records.append({
                    "variant": variant["name"], "state": name,
                    "repetition": rep + 1, "group": gi,
                    "seed_entropy": entropy, "seed": seed,
                    "counts": counts, "statistics": statistic,
                })
            energy = complete_energy(stats)
            residual = energy["energy"] - exact[name]["measurement"]
            se = energy["sampling_se"]
            require(
                abs(residual) <= 6 * se if se > 0 else abs(residual) <= ENERGY_TOL,
                "Local complete energy failed six-SE statistical validation.",
            )
            local_energies.append({
```

```
                "variant": variant["name"], "state": name,
                "repetition": rep + 1, **energy,
                "difference_from_exact": residual,
            })

save_json("groups.json", [
    {k: v for k, v in g.items() if k != "score"} for g in groups
])
save_json("exact_validation.json", {
    "reconstruction_error": reconstruction_error,
    "states": exact, "logical_checks": deterministic,
})
save_json("negative_controls.json", negative_controls)
save_json("local_counts_and_statistics.json", local_records)
save_json("local_complete_energies.json", local_energies)

print("Versions:", versions)
print("Evidence directory:", directory.resolve())
print("Baseline:", manifest["baseline"])
print("All variants are UNMITIGATED; no additional optimization.")
print(
    f"Local deterministic validation PASS: {len(original)} Pauli terms, "
    f"{len(groups)} groups; reconstruction error={reconstruction_error:.3e}"
)
for row in local_energies:
    print(
        f"LOCAL {row['variant']}/{row['state']}/rep{row['repetition']}: "
        f"{row['energy']:.12f} +/- {row['sampling_se']:.6f} Ha (sampling SE)"
    )

# IBM access starts only after deterministic and local sampling validation.
try:
    service = QiskitRuntimeService(channel=CHANNEL)
except AccountNotFoundError:
    token = getpass.getpass("IBM Quantum API token (hidden, not saved): ")
    try:
        require(bool(token.strip()), "Empty API token.")
        service = QiskitRuntimeService(channel=CHANNEL, token=token)
    finally:
        del token

backend = service.backend(BACKEND_NAME)
require(backend.name == BACKEND_NAME, "Unexpected backend; no substitution allowed.")
require(backend.status().operational, "ibm_kingston is not operational.")
width = int(backend.num_qubits)
manifest["physical_backend_width"] = width
coupling = backend.coupling_map
require(coupling is not None, "Missing coupling map.")
adjacency = {q: set() for q in range(width)}
for a, b in coupling.get_edges():
    adjacency[a].add(b)
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    adjacency[b].add(a)

def baseline_path():
    def extend(path):
        if len(path) == NQ:
            return path
        for neighbor in sorted(adjacency[path[-1]]):
            if neighbor not in path:
                found = extend(path + [neighbor])
                if found is not None:
                    return found
        return None
    for start in sorted(adjacency, key=lambda q: (-len(adjacency[q]), q)):
        found = extend([start])
        if found is not None:
            return found
    raise RuntimeError("No five-site connected path on ibm_kingston.")

baseline_layout = baseline_path()
properties = backend.properties()
target_records, edge_errors = [], {}
readout_errors = {q: None for q in range(width)}
single_errors = {q: [] for q in range(width)}
for operation_name in backend.target.operation_names:
    for qargs, props in backend.target[operation_name].items():
        if qargs is None:
            continue
        error = None if props is None else props.error
        duration = None if props is None else props.duration
        target_records.append({
            "operation": operation_name, "qubits": qargs,
            "error": error, "duration_seconds": duration,
        })
        if error is None or not np.isfinite(error):
            continue
        if len(qargs) == 1 and operation_name == "measure":
            readout_errors[qargs[0]] = float(error)
        elif len(qargs) == 1 and operation_name not in ("delay", "reset"):
            single_errors[qargs[0]].append(float(error))
        elif len(qargs) == 2:
            edge = tuple(sorted(qargs))
            edge_errors[edge] = min(edge_errors.get(edge, float("inf")), float(error))

known_readout = [e for e in readout_errors.values() if e is not None]
require(known_readout and edge_errors, "Insufficient calibration-informed placement data.")
missing_readout = max(known_readout) + 0.1
missing_edge = max(edge_errors.values()) + 0.1

def qubit_cost(q):
    return (
        readout_errors[q] if readout_errors[q] is not None else missing_readout
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    ) + (float(np.mean(single_errors[q])) if single_errors[q] else 0)

# Calibration-scored beam search over connected five-site paths.
# This is a reproducible heuristic, not a global optimum or error guarantee.
paths = [(qubit_cost(q), (q,)) for q in range(width)]
for length in range(1, NQ):
    extended = []
    for cost, path in paths:
        for q in sorted(adjacency[path[-1]]):
            if q not in path:
                extended.append((
                    cost + qubit_cost(q)
                    + edge_errors.get(tuple(sorted((path[-1], q))), missing_edge),
                    path + (q,),
                ))
    require(bool(extended), "No calibration-informed five-site path.")
    paths = sorted(extended)[:5000]
calibration_score, calibration_layout = min(paths)
manifest["placement"] = {
    "baseline": baseline_layout,
    "calibration_informed": calibration_layout,
    "heuristic_score": calibration_score,
    "method": "Calibration-scored connected-path beam search; beam width 5000.",
    "routing": (
        "Backend preset pass manager; calibrated variant level 3. "
        "No claim that the router minimizes calibration-weighted error."
    ),
}
save_json("calibration_before.json", {
    "retrieved_utc": utc(),
    "properties": None if properties is None else properties.to_dict(),
    "target_instructions": target_records,
})

def compiled_probabilities(circuit, corruption=None):
    require(
        len(circuit.cregs) == 1 and circuit.cregs[0].name == REGISTER
        and circuit.num_clbits == NQ,
        "Compiled register changed.",
    )
    mapping = {}
    active = set()
    for instruction in circuit.data:
        physical = [circuit.find_bit(q).index for q in instruction.qubits]
        if instruction.operation.name == "measure":
            require(len(instruction.clbits) == 1, "Invalid measurement.")
            c = circuit.find_bit(instruction.clbits[0]).index
            require(c not in mapping, "Repeated measurement destination.")
            mapping[c] = physical[0]
        if instruction.operation.name not in ("barrier", "delay"):
            active.update(physical)
```

```
    require(
        set(mapping) == set(range(NQ)) and len(set(mapping.values())) == NQ,
        "Invalid compiled measurement map.",
    )
    require(circuit.layout is not None, "Missing transpilation layout.")
    initial = list(circuit.layout.initial_index_layout(filter_ancillas=True))
    final = list(circuit.layout.final_index_layout(filter_ancillas=True))
    require(len(initial) == NQ and len(final) == NQ, "Unexpected layout width.")
    require(
        all(mapping[q] == final[q] for q in range(NQ)),
        "Logical/physical/classical mappings disagree.",
    )
    require(
        len(active) <= MAX_ACTIVE_QUBITS,
        f"Compiled circuit has {len(active)} active wires; exact compact "
        f"validation cap is {MAX_ACTIVE_QUBITS}. No submission or fallback.",
    )
    wires = sorted(active)
    compact_index = {physical: i for i, physical in enumerate(wires)}
    compact = QuantumCircuit(len(wires))
    compact.global_phase = circuit.global_phase
    touched, measured = set(), set()
    for instruction in circuit.data:
        operation = instruction.operation
        physical = [circuit.find_bit(q).index for q in instruction.qubits]
        if operation.name == "barrier":
            continue
        if operation.name == "measure":
            measured.update(physical)
            continue
        require(
            not instruction.clbits and getattr(operation, "condition", None) is None,
            "Dynamic classical control is unsupported.",
        )
        require(not measured.intersection(physical), "Nonterminal measurement.")
        if operation.name == "delay":
            continue
        if operation.name == "reset":
            # Initialize resets are redundant only on still-untouched |0>.
            require(not touched.intersection(physical), "Noninitial reset.")
            continue
        compact.append(operation.copy(), [compact_index[q] for q in physical])
        touched.update(physical)
    if corruption is not None:
        compact.x(compact_index[corruption])
    probabilities = Statevector.from_instruction(compact).probabilities(
        qargs=[compact_index[mapping[c]] for c in range(NQ)]
    )
    return probabilities, {
        "initial_logical_to_physical": initial,
        "final_logical_to_physical": final,
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        "classical_to_physical": mapping,
        "active_simulation_qubits": len(wires),
        "compiled_width": circuit.num_qubits,
    }

compiled, circuit_records = {}, []
for variant in variants:
    layout = (
        list(calibration_layout)
        if variant["placement"] == "calibration_path"
        else baseline_layout
    )
    manager = generate_preset_pass_manager(
        backend=backend, optimization_level=variant["level"],
        initial_layout=list(layout), seed_transpiler=TRANSPILE_SEED,
    )
    for name in names:
        circuits = manager.run(logical[(variant["name"], name)])
        compiled[(variant["name"], name)] = circuits
        for gi, circuit in enumerate(circuits):
            circuit.name = f"{variant['name']}_{name}_g{gi}"
            for instruction in circuit.data:
                if instruction.operation.name == "barrier":
                    continue
                qargs = tuple(circuit.find_bit(q).index for q in instruction.qubits)
                require(
                    backend.target.instruction_supported(
                        operation_name=instruction.operation.name, qargs=qargs
                    ),
                    "Compiled circuit contains an unsupported backend instruction.",
                )
            probabilities, mapping = compiled_probabilities(circuit)
            error = assert_probability(
                probabilities, ideal[name][gi],
                f"{variant['name']}/{name}/compiled/{gi}",
            )
            entangling = Counter(
                ins.operation.name for ins in circuit.data
                if len(ins.qubits) == 2
                and ins.operation.name not in ("barrier", "delay")
            )
            circuit_records.append({
                "variant": variant["name"], "state": name, "group": gi,
                "basis": groups[gi]["basis"], "depth": circuit.depth(),
                "entangling_gate_count": sum(entangling.values()),
                "entangling_gates": dict(entangling),
                "operation_counts": dict(circuit.count_ops()),
                "probability_discrepancy": error,
                "layout": str(circuit.layout), **mapping,
            })
```

```
# Corrupt a physical output wire of an already validated compiled baseline.
candidates = []
for gi, circuit in enumerate(compiled[("baseline", "COBYLA")]):
    _, mapping = compiled_probabilities(circuit)
    for physical in mapping["classical_to_physical"].values():
        bad, _ = compiled_probabilities(circuit, corruption=physical)
        candidates.append((
            float(np.max(np.abs(bad - ideal["COBYLA"][gi]))), gi, physical, bad
        ))
distance, gi, physical, bad = max(candidates, key=lambda x: x[0])
require(distance > PROB_TOL, "No meaningful compiled corruption found.")
expect_failure(
    lambda: assert_probability(bad, ideal["COBYLA"][gi], "Corrupt compiled circuit"),
    f"compiled-physical-X/group{gi}/physical{physical}",
)

save_json("negative_controls.json", negative_controls)
save_json("compiled_validation_layouts_and_metrics.json", circuit_records)
save_qpy(
    "validated_submission_circuits.qpy",
    [c for circuits in compiled.values() for c in circuits],
)

comparisons = []
print("\nVALIDATED COMPILED CIRCUITS:")
for record in circuit_records:
    print(
        f"{record['variant']:23s} {record['state']:6s} "
        f"g{record['group']:03d} {record['basis']} "
        f"depth={record['depth']:4d} 2q={record['entangling_gate_count']:4d} "
        f"initial={record['initial_logical_to_physical']} "
        f"final={record['final_logical_to_physical']} "
        f"c->physical={record['classical_to_physical']}"
    )
print("\nCOMPUTED CIRCUIT COMPARISONS:")
for name in names:
    baseline_records = [
        x for x in circuit_records if x["state"] == name and x["variant"] == "baseline"
    ]
    bd = sum(x["depth"] for x in baseline_records)
    bg = sum(x["entangling_gate_count"] for x in baseline_records)
    for variant in variant_names:
        records = [
            x for x in circuit_records if x["state"] == name and x["variant"] == variant
        ]
        depth = sum(x["depth"] for x in records)
        gates = sum(x["entangling_gate_count"] for x in records)
        row = {
            "state": name, "variant": variant,
            "mean_depth": depth / len(records), "depth_sum": depth,
            "entangling_gate_sum": gates,
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
            "depth_sum_difference_from_baseline": depth - bd,
            "entangling_gate_difference_from_baseline": gates - bg,
            "max_compiled_probability_discrepancy": max(
                x["probability_discrepancy"] for x in records
            ),
        }
        comparisons.append(row)
        print(
            f"{name}/{variant}: mean depth={row['mean_depth']:.2f}, "
            f"sum2q={gates}, delta depth={depth-bd:+d}, delta2q={gates-bg:+d}; "
            f"max probability error={row['max_compiled_probability_discrepancy']:.3e}"
        )
save_json("computed_circuit_comparisons.json", comparisons)

print("\nRELEVANT CALIBRATION:")
print("Calibration timestamp:", None if properties is None else properties.last_update_date)
print("Baseline layout:", baseline_layout)
print("Calibration-informed layout:", list(calibration_layout))
print("Placement heuristic score:", calibration_score)
selected = set(baseline_layout) | set(calibration_layout)
for q in sorted(selected):
    quantities = {}
    if properties is not None:
        for label, method in (
            ("T1_seconds", properties.t1),
            ("T2_seconds", properties.t2),
            ("frequency_Hz", properties.frequency),
        ):
            try:
                quantities[label] = method(q)
            except Exception:
                quantities[label] = None
    print(f"physical {q}: readout error={readout_errors[q]}, {quantities}")
for item in target_records:
    if len(item["qubits"]) == 2 and set(item["qubits"]).issubset(selected):
        print(item)
print("Full calibration snapshot saved; calibration scores are not measured improvements.")
print("Negative controls detected:", [x["control"] for x in negative_controls])

signature = inspect.signature(SamplerV2)
run_signature = inspect.signature(SamplerV2.run)
require("mode" in signature.parameters, "SamplerV2 backend job mode unavailable.")
require("shots" in run_signature.parameters, "SamplerV2 shots interface unavailable.")
save_json("installed_sampler_api.json", {
    "constructor_signature": str(signature),
    "run_signature": str(run_signature),
    "class_docstring": inspect.getdoc(SamplerV2),
    "run_docstring": inspect.getdoc(SamplerV2.run),
    "result_access": f"publication.data.{REGISTER}.get_counts()",
    "source": "Installed SDK inspection; no claim of external documentation retrieval.",
})
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
options = {
    "default_shots": SHOTS,
    "dynamical_decoupling": {"enable": False},
    "twirling": {"enable_gates": False, "enable_measure": False},
}
sampler = SamplerV2(mode=backend, options=options)
manifest["sampler_options"] = options

schedule = []
for rep in range(REPS):
    order = variant_names if rep == 0 else list(reversed(variant_names))
    for name in names:
        for variant in order:
            schedule.append({
                "index": len(schedule), "variant": variant, "state": name,
                "repetition": rep + 1, "circuits": len(groups),
                "shots_per_circuit": SHOTS, "requested_shots": len(groups) * SHOTS,
            })
total_shots = sum(row["requested_shots"] for row in schedule)
manifest.update({
    "pauli_term_count": len(original), "measurement_group_count": len(groups),
    "schedule": schedule, "job_count": len(schedule),
    "total_circuit_executions": sum(row["circuits"] for row in schedule),
    "total_requested_shots": total_shots,
    "status": "validated_awaiting_confirmation",
})
save_json("manifest.json", manifest)

print("\nFULL QPU WORKLOAD — NOTHING SUBMITTED:")
print(
    f"Backend={BACKEND_NAME}; logical qubits={NQ}; physical width={width}; "
    f"Pauli terms={len(original)}; groups={len(groups)}; "
    f"complete repetitions/state/variant={REPS}"
)
for row in schedule:
    print(
        f"Job {row['index']:02d}: {row['variant']:23s} {row['state']:6s} "
        f"rep{row['repetition']}: {row['circuits']} circuits x {SHOTS} shots "
        f"= {row['requested_shots']:,} shots"
    )
print(
    f"TOTAL: {len(schedule)} job-mode Sampler jobs, "
    f"{manifest['total_circuit_executions']} circuit executions, "
    f"{total_shots:,} requested shots."
)
print("Exactly three variants; no Session, Estimator, mitigation, or new VQE loop.")
print(
    "Initialize is equivalent here only for preparation from |00000>; "
    "it is not generally the same quantum channel as a unitary StatePreparation."
)
print(
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    "Compiler/placement factors are benchmarked, not assumed improvements. "
    "The calibrated variant changes placement and compiler level together."
)
confirmation = input("Type YES to authorize ALL printed QPU jobs: ").strip()
if confirmation != "YES":
    manifest["status"] = "not_submitted"
    save_json("manifest.json", manifest)
    print("No QPU jobs submitted. Local results and validation evidence preserved.")
    return {
        "evidence_directory": str(directory.resolve()),
        "manifest": manifest, "exact": exact,
        "local": local_energies, "hardware": None,
        "circuit_comparisons": comparisons,
    }

hardware_groups, hardware_energies = [], []
stage = "starting"
try:
    manifest["status"] = "authorized"
    manifest["confirmation_utc"] = utc()
    save_json("manifest.json", manifest)
    for row in schedule:
        stage = f"job-{row['index']}-submit"
        circuits = compiled[(row["variant"], row["state"])]
        require(len(circuits) == row["circuits"], "Workload changed.")
        # Submit exactly the validated, QPY-saved circuit objects.
        job = sampler.run(circuits, shots=SHOTS)
        job_record = {
            **row, "job_id": job.job_id(),
            "submitted_utc": utc(), "status": "submitted",
        }
        manifest["jobs"].append(job_record)
        save_json("manifest.json", manifest)

        stage = f"job-{row['index']}-retrieve"
        result = job.result()
        save_pickle(f"job_{row['index']:02d}_raw_primitive_result.pkl", result)
        require(len(result) == len(groups), "Returned publication count mismatch.")
        raw_counts = [counts_from_publication(pub) for pub in result]
        # Persist every returned count BEFORE decoding/statistical checks.
        save_json(f"job_{row['index']:02d}_raw_counts.json", {
            "job_id": job.job_id(), "register": REGISTER,
            "counts": raw_counts,
            "publication_metadata": [pub.metadata for pub in result],
            "result_metadata": getattr(result, "metadata", None),
        })
        stats = []
        for gi, counts in enumerate(raw_counts):
            statistic = group_statistics(counts, gi, ideal[row["state"]][gi])
            stats.append(statistic)
            hardware_groups.append({
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
            **row, "job_id": job.job_id(), "group": gi,
            "basis": groups[gi]["basis"], "statistics": statistic,
        })
        z = statistic["z_against_ideal_sampling"]
        print(
            f"GROUP {row['variant']}/{row['state']}/rep{row['repetition']}/g{gi}: "
            f"ideal={statistic['ideal_energy']:+.9f}, "
            f"observed={statistic['energy']:+.9f}, "
            f"residual={statistic['residual']:+.6f}, "
            f"sampling SE={statistic['sampling_se']:.6f}, "
            f"ideal-sampling z={z}, "
            f"TV={statistic['probability_total_variation']:.5f}"
        )
    energy = complete_energy(stats)
    hardware_energies.append({
        **row, "job_id": job.job_id(), **energy,
        "difference_from_exact": (
            energy["energy"] - exact[row["state"]]["measurement"]
        ),
    })
    job_record["status"] = "completed"
    job_record["completed_utc"] = utc()
    try:
        job_record["metrics"] = job.metrics()
    except Exception as error:
        job_record["metrics_unavailable_exception_type"] = type(error).__name__
    save_json("hardware_group_statistics.json", hardware_groups)
    save_json("hardware_complete_repetition_energies.json", hardware_energies)
    save_json("manifest.json", manifest)

summary = []
print("\nFINAL COMPARISON — ALL HARDWARE RESULTS UNMITIGATED:")
print("Sampling SE excludes systematic hardware error and drift.")
for name in names:
    for variant in variant_names:
        rows = [
            x for x in hardware_energies
            if x["state"] == name and x["variant"] == variant
        ]
        require(len(rows) == REPS, "Missing complete repetitions.")
        energies = [x["energy"] for x in rows]
        mean = float(np.mean(energies))
        mean_variance = math.fsum(
            x["sampling_variance"] for x in rows
        ) / REPS**2
        local_rows = [
            x for x in local_energies
            if x["state"] == name and x["variant"] == variant
        ]
        local_mean = float(np.mean([x["energy"] for x in local_rows]))
        local_se = math.sqrt(math.fsum(
```

```
            x["sampling_variance"] for x in local_rows
        ) / REPS**2)
        group_rows = [
            x for x in hardware_groups
            if x["state"] == name and x["variant"] == variant
        ]
        item = {
            "state": name, "variant": variant,
            "E_exact": benchmark, "E0_fd": fd_reference,
            "classical_energy": classical[name],
            "exact_pauli_energy": exact[name]["direct_pauli"],
            "exact_measurement_energy": exact[name]["measurement"],
            "local_repetitions": local_rows,
            "local_mean_energy": local_mean, "local_mean_sampling_se": local_se,
            "hardware_repetition_energies": energies,
            "hardware_repetition_sampling_se": [x["sampling_se"] for x in rows],
            "hardware_mean_energy": mean,
            "hardware_mean_sampling_variance": mean_variance,
            "hardware_mean_sampling_se": math.sqrt(mean_variance),
            "between_repetition_standard_deviation": float(np.std(energies, ddof=1)),
            "finite_representation_difference": fd_reference - benchmark,
            "classical_variational_residual": classical[name] - fd_reference,
            "quantum_implementation_difference": (
                exact[name]["measurement"] - classical[name]
            ),
            "hardware_difference": mean - exact[name]["measurement"],
            "mean_group_probability_TV": float(np.mean([
                x["statistics"]["probability_total_variation"] for x in group_rows
            ])),
            "job_ids": [x["job_id"] for x in rows],
        }
        summary.append(item)
        print(f"\n{name}/{variant}")
        for key in (
            "E_exact", "E0_fd", "classical_energy",
            "exact_pauli_energy", "exact_measurement_energy",
            "local_mean_energy", "local_mean_sampling_se",
            "hardware_repetition_energies", "hardware_repetition_sampling_se",
            "hardware_mean_energy", "hardware_mean_sampling_se",
            "between_repetition_standard_deviation",
            "finite_representation_difference", "classical_variational_residual",
            "quantum_implementation_difference", "hardware_difference",
            "mean_group_probability_TV", "job_ids",
        ):
            print(f"  {key}: {item[key]}")

    measured_comparisons = []
    print("\nMEASURED CHANGES VS BASELINE — POSITIVE REDUCTION MEANS
IMPROVEMENT:")
    for name in names:
        baseline = next(
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
            x for x in summary if x["state"] == name and x["variant"] == "baseline"
        )
        for variant in variant_names[1:]:
            alternative = next(
                x for x in summary if x["state"] == name and x["variant"] == variant
            )
            row = {
                "state": name, "alternative": variant,
                "energy_difference": (
                    alternative["hardware_mean_energy"] - baseline["hardware_mean_energy"]
                ),
                "energy_difference_sampling_se": math.sqrt(
                    alternative["hardware_mean_sampling_variance"]
                    + baseline["hardware_mean_sampling_variance"]
                ),
                "absolute_hardware_difference_reduction": (
                    abs(baseline["hardware_difference"])
                    - abs(alternative["hardware_difference"])
                ),
                "mean_group_TV_reduction": (
                    baseline["mean_group_probability_TV"]
                    - alternative["mean_group_probability_TV"]
                ),
                "interpretation": (
                    "Observed point estimates; two repetitions and sampling-only "
                    "errors do not establish drift-robust improvement. TV estimates "
                    "contain finite-shot fluctuations. No mitigation was applied."
                ),
            }
            measured_comparisons.append(row)
            print(row)

    save_json("final_numerical_summary.json", summary)
    save_json("measured_hardware_comparisons.json", measured_comparisons)
    try:
        after = backend.properties(refresh=True)
        save_json("calibration_after.json", {
            "retrieved_utc": utc(),
            "properties": None if after is None else after.to_dict(),
        })
    except Exception as error:
        save_json("calibration_after_unavailable.json", {
            "exception_type": type(error).__name__, "utc": utc(),
        })
    manifest["status"] = "completed"
    manifest["finished_utc"] = utc()
    save_json("manifest.json", manifest)
    print("\nEvidence saved:", directory.resolve())
    print("Job IDs:", [x["job_id"] for x in manifest["jobs"]])
    return {
        "evidence_directory": str(directory.resolve()),
```

```
            "manifest": manifest, "summary": summary,
            "circuit_comparisons": comparisons,
            "measured_hardware_comparisons": measured_comparisons,
        }

    except BaseException as error:
        manifest["status"] = "interrupted_or_failed"
        manifest["failure"] = {
            "stage": stage, "exception_type": type(error).__name__, "utc": utc()
        }
        save_json("manifest.json", manifest)
        save_json("hardware_group_statistics.json", hardware_groups)
        save_json("hardware_complete_repetition_energies.json", hardware_energies)
        print(
            f"Stopped at {stage}. Evidence and submitted job IDs remain in "
            f"{directory.resolve()}. Inspect the manifest before rerunning."
        )
        raise


# Strict dependency sweep: only these ten Part I symbols are read.
# Every other working symbol is imported or defined inside this cell.
_part_ii_required = (
    "H", "r", "h", "n_interior", "R_max", "E_exact", "E0_fd",
    "alpha_cobyla", "alpha_spsa", "normalized_trial_state",
)
_part_ii_missing = [name for name in _part_ii_required if name not in globals()]
if _part_ii_missing:
    raise NameError("Run authoritative Part I first. Missing: " + ", ".join(_part_ii_missing))

part_ii_benchmark_results = _hydrogen_part_ii_benchmark(
    **{name: globals()[name] for name in _part_ii_required}
)
```

Second-run results (verbatim)

```
Versions: {'numpy': '2.1.3', 'scipy': '1.16.3', 'qiskit': '2.5.2', 'qiskit-ibm-runtime': '0.50.0'}
Evidence directory: /content/hydrogen_part_II_20261013T170736_010196Z_71f433e3
Baseline: Preceding actual Part II: StatePreparation, full unpruned QWC Hamiltonian including identity,
optimization level 2, seed 731, first deterministic five-site path ordered by (-degree, index).
All variants are UNMITIGATED; no additional optimization.
Local deterministic validation PASS: 63 Pauli terms, 17 groups; reconstruction error=1.776e-15
LOCAL baseline/COBYLA/rep1: -0.475873950777 +/- 0.019796 Ha (sampling SE)
LOCAL baseline/COBYLA/rep2: -0.458351316906 +/- 0.019753 Ha (sampling SE)
LOCAL baseline/SPSA/rep1: -0.469823312305 +/- 0.020091 Ha (sampling SE)
LOCAL baseline/SPSA/rep2: -0.481874863761 +/- 0.019844 Ha (sampling SE)
LOCAL initialize_matched/COBYLA/rep1: -0.448455311003 +/- 0.019739 Ha (sampling SE)
LOCAL initialize_matched/COBYLA/rep2: -0.480818913778 +/- 0.019785 Ha (sampling SE)
LOCAL initialize_matched/SPSA/rep1: -0.468814779009 +/- 0.019929 Ha (sampling SE)
LOCAL initialize_matched/SPSA/rep2: -0.439763655890 +/- 0.019985 Ha (sampling SE)
LOCAL stateprep_calibrated/COBYLA/rep1: -0.425467344012 +/- 0.019866 Ha (sampling SE)
LOCAL stateprep_calibrated/COBYLA/rep2: -0.470131665184 +/- 0.019861 Ha (sampling SE)
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
LOCAL stateprep_calibrated/SPSA/rep1: -0.464087637038 +/- 0.019835 Ha (sampling SE)
LOCAL stateprep_calibrated/SPSA/rep2: -0.427716620309 +/- 0.019878 Ha (sampling SE)
qiskit_runtime_service._discover_account:WARNING:2026-10-13 17:09:32,597: Loading account with the given token. A saved account will not be used.
qiskit_runtime_service.__init__:WARNING:2026-10-13 17:09:34,795: Instance was not set at service instantiation. Free and trial plan instances will be prioritized. Based on the following filters: (tags: None, region: us-east, eu-de), and available plans: (open), the available account instances are: open-instance. If you need a specific instance set it explicitly either by using a saved account with a saved default instance or passing it in directly to QiskitRuntimeService(). Alternatively, pass instance='auto' or save it to your account for auto-selection without warning.
qiskit_runtime_service.backends:WARNING:2026-10-13 17:09:34,796: Using instance: open-instance, plan: open
VALIDATED COMPILED CIRCUITS:
baseline            COBYLA g000 YXXXY depth= 160 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline            COBYLA g001 XYXXY depth= 160 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline            COBYLA g002 YXXYX depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline            COBYLA g003 YXYYY depth= 160 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline            COBYLA g004 YYYXY depth= 160 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline            COBYLA g005 XXYXY depth= 160 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline            COBYLA g006 XYXYX depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline            COBYLA g007 XYYYY depth= 160 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline            COBYLA g008 YXYXX depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline            COBYLA g009 YYXYY depth= 160 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline            COBYLA g010 YYYYX depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline            COBYLA g011 XXXYY depth= 160 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline            COBYLA g012 XXYYX depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline            COBYLA g013 XYYXX depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline            COBYLA g014 YYXXX depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline            COBYLA g015 XXXXX depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline            COBYLA g016 ZZZZZ depth= 159 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline            SPSA   g000 YXXXY depth= 160 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline            SPSA   g001 XYXXY depth= 160 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
```

```
baseline            SPSA   g002 YXXYX depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline            SPSA   g003 YXYYY depth= 159 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline            SPSA   g004 YYYXY depth= 160 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline            SPSA   g005 XXYXY depth= 160 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline            SPSA   g006 XYXYX depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline            SPSA   g007 XYYYY depth= 159 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline            SPSA   g008 YXYXX depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline            SPSA   g009 YYXYY depth= 159 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline            SPSA   g010 YYYYX depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline            SPSA   g011 XXXYY depth= 159 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline            SPSA   g012 XXYYX depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline            SPSA   g013 XYYXX depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline            SPSA   g014 YYXXX depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline            SPSA   g015 XXXXX depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline            SPSA   g016 ZZZZZ depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
initialize_matched     COBYLA g000 YXXXY depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
initialize_matched     COBYLA g001 XYXXY depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
initialize_matched     COBYLA g002 YXXYX depth= 162 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
initialize_matched     COBYLA g003 YXYYY depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
initialize_matched     COBYLA g004 YYYXY depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
initialize_matched     COBYLA g005 XXYXY depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
initialize_matched     COBYLA g006 XYXYX depth= 162 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
initialize_matched     COBYLA g007 XYYYY depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
initialize_matched     COBYLA g008 YXYXX depth= 162 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
initialize_matched     COBYLA g009 YYXYY depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
initialize_matched     COBYLA g010 YYYYX depth= 162 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
initialize_matched     COBYLA g011 XXXYY depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
initialize_matched     COBYLA g012 XXYYX depth= 162 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
initialize_matched     COBYLA g013 XYYXX depth= 162 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
initialize_matched     COBYLA g014 YYXXX depth= 162 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
initialize_matched     COBYLA g015 XXXXX depth= 162 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
initialize_matched     COBYLA g016 ZZZZZ depth= 160 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
initialize_matched     SPSA   g000 YXXXY depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
initialize_matched     SPSA   g001 XYXXY depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
initialize_matched     SPSA   g002 YXXYX depth= 162 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
initialize_matched     SPSA   g003 YXYYY depth= 160 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
initialize_matched     SPSA   g004 YYYXY depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
initialize_matched     SPSA   g005 XXYXY depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
initialize_matched     SPSA   g006 XYXYX depth= 162 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
initialize_matched     SPSA   g007 XYYYY depth= 160 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
initialize_matched     SPSA   g008 YXYXX depth= 162 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
initialize_matched     SPSA   g009 YYXYY depth= 160 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
initialize_matched     SPSA   g010 YYYYX depth= 162 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
initialize_matched     SPSA   g011 XXXYY depth= 160 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
initialize_matched     SPSA   g012 XXYYX depth= 162 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
initialize_matched     SPSA   g013 XYYXX depth= 162 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
initialize_matched     SPSA   g014 YYXXX depth= 162 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
initialize_matched     SPSA   g015 XXXXX depth= 162 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
initialize_matched     SPSA   g016 ZZZZZ depth= 162 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c-
>physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
stateprep_calibrated    COBYLA g000 YXXXY depth= 160 2q=  46 initial=[37, 45, 44, 43, 42] final=[44, 45, 43,
37, 42] c->physical={0: 44, 1: 45, 2: 43, 3: 37, 4: 42}
stateprep_calibrated    COBYLA g001 XYXXY depth= 160 2q=  46 initial=[37, 45, 44, 43, 42] final=[44, 45, 43,
37, 42] c->physical={0: 44, 1: 45, 2: 43, 3: 37, 4: 42}
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
stateprep_calibrated    COBYLA g002 YXXYX depth= 161 2q=  46 initial=[37, 45, 44, 43, 42] final=[44, 45, 43,
37, 42] c->physical={0: 44, 1: 45, 2: 43, 3: 37, 4: 42}
stateprep_calibrated    COBYLA g003 YXYYY depth= 160 2q=  46 initial=[37, 45, 44, 43, 42] final=[44, 45, 43,
37, 42] c->physical={0: 44, 1: 45, 2: 43, 3: 37, 4: 42}
stateprep_calibrated    COBYLA g004 YYYXY depth= 160 2q=  46 initial=[37, 45, 44, 43, 42] final=[44, 45, 43,
37, 42] c->physical={0: 44, 1: 45, 2: 43, 3: 37, 4: 42}
stateprep_calibrated    COBYLA g005 XXYXY depth= 160 2q=  46 initial=[37, 45, 44, 43, 42] final=[44, 45, 43,
37, 42] c->physical={0: 44, 1: 45, 2: 43, 3: 37, 4: 42}
stateprep_calibrated    COBYLA g006 XYXYX depth= 161 2q=  46 initial=[37, 45, 44, 43, 42] final=[44, 45, 43,
37, 42] c->physical={0: 44, 1: 45, 2: 43, 3: 37, 4: 42}
stateprep_calibrated    COBYLA g007 XYYYY depth= 160 2q=  46 initial=[37, 45, 44, 43, 42] final=[44, 45, 43,
37, 42] c->physical={0: 44, 1: 45, 2: 43, 3: 37, 4: 42}
stateprep_calibrated    COBYLA g008 YXYXX depth= 161 2q=  46 initial=[37, 45, 44, 43, 42] final=[44, 45, 43,
37, 42] c->physical={0: 44, 1: 45, 2: 43, 3: 37, 4: 42}
stateprep_calibrated    COBYLA g009 YYXYY depth= 160 2q=  46 initial=[37, 45, 44, 43, 42] final=[44, 45, 43,
37, 42] c->physical={0: 44, 1: 45, 2: 43, 3: 37, 4: 42}
stateprep_calibrated    COBYLA g010 YYYYX depth= 161 2q=  46 initial=[37, 45, 44, 43, 42] final=[44, 45, 43,
37, 42] c->physical={0: 44, 1: 45, 2: 43, 3: 37, 4: 42}
stateprep_calibrated    COBYLA g011 XXXYY depth= 160 2q=  46 initial=[37, 45, 44, 43, 42] final=[44, 45, 43,
37, 42] c->physical={0: 44, 1: 45, 2: 43, 3: 37, 4: 42}
stateprep_calibrated    COBYLA g012 XXYYX depth= 161 2q=  46 initial=[37, 45, 44, 43, 42] final=[44, 45, 43,
37, 42] c->physical={0: 44, 1: 45, 2: 43, 3: 37, 4: 42}
stateprep_calibrated    COBYLA g013 XYYXX depth= 161 2q=  46 initial=[37, 45, 44, 43, 42] final=[44, 45, 43,
37, 42] c->physical={0: 44, 1: 45, 2: 43, 3: 37, 4: 42}
stateprep_calibrated    COBYLA g014 YYXXX depth= 161 2q=  46 initial=[37, 45, 44, 43, 42] final=[44, 45, 43,
37, 42] c->physical={0: 44, 1: 45, 2: 43, 3: 37, 4: 42}
stateprep_calibrated    COBYLA g015 XXXXX depth= 161 2q=  46 initial=[37, 45, 44, 43, 42] final=[44, 45, 43,
37, 42] c->physical={0: 44, 1: 45, 2: 43, 3: 37, 4: 42}
stateprep_calibrated    COBYLA g016 ZZZZZ depth= 159 2q=  46 initial=[37, 45, 44, 43, 42] final=[44, 45, 43,
37, 42] c->physical={0: 44, 1: 45, 2: 43, 3: 37, 4: 42}
stateprep_calibrated    SPSA   g000 YXXXY depth= 160 2q=  46 initial=[37, 45, 44, 43, 42] final=[44, 45, 43,
37, 42] c->physical={0: 44, 1: 45, 2: 43, 3: 37, 4: 42}
stateprep_calibrated    SPSA   g001 XYXXY depth= 160 2q=  46 initial=[37, 45, 44, 43, 42] final=[44, 45, 43,
37, 42] c->physical={0: 44, 1: 45, 2: 43, 3: 37, 4: 42}
stateprep_calibrated    SPSA   g002 YXXYX depth= 161 2q=  46 initial=[37, 45, 44, 43, 42] final=[44, 45, 43,
37, 42] c->physical={0: 44, 1: 45, 2: 43, 3: 37, 4: 42}
stateprep_calibrated    SPSA   g003 YXYYY depth= 160 2q=  46 initial=[37, 45, 44, 43, 42] final=[44, 45, 43,
37, 42] c->physical={0: 44, 1: 45, 2: 43, 3: 37, 4: 42}
stateprep_calibrated    SPSA   g004 YYYXY depth= 160 2q=  46 initial=[37, 45, 44, 43, 42] final=[44, 45, 43,
37, 42] c->physical={0: 44, 1: 45, 2: 43, 3: 37, 4: 42}
stateprep_calibrated    SPSA   g005 XXYXY depth= 160 2q=  46 initial=[37, 45, 44, 43, 42] final=[44, 45, 43,
37, 42] c->physical={0: 44, 1: 45, 2: 43, 3: 37, 4: 42}
stateprep_calibrated    SPSA   g006 XYXYX depth= 161 2q=  46 initial=[37, 45, 44, 43, 42] final=[44, 45, 43,
37, 42] c->physical={0: 44, 1: 45, 2: 43, 3: 37, 4: 42}
stateprep_calibrated    SPSA   g007 XYYYY depth= 160 2q=  46 initial=[37, 45, 44, 43, 42] final=[44, 45, 43,
37, 42] c->physical={0: 44, 1: 45, 2: 43, 3: 37, 4: 42}
stateprep_calibrated    SPSA   g008 YXYXX depth= 161 2q=  46 initial=[37, 45, 44, 43, 42] final=[44, 45, 43,
37, 42] c->physical={0: 44, 1: 45, 2: 43, 3: 37, 4: 42}
stateprep_calibrated    SPSA   g009 YYXYY depth= 160 2q=  46 initial=[37, 45, 44, 43, 42] final=[44, 45, 43,
37, 42] c->physical={0: 44, 1: 45, 2: 43, 3: 37, 4: 42}
stateprep_calibrated    SPSA   g010 YYYYX depth= 161 2q=  46 initial=[37, 45, 44, 43, 42] final=[44, 45, 43,
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

37, 42] c->physical={0: 44, 1: 45, 2: 43, 3: 37, 4: 42}
stateprep_calibrated SPSA g011 XXXYY depth= 160 2q= 46 initial=[37, 45, 44, 43, 42] final=[44, 45, 43, 37, 42] c->physical={0: 44, 1: 45, 2: 43, 3: 37, 4: 42}
stateprep_calibrated SPSA g012 XXYYX depth= 161 2q= 46 initial=[37, 45, 44, 43, 42] final=[44, 45, 43, 37, 42] c->physical={0: 44, 1: 45, 2: 43, 3: 37, 4: 42}
stateprep_calibrated SPSA g013 XYYXX depth= 161 2q= 46 initial=[37, 45, 44, 43, 42] final=[44, 45, 43, 37, 42] c->physical={0: 44, 1: 45, 2: 43, 3: 37, 4: 42}
stateprep_calibrated SPSA g014 YYXXX depth= 161 2q= 46 initial=[37, 45, 44, 43, 42] final=[44, 45, 43, 37, 42] c->physical={0: 44, 1: 45, 2: 43, 3: 37, 4: 42}
stateprep_calibrated SPSA g015 XXXXX depth= 161 2q= 46 initial=[37, 45, 44, 43, 42] final=[44, 45, 43, 37, 42] c->physical={0: 44, 1: 45, 2: 43, 3: 37, 4: 42}
stateprep_calibrated SPSA g016 ZZZZZ depth= 161 2q= 46 initial=[37, 45, 44, 43, 42] final=[44, 45, 43, 37, 42] c->physical={0: 44, 1: 45, 2: 43, 3: 37, 4: 42}

COMPUTED CIRCUIT COMPARISONS:
COBYLA/baseline: mean depth=160.41, sum2q=782, delta depth=+0, delta2q=+0; max probability error=6.384e-15
COBYLA/initialize_matched: mean depth=161.41, sum2q=782, delta depth=+17, delta2q=+0; max probability error=6.384e-15
COBYLA/stateprep_calibrated: mean depth=160.41, sum2q=782, delta depth=+0, delta2q=+0; max probability error=6.453e-15
SPSA/baseline: mean depth=160.29, sum2q=782, delta depth=+0, delta2q=+0; max probability error=5.607e-15
SPSA/initialize_matched: mean depth=161.29, sum2q=782, delta depth=+17, delta2q=+0; max probability error=5.607e-15
SPSA/stateprep_calibrated: mean depth=160.53, sum2q=782, delta depth=+4, delta2q=+0; max probability error=5.496e-15

RELEVANT CALIBRATION:
Calibration timestamp: 2026-10-13 17:01:12+00:00
Baseline layout: [3, 4, 5, 6, 7]
Calibration-informed layout: [37, 45, 44, 43, 42]
Placement heuristic score: 0.04004538109888418
physical 3: readout error=0.005615234375, {'T1_seconds': 0.00012094925574410159, 'T2_seconds': 0.00015136115822525202, 'frequency_Hz': None}
physical 4: readout error=0.0125732421875, {'T1_seconds': 0.00013187793069416195, 'T2_seconds': 5.06634862566277e-05, 'frequency_Hz': None}
physical 5: readout error=0.005126953125, {'T1_seconds': 0.00011196143593272822, 'T2_seconds': 4.7177116976203943e-05, 'frequency_Hz': None}
physical 6: readout error=0.0069580078125, {'T1_seconds': 0.00020828953299024448, 'T2_seconds': 0.0001000767949567485, 'frequency_Hz': None}
physical 7: readout error=0.0198974609375, {'T1_seconds': 0.0003940070908692518, 'T2_seconds': 7.020455629974175e-05, 'frequency_Hz': None}
physical 37: readout error=0.0057373046875, {'T1_seconds': 0.0001370170151913028, 'T2_seconds': 2.8466413165535166e-05, 'frequency_Hz': None}
physical 42: readout error=0.003662109375, {'T1_seconds': 0.00021444908350800796, 'T2_seconds': 8.667321662068173e-05, 'frequency_Hz': None}
physical 43: readout error=0.0064697265625, {'T1_seconds': 0.00013680039488212745, 'T2_seconds': 3.917840573327137e-05, 'frequency_Hz': None}
physical 44: readout error=0.005126953125, {'T1_seconds': 0.00018131295699302762, 'T2_seconds': 0.00022567673938935425, 'frequency_Hz': None}
physical 45: readout error=0.0029296875, {'T1_seconds': 0.00013122849071772158, 'T2_seconds':

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
7.248849813238011e-05, 'frequency_Hz': None}
{'operation': 'cz', 'qubits': (3, 4), 'error': 0.002376075790546134, 'duration_seconds': 6.8e-08}
{'operation': 'cz', 'qubits': (4, 3), 'error': 0.002376075790546134, 'duration_seconds': 6.8e-08}
{'operation': 'cz', 'qubits': (4, 5), 'error': 0.0027843012560479774, 'duration_seconds': 6.8e-08}
{'operation': 'cz', 'qubits': (5, 4), 'error': 0.0027843012560479774, 'duration_seconds': 6.8e-08}
{'operation': 'cz', 'qubits': (5, 6), 'error': 0.003253426844997992, 'duration_seconds': 6.8e-08}
{'operation': 'cz', 'qubits': (6, 5), 'error': 0.003253426844997992, 'duration_seconds': 6.8e-08}
{'operation': 'cz', 'qubits': (6, 7), 'error': 0.008121907996348404, 'duration_seconds': 8e-08}
{'operation': 'cz', 'qubits': (7, 6), 'error': 0.008121907996348404, 'duration_seconds': 8e-08}
{'operation': 'cz', 'qubits': (37, 45), 'error': 0.001996882908662234, 'duration_seconds': 6.8e-08}
{'operation': 'cz', 'qubits': (42, 43), 'error': 0.0030298112073647576, 'duration_seconds': 6.8e-08}
{'operation': 'cz', 'qubits': (43, 42), 'error': 0.0030298112073647576, 'duration_seconds': 6.8e-08}
{'operation': 'cz', 'qubits': (43, 44), 'error': 0.003193997398737436, 'duration_seconds': 6.8e-08}
{'operation': 'cz', 'qubits': (44, 43), 'error': 0.003193997398737436, 'duration_seconds': 6.8e-08}
{'operation': 'cz', 'qubits': (44, 45), 'error': 0.002324633109004326, 'duration_seconds': 6.8e-08}
{'operation': 'cz', 'qubits': (45, 37), 'error': 0.001996882908662234, 'duration_seconds': 6.8e-08}
{'operation': 'cz', 'qubits': (45, 44), 'error': 0.002324633109004326, 'duration_seconds': 6.8e-08}
Full calibration snapshot saved; calibration scores are not measured improvements.
Negative controls detected: ['corrupted-preparation/x/q4', 'corrupted-basis/group16/q0/X', 'compiled-physical-X/group16/physical7']

FULL QPU WORKLOAD — NOTHING SUBMITTED:
Backend=ibm_kingston; logical qubits=5; physical width=156; Pauli terms=63; groups=17; complete repetitions/state/variant=2
Job 00: baseline              COBYLA rep1: 17 circuits x 4096 shots = 69,632 shots
Job 01: initialize_matched    COBYLA rep1: 17 circuits x 4096 shots = 69,632 shots
Job 02: stateprep_calibrated  COBYLA rep1: 17 circuits x 4096 shots = 69,632 shots
Job 03: baseline              SPSA   rep1: 17 circuits x 4096 shots = 69,632 shots
Job 04: initialize_matched    SPSA   rep1: 17 circuits x 4096 shots = 69,632 shots
Job 05: stateprep_calibrated  SPSA   rep1: 17 circuits x 4096 shots = 69,632 shots
Job 06: stateprep_calibrated  COBYLA rep2: 17 circuits x 4096 shots = 69,632 shots
Job 07: initialize_matched    COBYLA rep2: 17 circuits x 4096 shots = 69,632 shots
Job 08: baseline              COBYLA rep2: 17 circuits x 4096 shots = 69,632 shots
Job 09: stateprep_calibrated  SPSA   rep2: 17 circuits x 4096 shots = 69,632 shots
Job 10: initialize_matched    SPSA   rep2: 17 circuits x 4096 shots = 69,632 shots
Job 11: baseline              SPSA   rep2: 17 circuits x 4096 shots = 69,632 shots
TOTAL: 12 job-mode Sampler jobs, 204 circuit executions, 835,584 requested shots.
Exactly three variants; no Session, Estimator, mitigation, or new VQE loop.
Initialize is equivalent here only for preparation from |00000>; it is not generally the same quantum channel as a unitary StatePreparation.
Compiler/placement factors are benchmarked, not assumed improvements. The calibrated variant changes placement and compiler level together.
/tmp/ipykernel_5433/2629100203.py:863: DeprecationWarning: The SamplerV2 class is deprecated as of qiskit-ibm-runtime 0.50.0 and will be removed no sooner than 3 months after the release date.
`qiskit_ibm_runtime.executor_sampler.Sampler` provides a new Sampler implementation that runs all the pre- and post-processing in the client side. The top-level `qiskit_ibm_runtime.Sampler` import will be updated to point to the client-side sampler in an upcoming release, with the legacy Sampler being available until the deprecation is completed.
  sampler = SamplerV2(mode=backend, options=options)
WARNING:urllib3.connectionpool:Retrying (PostForcelistRetry(total=4, connect=3, read=None, redirect=None, status=None)) after connection broken by 'ConnectionResetError(104, 'Connection reset by peer')':
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

/api/v1/jobs/db76bouil99s73fndkq0?exclude_params=true
GROUP baseline/COBYLA/rep1/g0: ideal=+2.513233743, observed=+2.515934075, residual=+0.002700, sampling SE=0.001329, ideal-sampling z=2.0313240415580793, TV=0.33029
GROUP baseline/COBYLA/rep1/g1: ideal=-0.000479405, observed=-0.001121635, residual=-0.000642, sampling SE=0.003130, ideal-sampling z=-0.2162224966414949, TV=0.33300
GROUP baseline/COBYLA/rep1/g2: ideal=-1.326226828, observed=-1.142198753, residual=+0.184028, sampling SE=0.011711, ideal-sampling z=36.99342889139037, TV=0.33249
GROUP baseline/COBYLA/rep1/g3: ideal=-0.118742890, observed=-0.033233643, residual=+0.085509, sampling SE=0.010745, ideal-sampling z=8.10076598364171, TV=0.16802
GROUP baseline/COBYLA/rep1/g4: ideal=-0.023664307, observed=-0.006771355, residual=+0.016893, sampling SE=0.005495, ideal-sampling z=3.089180696531248, TV=0.16709
GROUP baseline/COBYLA/rep1/g5: ideal=+0.000239752, observed=-0.002617149, residual=-0.002857, sampling SE=0.002821, ideal-sampling z=-0.9603763739205144, TV=0.19788
GROUP baseline/COBYLA/rep1/g6: ideal=-0.000958810, observed=+0.001827850, residual=+0.002787, sampling SE=0.003137, ideal-sampling z=0.9382062673497921, TV=0.33225
GROUP baseline/COBYLA/rep1/g7: ideal=-0.000000000, observed=-0.005732803, residual=-0.005733, sampling SE=0.003091, ideal-sampling z=-1.9300820129406557, TV=0.13752
GROUP baseline/COBYLA/rep1/g8: ideal=-0.637865170, observed=-0.468968239, residual=+0.168897, sampling SE=0.007798, ideal-sampling z=42.85608008269418, TV=0.20114
GROUP baseline/COBYLA/rep1/g9: ideal=+0.011832153, observed=+0.009014626, residual=-0.002818, sampling SE=0.005511, ideal-sampling z=-0.5143318223443148, TV=0.28358
GROUP baseline/COBYLA/rep1/g10: ideal=-0.047328614, observed=-0.013875046, residual=+0.033454, sampling SE=0.005424, ideal-sampling z=6.159978380820053, TV=0.12922
GROUP baseline/COBYLA/rep1/g11: ideal=+0.000119876, observed=-0.001370888, residual=-0.001491, sampling SE=0.002786, ideal-sampling z=-0.501135064435905, TV=0.30186
GROUP baseline/COBYLA/rep1/g12: ideal=+0.000479504, observed=+0.003364906, residual=+0.002885, sampling SE=0.002827, ideal-sampling z=0.9699597775495185, TV=0.16949
GROUP baseline/COBYLA/rep1/g13: ideal=-0.001917621, observed=+0.001827850, residual=+0.003745, sampling SE=0.003132, ideal-sampling z=1.2610643638096615, TV=0.15813
GROUP baseline/COBYLA/rep1/g14: ideal=-0.177648296, observed=-0.001246262, residual=+0.176402, sampling SE=0.005474, ideal-sampling z=37.32322232963583, TV=0.33549
GROUP baseline/COBYLA/rep1/g15: ideal=-0.006115345, observed=+0.000830841, residual=+0.006946, sampling SE=0.002797, ideal-sampling z=2.3362346233410585, TV=0.35249
GROUP baseline/COBYLA/rep1/g16: ideal=-0.645931671, observed=-0.757967723, residual=-0.112036, sampling SE=0.008849, ideal-sampling z=-14.885474142532706, TV=0.21899
GROUP initialize_matched/COBYLA/rep1/g0: ideal=+2.513233743, observed=+2.512901505, residual=-0.000332, sampling SE=0.001329, ideal-sampling z=-0.24992595844370996, TV=0.33349
GROUP initialize_matched/COBYLA/rep1/g1: ideal=-0.000479405, observed=-0.000664673, residual=-0.000185, sampling SE=0.003112, ideal-sampling z=-0.06237489042528336, TV=0.32934
GROUP initialize_matched/COBYLA/rep1/g2: ideal=-1.326226828, observed=-1.109962120, residual=+0.216265, sampling SE=0.012316, ideal-sampling z=43.47365538527731, TV=0.31735
GROUP initialize_matched/COBYLA/rep1/g3: ideal=-0.118742890, observed=-0.015204391, residual=+0.103538, sampling SE=0.010754, ideal-sampling z=9.808777079067346, TV=0.17919
GROUP initialize_matched/COBYLA/rep1/g4: ideal=-0.023664307, observed=-0.002201729, residual=+0.021463, sampling SE=0.005447, ideal-sampling z=3.9248191172197506, TV=0.17124
GROUP initialize_matched/COBYLA/rep1/g5: ideal=+0.000239752, observed=+0.000124626, residual=-0.000115, sampling SE=0.002833, ideal-sampling z=-0.03870065220663721, TV=0.19691
GROUP initialize_matched/COBYLA/rep1/g6: ideal=-0.000958810, observed=-0.001121635, residual=-0.000163, sampling SE=0.003153, ideal-sampling z=-0.054819573743778945, TV=0.31442
GROUP initialize_matched/COBYLA/rep1/g7: ideal=-0.000000000, observed=+0.000083084, residual=+0.000083, sampling SE=0.003119, ideal-sampling z=0.02797220308614707, TV=0.14341

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

GROUP initialize_matched/COBYLA/rep1/g8: ideal=-0.637865170, observed=-0.449651184, residual=+0.188214, sampling SE=0.008047, ideal-sampling z=47.757609309342804, TV=0.20603
GROUP initialize_matched/COBYLA/rep1/g9: ideal=+0.011832153, observed=-0.002492523, residual=-0.014325, sampling SE=0.005482, ideal-sampling z=-2.61492954957423, TV=0.28309
GROUP initialize_matched/COBYLA/rep1/g10: ideal=-0.047328614, observed=-0.010842476, residual=+0.036486, sampling SE=0.005466, ideal-sampling z=6.718381144767565, TV=0.12922
GROUP initialize_matched/COBYLA/rep1/g11: ideal=+0.000119876, observed=-0.006189766, residual=-0.006310, sampling SE=0.002845, ideal-sampling z=-2.121049005926857, TV=0.28782
GROUP initialize_matched/COBYLA/rep1/g12: ideal=+0.000479504, observed=-0.003531075, residual=-0.004011, sampling SE=0.002800, ideal-sampling z=-1.3481998247102638, TV=0.17106
GROUP initialize_matched/COBYLA/rep1/g13: ideal=-0.001917621, observed=-0.001537056, residual=+0.000381, sampling SE=0.003102, ideal-sampling z=0.1281324930155059, TV=0.16228
GROUP initialize_matched/COBYLA/rep1/g14: ideal=-0.177648296, observed=+0.003780327, residual=+0.181429, sampling SE=0.005493, ideal-sampling z=38.38675019090281, TV=0.34233
GROUP initialize_matched/COBYLA/rep1/g15: ideal=-0.006115345, observed=+0.004029579, residual=+0.010145, sampling SE=0.002815, ideal-sampling z=3.4120772553403262, TV=0.34248
GROUP initialize_matched/COBYLA/rep1/g16: ideal=-0.645931671, observed=-0.754731987, residual=-0.108800, sampling SE=0.008870, ideal-sampling z=-14.455563727756346, TV=0.21850
WARNING:urllib3.connectionpool:Retrying (PostForcelistRetry(total=4, connect=3, read=None, redirect=None, status=None)) after connection broken by 'ConnectionResetError(104, 'Connection reset by peer')': /api/v1/jobs/db76r0gr1c5c738js6b0?exclude_params=true
GROUP stateprep_calibrated/COBYLA/rep1/g0: ideal=+2.513233743, observed=+2.512735337, residual=-0.000498, sampling SE=0.001329, ideal-sampling z=-0.37492595844395443, TV=0.19262
GROUP stateprep_calibrated/COBYLA/rep1/g1: ideal=-0.000479405, observed=+0.006023598, residual=+0.006503, sampling SE=0.003029, ideal-sampling z=2.1893946187392475, TV=0.20525
GROUP stateprep_calibrated/COBYLA/rep1/g2: ideal=-1.326226828, observed=-1.156115341, residual=+0.170111, sampling SE=0.011217, ideal-sampling z=34.195908433333564, TV=0.21173
GROUP stateprep_calibrated/COBYLA/rep1/g3: ideal=-0.118742890, observed=-0.065137939, residual=+0.053605, sampling SE=0.010659, ideal-sampling z=5.078294736436894, TV=0.21004
GROUP stateprep_calibrated/COBYLA/rep1/g4: ideal=-0.023664307, observed=+0.009596214, residual=+0.033261, sampling SE=0.005463, ideal-sampling z=6.08228558517916, TV=0.20243
GROUP stateprep_calibrated/COBYLA/rep1/g5: ideal=+0.000239752, observed=+0.001744766, residual=+0.001505, sampling SE=0.002977, ideal-sampling z=0.5059259106242887, TV=0.18880
GROUP stateprep_calibrated/COBYLA/rep1/g6: ideal=-0.000958810, observed=-0.001080093, residual=-0.000121, sampling SE=0.003009, ideal-sampling z=-0.04083329429175864, TV=0.21481
GROUP stateprep_calibrated/COBYLA/rep1/g7: ideal=-0.000000000, observed=+0.000581589, residual=+0.000582, sampling SE=0.002980, ideal-sampling z=0.19580542160272899, TV=0.20614
GROUP stateprep_calibrated/COBYLA/rep1/g8: ideal=-0.637865170, observed=-0.522308235, residual=+0.115557, sampling SE=0.006926, ideal-sampling z=29.32153486330317, TV=0.18760
GROUP stateprep_calibrated/COBYLA/rep1/g9: ideal=+0.011832153, observed=+0.016782990, residual=+0.004951, sampling SE=0.005471, ideal-sampling z=0.9037612281682992, TV=0.17367
GROUP stateprep_calibrated/COBYLA/rep1/g10: ideal=-0.047328614, observed=-0.019566307, residual=+0.027762, sampling SE=0.005505, ideal-sampling z=5.112017029302118, TV=0.18617
GROUP stateprep_calibrated/COBYLA/rep1/g11: ideal=+0.000119876, observed=+0.001370888, residual=+0.001251, sampling SE=0.002944, ideal-sampling z=0.42054010917101686, TV=0.19309
GROUP stateprep_calibrated/COBYLA/rep1/g12: ideal=+0.000479504, observed=+0.002201729, residual=+0.001722, sampling SE=0.002914, ideal-sampling z=0.5789449048791943, TV=0.19457
GROUP stateprep_calibrated/COBYLA/rep1/g13: ideal=-0.001917621, observed=+0.004154205, residual=+0.006072, sampling SE=0.002986, ideal-sampling z=2.044325904111792, TV=0.20152
GROUP stateprep_calibrated/COBYLA/rep1/g14: ideal=-0.177648296, observed=-0.059737473, residual=+0.117911, sampling SE=0.005412, ideal-sampling z=24.947625398529205, TV=0.23055

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

GROUP stateprep_calibrated/COBYLA/rep1/g15: ideal=-0.006115345, observed=-0.001869392, residual=+0.004246, sampling SE=0.002932, ideal-sampling z=1.4280557781468732, TV=0.23263
GROUP stateprep_calibrated/COBYLA/rep1/g16: ideal=-0.645931671, observed=-0.645729814, residual=+0.000202, sampling SE=0.008256, ideal-sampling z=0.02681927130809338, TV=0.09580
GROUP baseline/SPSA/rep1/g0: ideal=+2.513233773, observed=+2.514313935, residual=+0.001080, sampling SE=0.001329, ideal-sampling z=0.8125516267483116, TV=0.13845
GROUP baseline/SPSA/rep1/g1: ideal=-0.000411417, observed=-0.003780327, residual=-0.003369, sampling SE=0.003078, ideal-sampling z=-1.134073672942215, TV=0.16094
GROUP baseline/SPSA/rep1/g2: ideal=-1.326200948, observed=-1.163052864, residual=+0.163148, sampling SE=0.011171, ideal-sampling z=32.78505227050502, TV=0.10664
GROUP baseline/SPSA/rep1/g3: ideal=-0.118951480, observed=-0.064639435, residual=+0.054312, sampling SE=0.010672, ideal-sampling z=5.145559512125235, TV=0.15540
GROUP baseline/SPSA/rep1/g4: ideal=-0.022529043, observed=-0.044657707, residual=-0.022129, sampling SE=0.005422, ideal-sampling z=-4.045736929687591, TV=0.17768
GROUP baseline/SPSA/rep1/g5: ideal=+0.000205743, observed=+0.000997009, residual=+0.000791, sampling SE=0.002829, ideal-sampling z=0.26602753135422985, TV=0.18434
GROUP baseline/SPSA/rep1/g6: ideal=-0.000822834, observed=-0.004860420, residual=-0.004038, sampling SE=0.003122, ideal-sampling z=-1.359179243158545, TV=0.13957
GROUP baseline/SPSA/rep1/g7: ideal=+0.000000000, observed=+0.006895981, residual=+0.006896, sampling SE=0.003104, ideal-sampling z=2.3213836399888685, TV=0.16317
GROUP baseline/SPSA/rep1/g8: ideal=-0.634548430, observed=-0.462321510, residual=+0.172227, sampling SE=0.007853, ideal-sampling z=42.31996500427205, TV=0.20566
GROUP baseline/SPSA/rep1/g9: ideal=+0.011264521, observed=+0.011216354, residual=-0.000048, sampling SE=0.005454, ideal-sampling z=-0.008792319887067473, TV=0.10849
GROUP baseline/SPSA/rep1/g10: ideal=-0.045058085, observed=-0.016782990, residual=+0.028275, sampling SE=0.005477, ideal-sampling z=5.201896372325457, TV=0.15000
GROUP baseline/SPSA/rep1/g11: ideal=+0.000102871, observed=+0.003032570, residual=+0.002930, sampling SE=0.002805, ideal-sampling z=0.9849781134784054, TV=0.15611
GROUP baseline/SPSA/rep1/g12: ideal=+0.000411486, observed=+0.001952477, residual=+0.001541, sampling SE=0.002851, ideal-sampling z=0.5180893347896876, TV=0.15304
GROUP baseline/SPSA/rep1/g13: ideal=-0.001645668, observed=+0.004818878, residual=+0.006465, sampling SE=0.003119, ideal-sampling z=2.176232079758426, TV=0.18904
GROUP baseline/SPSA/rep1/g14: ideal=-0.169096156, observed=-0.154744148, residual=+0.014352, sampling SE=0.004898, ideal-sampling z=2.988582320156833, TV=0.14934
GROUP baseline/SPSA/rep1/g15: ideal=-0.005247610, observed=+0.003904953, residual=+0.009153, sampling SE=0.002826, ideal-sampling z=3.07830321580904, TV=0.16531
GROUP baseline/SPSA/rep1/g16: ideal=-0.661485936, observed=-0.647534759, residual=+0.013951, sampling SE=0.009021, ideal-sampling z=1.8442659503969465, TV=0.23798
WARNING:urllib3.connectionpool:Retrying (PostForcelistRetry(total=4, connect=3, read=None, redirect=None, status=None)) after connection broken by 'ConnectionResetError(104, 'Connection reset by peer')': /api/v1/jobs/db770p6il99s73fnetog?exclude_params=true
GROUP initialize_matched/SPSA/rep1/g0: ideal=+2.513233773, observed=+2.512610711, residual=-0.000623, sampling SE=0.001329, ideal-sampling z=-0.4686983732517508, TV=0.14409
GROUP initialize_matched/SPSA/rep1/g1: ideal=-0.000411417, observed=+0.000041542, residual=+0.000453, sampling SE=0.003133, ideal-sampling z=0.1524793085526822, TV=0.16126
GROUP initialize_matched/SPSA/rep1/g2: ideal=-1.326200948, observed=-1.151836510, residual=+0.174364, sampling SE=0.011414, ideal-sampling z=35.03900926095611, TV=0.11009
GROUP initialize_matched/SPSA/rep1/g3: ideal=-0.118951480, observed=-0.059654388, residual=+0.059297, sampling SE=0.010645, ideal-sampling z=5.617846140032978, TV=0.14611
GROUP initialize_matched/SPSA/rep1/g4: ideal=-0.022529043, observed=-0.037470932, residual=-0.014942, sampling SE=0.005466, ideal-sampling z=-2.7317940229799444, TV=0.18734

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

GROUP initialize_matched/SPSA/rep1/g5: ideal=+0.000205743, observed=+0.000789299, residual=+0.000584, sampling SE=0.002846, ideal-sampling z=0.19619435272507477, TV=0.17650
GROUP initialize_matched/SPSA/rep1/g6: ideal=-0.000822834, observed=-0.000830841, residual=-0.000008, sampling SE=0.003109, ideal-sampling z=-0.002695375663433781, TV=0.13957
GROUP initialize_matched/SPSA/rep1/g7: ideal=+0.000000000, observed=-0.000872383, residual=-0.000872, sampling SE=0.003093, ideal-sampling z=-0.2936690146974002, TV=0.16592
GROUP initialize_matched/SPSA/rep1/g8: ideal=-0.634548430, observed=-0.455342445, residual=+0.179206, sampling SE=0.008011, ideal-sampling z=44.034875683012366, TV=0.19613
GROUP initialize_matched/SPSA/rep1/g9: ideal=+0.011264521, observed=+0.013542709, residual=+0.002278, sampling SE=0.005459, ideal-sampling z=0.41585692259923646, TV=0.10727
GROUP initialize_matched/SPSA/rep1/g10: ideal=-0.045058085, observed=-0.011340981, residual=+0.033717, sampling SE=0.005485, ideal-sampling z=6.203087207114934, TV=0.14806
GROUP initialize_matched/SPSA/rep1/g11: ideal=+0.000102871, observed=-0.001370888, residual=-0.001474, sampling SE=0.002853, ideal-sampling z=-0.4954846249358678, TV=0.15144
GROUP initialize_matched/SPSA/rep1/g12: ideal=+0.000411486, observed=+0.000124626, residual=-0.000287, sampling SE=0.002859, ideal-sampling z=-0.09644371394468365, TV=0.13279
GROUP initialize_matched/SPSA/rep1/g13: ideal=-0.001645668, observed=+0.005857430, residual=+0.007503, sampling SE=0.003117, ideal-sampling z=2.5258511474760104, TV=0.19636
GROUP initialize_matched/SPSA/rep1/g14: ideal=-0.169096156, observed=-0.151088448, residual=+0.018008, sampling SE=0.004889, ideal-sampling z=3.749825132895434, TV=0.16618
GROUP initialize_matched/SPSA/rep1/g15: ideal=-0.005247610, observed=+0.002700233, residual=+0.007948, sampling SE=0.002799, ideal-sampling z=2.6731170316557833, TV=0.18533
GROUP initialize_matched/SPSA/rep1/g16: ideal=-0.661485936, observed=-0.651346170, residual=+0.010140, sampling SE=0.009124, ideal-sampling z=1.3404191839932686, TV=0.25360
GROUP stateprep_calibrated/SPSA/rep1/g0: ideal=+2.513233773, observed=+2.511738328, residual=-0.001495, sampling SE=0.001329, ideal-sampling z=-1.1249483732521413, TV=0.14630
GROUP stateprep_calibrated/SPSA/rep1/g1: ideal=-0.000411417, observed=-0.006148224, residual=-0.005737, sampling SE=0.002962, ideal-sampling z=-1.9311771506075295, TV=0.17534
GROUP stateprep_calibrated/SPSA/rep1/g2: ideal=-1.326200948, observed=-1.171859779, residual=+0.154341, sampling SE=0.010903, ideal-sampling z=31.015278633558225, TV=0.14787
GROUP stateprep_calibrated/SPSA/rep1/g3: ideal=-0.118951480, observed=-0.085742798, residual=+0.033209, sampling SE=0.010594, ideal-sampling z=3.1462127873157826, TV=0.19554
GROUP stateprep_calibrated/SPSA/rep1/g4: ideal=-0.022529043, observed=-0.028497849, residual=-0.005969, sampling SE=0.005420, ideal-sampling z=-1.0912641510212642, TV=0.19899
GROUP stateprep_calibrated/SPSA/rep1/g5: ideal=+0.000205743, observed=-0.006563644, residual=-0.006769, sampling SE=0.002987, ideal-sampling z=-2.2759001707470223, TV=0.19336
GROUP stateprep_calibrated/SPSA/rep1/g6: ideal=-0.000822834, observed=-0.000083084, residual=+0.000740, sampling SE=0.002957, ideal-sampling z=0.2490232801604008, TV=0.16283
GROUP stateprep_calibrated/SPSA/rep1/g7: ideal=+0.000000000, observed=-0.001620140, residual=-0.001620, sampling SE=0.002960, ideal-sampling z=-0.5453853130094467, TV=0.21444
GROUP stateprep_calibrated/SPSA/rep1/g8: ideal=-0.634548430, observed=-0.550972252, residual=+0.083576, sampling SE=0.006443, ideal-sampling z=20.536516263606046, TV=0.20795
GROUP stateprep_calibrated/SPSA/rep1/g9: ideal=+0.011264521, observed=+0.007103691, residual=-0.004161, sampling SE=0.005524, ideal-sampling z=-0.7595115164253587, TV=0.17743
GROUP stateprep_calibrated/SPSA/rep1/g10: ideal=-0.045058085, observed=+0.000623131, residual=+0.045681, sampling SE=0.005487, ideal-sampling z=8.404178508026149, TV=0.22696
GROUP stateprep_calibrated/SPSA/rep1/g11: ideal=+0.000102871, observed=+0.002907944, residual=+0.002805, sampling SE=0.002985, ideal-sampling z=0.94307822465536, TV=0.17623
GROUP stateprep_calibrated/SPSA/rep1/g12: ideal=+0.000411486, observed=+0.001827850, residual=+0.001416, sampling SE=0.002962, ideal-sampling z=0.4761893541941631, TV=0.21913
GROUP stateprep_calibrated/SPSA/rep1/g13: ideal=-0.001645668, observed=+0.005857430,

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

residual=+0.007503, sampling SE=0.002980, ideal-sampling z=2.525851147476012, TV=0.20616
GROUP stateprep_calibrated/SPSA/rep1/g14: ideal=-0.169096156, observed=-0.104561348,
residual=+0.064535, sampling SE=0.005211, ideal-sampling z=13.43837002229589, TV=0.15160
GROUP stateprep_calibrated/SPSA/rep1/g15: ideal=-0.005247610, observed=-0.004486542,
residual=+0.000761, sampling SE=0.002986, ideal-sampling z=0.2559718641208359, TV=0.17332
GROUP stateprep_calibrated/SPSA/rep1/g16: ideal=-0.661485936, observed=-0.539031750,
residual=+0.122454, sampling SE=0.007469, ideal-sampling z=16.18774408543993, TV=0.13712
GROUP stateprep_calibrated/COBYLA/rep2/g0: ideal=+2.513233743, observed=+2.512278374, residual=-0.000955, sampling SE=0.001329, ideal-sampling z=-0.7186759584442927, TV=0.20502
GROUP stateprep_calibrated/COBYLA/rep2/g1: ideal=-0.000479405, observed=-0.000456963,
residual=+0.000022, sampling SE=0.002998, ideal-sampling z=0.007555839672994459, TV=0.19890
GROUP stateprep_calibrated/COBYLA/rep2/g2: ideal=-1.326226828, observed=-1.174809265,
residual=+0.151418, sampling SE=0.010788, ideal-sampling z=30.438045131466172, TV=0.20050
GROUP stateprep_calibrated/COBYLA/rep2/g3: ideal=-0.118742890, observed=-0.092306442,
residual=+0.026436, sampling SE=0.010636, ideal-sampling z=2.504471564989044, TV=0.19320
GROUP stateprep_calibrated/COBYLA/rep2/g4: ideal=-0.023664307, observed=+0.016782990,
residual=+0.040447, sampling SE=0.005466, ideal-sampling z=7.396516919534714, TV=0.21318
GROUP stateprep_calibrated/COBYLA/rep2/g5: ideal=+0.000239752, observed=-0.005275841, residual=-0.005516, sampling SE=0.002938, ideal-sampling z=-1.8541225283097285, TV=0.19618
GROUP stateprep_calibrated/COBYLA/rep2/g6: ideal=-0.000958810, observed=-0.000997009, residual=-0.000038, sampling SE=0.002954, ideal-sampling z=-0.012860735387713648, TV=0.20480
GROUP stateprep_calibrated/COBYLA/rep2/g7: ideal=-0.000000000, observed=+0.005857430,
residual=+0.005857, sampling SE=0.002999, ideal-sampling z=1.9720403175698993, TV=0.21371
GROUP stateprep_calibrated/COBYLA/rep2/g8: ideal=-0.637865170, observed=-0.526047020,
residual=+0.111818, sampling SE=0.006869, ideal-sampling z=28.372851787177627, TV=0.19535
GROUP stateprep_calibrated/COBYLA/rep2/g9: ideal=+0.011832153, observed=+0.019109344,
residual=+0.007277, sampling SE=0.005495, ideal-sampling z=1.328430804792398, TV=0.19244
GROUP stateprep_calibrated/COBYLA/rep2/g10: ideal=-0.047328614, observed=-0.019233971,
residual=+0.028095, sampling SE=0.005457, ideal-sampling z=5.173211852748422, TV=0.19056
GROUP stateprep_calibrated/COBYLA/rep2/g11: ideal=+0.000119876, observed=+0.006148224,
residual=+0.006028, sampling SE=0.002960, ideal-sampling z=2.0264892753042885, TV=0.19762
GROUP stateprep_calibrated/COBYLA/rep2/g12: ideal=+0.000479504, observed=+0.003988037,
residual=+0.003509, sampling SE=0.002970, ideal-sampling z=1.1794320307657635, TV=0.20646
GROUP stateprep_calibrated/COBYLA/rep2/g13: ideal=-0.001917621, observed=+0.004029579,
residual=+0.005947, sampling SE=0.002980, ideal-sampling z=2.00236546445275, TV=0.20640
GROUP stateprep_calibrated/COBYLA/rep2/g14: ideal=-0.177648296, observed=-0.061274529,
residual=+0.116374, sampling SE=0.005429, ideal-sampling z=24.62241439962938, TV=0.22392
GROUP stateprep_calibrated/COBYLA/rep2/g15: ideal=-0.006115345, observed=-0.001453972,
residual=+0.004661, sampling SE=0.002957, ideal-sampling z=1.5677756004844399, TV=0.22001
GROUP stateprep_calibrated/COBYLA/rep2/g16: ideal=-0.645931671, observed=-0.652948353, residual=-0.007017, sampling SE=0.008210, ideal-sampling z=-0.9322591901159925, TV=0.08297
WARNING:urllib3.connectionpool:Retrying (PostForcelistRetry(total=4, connect=3, read=None, redirect=None, status=None)) after connection broken by 'ConnectionResetError(104, 'Connection reset by peer')': /api/v1/jobs/db77ce6r6oqs7386tr9g?exclude_params=true
GROUP initialize_matched/COBYLA/rep2/g0: ideal=+2.513233743, observed=+2.511862954, residual=-0.001371, sampling SE=0.001329, ideal-sampling z=-1.0311759584439018, TV=0.32260
GROUP initialize_matched/COBYLA/rep2/g1: ideal=-0.000479405, observed=+0.006895981,
residual=+0.007375, sampling SE=0.003088, ideal-sampling z=2.4831036851520154, TV=0.34169
GROUP initialize_matched/COBYLA/rep2/g2: ideal=-1.326226828, observed=-1.145355949,
residual=+0.180871, sampling SE=0.011560, ideal-sampling z=36.3587675337417, TV=0.31931
GROUP initialize_matched/COBYLA/rep2/g3: ideal=-0.118742890, observed=-0.011091728,

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

residual=+0.107651, sampling SE=0.010720, ideal-sampling z=10.19839251327734, TV=0.17043
GROUP initialize_matched/COBYLA/rep2/g4: ideal=-0.023664307, observed=+0.005275841,
residual=+0.028940, sampling SE=0.005460, ideal-sampling z=5.292227441982754, TV=0.18344
GROUP initialize_matched/COBYLA/rep2/g5: ideal=+0.000239752, observed=+0.000373878,
residual=+0.000134, sampling SE=0.002806, ideal-sampling z=0.04508804976735126, TV=0.19252
GROUP initialize_matched/COBYLA/rep2/g6: ideal=-0.000958810, observed=+0.005774345,
residual=+0.006733, sampling SE=0.003118, ideal-sampling z=2.2669028152918957, TV=0.32004
GROUP initialize_matched/COBYLA/rep2/g7: ideal=-0.000000000, observed=+0.003323364,
residual=+0.003323, sampling SE=0.003118, ideal-sampling z=1.1188881234439358, TV=0.15610
GROUP initialize_matched/COBYLA/rep2/g8: ideal=-0.637865170, observed=-0.473288612,
residual=+0.164577, sampling SE=0.007733, ideal-sampling z=41.75982408361577, TV=0.20236
GROUP initialize_matched/COBYLA/rep2/g9: ideal=+0.011832153, observed=+0.009014626, residual=-
0.002818, sampling SE=0.005478, ideal-sampling z=-0.5143318223443143, TV=0.28391
GROUP initialize_matched/COBYLA/rep2/g10: ideal=-0.047328614, observed=-0.015329018,
residual=+0.032000, sampling SE=0.005439, ideal-sampling z=5.892251028242478, TV=0.12903
GROUP initialize_matched/COBYLA/rep2/g11: ideal=+0.000119876, observed=-0.004735794, residual=-
0.004856, sampling SE=0.002827, ideal-sampling z=-1.6322818684080347, TV=0.30723
GROUP initialize_matched/COBYLA/rep2/g12: ideal=+0.000479504, observed=+0.001163177,
residual=+0.000684, sampling SE=0.002829, ideal-sampling z=0.22982448285211785, TV=0.16704
GROUP initialize_matched/COBYLA/rep2/g13: ideal=-0.001917621, observed=+0.002534065,
residual=+0.004452, sampling SE=0.003092, ideal-sampling z=1.498840188544234, TV=0.15270
GROUP initialize_matched/COBYLA/rep2/g14: ideal=-0.177648296, observed=+0.003157196,
residual=+0.180805, sampling SE=0.005505, ideal-sampling z=38.254907894051534, TV=0.34648
GROUP initialize_matched/COBYLA/rep2/g15: ideal=-0.006115345, observed=+0.001620140,
residual=+0.007735, sampling SE=0.002817, ideal-sampling z=2.6017022857824332, TV=0.35128
GROUP initialize_matched/COBYLA/rep2/g16: ideal=-0.645931671, observed=-0.755212641, residual=-
0.109281, sampling SE=0.008829, ideal-sampling z=-14.519425045975073, TV=0.21753
GROUP baseline/COBYLA/rep2/g0: ideal=+2.513233743, observed=+2.512693795, residual=-0.000540,
sampling SE=0.001329, ideal-sampling z=-0.40617595844401555, TV=0.32391
GROUP baseline/COBYLA/rep2/g1: ideal=-0.000479405, observed=+0.000706215, residual=+0.001186,
sampling SE=0.003110, ideal-sampling z=0.3991679282233478, TV=0.31371
GROUP baseline/COBYLA/rep2/g2: ideal=-1.326226828, observed=-1.130151558, residual=+0.196075,
sampling SE=0.011938, ideal-sampling z=39.415163019260504, TV=0.32150
GROUP baseline/COBYLA/rep2/g3: ideal=-0.118742890, observed=-0.005358925, residual=+0.113384,
sampling SE=0.010728, ideal-sampling z=10.741492815509456, TV=0.18604
GROUP baseline/COBYLA/rep2/g4: ideal=-0.023664307, observed=-0.006522102, residual=+0.017142,
sampling SE=0.005461, ideal-sampling z=3.134760974023348, TV=0.17666
GROUP baseline/COBYLA/rep2/g5: ideal=+0.000239752, observed=+0.000332336, residual=+0.000093,
sampling SE=0.002800, ideal-sampling z=0.031123266105018196, TV=0.20032
GROUP baseline/COBYLA/rep2/g6: ideal=-0.000958810, observed=+0.000249252, residual=+0.001208,
sampling SE=0.003106, ideal-sampling z=0.40672764817295254, TV=0.33127
GROUP baseline/COBYLA/rep2/g7: ideal=-0.000000000, observed=-0.000498505, residual=-0.000499,
sampling SE=0.003106, ideal-sampling z=-0.16783321851653316, TV=0.15271
GROUP baseline/COBYLA/rep2/g8: ideal=-0.637865170, observed=-0.463858566, residual=+0.174007,
sampling SE=0.007848, ideal-sampling z=44.152613620065765, TV=0.20896
GROUP baseline/COBYLA/rep2/g9: ideal=+0.011832153, observed=-0.004943504, residual=-0.016776,
sampling SE=0.005477, ideal-sampling z=-3.0623492820889053, TV=0.28359
GROUP baseline/COBYLA/rep2/g10: ideal=-0.047328614, observed=-0.017406120, residual=+0.029922,
sampling SE=0.005462, ideal-sampling z=5.509783381703087, TV=0.12476
GROUP baseline/COBYLA/rep2/g11: ideal=+0.000119876, observed=+0.001163177, residual=+0.001043,
sampling SE=0.002834, ideal-sampling z=0.3507162323826145, TV=0.31523

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

GROUP baseline/COBYLA/rep2/g12: ideal=+0.000479504, observed=+0.000955467, residual=+0.000476, sampling SE=0.002829, ideal-sampling z=0.16000039844670233, TV=0.16390
GROUP baseline/COBYLA/rep2/g13: ideal=-0.001917621, observed=-0.000830841, residual=+0.001087, sampling SE=0.003135, ideal-sampling z=0.3659083177500786, TV=0.16374
GROUP baseline/COBYLA/rep2/g14: ideal=-0.177648296, observed=+0.000290794, residual=+0.177939, sampling SE=0.005495, ideal-sampling z=37.64843332853565, TV=0.34941
GROUP baseline/COBYLA/rep2/g15: ideal=-0.006115345, observed=-0.001412430, residual=+0.004703, sampling SE=0.002818, ideal-sampling z=1.5817475827181977, TV=0.34794
GROUP baseline/COBYLA/rep2/g16: ideal=-0.645931671, observed=-0.760489656, residual=-0.114558, sampling SE=0.008767, ideal-sampling z=-15.220546449633854, TV=0.21216
GROUP stateprep_calibrated/SPSA/rep2/g0: ideal=+2.513233773, observed=+2.511530617, residual=-0.001703, sampling SE=0.001329, ideal-sampling z=-1.2811983732520593, TV=0.23241
GROUP stateprep_calibrated/SPSA/rep2/g1: ideal=-0.000411417, observed=+0.002824860, residual=+0.003236, sampling SE=0.002977, ideal-sampling z=1.0894255015978807, TV=0.26559
GROUP stateprep_calibrated/SPSA/rep2/g2: ideal=-1.326200948, observed=-1.051844788, residual=+0.274356, sampling SE=0.013602, ideal-sampling z=55.13261843138453, TV=0.28942
GROUP stateprep_calibrated/SPSA/rep2/g3: ideal=-0.118951480, observed=-0.244641151, residual=-0.125690, sampling SE=0.009986, ideal-sampling z=-11.907923477243566, TV=0.31266
GROUP stateprep_calibrated/SPSA/rep2/g4: ideal=-0.022529043, observed=-0.020314064, residual=+0.002215, sampling SE=0.005468, ideal-sampling z=0.4049598525706753, TV=0.27347
GROUP stateprep_calibrated/SPSA/rep2/g5: ideal=+0.000205743, observed=-0.004777336, residual=-0.004983, sampling SE=0.002960, ideal-sampling z=-1.6753348345362893, TV=0.27973
GROUP stateprep_calibrated/SPSA/rep2/g6: ideal=-0.000822834, observed=+0.003240280, residual=+0.004063, sampling SE=0.002986, ideal-sampling z=1.3677728615996703, TV=0.28873
GROUP stateprep_calibrated/SPSA/rep2/g7: ideal=+0.000000000, observed=-0.000540047, residual=-0.000540, sampling SE=0.002955, ideal-sampling z=-0.18179510433649015, TV=0.32033
GROUP stateprep_calibrated/SPSA/rep2/g8: ideal=-0.634548430, observed=-0.401296234, residual=+0.233252, sampling SE=0.008683, ideal-sampling z=57.31522564159084, TV=0.28827
GROUP stateprep_calibrated/SPSA/rep2/g9: ideal=+0.011264521, observed=+0.022806587, residual=+0.011542, sampling SE=0.005484, ideal-sampling z=2.106870870357204, TV=0.28409
GROUP stateprep_calibrated/SPSA/rep2/g10: ideal=-0.045058085, observed=+0.066591911, residual=+0.111650, sampling SE=0.005354, ideal-sampling z=20.540751375550478, TV=0.35580
GROUP stateprep_calibrated/SPSA/rep2/g11: ideal=+0.000102871, observed=+0.001495514, residual=+0.001393, sampling SE=0.002976, ideal-sampling z=0.4682128179941786, TV=0.32493
GROUP stateprep_calibrated/SPSA/rep2/g12: ideal=+0.000411486, observed=-0.003572617, residual=-0.003984, sampling SE=0.002943, ideal-sampling z=-1.3394764716119312, TV=0.37897
GROUP stateprep_calibrated/SPSA/rep2/g13: ideal=-0.001645668, observed=+0.008848457, residual=+0.010494, sampling SE=0.002980, ideal-sampling z=3.5327540625026432, TV=0.29580
GROUP stateprep_calibrated/SPSA/rep2/g14: ideal=-0.169096156, observed=-0.051719856, residual=+0.117376, sampling SE=0.005431, ideal-sampling z=24.441788860972114, TV=0.26617
GROUP stateprep_calibrated/SPSA/rep2/g15: ideal=-0.005247610, observed=+0.002243271, residual=+0.007491, sampling SE=0.002952, ideal-sampling z=2.5194257204252364, TV=0.25947
GROUP stateprep_calibrated/SPSA/rep2/g16: ideal=-0.661485936, observed=-0.411942005, residual=+0.249544, sampling SE=0.006174, ideal-sampling z=32.98828243000627, TV=0.28854
GROUP initialize_matched/SPSA/rep2/g0: ideal=+2.513233773, observed=+2.514313935, residual=+0.001080, sampling SE=0.001329, ideal-sampling z=0.8125516267486457, TV=0.14313
GROUP initialize_matched/SPSA/rep2/g1: ideal=-0.000411417, observed=-0.002118645, residual=-0.001707, sampling SE=0.003082, ideal-sampling z=-0.5747028114226935, TV=0.15976
GROUP initialize_matched/SPSA/rep2/g2: ideal=-1.326200948, observed=-1.144483566, residual=+0.181717, sampling SE=0.011604, ideal-sampling z=36.51660328802956, TV=0.11626
GROUP initialize_matched/SPSA/rep2/g3: ideal=-0.118951480, observed=-0.049268875, residual=+0.069683,

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

sampling SE=0.010709, ideal-sampling z=6.601776614840777, TV=0.15350
GROUP initialize_matched/SPSA/rep2/g4: ideal=-0.022529043, observed=-0.040586586, residual=-0.018058, sampling SE=0.005483, ideal-sampling z=-3.3014224507433747, TV=0.17768
GROUP initialize_matched/SPSA/rep2/g5: ideal=+0.000205743, observed=-0.001786308, residual=-0.001992, sampling SE=0.002810, ideal-sampling z=-0.6697370622764514, TV=0.19529
GROUP initialize_matched/SPSA/rep2/g6: ideal=-0.000822834, observed=+0.000872383, residual=+0.001695, sampling SE=0.003101, ideal-sampling z=0.5706637848241937, TV=0.12566
GROUP initialize_matched/SPSA/rep2/g7: ideal=+0.000000000, observed=+0.003697243, residual=+0.003697, sampling SE=0.003109, ideal-sampling z=1.24459725276511, TV=0.14851
GROUP initialize_matched/SPSA/rep2/g8: ideal=-0.634548430, observed=-0.449941978, residual=+0.184606, sampling SE=0.008073, ideal-sampling z=45.3618898987043, TV=0.20004
GROUP initialize_matched/SPSA/rep2/g9: ideal=+0.011264521, observed=+0.017073784, residual=+0.005809, sampling SE=0.005478, ideal-sampling z=1.0604138085159494, TV=0.11578
GROUP initialize_matched/SPSA/rep2/g10: ideal=-0.045058085, observed=-0.017738457, residual=+0.027320, sampling SE=0.005486, ideal-sampling z=5.026114775377686, TV=0.14487
GROUP initialize_matched/SPSA/rep2/g11: ideal=+0.000102871, observed=+0.001910934, residual=+0.001808, sampling SE=0.002817, ideal-sampling z=0.6078791140709973, TV=0.14441
GROUP initialize_matched/SPSA/rep2/g12: ideal=+0.000411486, observed=+0.006189766, residual=+0.005778, sampling SE=0.002826, ideal-sampling z=1.9426886750375456, TV=0.15850
GROUP initialize_matched/SPSA/rep2/g13: ideal=-0.001645668, observed=+0.008308411, residual=+0.009954, sampling SE=0.003085, ideal-sampling z=3.350952147289501, TV=0.18886
GROUP initialize_matched/SPSA/rep2/g14: ideal=-0.169096156, observed=-0.163634148, residual=+0.005462, sampling SE=0.004815, ideal-sampling z=1.1373782073606704, TV=0.16374
GROUP initialize_matched/SPSA/rep2/g15: ideal=-0.005247610, observed=+0.001287804, residual=+0.006535, sampling SE=0.002801, ideal-sampling z=2.198071160579551, TV=0.19290
GROUP initialize_matched/SPSA/rep2/g16: ideal=-0.661485936, observed=-0.627285228, residual=+0.034201, sampling SE=0.009046, ideal-sampling z=4.521138323567715, TV=0.24310
GROUP baseline/SPSA/rep2/g0: ideal=+2.513233773, observed=+2.513981598, residual=+0.000748, sampling SE=0.001329, ideal-sampling z=0.5625516267485764, TV=0.14091
GROUP baseline/SPSA/rep2/g1: ideal=-0.000411417, observed=-0.001121635, residual=-0.000710, sampling SE=0.003111, ideal-sampling z=-0.23908029451098023, TV=0.15508
GROUP baseline/SPSA/rep2/g2: ideal=-1.326200948, observed=-1.168453331, residual=+0.157748, sampling SE=0.010918, ideal-sampling z=31.699813719547095, TV=0.11490
GROUP baseline/SPSA/rep2/g3: ideal=-0.118951480, observed=-0.054087753, residual=+0.064864, sampling SE=0.010709, ideal-sampling z=6.145232874529959, TV=0.15551
GROUP baseline/SPSA/rep2/g4: ideal=-0.022529043, observed=-0.041749763, residual=-0.019221, sampling SE=0.005478, ideal-sampling z=-3.5140837304417234, TV=0.17112
GROUP baseline/SPSA/rep2/g5: ideal=+0.000205743, observed=+0.001952477, residual=+0.001747, sampling SE=0.002823, ideal-sampling z=0.5872601530483451, TV=0.17922
GROUP baseline/SPSA/rep2/g6: ideal=-0.000822834, observed=-0.002118645, residual=-0.001296, sampling SE=0.003126, ideal-sampling z=-0.4362108384711518, TV=0.14119
GROUP baseline/SPSA/rep2/g7: ideal=+0.000000000, observed=-0.003364906, residual=-0.003365, sampling SE=0.003097, ideal-sampling z=-1.1327233424042236, TV=0.15569
GROUP baseline/SPSA/rep2/g8: ideal=-0.634548430, observed=-0.468552818, residual=+0.165996, sampling SE=0.007816, ideal-sampling z=40.78879475539674, TV=0.18881
GROUP baseline/SPSA/rep2/g9: ideal=+0.011264521, observed=+0.013501167, residual=+0.002237, sampling SE=0.005436, ideal-sampling z=0.4082739004119804, TV=0.10819
GROUP baseline/SPSA/rep2/g10: ideal=-0.045058085, observed=+0.002326355, residual=+0.047384, sampling SE=0.005508, ideal-sampling z=8.71752831128087, TV=0.14074
GROUP baseline/SPSA/rep2/g11: ideal=+0.000102871, observed=+0.003323364, residual=+0.003220, sampling SE=0.002813, ideal-sampling z=1.0827445207321793, TV=0.14249

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

GROUP baseline/SPSA/rep2/g12: ideal=+0.000411486, observed=+0.003198738, residual=+0.002787, sampling SE=0.002848, ideal-sampling z=0.9370891407449399, TV=0.15207
GROUP baseline/SPSA/rep2/g13: ideal=-0.001645668, observed=+0.001495514, residual=+0.003141, sampling SE=0.003110, ideal-sampling z=1.0574510630621703, TV=0.18496
GROUP baseline/SPSA/rep2/g14: ideal=-0.169096156, observed=-0.148346672, residual=+0.020749, sampling SE=0.004982, ideal-sampling z=4.320757242449388, TV=0.15715
GROUP baseline/SPSA/rep2/g15: ideal=-0.005247610, observed=+0.005774345, residual=+0.011022, sampling SE=0.002821, ideal-sampling z=3.707040398115814, TV=0.18801
GROUP baseline/SPSA/rep2/g16: ideal=-0.661485936, observed=-0.642245126, residual=+0.019241, sampling SE=0.009021, ideal-sampling z=2.5435251255176743, TV=0.24017

FINAL COMPARISON — ALL HARDWARE RESULTS UNMITIGATED:
Sampling SE excludes systematic hardware error and drift.

COBYLA/baseline
 E_exact: -0.5
 E0_fd: -0.46097392795978287
 classical_energy: -0.46097392795976344
 exact_pauli_energy: -0.4609739279597611
 exact_measurement_energy: -0.4609739279597616
 local_mean_energy: -0.467112633841378
 local_mean_sampling_se: 0.01398262958485123
 hardware_repetition_energies: [0.09769665310172276, 0.12491882953451995]
 hardware_repetition_sampling_se: [0.024165227008386244, 0.0242540454395169]
 hardware_mean_energy: 0.11130774131812135
 hardware_mean_sampling_se: 0.017118826745348165
 between_repetition_standard_deviation: 0.01924898555428751
 finite_representation_difference: 0.03902607204021713
 classical_variational_residual: 1.942890293094024e-14
 quantum_implementation_difference: 1.8318679906315083e-15
 hardware_difference: 0.5722816692778829
 mean_group_probability_TV: 0.24484544268801478
 job_ids: ['db76bouil99s73fndkq0', 'db77l08r1c5c738jts1g']

COBYLA/initialize_matched
 E_exact: -0.5
 E0_fd: -0.46097392795978287
 classical_energy: -0.46097392795976344
 exact_pauli_energy: -0.4609739279597611
 exact_measurement_energy: -0.4609739279597616
 local_mean_energy: -0.46463711239042715
 local_mean_sampling_se: 0.013973924600154255
 hardware_repetition_energies: [0.16278850642608017, 0.14598182501946044]
 hardware_repetition_sampling_se: [0.024558607815404063, 0.024044325785894532]
 hardware_mean_energy: 0.1543851657227703
 hardware_mean_sampling_se: 0.017184693918783207
 between_repetition_standard_deviation: 0.011884118391862673
 finite_representation_difference: 0.03902607204021713
 classical_variational_residual: 1.942890293094024e-14
 quantum_implementation_difference: 1.8318679906315083e-15
 hardware_difference: 0.6153590936825319

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
  mean_group_probability_TV: 0.24387788352531714
  job_ids: ['db76np2raffc73f9s5s0', 'db77ce6r6oqs7386tr9g']

COBYLA/stateprep_calibrated
  E_exact: -0.5
  E0_fd: -0.46097392795978287
  classical_energy: -0.46097392795976344
  exact_pauli_energy: -0.4609739279597611
  exact_measurement_energy: -0.4609739279597616
  local_mean_energy: -0.44779950459798146
  local_mean_sampling_se: 0.014045702180928682
  hardware_repetition_energies: [0.08364671974796493, 0.03339061434990134]
  hardware_repetition_sampling_se: [0.023395475035670122, 0.023145343415879392]
  hardware_mean_energy: 0.05851866704893314
  hardware_mean_sampling_se: 0.01645490180754397
  between_repetition_standard_deviation: 0.03553643292299662
  finite_representation_difference: 0.03902607204021713
  classical_variational_residual: 1.942890293094024e-14
  quantum_implementation_difference: 1.8318679906315083e-15
  hardware_difference: 0.5194925950086947
  mean_group_probability_TV: 0.19613723732784566
  job_ids: ['db76r0gr1c5c738js6b0', 'db779huil99s73fnfg50']

SPSA/baseline
  E_exact: -0.5
  E0_fd: -0.46097392795978287
  classical_energy: -0.46077921338321903
  exact_pauli_energy: -0.46077921338321587
  exact_measurement_energy: -0.4607792133832168
  local_mean_energy: -0.4758490880330404
  local_mean_sampling_se: 0.014119419258267076
  hardware_repetition_energies: [-0.015242003074395724, 0.015512907814209042]
  hardware_repetition_sampling_se: [0.023816031723718618, 0.023737953178577655]
  hardware_mean_energy: 0.00013545236990665857
  hardware_mean_sampling_se: 0.016812895260584333
  between_repetition_standard_deviation: 0.02174700604412042
  finite_representation_difference: 0.03902607204021713
  classical_variational_residual: 0.00019471457656383695
  quantum_implementation_difference: 2.220446049250313e-15
  hardware_difference: 0.46091466575312345
  mean_group_probability_TV: 0.1605100213206565
  job_ids: ['db76tduil99s73fnenf0', 'db77qliraffc73f9ud10']

SPSA/initialize_matched
  E_exact: -0.5
  E0_fd: -0.46097392795978287
  classical_energy: -0.46077921338321903
  exact_pauli_energy: -0.46077921338321587
  exact_measurement_energy: -0.4607792133832168
  local_mean_energy: -0.45428921744936984
  local_mean_sampling_se: 0.01411186307200074
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

hardware_repetition_energies: [0.014512564930469792, 0.05681046847405706]
hardware_repetition_sampling_se: [0.02403006142672662, 0.024112356408639395]
hardware_mean_energy: 0.035661516702263425
hardware_mean_sampling_se: 0.01702093992520322
between_repetition_standard_deviation: 0.029909134425645058
finite_representation_difference: 0.03902607204021713
classical_variational_residual: 0.00019471457656383695
quantum_implementation_difference: 2.220446049250313e-15
hardware_difference: 0.49644073008548023
mean_group_probability_TV: 0.16294235410999414
job_ids: ['db770p6il99s73fnetog', 'db77o66il99s73fnges0']

SPSA/stateprep_calibrated
E_exact: -0.5
E0_fd: -0.46097392795978287
classical_energy: -0.46077921338321903
exact_pauli_energy: -0.46077921338321587
exact_measurement_energy: -0.4607792133832168
local_mean_energy: -0.445902128673735
local_mean_sampling_se: 0.014040620517251438
hardware_repetition_energies: [0.030490963702046577, 0.4289333995088804]
hardware_repetition_sampling_se: [0.022759472794289654, 0.02426005751478568]
hardware_mean_energy: 0.22971218160546347
hardware_mean_sampling_se: 0.016632378005675557
between_repetition_standard_deviation: 0.2817413482714978
finite_representation_difference: 0.03902607204021713
classical_variational_residual: 0.00019471457656383695
quantum_implementation_difference: 2.220446049250313e-15
hardware_difference: 0.6904913949886803
mean_group_probability_TV: 0.23867514172633678
job_ids: ['db773mmil99s73fnf410', 'db77mcer6oqs7386uh00']

MEASURED CHANGES VS BASELINE — POSITIVE REDUCTION MEANS IMPROVEMENT:
{'state': 'COBYLA', 'alternative': 'initialize_matched', 'energy_difference': 0.043077424404648956, 'energy_difference_sampling_se': 0.024256296795255292, 'absolute_hardware_difference_reduction': -0.04307742440464901, 'mean_group_TV_reduction': 0.0009675591626976399, 'interpretation': 'Observed point estimates; two repetitions and sampling-only errors do not establish drift-robust improvement. TV estimates contain finite-shot fluctuations. No mitigation was applied.'}
{'state': 'COBYLA', 'alternative': 'stateprep_calibrated', 'energy_difference': -0.05278907426918821, 'energy_difference_sampling_se': 0.023744852550251, 'absolute_hardware_difference_reduction': 0.05278907426918822, 'mean_group_TV_reduction': 0.04870820536016912, 'interpretation': 'Observed point estimates; two repetitions and sampling-only errors do not establish drift-robust improvement. TV estimates contain finite-shot fluctuations. No mitigation was applied.'}
{'state': 'SPSA', 'alternative': 'initialize_matched', 'energy_difference': 0.03552606433235676, 'energy_difference_sampling_se': 0.023924586579098, 'absolute_hardware_difference_reduction': -0.03552606433235678, 'mean_group_TV_reduction': -0.0024323327893376323, 'interpretation': 'Observed point estimates; two repetitions and sampling-only errors do not establish drift-robust improvement. TV estimates contain finite-shot fluctuations. No mitigation was applied.'}
{'state': 'SPSA', 'alternative': 'stateprep_calibrated', 'energy_difference': 0.2295767292355568, 'energy_difference_sampling_se': 0.023649723997693065, 'absolute_hardware_difference_reduction': -0.22957672923555683, 'mean_group_TV_reduction': -0.07816512040568027, 'interpretation': 'Observed point

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

estimates; two repetitions and sampling-only errors do not establish drift-robust improvement. TV estimates contain finite-shot fluctuations. No mitigation was applied.'}

Evidence saved: /content/hydrogen_part_II_20261013T170736_010196Z_71f433e3
Job IDs: ['db76bouil99s73fndkq0', 'db76np2raffc73f9s5s0', 'db76r0gr1c5c738js6b0', 'db76tduil99s73fnenf0', 'db770p6il99s73fnetog', 'db773mmil99s73fnf410', 'db779huil99s73fnfg50', 'db77ce6r6oqs7386tr9g', 'db77l08r1c5c738jts1g', 'db77mcer6oqs7386uh00', 'db77o66il99s73fnges0', 'db77qliraffc73f9ud10']

Second-run conclusion

Run 2 compared three unmitigated implementations on the **ibm_kingston** backend using 12 jobs, 204 circuit instances, and 835,584 requested shots. Reported deterministic and compiled-probability checks passed below 6.5×10^{-15}, and corruption controls detected the injected faults. All variants nevertheless retained 46 two-qubit gates per measurement circuit. Their mean-energy discrepancies ranged from about 0.461 to 0.690 Ha; calibration-informed compilation produced mixed results and substantial variability for the fixed SPSA-derived state. Because that variant changed placement and compiler level together, it did not isolate either effect. Initialize uses StatePreparation with reset semantics, so it was not a fundamentally new low-cost synthesis algorithm [34]. This run improved diagnostic coverage, not demonstrated circuit efficiency or reliable energy accuracy.

Prompt sent to Quantum Flow to further refine the second code version

The supplied results repeat the previous experiment; do not treat them as new validation evidence. Preserve this benchmark and all working scientific checks.
Implement the previously requested genuinely different exact state synthesis, investigating the proposed bond-dimension-two sequential preparation. Preserve the Hamiltonian, five-qubit encoding, and both fixed Part I states. Compare actual compiled depth and two-qubit-gate counts against the 46-gate baseline; do not silently fall back to generic StatePreparation or present Initialize as a new synthesis method.
Separate synthesis, compiler, and placement comparisons so each changes one factor. Analyze dominant group residuals, cancellation, and the calibrated SPSA repetition discrepancy without assuming its cause. Use balanced repeated execution blocks within an explicitly approved budget.
Preserve independent local sampling, negative controls, compiled-output validation, circuit reuse, and separate shot/repetition uncertainties. Add source and circuit hashes, explicit job-to-circuit associations, and clear mean-versus-summed depth labels. Return one complete runnable cell and a change summary. Print local acceptance results, circuit comparisons, backend, and full workload before requesting QPU authorization.\

Third optimized code version (verbatim)

```
# Initial code produced by Stellora.AI Quantum Flow

def _run_atomic_hydrogen_part_ii(
    H, r, h, n_interior, R_max, E_exact, E0_fd,
    alpha_cobyla, alpha_spsa, normalized_trial_state,
):
    import os
    import json
    import math
    import pickle
    import marshal
    import hashlib
    import inspect
    import getpass
    import uuid
    from pathlib import Path
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
from datetime import datetime, timezone
from collections import Counter
from importlib.metadata import version

import numpy as np
import scipy
import qiskit
from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister, qpy
from qiskit.circuit.library import StatePreparation, UnitaryGate
from qiskit.quantum_info import Statevector, Operator, SparsePauliOp
from qiskit.primitives import StatevectorSampler
from qiskit.transpiler.preset_passmanagers import generate_preset_pass_manager
from qiskit_ibm_runtime import QiskitRuntimeService, SamplerV2
from qiskit_ibm_runtime.accounts.exceptions import AccountNotFoundError

NQ, DIM = 5, 32
SHOTS, REPETITIONS = 4096, 2
REGISTER = "hydrogen"
BACKEND_NAME = "ibm_fez"
CHANNEL = "ibm_quantum_platform"
BASELINE_TRANSPILE_SEED = 731
LOCAL_SEED = 20261012
MATRIX_TOL = 1e-12
ENERGY_TOL = 1e-10
PROBABILITY_TOL = 1e-10
SCHMIDT_TOL = 1e-12
MAX_COMPACT_QUBITS = 20
HISTORICAL_TWO_QUBIT_REFERENCE = 46

def require(condition, message):
    if not bool(condition):
        raise RuntimeError(message)

def real_scalar(value, label):
    value = complex(value)
    require(
        np.isfinite(value.real)
        and np.isfinite(value.imag)
        and abs(value.imag) <= MATRIX_TOL,
        f"{label}: nonfinite or unexpectedly complex value.",
    )
    return float(value.real)

def utc():
    return datetime.now(timezone.utc).isoformat()

def sha256(data):
    return hashlib.sha256(data).hexdigest()

def jsonable(value):
    if isinstance(value, np.ndarray):
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        return jsonable(value.tolist())
    if isinstance(value, np.generic):
        return jsonable(value.item())
    if isinstance(value, complex):
        return {"real": value.real, "imag": value.imag}
    if isinstance(value, datetime):
        return value.isoformat()
    if isinstance(value, dict):
        return {str(k): jsonable(v) for k, v in value.items()}
    if isinstance(value, (list, tuple)):
        return [jsonable(v) for v in value]
    if isinstance(value, float) and not math.isfinite(value):
        return None
    if value is None or isinstance(value, (str, int, float, bool)):
        return value
    return repr(value)

# Validate only the permitted Part I interface.
matrix = np.array(H, dtype=complex, copy=True)
grid = np.array(r, dtype=float, copy=True)
require(matrix.shape == (DIM, DIM), "H must be 32 x 32.")
require(grid.shape == (DIM,), "r must contain exactly 32 elements.")
require(np.all(np.isfinite(matrix)), "H contains nonfinite entries.")
require(np.all(np.isfinite(grid)), "r contains nonfinite entries.")
hermitian_error = float(np.max(np.abs(matrix - matrix.conj().T)))
require(hermitian_error <= MATRIX_TOL, "H fails Hermiticity validation.")
require(int(n_interior) == DIM, "n_interior must be 32.")
require(np.isfinite(h) and float(h) > 0, "Invalid grid spacing.")
require(np.isfinite(R_max) and float(R_max) > 0, "Invalid R_max.")
require(callable(normalized_trial_state), "Invalid trial-state function.")

analytical_reference = real_scalar(E_exact, "E_exact")
finite_reference = real_scalar(E0_fd, "E0_fd")
alphas = {"COBYLA": alpha_cobyla, "SPSA": alpha_spsa}
states, classical = {}, {}
for name, alpha in alphas.items():
    require(
        np.ndim(alpha) == 0 and np.isfinite(alpha) and float(alpha) > 0,
        f"{name}: alpha must be finite and positive.",
    )
    vector = np.array(
        normalized_trial_state(alpha, grid.copy()), dtype=complex, copy=True
    )
    require(vector.shape == (DIM,), f"{name}: incorrect state size.")
    require(np.all(np.isfinite(vector)), f"{name}: nonfinite amplitudes.")
    require(
        abs(real_scalar(np.vdot(vector, vector), "State norm") - 1)
        <= MATRIX_TOL,
        f"{name}: state is not normalized.",
    )
    states[name] = vector
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    classical[name] = real_scalar(
        np.vdot(vector, matrix @ vector), f"{name} classical energy"
    )

versions = {
    "numpy": np.__version__,
    "scipy": scipy.__version__,
    "qiskit": qiskit.__version__,
    "qiskit-ibm-runtime": version("qiskit-ibm-runtime"),
}
require(versions["qiskit"] == "2.5.2", "Requires Qiskit 2.5.2.")
require(
    versions["qiskit-ibm-runtime"] == "0.50.0",
    "Requires qiskit-ibm-runtime 0.50.0.",
)

# All comparisons change exactly one factor relative to the named comparator.
# The actual earlier baseline used StatePreparation, level 2, seed 731,
# and the first deterministic five-site path. Initialize is NOT a variant.
variants = [
    {
        "name": "baseline",
        "synthesis": "generic",
        "optimization_level": 2,
        "placement": "baseline",
        "compare_to": None,
        "changed_factor": None,
    },
    {
        "name": "sequential",
        "synthesis": "bond_two_sequential",
        "optimization_level": 2,
        "placement": "baseline",
        "compare_to": "baseline",
        "changed_factor": "synthesis",
    },
    {
        "name": "sequential_level3",
        "synthesis": "bond_two_sequential",
        "optimization_level": 3,
        "placement": "baseline",
        "compare_to": "sequential",
        "changed_factor": "compiler optimization level",
    },
    {
        "name": "sequential_calibrated",
        "synthesis": "bond_two_sequential",
        "optimization_level": 2,
        "placement": "calibrated",
        "compare_to": "sequential",
        "changed_factor": "placement",
```

```
    },
]
state_names = list(states)
variant_names = [v["name"] for v in variants]

timestamp = datetime.now(timezone.utc).strftime("%Y%m%dT%H%M%S_%fZ")
evidence = Path.cwd() / (
    f"hydrogen_part_ii_{timestamp}_{uuid.uuid4().hex[:8]}"
)
evidence.mkdir(mode=0o700, exist_ok=False)

def save_json(filename, content):
    temporary = evidence / f".{filename}.{uuid.uuid4().hex}.tmp"
    with temporary.open("x", encoding="utf-8") as handle:
        os.chmod(temporary, 0o600)
        json.dump(jsonable(content), handle, indent=2, allow_nan=False)
        handle.flush()
        os.fsync(handle.fileno())
    os.replace(temporary, evidence / filename)

def save_pickle(filename, content):
    with (evidence / filename).open("xb") as handle:
        os.chmod(evidence / filename, 0o600)
        pickle.dump(content, handle, protocol=pickle.HIGHEST_PROTOCOL)
        handle.flush()
        os.fsync(handle.fileno())

def save_qpy(filename, circuits):
    path = evidence / filename
    with path.open("xb") as handle:
        os.chmod(path, 0o600)
        qpy.dump(circuits, handle)
    return sha256(path.read_bytes())

# Semantic circuit hashes exclude presentation-only names.
# QPY artifact hashes are also recorded separately.
def circuit_hash(circuit):
    records = []
    for instruction in circuit.data:
        operation = instruction.operation
        item = {
            "operation": operation.name,
            "qubits": [circuit.find_bit(q).index for q in instruction.qubits],
            "clbits": [circuit.find_bit(c).index for c in instruction.clbits],
            "parameters": list(operation.params),
        }
        if operation.name in ("unitary", "state_preparation"):
            item["operation_matrix_or_parameters"] = list(operation.params)
        records.append(item)
    payload = {
        "num_qubits": circuit.num_qubits,
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        "num_clbits": circuit.num_clbits,
        "global_phase": circuit.global_phase,
        "classical_registers": [(c.name, len(c)) for c in circuit.cregs],
        "instructions": records,
    }
    encoded = json.dumps(
        jsonable(payload), sort_keys=True, separators=(",", ":"),
        allow_nan=False,
    ).encode("utf-8")
    return sha256(encoded)

source_evidence = {
    "trial_function_code_object_sha256": sha256(
        marshal.dumps(normalized_trial_state.__code__)
    ) if hasattr(normalized_trial_state, "__code__") else None,
    "trial_function_source_sha256": None,
    "part_ii_cell_source_sha256": None,
    "note": (
        "Source is captured when accessible; code-object hashes are "
        "Python-version-specific and are labeled separately."
    ),
}
try:
    trial_source = inspect.getsource(normalized_trial_state)
except (OSError, TypeError):
    trial_source = None
if trial_source is not None:
    source_evidence["trial_function_source_sha256"] = sha256(
        trial_source.encode("utf-8")
    )
    (evidence / "trial_function_source.py").write_text(
        trial_source, encoding="utf-8"
    )
    os.chmod(evidence / "trial_function_source.py", 0o600)

# Optional Jupyter source capture reads the current cell, not previous variables.
try:
    from IPython import get_ipython
    shell = get_ipython()
    current_source = (
        shell.history_manager.input_hist_raw[-1]
        if shell is not None else None
    )
except Exception:
    current_source = None
if current_source:
    source_evidence["part_ii_cell_source_sha256"] = sha256(
        current_source.encode("utf-8")
    )
    (evidence / "part_ii_cell_source.py").write_text(
        current_source, encoding="utf-8"
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        )
        os.chmod(evidence / "part_ii_cell_source.py", 0o600)
    save_json("source_hashes.json", source_evidence)

    manifest = {
        "created_utc": utc(),
        "versions": versions,
        "backend": BACKEND_NAME,
        "channel": CHANNEL,
        "logical_qubits": NQ,
        "shots_per_circuit": SHOTS,
        "complete_repetitions_per_state_per_variant": REPETITIONS,
        "transpiler_seed": BASELINE_TRANSPILE_SEED,
        "local_seed": LOCAL_SEED,
        "variants": variants,
        "mitigation": "NONE: all variants use unadjusted returned counts.",
        "baseline_definition": (
            "StatePreparation; optimization level 2; seed 731; "
            "first connected five-site path using (-degree,index) starts."
        ),
        "historical_46_gate_reference": {
            "value": HISTORICAL_TWO_QUBIT_REFERENCE,
            "provenance": "User-supplied historical reference.",
            "qualification": (
                "Historical circuits/settings/counting scope were not supplied. "
                "46 is not asserted to be reproduced or new validation evidence."
            ),
        },
        "previous_experiment_records": (
            "Not imported or counted as new validation. The preceding code is "
            "available, but no historical numeric SPSA repetitions/counts/job "
            "associations are present in this conversation."
        ),
        "hamiltonian_sha256": sha256(matrix.tobytes()),
        "state_sha256": {k: sha256(v.tobytes()) for k, v in states.items()},
        "optimized_parameters": {k: float(v) for k, v in alphas.items()},
        "reference_energies": {
            "E_exact": analytical_reference, "E0_fd": finite_reference
        },
        "classical_energies": classical,
        "grid": {
            "n_interior": int(n_interior), "h": float(h), "R_max": float(R_max)
        },
        "source_hashes": source_evidence,
        "tolerances": {
            "matrix": MATRIX_TOL, "energy": ENERGY_TOL,
            "probability": PROBABILITY_TOL, "Schmidt_tail": SCHMIDT_TOL,
        },
        "jobs": [],
        "status": "local_validation",
    }
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
save_json("manifest.json", manifest)
np.savez_compressed(
    evidence / "scientific_inputs.npz",
    H=matrix, r=grid, COBYLA=states["COBYLA"], SPSA=states["SPSA"],
)
os.chmod(evidence / "scientific_inputs.npz", 0o600)

# Full Hamiltonian: no threshold-based intentional term pruning.
pauli = SparsePauliOp.from_operator(Operator(matrix), atol=0, rtol=0)
original_terms = pauli.to_list()
labels = [label for label, coefficient in original_terms]
require(len(labels) == len(set(labels)), "Duplicate original Pauli terms.")
for label, coefficient in original_terms:
    real_scalar(coefficient, label)
reconstruction_error = float(np.max(np.abs(pauli.to_matrix() - matrix)))
require(reconstruction_error <= MATRIX_TOL, "Pauli reconstruction failed.")

group_operators = pauli.group_commuting(qubit_wise=True)
grouped_terms = [
    term for group in group_operators for term in group.to_list()
]
require(
    Counter(label for label, coefficient in grouped_terms) == Counter(labels),
    "Missing or duplicated grouped terms.",
)
original_coefficients = dict(original_terms)
require(
    all(c == original_coefficients[label] for label, c in grouped_terms),
    "Grouping changed a coefficient.",
)
require(
    labels.count("IIIII") == 1
    and sum(label == "IIIII" for label, c in grouped_terms) == 1,
    "Identity must be counted exactly once.",
)

groups = []
for index, operator in enumerate(group_operators):
    axes = ["I"] * NQ
    weights = np.zeros(DIM, dtype=float)
    for label, coefficient in operator.to_list():
        mask = 0
        for q in range(NQ):
            axis = label[NQ - 1 - q]
            if axis != "I":
                require(axes[q] in ("I", axis), "Invalid QWC grouping.")
                axes[q] = axis
                mask |= 1 << q
        weights += real_scalar(coefficient, label) * np.array([
            -1.0 if (k & mask).bit_count() % 2 else 1.0
            for k in range(DIM)
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        ])
    axes = ["Z" if axis == "I" else axis for axis in axes]
    groups.append({
        "index": index,
        "axes": axes,
        "basis": "".join(reversed(axes)),
        "terms": operator.to_list(),
        "weights": weights,
    })

def state_error(actual, intended, label):
    overlap = np.vdot(intended, actual)
    require(abs(overlap) > 0, f"{label}: zero overlap.")
    error = float(np.linalg.norm(
        actual - (overlap / abs(overlap)) * intended
    ))
    require(error <= MATRIX_TOL, f"{label}: state mismatch {error:.3e}.")
    return error

def probability_error(actual, intended, label):
    actual = np.asarray(actual, dtype=float)
    require(
        actual.shape == (DIM,)
        and np.all(np.isfinite(actual))
        and np.min(actual) >= -MATRIX_TOL,
        f"{label}: invalid probabilities.",
    )
    require(
        abs(float(actual.sum()) - 1) <= PROBABILITY_TOL,
        f"{label}: invalid probability normalization.",
    )
    error = float(np.max(np.abs(actual - intended)))
    require(
        error <= PROBABILITY_TOL,
        f"{label}: probability discrepancy {error:.3e}.",
    )
    return error

# Genuine sequential synthesis.
#
# For the specified equally spaced radial trial family:
# k = x + 2**m*y,
# psi[x,y] ∝ (x+1+2**m*y)*exp[-alpha*h*(x+1+2**m*y)].
# This is the sum of two products across every cut, hence rank <= 2.
# We nevertheless test ALL cuts of BOTH actual supplied states.
#
# order="F" makes tensor[b0,b1,...,b4] correspond to psi[k],
# with k = b0 + 2*b1 + ... + 16*b4.
# The bond lives successively on q0,q1,...,q4; no extra qubit is used.
def sequential_synthesis(vector, name):
    cut_records = []
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    for cut in range(1, NQ):
        singular = np.linalg.svd(
            vector.reshape(1 << cut, 1 << (NQ - cut), order="F"),
            compute_uv=False,
        )
        tail = float(np.linalg.norm(singular[2:]))
        numerical_rank = int(np.count_nonzero(singular > SCHMIDT_TOL))
        cut_records.append({
            "cut": cut,
            "singular_values": singular.tolist(),
            "numerical_rank_at_tolerance": numerical_rank,
            "tail_norm_beyond_rank_two": tail,
        })
        require(
            tail <= SCHMIDT_TOL,
            f"{name}: cut {cut} obstructs bond-two exact synthesis; "
            "stopping without generic fallback.",
        )

    tensors = [None] * NQ
    residual = vector.copy()
    next_rank = 1
    discarded = []
    for q in range(NQ - 1, 0, -1):
        unfolding = residual.reshape(1 << q, 2 * next_rank, order="F")
        u, singular, vh = np.linalg.svd(unfolding, full_matrices=False)
        discarded.append(float(np.linalg.norm(singular[2:])))
        require(
            discarded[-1] <= SCHMIDT_TOL,
            f"{name}: MPS factorization exceeds numerical rank-two tolerance.",
        )
        tensors[q] = vh[:2].reshape(2, 2, next_rank, order="F")
        residual = u[:, :2] * singular[:2]
        next_rank = 2

    retained_norm = float(np.linalg.norm(residual))
    require(
        abs(retained_norm - 1) <= MATRIX_TOL,
        f"{name}: sequential retained norm is invalid.",
    )
    # Only floating-point norm cleanup, independently bounded below;
    # no physical state or optimized parameter is fitted or changed.
    tensors[0] = (residual / retained_norm).reshape(1, 2, 2, order="F")
    circuit = QuantumCircuit(NQ, name=f"{name}_bond_two")
    gates = []
    isometry_errors = []

    for q, tensor in enumerate(tensors):
        left_rank, _, right_rank = tensor.shape
        isometry = tensor.transpose(1, 2, 0).reshape(
            2 * right_rank, left_rank, order="F"
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        )
        iso_error = float(np.max(np.abs(
            isometry.conj().T @ isometry - np.eye(left_rank)
        )))
        require(iso_error <= MATRIX_TOL, "Sequential isometry invalid.")
        _, _, vh = np.linalg.svd(isometry.conj().T, full_matrices=True)
        complement = vh.conj().T[:, left_rank:]
        unitary = np.column_stack((isometry, complement))
        require(
            np.max(np.abs(unitary.conj().T @ unitary - np.eye(len(unitary))))
            <= MATRIX_TOL,
            "Sequential unitary completion failed.",
        )
        wires = [q, q + 1] if q < NQ - 1 else [q]
        circuit.append(UnitaryGate(unitary, label=f"B2_{q}"), wires)
        gates.append((wires, unitary))
        isometry_errors.append(iso_error)

    # Independent dense-vector gate application with explicit index semantics.
    independent = np.zeros(DIM, dtype=complex)
    independent[0] = 1
    for wires, unitary in gates:
        q = wires[0]
        local_dimension = len(unitary)
        mask = (local_dimension - 1) << q
        output = np.zeros_like(independent)
        for k, amplitude in enumerate(independent):
            column = (k >> q) & (local_dimension - 1)
            base = k & ~mask
            for row in range(local_dimension):
                output[base | (row << q)] += unitary[row, column] * amplitude
        independent = output

    independent_error = state_error(
        independent, vector, f"{name} independent sequential synthesis"
    )
    qiskit_error = state_error(
        Statevector.from_instruction(circuit).data,
        vector,
        f"{name} Qiskit sequential synthesis",
    )
    return circuit, {
        "state": name,
        "cuts": cut_records,
        "right_canonical_discarded_tail_norms": discarded,
        "retained_norm_before_numerical_cleanup": retained_norm,
        "isometry_errors": isometry_errors,
        "independent_vector_error_up_to_global_phase": independent_error,
        "qiskit_state_error_up_to_global_phase": qiskit_error,
        "construction": (
            "Four adjacent two-qubit isometry completions followed by one "
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
            "one-qubit completion; exactly five qubits, no ancillary qubit."
        ),
        "exactness_qualification": (
            "Analytic trial-family rank bound is exact. Numerical factorization "
            "and circuit equality are independently checked at stated tolerances."
        ),
    }

preparations = {}
synthesis_records = []
for name, vector in states.items():
    generic = QuantumCircuit(NQ, name=f"{name}_generic")
    generic.append(StatePreparation(vector), range(NQ))
    state_error(
        Statevector.from_instruction(generic).data, vector,
        f"{name} baseline preparation",
    )
    sequential, record = sequential_synthesis(vector, name)
    preparations[("generic", name)] = generic
    preparations[("bond_two_sequential", name)] = sequential
    synthesis_records.append(record)
save_json("sequential_synthesis_validation.json", synthesis_records)

def measurement_circuit(prep, group, axes_override=None):
    qr = QuantumRegister(NQ, "q")
    cr = ClassicalRegister(NQ, REGISTER)
    circuit = QuantumCircuit(qr, cr)
    circuit.compose(prep, qubits=list(qr), inplace=True)
    axes = group["axes"] if axes_override is None else axes_override
    for q, axis in enumerate(axes):
        if axis == "X":
            circuit.h(q)
        elif axis == "Y":
            circuit.sdg(q)
            circuit.h(q)
        else:
            require(axis == "Z", "Invalid measurement axis.")
    for q in range(NQ):
        circuit.measure(qr[q], cr[q])
    return circuit

exact, ideal, logical = {}, {}, {}
logical_checks = []
for name in state_names:
    intended = Statevector(states[name])
    direct = real_scalar(intended.expectation_value(pauli), "Pauli energy")
    require(abs(direct - classical[name]) <= ENERGY_TOL, "Pauli/classical mismatch.")
    ideal[name], contributions = [], []
    for gi, group in enumerate(groups):
        measured = measurement_circuit(preparations[("generic", name)], group)
        probabilities = Statevector.from_instruction(
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
            measured.remove_final_measurements(inplace=False)
        ).probabilities()
        group_direct = real_scalar(
            intended.expectation_value(group_operators[gi]), "Direct group"
        )
        group_measured = float(probabilities @ group["weights"])
        require(
            abs(group_direct - group_measured) <= ENERGY_TOL,
            f"{name}/group{gi}: independent deterministic check failed.",
        )
        ideal[name].append(probabilities)
        contributions.append(group_measured)
    measured_energy = math.fsum(contributions)
    require(
        abs(measured_energy - direct) <= ENERGY_TOL
        and abs(measured_energy - classical[name]) <= ENERGY_TOL,
        "Exact group sum failed.",
    )
    exact[name] = {
        "direct_pauli_energy": direct,
        "exact_measurement_energy": measured_energy,
        "group_contributions": contributions,
    }
    for variant in variants:
        key = (variant["name"], name)
        logical[key] = []
        for gi, group in enumerate(groups):
            circuit = measurement_circuit(
                preparations[(variant["synthesis"], name)], group
            )
            circuit.name = f"{variant['name']}_{name}_g{gi:03d}"
            error = probability_error(
                Statevector.from_instruction(
                    circuit.remove_final_measurements(inplace=False)
                ).probabilities(),
                ideal[name][gi],
                circuit.name,
            )
            logical[key].append(circuit)
            logical_checks.append({
                "variant": variant["name"], "state": name, "group": gi,
                "probability_error": error,
                "logical_circuit_hash": circuit_hash(circuit),
            })

negative_controls = []

def expect_failure(function, label):
    try:
        function()
    except RuntimeError:
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        negative_controls.append({"control": label, "detected": True})
    else:
        raise RuntimeError(f"Negative control was not detected: {label}")

# Preparation-corruption control.
control_state = states["COBYLA"]
candidates = []
for q in range(NQ):
    for gate in ("x", "z"):
        bad = preparations[("bond_two_sequential", "COBYLA")].copy()
        getattr(bad, gate)(q)
        actual = Statevector.from_instruction(bad).data
        candidates.append((
            1 - abs(np.vdot(control_state, actual)) ** 2, q, gate, actual
        ))
distance, q, gate, actual = max(candidates, key=lambda x: x[0])
require(distance > ENERGY_TOL, "No meaningful preparation corruption found.")
expect_failure(
    lambda: state_error(actual, control_state, "Corrupted state"),
    f"sequential-preparation/{gate}/q{q}",
)

# Wrong-basis corruption control.
candidates = []
for gi, group in enumerate(groups):
    for q in range(NQ):
        for axis in ("X", "Y", "Z"):
            if axis == group["axes"][q]:
                continue
            bad_axes = list(group["axes"])
            bad_axes[q] = axis
            bad = measurement_circuit(
                preparations[("bond_two_sequential", "COBYLA")],
                group, bad_axes,
            )
            probabilities = Statevector.from_instruction(
                bad.remove_final_measurements(inplace=False)
            ).probabilities()
            candidates.append((
                float(np.max(np.abs(probabilities - ideal["COBYLA"][gi]))),
                gi, q, axis, probabilities,
            ))
distance, gi, q, axis, actual = max(candidates, key=lambda x: x[0])
require(distance > PROBABILITY_TOL, "No meaningful wrong-basis control.")
expect_failure(
    lambda: probability_error(actual, ideal["COBYLA"][gi], "Wrong basis"),
    f"wrong-basis/group{gi}/q{q}/{axis}",
)

# Classical-bit permutation corruption control.
candidates = []
```

```
    for gi, probabilities in enumerate(ideal["COBYLA"]):
        for a in range(NQ):
            for b in range(a + 1, NQ):
                permutation = []
                for k in range(DIM):
                    swap = ((k >> a) ^ (k >> b)) & 1
                    permutation.append(k ^ ((swap << a) | (swap << b)))
                bad = probabilities[np.array(permutation)]
                candidates.append((
                    float(np.max(np.abs(bad - probabilities))), gi, a, b, bad
                ))
    distance, gi, a, b, actual = max(candidates, key=lambda x: x[0])
    require(distance > PROBABILITY_TOL, "No meaningful bit-mapping control.")
    expect_failure(
        lambda: probability_error(actual, ideal["COBYLA"][gi], "Wrong bit mapping"),
        f"classical-bit-swap/group{gi}/c{a}/c{b}",
    )

    def returned_counts(publication):
        require(hasattr(publication.data, REGISTER), "Missing returned register.")
        bitarray = getattr(publication.data, REGISTER)
        require(bitarray.num_bits == NQ, "Returned register has wrong width.")
        return bitarray.get_counts()

    def group_statistics(counts, gi, expected_probabilities):
        frequencies = np.zeros(DIM, dtype=np.int64)
        require(isinstance(counts, dict) and bool(counts), "Missing counts.")
        for bitstring, count in counts.items():
            require(
                isinstance(bitstring, str)
                and len(bitstring) == NQ
                and set(bitstring).issubset({"0", "1"}),
                f"Unexpected count string {bitstring!r}.",
            )
            require(
                isinstance(count, (int, np.integer)) and count >= 0,
                "Invalid multiplicity.",
            )
            # c[4]...c[0]: int(string,2) has c[j] at bit j. Never reverse.
            frequencies[int(bitstring, 2)] += int(count)
        total = int(frequencies.sum())
        require(total == SHOTS and total > 1, "Unexpected shot count.")
        weights = groups[gi]["weights"]
        mean = float(frequencies @ weights / total)
        variance = float(
            frequencies @ ((weights - mean) ** 2) / (total * (total - 1))
        )
        intended = float(expected_probabilities @ weights)
        ideal_variance = float(
            expected_probabilities @ ((weights - intended) ** 2) / total
        )
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    residual = mean - intended
    return {
        "energy": mean,
        "sampling_variance": variance,
        "sampling_standard_error": math.sqrt(variance),
        "ideal_energy": intended,
        "residual": residual,
        "ideal_sampling_standard_error": math.sqrt(ideal_variance),
        "ideal_sampling_z": (
            residual / math.sqrt(ideal_variance) if ideal_variance > 0 else None
        ),
        "probability_total_variation": float(
            0.5 * np.abs(frequencies / total - expected_probabilities).sum()
        ),
        "shots": total,
    }

def complete_statistics(group_stats):
    variance = math.fsum(x["sampling_variance"] for x in group_stats)
    return {
        "energy": math.fsum(x["energy"] for x in group_stats),
        "sampling_variance": variance,
        "sampling_standard_error": math.sqrt(variance),
    }

# Independent deterministic streams: one seed per variant/state/repetition/group.
local_records, local_complete = [], []
for vi, variant in enumerate(variants):
    for si, name in enumerate(state_names):
        for rep in range(REPETITIONS):
            statistics = []
            for gi in range(len(groups)):
                entropy = [LOCAL_SEED, vi, si, rep, gi]
                seed = int(
                    np.random.SeedSequence(entropy).generate_state(1)[0]
                )
                local_sampler = StatevectorSampler(
                    default_shots=SHOTS, seed=seed
                )
                publication = local_sampler.run(
                    [logical[(variant["name"], name)][gi]], shots=SHOTS
                ).result()[0]
                counts = returned_counts(publication)
                stats = group_statistics(counts, gi, ideal[name][gi])
                statistics.append(stats)
                local_records.append({
                    "variant": variant["name"], "state": name,
                    "repetition": rep + 1, "group": gi,
                    "seed_entropy": entropy, "seed": seed,
                    "counts": counts, "statistics": stats,
                })
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
            complete = complete_statistics(statistics)
            difference = (
                complete["energy"] - exact[name]["exact_measurement_energy"]
            )
            se = complete["sampling_standard_error"]
            require(
                abs(difference) <= 6 * se
                if se > 0 else abs(difference) <= ENERGY_TOL,
                f"{variant['name']}/{name}: local six-SE acceptance failed.",
            )
            local_complete.append({
                "variant": variant["name"], "state": name,
                "repetition": rep + 1, **complete,
                "difference_from_exact": difference,
            })

save_json("measurement_groups.json", [
    {k: v for k, v in g.items() if k != "weights"} for g in groups
])
save_json("exact_validation.json", {
    "hermitian_error": hermitian_error,
    "hamiltonian_reconstruction_error": reconstruction_error,
    "states": exact,
    "logical_circuit_checks": logical_checks,
})
save_json("negative_controls.json", negative_controls)
save_json("local_counts_and_group_statistics.json", local_records)
save_json("local_complete_energies.json", local_complete)
save_qpy(
    "logical_measurement_circuits.qpy",
    [c for circuits in logical.values() for c in circuits],
)

print("=== CHANGE SUMMARY ===")
print(
    "1. Exact bond-two sequential synthesis replaces the proposed Initialize "
    "alternative; all Schmidt cuts, isometries, and both output states tested."
)
print(
    "2. Synthesis, compiler-level, and placement contrasts each change one factor."
)
print(
    "3. Independent local streams; preparation/basis/bit-map/compiled corruption "
    "controls; checked circuits reused without recompilation."
)
print(
    "4. Balanced forward/reverse execution blocks; full workload requires YES."
)
print(
    "5. Per-group residuals, cancellation, repetition differences, separate "
    "sampling/repetition uncertainties, hashes, and job-circuit evidence."
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
)
print(
    "No new optimization, Hamiltonian changes, parameter changes, mitigation, "
    "offsets, postselection, or historical-result revalidation."
)
print("\nVersions:", versions)
print("Evidence:", evidence.resolve())
print("\n=== LOCAL ACCEPTANCE RESULTS ===")
print(
    f"Pauli terms={len(original_terms)}, QWC groups={len(groups)}, "
    f"H reconstruction error={reconstruction_error:.3e}"
)
for record in synthesis_records:
    print(
        record["state"],
        "Schmidt ranks:",
        [x["numerical_rank_at_tolerance"] for x in record["cuts"]],
        "max rank-two tail:",
        max(x["tail_norm_beyond_rank_two"] for x in record["cuts"]),
        "independent state error:",
        record["independent_vector_error_up_to_global_phase"],
    )
for name in state_names:
    print(
        f"{name}: classical={classical[name]:.16f}, "
        f"Pauli={exact[name]['direct_pauli_energy']:.16f}, "
        f"measurement={exact[name]['exact_measurement_energy']:.16f} Ha"
    )
for row in local_complete:
    print(
        f"LOCAL {row['variant']}/{row['state']}/rep{row['repetition']}: "
        f"{row['energy']:.12f} +/- "
        f"{row['sampling_standard_error']:.6f} Ha (sampling SE)"
    )
print("Local acceptance: PASS. Corruption controls so far:", negative_controls)

# No IBM access before all deterministic/local acceptance checks above.
try:
    service = QiskitRuntimeService(channel=CHANNEL)
except AccountNotFoundError:
    token = getpass.getpass("IBM Quantum API token (hidden; not saved): ")
    try:
        require(bool(token.strip()), "Empty API token.")
        service = QiskitRuntimeService(channel=CHANNEL, token=token)
    finally:
        del token

backend = service.backend(BACKEND_NAME)
require(backend.name == BACKEND_NAME, "Backend substitution is forbidden.")
require(backend.status().operational, "ibm_fez is not operational.")
width = int(backend.num_qubits)
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
manifest["physical_backend_width"] = width
coupling = backend.coupling_map
require(coupling is not None, "Missing backend coupling map.")
adjacency = {q: set() for q in range(width)}
for a, b in coupling.get_edges():
    adjacency[a].add(b)
    adjacency[b].add(a)

def find_baseline_path():
    def extend(path):
        if len(path) == NQ:
            return path
        for neighbor in sorted(adjacency[path[-1]]):
            if neighbor not in path:
                found = extend(path + [neighbor])
                if found is not None:
                    return found
        return None

    for start in sorted(adjacency, key=lambda q: (-len(adjacency[q]), q)):
        found = extend([start])
        if found is not None:
            return found
    raise RuntimeError("No connected five-site baseline path.")

baseline_layout = find_baseline_path()
properties = backend.properties()
target_calibration, readout_error, single_errors, edge_error = [], {}, {}, {}
for q in range(width):
    readout_error[q] = None
    single_errors[q] = []
for operation_name in backend.target.operation_names:
    for qargs, props in backend.target[operation_name].items():
        if qargs is None:
            continue
        error = None if props is None else props.error
        duration = None if props is None else props.duration
        target_calibration.append({
            "operation": operation_name, "qubits": list(qargs),
            "error": error, "duration_seconds": duration,
        })
        if error is None or not np.isfinite(error):
            continue
        if len(qargs) == 1 and operation_name == "measure":
            readout_error[qargs[0]] = float(error)
        elif len(qargs) == 1 and operation_name not in ("delay", "reset"):
            single_errors[qargs[0]].append(float(error))
        elif len(qargs) == 2:
            edge = tuple(sorted(qargs))
            edge_error[edge] = min(
                edge_error.get(edge, float("inf")), float(error)
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        )

known_readout = [x for x in readout_error.values() if x is not None]
require(
    known_readout and edge_error,
    "Insufficient calibration evidence for the requested placement comparison.",
)
missing_readout = max(known_readout) + 0.1
missing_edge = max(edge_error.values()) + 0.1

def qubit_cost(q):
    return (
        readout_error[q] if readout_error[q] is not None else missing_readout
    ) + (float(np.mean(single_errors[q])) if single_errors[q] else 0)

# A declared heuristic, not a proof of global optimality or QPU improvement.
paths = [(qubit_cost(q), (q,)) for q in range(width)]
for length in range(1, NQ):
    extended = []
    for cost, path in paths:
        for q in sorted(adjacency[path[-1]]):
            if q not in path:
                extended.append((
                    cost + qubit_cost(q)
                    + edge_error.get(
                        tuple(sorted((path[-1], q))), missing_edge
                    ),
                    path + (q,),
                ))
    require(bool(extended), "Calibration path search failed.")
    paths = sorted(extended)[:5000]
placement_score, calibrated_layout = min(paths)
calibrated_layout = list(calibrated_layout)
manifest["placements"] = {
    "baseline": baseline_layout,
    "calibrated": calibrated_layout,
    "calibration_score": placement_score,
    "method": "Connected-path beam search, width 5000.",
    "routing": (
        "Same backend preset workflow and seed for placement contrast. "
        "Placement can change routing outcomes; no calibration-optimal "
        "routing claim is made."
    ),
}
save_json("calibration_before.json", {
    "retrieved_utc": utc(),
    "backend_properties": None if properties is None else properties.to_dict(),
    "target_instructions": target_calibration,
})

def compact_compiled_probabilities(circuit, output_x=None):
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    require(
        len(circuit.cregs) == 1
        and circuit.cregs[0].name == REGISTER
        and len(circuit.cregs[0]) == NQ
        and circuit.num_clbits == NQ,
        "Compiled classical register changed.",
    )
    mapping, active = {}, set()
    for instruction in circuit.data:
        physical = [
            circuit.find_bit(q).index for q in instruction.qubits
        ]
        if instruction.operation.name == "measure":
            require(
                len(physical) == 1 and len(instruction.clbits) == 1,
                "Invalid compiled measurement.",
            )
            c = circuit.find_bit(instruction.clbits[0]).index
            require(c not in mapping, "Duplicate measurement destination.")
            mapping[c] = physical[0]
        if instruction.operation.name not in ("barrier", "delay"):
            active.update(physical)
    require(
        set(mapping) == set(range(NQ)) and len(set(mapping.values())) == NQ,
        "Invalid compiled measurement map.",
    )
    require(circuit.layout is not None, "Missing compiled layout.")
    initial = list(circuit.layout.initial_index_layout(filter_ancillas=True))
    final = list(circuit.layout.final_index_layout(filter_ancillas=True))
    require(len(initial) == NQ and len(final) == NQ, "Invalid logical layout size.")
    require(
        all(mapping[q] == final[q] for q in range(NQ)),
        "Logical/physical/classical mapping mismatch.",
    )
    require(
        len(active) <= MAX_COMPACT_QUBITS,
        f"Exact compact validator would need {len(active)} active qubits "
        f"(cap {MAX_COMPACT_QUBITS}). Stop; no fallback or submission.",
    )
    wires = sorted(active)
    index = {physical: j for j, physical in enumerate(wires)}
    compact = QuantumCircuit(len(wires))
    compact.global_phase = circuit.global_phase
    touched, measured = set(), set()
    for instruction in circuit.data:
        operation = instruction.operation
        physical = [
            circuit.find_bit(q).index for q in instruction.qubits
        ]
        if operation.name == "barrier":
            continue
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        if operation.name == "measure":
            measured.update(physical)
            continue
        require(
            not instruction.clbits
            and getattr(operation, "condition", None) is None,
            "Dynamic classical control unsupported by exact validator.",
        )
        require(
            not measured.intersection(physical),
            "Operation after measurement on the same wire.",
        )
        if operation.name == "delay":
            continue
        if operation.name == "reset":
            require(not touched.intersection(physical), "Noninitial reset.")
            continue
        compact.append(operation.copy(), [index[q] for q in physical])
        touched.update(physical)
    if output_x is not None:
        compact.x(index[output_x])
    probabilities = Statevector.from_instruction(compact).probabilities(
        qargs=[index[mapping[c]] for c in range(NQ)]
    )
    # qargs order directly makes returned marginal bit j equal c[j].
    return probabilities, {
        "initial_logical_to_physical": initial,
        "final_logical_to_physical": final,
        "classical_to_physical": mapping,
        "compiled_width": circuit.num_qubits,
        "active_validation_qubits": len(wires),
    }

# Independently exercise compact physical placement and c-bit decoding.
validator = QuantumCircuit(
    QuantumRegister(NQ, "q"), ClassicalRegister(NQ, REGISTER)
)
validator.ry(0.37, 0)
validator.h(2)
validator.cx(0, 4)
validator.rz(-0.29, 4)
validator.cx(4, 2)
validator.sdg(2)
for q in range(NQ):
    validator.measure(q, q)
baseline_manager = generate_preset_pass_manager(
    backend=backend, optimization_level=2,
    initial_layout=baseline_layout,
    seed_transpiler=BASELINE_TRANSPILE_SEED,
)
validator_compiled = baseline_manager.run(validator)
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
validator_probabilities, validator_mapping = compact_compiled_probabilities(
    validator_compiled
)
validator_error = probability_error(
    validator_probabilities,
    Statevector.from_instruction(
        validator.remove_final_measurements(inplace=False)
    ).probabilities(),
    "Compact compiled-validator self-test",
)
save_json("compiled_validator_self_test.json", {
    "maximum_probability_error": validator_error,
    "mapping": validator_mapping,
})

def metrics(circuit):
    entangling = Counter(
        ins.operation.name for ins in circuit.data
        if len(ins.qubits) == 2
        and ins.operation.name not in ("barrier", "delay")
    )
    return {
        "depth": circuit.depth(),
        "two_qubit_gate_count": sum(entangling.values()),
        "two_qubit_gate_types": dict(entangling),
        "operation_counts": dict(circuit.count_ops()),
    }

def calibration_proxy(circuit):
    # A descriptive additive calibration score, NOT a predicted energy error.
    score, missing = 0.0, []
    for ins in circuit.data:
        if ins.operation.name in ("barrier", "delay"):
            continue
        qargs = tuple(circuit.find_bit(q).index for q in ins.qubits)
        props = backend.target[ins.operation.name].get(qargs)
        error = None if props is None else props.error
        if error is None or not np.isfinite(error):
            missing.append({"operation": ins.operation.name, "qubits": qargs})
        else:
            score += float(error)
    return {
        "sum_reported_instruction_errors": score,
        "missing_calibration_entries": missing,
        "interpretation": "Descriptive proxy only; not a QPU energy prediction.",
    }

compiled, circuit_records, preparation_records = {}, [], []
for variant in variants:
    layout = (
        baseline_layout if variant["placement"] == "baseline"
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        else calibrated_layout
    )
    manager = generate_preset_pass_manager(
        backend=backend,
        optimization_level=variant["optimization_level"],
        initial_layout=list(layout),
        seed_transpiler=BASELINE_TRANSPILE_SEED,
    )
    for name in state_names:
        # Preparation-only comparison has a separately labeled scope.
        prep_source = preparations[(variant["synthesis"], name)]
        prep_compiled = manager.run(prep_source)
        probe = prep_compiled.copy()
        probe.add_register(ClassicalRegister(NQ, REGISTER))
        output_positions = prep_compiled.layout.final_index_layout(
            filter_ancillas=True
        )
        for q, physical in enumerate(output_positions):
            probe.measure(probe.qubits[physical], probe.cregs[0][q])
        probabilities, placement = compact_compiled_probabilities(probe)
        probability_error(
            probabilities, np.abs(states[name]) ** 2,
            f"{variant['name']}/{name}/preparation-only output",
        )
        prep_metrics = metrics(prep_compiled)
        preparation_records.append({
            "variant": variant["name"], "state": name,
            "scope": "Compiled preparation only; one circuit, no measurement rotations.",
            **prep_metrics,
            "difference_from_historical_46": (
                prep_metrics["two_qubit_gate_count"] - HISTORICAL_TWO_QUBIT_REFERENCE
            ),
            "historical_comparability": (
                "46 has unknown historical scope; this arithmetic comparison "
                "does not certify baseline reproduction."
            ),
            "circuit_hash": circuit_hash(prep_compiled),
            "mapping": placement,
        })

        circuits = manager.run(logical[(variant["name"], name)])
        compiled[(variant["name"], name)] = circuits
        for gi, circuit in enumerate(circuits):
            for ins in circuit.data:
                if ins.operation.name == "barrier":
                    continue
                qargs = tuple(circuit.find_bit(q).index for q in ins.qubits)
                require(
                    backend.target.instruction_supported(
                        operation_name=ins.operation.name, qargs=qargs
                    ),
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
                f"Unsupported compiled instruction {ins.operation.name}.",
            )
        probabilities, placement = compact_compiled_probabilities(circuit)
        discrepancy = probability_error(
            probabilities, ideal[name][gi],
            f"{variant['name']}/{name}/compiled/group{gi}",
        )
        record = {
            "variant": variant["name"], "state": name, "group": gi,
            "basis": groups[gi]["basis"],
            **metrics(circuit),
            **placement,
            "probability_discrepancy": discrepancy,
            "layout_repr": str(circuit.layout),
            "logical_circuit_hash": circuit_hash(
                logical[(variant["name"], name)][gi]
            ),
            "compiled_circuit_hash": circuit_hash(circuit),
            "calibration_proxy": calibration_proxy(circuit),
        }
        circuit_records.append(record)

# Negative control on an actual checked hardware-compiled circuit.
candidates = []
for gi, circuit in enumerate(compiled[("sequential", "COBYLA")]):
    _, placement = compact_compiled_probabilities(circuit)
    for physical in placement["classical_to_physical"].values():
        bad, _ = compact_compiled_probabilities(circuit, output_x=physical)
        candidates.append((
            float(np.max(np.abs(bad - ideal["COBYLA"][gi]))),
            gi, physical, bad,
        ))
distance, gi, physical, actual = max(candidates, key=lambda x: x[0])
require(distance > PROBABILITY_TOL, "No meaningful compiled corruption.")
expect_failure(
    lambda: probability_error(
        actual, ideal["COBYLA"][gi], "Corrupt compiled output"
    ),
    f"compiled-output-X/group{gi}/physical{physical}",
)
save_json("negative_controls.json", negative_controls)

all_compiled = [
    c for variant in variants for name in state_names
    for c in compiled[(variant["name"], name)]
]
manifest["validated_compiled_qpy_sha256"] = save_qpy(
    "validated_submission_circuits.qpy", all_compiled
)
save_json("compiled_circuit_records.json", circuit_records)
save_json("preparation_only_metrics.json", preparation_records)
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
aggregate = {}
for variant in variant_names:
    for name in state_names:
        records = [
            x for x in circuit_records
            if x["variant"] == variant and x["state"] == name
        ]
        aggregate[(variant, name)] = {
            "variant": variant, "state": name,
            "measurement_circuit_count": len(records),
            "mean_depth_per_measurement_circuit": float(np.mean([
                x["depth"] for x in records
            ])),
            "summed_depth_across_measurement_circuits": sum(
                x["depth"] for x in records
            ),
            "mean_two_qubit_gates_per_measurement_circuit": float(np.mean([
                x["two_qubit_gate_count"] for x in records
            ])),
            "summed_two_qubit_gates_across_measurement_circuits": sum(
                x["two_qubit_gate_count"] for x in records
            ),
            "maximum_compiled_probability_discrepancy": max(
                x["probability_discrepancy"] for x in records
            ),
        }
factor_comparisons = []
for variant in variants:
    if variant["compare_to"] is None:
        continue
    for name in state_names:
        current = aggregate[(variant["name"], name)]
        reference = aggregate[(variant["compare_to"], name)]
        factor_comparisons.append({
            "state": name,
            "variant": variant["name"],
            "comparator": variant["compare_to"],
            "changed_factor": variant["changed_factor"],
            "mean_depth_difference": (
                current["mean_depth_per_measurement_circuit"]
                - reference["mean_depth_per_measurement_circuit"]
            ),
            "summed_two_qubit_gate_difference": (
                current["summed_two_qubit_gates_across_measurement_circuits"]
                - reference["summed_two_qubit_gates_across_measurement_circuits"]
            ),
            "interpretation": "Negative differences are measured compiled-cost reductions.",
        })
save_json("compiled_aggregate_metrics.json", list(aggregate.values()))
save_json("one_factor_compiled_comparisons.json", factor_comparisons)
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
print("\n=== BACKEND AND COMPILED ACCEPTANCE ===")
print(f"Backend={BACKEND_NAME}; logical qubits={NQ}; physical width={width}")
print("Baseline layout:", baseline_layout)
print("Calibrated layout:", calibrated_layout)
print("Calibration heuristic score:", placement_score)
print("Calibration timestamp:", None if properties is None else properties.last_update_date)
print("Compact compiled-validator self-test error:", validator_error)
print("All compiled measurement probabilities: PASS")
print("All negative controls:", negative_controls)

print("\nPREPARATION-ONLY METRICS; HISTORICAL 46-GATE COMPARISON:")
for row in preparation_records:
    print(
        f"{row['variant']:24s} {row['state']:6s}: "
        f"depth={row['depth']}, two-qubit gates={row['two_qubit_gate_count']}, "
        f"count-46={row['difference_from_historical_46']:+d}"
    )
print(
    "Historical 46-gate settings/counting scope were not supplied. "
    "The actual freshly compiled baseline above is the controlled comparator; "
    "a mismatch is reported, not repaired or silently relabeled."
)

print("\nCOMPLETE MEASUREMENT-CIRCUIT METRICS AND MAPPINGS:")
for row in circuit_records:
    print(
        f"{row['variant']:24s} {row['state']:6s} g{row['group']:03d} "
        f"{row['basis']} depth={row['depth']:4d} "
        f"2q={row['two_qubit_gate_count']:4d} "
        f"initial={row['initial_logical_to_physical']} "
        f"final={row['final_logical_to_physical']} "
        f"c->physical={row['classical_to_physical']}"
    )
print("\nMEAN VERSUS SUMMED METRICS (SUMMED DEPTH IS NOT EXECUTION LATENCY):")
for row in aggregate.values():
    print(row)
print("\nONE-FACTOR COMPARISONS:")
for row in factor_comparisons:
    print(row)

selected_qubits = {
    q for record in circuit_records
    for q in record["initial_logical_to_physical"]
    + record["final_logical_to_physical"]
}
print("\nRELEVANT CALIBRATION; FULL SNAPSHOT SAVED:")
for q in sorted(selected_qubits):
    quantities = {}
    if properties is not None:
        for label, getter in (
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
            ("T1_seconds", properties.t1),
            ("T2_seconds", properties.t2),
            ("frequency_Hz", properties.frequency),
        ):
            try:
                quantities[label] = getter(q)
            except Exception:
                quantities[label] = None
    print(f"q{q}: readout_error={readout_error[q]}, {quantities}")
for record in target_calibration:
    if len(record["qubits"]) == 2 and set(record["qubits"]).issubset(selected_qubits):
        print(record)
print("Calibration proxies are not evidence of energy improvement or a diagnosis.")

constructor_signature = inspect.signature(SamplerV2)
run_signature = inspect.signature(SamplerV2.run)
require("mode" in constructor_signature.parameters, "Missing SamplerV2 job mode.")
require("shots" in run_signature.parameters, "Missing SamplerV2 shots argument.")
save_json("installed_sampler_api.json", {
    "constructor_signature": str(constructor_signature),
    "run_signature": str(run_signature),
    "class_docstring": inspect.getdoc(SamplerV2),
    "run_docstring": inspect.getdoc(SamplerV2.run),
    "result_access": f"publication.data.{REGISTER}.get_counts()",
    "evidence": "Installed SDK inspection; no claim of external page retrieval.",
})
options = {
    "default_shots": SHOTS,
    "dynamical_decoupling": {"enable": False},
    "twirling": {"enable_gates": False, "enable_measure": False},
}
hardware_sampler = SamplerV2(mode=backend, options=options)
manifest["sampler_options"] = options

# Balanced blocks: the second block reverses the entire first block.
# Each block contains one complete repetition of every state/variant pair.
first_block = [
    (variant, name)
    for name in state_names for variant in variant_names
]
schedule = []
for block in range(REPETITIONS):
    order = first_block if block == 0 else list(reversed(first_block))
    for variant, name in order:
        records = sorted(
            [
                x for x in circuit_records
                if x["variant"] == variant and x["state"] == name
            ],
            key=lambda x: x["group"],
        )
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        schedule.append({
            "job_index": len(schedule),
            "block": block + 1,
            "repetition": block + 1,
            "variant": variant,
            "state": name,
            "publication_count": len(groups),
            "shots_per_publication": SHOTS,
            "requested_shots": len(groups) * SHOTS,
            "publication_circuit_associations": [
                {
                    "publication_index": x["group"],
                    "group_index": x["group"],
                    "basis": x["basis"],
                    "logical_circuit_hash": x["logical_circuit_hash"],
                    "compiled_circuit_hash": x["compiled_circuit_hash"],
                    "classical_register": REGISTER,
                    "classical_to_physical": x["classical_to_physical"],
                }
                for x in records
            ],
        })
manifest.update({
    "pauli_term_count": len(original_terms),
    "measurement_group_count": len(groups),
    "schedule": schedule,
    "sampler_job_count": len(schedule),
    "total_circuit_executions": sum(x["publication_count"] for x in schedule),
    "total_requested_shots": sum(x["requested_shots"] for x in schedule),
    "status": "validated_awaiting_confirmation",
    "budget": (
        "This exact proposed workload is the only budget authorized by YES. "
        "No automatic retries, extra repetitions, or adaptive jobs."
    ),
    "execution_balance": (
        "Two complete forward/reverse blocks. This balances ordinal position "
        "but cannot remove unknown device drift."
    ),
})
save_json("manifest.json", manifest)

print("\n=== FULL PROPOSED QPU WORKLOAD — NO JOBS SUBMITTED ===")
print(
    f"Backend={BACKEND_NAME}; logical qubits={NQ}; physical width={width}; "
    f"Pauli terms={len(original_terms)}; measurement groups={len(groups)}"
)
print(
    f"Variants={len(variants)}; states={len(state_names)}; "
    f"complete repetitions/state/variant={REPETITIONS}; shots/circuit={SHOTS}"
)
for row in schedule:
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    print(
        f"Job {row['job_index']:02d}, block {row['block']}: "
        f"{row['variant']:24s} {row['state']:6s}, "
        f"{row['publication_count']} circuits x {SHOTS} shots "
        f"= {row['requested_shots']:,} requested shots"
    )
print(
    f"TOTAL: {manifest['sampler_job_count']} job-mode Sampler jobs; "
    f"{manifest['total_circuit_executions']} circuit executions; "
    f"{manifest['total_requested_shots']:,} requested shots."
)
print("Every group is executed in every complete repetition, including identity.")
print("No Session, Estimator, mitigation, postselection, or additional optimization.")
print(
    "This is a new controlled benchmark. Historical repetitions are NOT counted "
    "as acceptance evidence or merged with these repetitions."
)

confirmation = input(
    "Type YES to approve exactly this complete QPU workload: "
).strip()
if confirmation != "YES":
    manifest["status"] = "not_submitted"
    save_json("manifest.json", manifest)
    print("No QPU jobs submitted; unavailable hardware uncertainties are not zero.")
    return {
        "manifest": manifest,
        "evidence_directory": str(evidence.resolve()),
        "exact": exact,
        "local": local_complete,
        "compiled_comparisons": factor_comparisons,
        "hardware": None,
    }

hardware_groups, hardware_complete = [], []
stage = "authorized"
try:
    manifest["status"] = "authorized"
    manifest["confirmation_utc"] = utc()
    save_json("manifest.json", manifest)

    # Submit/retrieve sequentially to preserve printed block order.
    # No second transpilation or unchecked copy is submitted.
    for scheduled in schedule:
        stage = f"job-{scheduled['job_index']}-pre_submission"
        circuits = compiled[(scheduled["variant"], scheduled["state"])]
        require(
            len(circuits) == scheduled["publication_count"],
            "Circuit workload changed after confirmation.",
        )
        for circuit, association in zip(
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        circuits, scheduled["publication_circuit_associations"]
    ):
        require(
            circuit_hash(circuit) == association["compiled_circuit_hash"],
            "Validated circuit changed before submission.",
        )

    stage = f"job-{scheduled['job_index']}-submission"
    job = hardware_sampler.run(circuits, shots=SHOTS)
    job_record = {
        **scheduled,
        "job_id": job.job_id(),
        "submitted_utc": utc(),
        "status": "submitted",
    }
    manifest["jobs"].append(job_record)
    save_json("manifest.json", manifest)
    print(
        f"Submitted job {scheduled['job_index']:02d}: "
        f"{job_record['job_id']}"
    )

    stage = f"job-{scheduled['job_index']}-retrieval"
    result = job.result()
    save_pickle(
        f"job_{scheduled['job_index']:02d}_raw_primitive_result.pkl",
        result,
    )
    require(
        len(result) == len(groups), "Returned publication count mismatch."
    )
    counts = [returned_counts(publication) for publication in result]
    save_json(
        f"job_{scheduled['job_index']:02d}_raw_counts.json",
        {
            "job_id": job.job_id(),
            "schedule": scheduled,
            "register": REGISTER,
            "counts": counts,
            "publication_metadata": [pub.metadata for pub in result],
            "result_metadata": getattr(result, "metadata", None),
        },
    )
    # Raw result and raw counts are persisted before statistical checks.
    statistics = []
    for gi, group_counts in enumerate(counts):
        stats = group_statistics(
            group_counts, gi, ideal[scheduled["state"]][gi]
        )
        statistics.append(stats)
        hardware_groups.append({
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
            "variant": scheduled["variant"],
            "state": scheduled["state"],
            "block": scheduled["block"],
            "repetition": scheduled["repetition"],
            "job_id": job.job_id(),
            "group": gi,
            "basis": groups[gi]["basis"],
            "statistics": stats,
        })
    complete = complete_statistics(statistics)
    hardware_complete.append({
        "variant": scheduled["variant"],
        "state": scheduled["state"],
        "block": scheduled["block"],
        "repetition": scheduled["repetition"],
        "job_id": job.job_id(),
        **complete,
        "difference_from_exact": (
            complete["energy"]
            - exact[scheduled["state"]]["exact_measurement_energy"]
        ),
    })
    job_record["status"] = "completed"
    job_record["completed_utc"] = utc()
    try:
        job_record["metrics"] = job.metrics()
    except Exception as error:
        job_record["metrics_unavailable_exception_type"] = type(error).__name__
    save_json("hardware_group_statistics.json", hardware_groups)
    save_json("hardware_complete_repetitions.json", hardware_complete)
    save_json("manifest.json", manifest)
    print(
        f"COMPLETE {scheduled['variant']}/{scheduled['state']}/"
        f"rep{scheduled['repetition']}: "
        f"{complete['energy']:.12f} +/- "
        f"{complete['sampling_standard_error']:.6f} Ha (sampling SE)"
    )

summary, group_analysis, repetition_analysis = [], [], []
print("\n=== FINAL SCIENTIFIC COMPARISON ===")
print(
    "Sampling SE excludes systematic device error. Between-repetition "
    "SD and SE are separate descriptive estimates from only two repeats."
)
for name in state_names:
    for variant in variant_names:
        rows = sorted(
            [
                x for x in hardware_complete
                if x["state"] == name and x["variant"] == variant
            ],
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        key=lambda x: x["repetition"],
    )
    require(len(rows) == REPETITIONS, "Incomplete hardware repetitions.")
    energies = [x["energy"] for x in rows]
    mean = float(np.mean(energies))
    shot_variance = math.fsum(
        x["sampling_variance"] for x in rows
    ) / REPETITIONS ** 2
    repetition_sd = float(np.std(energies, ddof=1))
    local_rows = [
        x for x in local_complete
        if x["state"] == name and x["variant"] == variant
    ]
    item = {
        "state": name,
        "variant": variant,
        "mitigation": "NONE",
        "E_exact": analytical_reference,
        "E0_fd": finite_reference,
        "classical_variational_energy": classical[name],
        "exact_pauli_energy": exact[name]["direct_pauli_energy"],
        "exact_measurement_energy": exact[name]["exact_measurement_energy"],
        "local_repetitions": local_rows,
        "local_mean_energy": float(np.mean([
            x["energy"] for x in local_rows
        ])),
        "local_mean_sampling_standard_error": math.sqrt(
            math.fsum(x["sampling_variance"] for x in local_rows)
            / REPETITIONS ** 2
        ),
        "hardware_repetition_energies": energies,
        "hardware_repetition_sampling_standard_errors": [
            x["sampling_standard_error"] for x in rows
        ],
        "hardware_mean_energy": mean,
        "hardware_mean_sampling_variance": shot_variance,
        "hardware_mean_sampling_standard_error": math.sqrt(shot_variance),
        "between_repetition_standard_deviation": repetition_sd,
        "between_repetition_standard_error_of_mean": (
            repetition_sd / math.sqrt(REPETITIONS)
        ),
        "finite_representation_difference": (
            finite_reference - analytical_reference
        ),
        "classical_variational_residual": (
            classical[name] - finite_reference
        ),
        "quantum_implementation_difference": (
            exact[name]["exact_measurement_energy"] - classical[name]
        ),
        "hardware_difference": (
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
            mean - exact[name]["exact_measurement_energy"]
        ),
        "job_ids": [x["job_id"] for x in rows],
    }
    summary.append(item)
    print(f"\n{name}/{variant}")
    for key, value in item.items():
        if key not in ("local_repetitions", "hardware_mean_sampling_variance"):
            print(f"  {key}: {value}")

    averaged_groups = []
    for gi, group in enumerate(groups):
        grow = sorted(
            [
                x for x in hardware_groups
                if x["state"] == name and x["variant"] == variant
                and x["group"] == gi
            ],
            key=lambda x: x["repetition"],
        )
        require(len(grow) == REPETITIONS, "Missing group repetition.")
        gs = [x["statistics"] for x in grow]
        group_mean = float(np.mean([x["energy"] for x in gs]))
        variance = math.fsum(
            x["sampling_variance"] for x in gs
        ) / REPETITIONS ** 2
        intended = exact[name]["group_contributions"][gi]
        averaged_groups.append({
            "group": gi,
            "basis": group["basis"],
            "ideal_energy": intended,
            "mean_hardware_contribution": group_mean,
            "mean_residual": group_mean - intended,
            "sampling_variance_of_mean": variance,
            "sampling_standard_error_of_mean": math.sqrt(variance),
            "mean_probability_total_variation": float(np.mean([
                x["probability_total_variation"] for x in gs
            ])),
        })

    signed = math.fsum(x["mean_residual"] for x in averaged_groups)
    absolute = math.fsum(abs(x["mean_residual"]) for x in averaged_groups)
    positive = math.fsum(
        max(0.0, x["mean_residual"]) for x in averaged_groups
    )
    negative = math.fsum(
        min(0.0, x["mean_residual"]) for x in averaged_groups
    )
    require(
        abs(signed - item["hardware_difference"]) <= ENERGY_TOL,
        "Group residuals do not reconstruct the hardware difference.",
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    )
    ranking = sorted(
        averaged_groups, key=lambda x: abs(x["mean_residual"]), reverse=True
    )
    analysis = {
        "state": name, "variant": variant,
        "groups_ranked_by_absolute_mean_residual": ranking,
        "signed_residual_sum": signed,
        "absolute_residual_sum": absolute,
        "positive_residual_sum": positive,
        "negative_residual_sum": negative,
        "cancellation_fraction": (
            1 - abs(signed) / absolute if absolute > 0 else None
        ),
        "interpretation": (
            "Descriptive cancellation, not a causal explanation. "
            "Small total energy residual can conceal large group residuals."
        ),
    }
    group_analysis.append(analysis)
    print(
        f"  Residual cancellation: signed={signed:+.6f}, "
        f"sum|residual|={absolute:.6f}, positive={positive:+.6f}, "
        f"negative={negative:+.6f}, "
        f"cancellation fraction={analysis['cancellation_fraction']}"
    )
    print("  Dominant group residuals:")
    for group in ranking[:min(10, len(ranking))]:
        print(
            f"    g{group['group']:03d} {group['basis']} "
            f"residual={group['mean_residual']:+.6f}, "
            f"sampling SE={group['sampling_standard_error_of_mean']:.6f}, "
            f"TV={group['mean_probability_total_variation']:.5f}"
        )

    differences = []
    for gi in range(len(groups)):
        repetitions = sorted(
            [
                x for x in hardware_groups
                if x["state"] == name and x["variant"] == variant
                and x["group"] == gi
            ],
            key=lambda x: x["repetition"],
        )
        first = repetitions[0]["statistics"]
        second = repetitions[1]["statistics"]
        delta = second["energy"] - first["energy"]
        delta_variance = (
            first["sampling_variance"] + second["sampling_variance"]
        )
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
            differences.append({
                "group": gi, "basis": groups[gi]["basis"],
                "repetition_2_minus_1": delta,
                "sampling_variance_of_difference": delta_variance,
                "sampling_standard_error_of_difference": math.sqrt(delta_variance),
            })
        delta = energies[1] - energies[0]
        delta_se = math.sqrt(
            rows[0]["sampling_variance"] + rows[1]["sampling_variance"]
        )
        require(
            abs(math.fsum(
                x["repetition_2_minus_1"] for x in differences
            ) - delta) <= ENERGY_TOL,
            "Group differences fail repetition reconstruction.",
        )
        rep_analysis = {
            "state": name, "variant": variant,
            "repetition_2_minus_1": delta,
            "sampling_standard_error_of_difference": delta_se,
            "sampling_only_z": delta / delta_se if delta_se > 0 else None,
            "groups_ranked_by_absolute_repetition_difference": sorted(
                differences,
                key=lambda x: abs(x["repetition_2_minus_1"]),
                reverse=True,
            ),
            "cause": (
                "Unresolved. Sampling-only comparison is descriptive; "
                "drift, coherent error, readout effects, and other mechanisms "
                "are not identified by this experiment alone."
            ),
        }
        repetition_analysis.append(rep_analysis)
        print(
            f"  Rep2-Rep1={delta:+.6f} +/- {delta_se:.6f} "
            f"(sampling SE), sampling-only z={rep_analysis['sampling_only_z']}"
        )
        if name == "SPSA" and variant == "sequential_calibrated":
            print("  CALIBRATED SPSA REPETITION DISCREPANCY:")
            for difference in rep_analysis[
                "groups_ranked_by_absolute_repetition_difference"
            ][:min(10, len(groups))]:
                print("   ", difference)
            print("  Cause remains unresolved; no causal label is assigned.")

measured_comparisons = []
print("\n=== ONE-FACTOR HARDWARE COMPARISONS ===")
for variant in variants:
    if variant["compare_to"] is None:
        continue
    for name in state_names:
```

```
        current = next(
            x for x in summary
            if x["state"] == name and x["variant"] == variant["name"]
        )
        reference = next(
            x for x in summary
            if x["state"] == name and x["variant"] == variant["compare_to"]
        )
        comparison = {
            "state": name,
            "variant": variant["name"],
            "comparator": variant["compare_to"],
            "changed_factor": variant["changed_factor"],
            "mean_energy_difference": (
                current["hardware_mean_energy"]
                - reference["hardware_mean_energy"]
            ),
            "sampling_standard_error_of_energy_difference": math.sqrt(
                current["hardware_mean_sampling_variance"]
                + reference["hardware_mean_sampling_variance"]
            ),
            "absolute_energy_residual_reduction": (
                abs(reference["hardware_difference"])
                - abs(current["hardware_difference"])
            ),
            "interpretation": (
                "Positive absolute-residual reduction is an observed point "
                "estimate of improvement, not proof of drift-robust superiority. "
                "No adjustment or mitigation has been applied."
            ),
        }
        measured_comparisons.append(comparison)
        print(comparison)

save_json("final_numerical_summary.json", summary)
save_json("group_residual_and_cancellation_analysis.json", group_analysis)
save_json("repetition_discrepancy_analysis.json", repetition_analysis)
save_json("one_factor_hardware_comparisons.json", measured_comparisons)
try:
    after = backend.properties(refresh=True)
    save_json("calibration_after.json", {
        "retrieved_utc": utc(),
        "backend_properties": None if after is None else after.to_dict(),
    })
except Exception as error:
    save_json("calibration_after_unavailable.json", {
        "exception_type": type(error).__name__, "utc": utc()
    })

manifest["status"] = "completed"
manifest["finished_utc"] = utc()
```

```
        save_json("manifest.json", manifest)
        print("\nJob IDs:", [x["job_id"] for x in manifest["jobs"]])
        print("Evidence directory:", evidence.resolve())
        print(
            "Historical calibrated SPSA repetitions were not supplied numerically. "
            "They have not been reinterpreted as new evidence; the analysis above "
            "applies only to the newly authorized executions."
        )
        return {
            "manifest": manifest,
            "evidence_directory": str(evidence.resolve()),
            "summary": summary,
            "compiled_comparisons": factor_comparisons,
            "hardware_comparisons": measured_comparisons,
            "group_analysis": group_analysis,
            "repetition_analysis": repetition_analysis,
        }

    except BaseException as error:
        manifest["status"] = "interrupted_or_failed"
        manifest["failure"] = {
            "stage": stage,
            "exception_type": type(error).__name__,
            "utc": utc(),
        }
        save_json("manifest.json", manifest)
        save_json("hardware_group_statistics.json", hardware_groups)
        save_json("hardware_complete_repetitions.json", hardware_complete)
        print(
            f"Stopped at {stage}. Submitted job IDs and partial/raw evidence remain "
            f"in {evidence.resolve()}. Inspect the manifest before rerunning; "
            "no automatic retry or extra QPU workload is authorized."
        )
        raise


# Strict Part I dependency sweep: only these ten existing symbols are read.
# All other working names are imported or defined inside this cell.
_part_ii_required = (
    "H", "r", "h", "n_interior", "R_max", "E_exact", "E0_fd",
    "alpha_cobyla", "alpha_spsa", "normalized_trial_state",
)
_part_ii_missing = [name for name in _part_ii_required if name not in globals()]
if _part_ii_missing:
    raise NameError(
        "Run authoritative Part I first. Missing: " + ", ".join(_part_ii_missing)
    )

part_ii_results = _run_atomic_hydrogen_part_ii(
    **{name: globals()[name] for name in _part_ii_required}
)
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

Third-run results (verbatim)

```
=== CHANGE SUMMARY ===
1. Exact bond-two sequential synthesis replaces the proposed Initialize alternative; all Schmidt cuts, isometries, and both output states tested.
2. Synthesis, compiler-level, and placement contrasts each change one factor.
3. Independent local streams; preparation/basis/bit-map/compiled corruption controls; checked circuits reused without recompilation.
4. Balanced forward/reverse execution blocks; full workload requires YES.
5. Per-group residuals, cancellation, repetition differences, separate sampling/repetition uncertainties, hashes, and job-circuit evidence.
No new optimization, Hamiltonian changes, parameter changes, mitigation, offsets, postselection, or historical-result revalidation.

Versions: {'numpy': '2.1.3', 'scipy': '1.16.3', 'qiskit': '2.5.2', 'qiskit-ibm-runtime': '0.50.0'}
Evidence: /content/hydrogen_part_ii_20261013T210806_159257Z_a657370a

=== LOCAL ACCEPTANCE RESULTS ===
Pauli terms=63, QWC groups=17, H reconstruction error=1.776e-15
COBYLA Schmidt ranks: [2, 2, 2, 2] max rank-two tail: 1.6205015931422403e-17 independent state error: 2.8960847954417557e-16
SPSA Schmidt ranks: [2, 2, 2, 2] max rank-two tail: 1.7360731119217328e-17 independent state error: 4.2119421759412193e-16
COBYLA: classical=-0.4609739279597634, Pauli=-0.4609739279597633, measurement=-0.4609739279597616 Ha
SPSA: classical=-0.4607792133832190, Pauli=-0.4607792133832186, measurement=-0.4607792133832168 Ha
LOCAL baseline/COBYLA/rep1: -0.475873950777 +/- 0.019796 Ha (sampling SE)
LOCAL baseline/COBYLA/rep2: -0.458351316906 +/- 0.019753 Ha (sampling SE)
LOCAL baseline/SPSA/rep1: -0.469823312305 +/- 0.020091 Ha (sampling SE)
LOCAL baseline/SPSA/rep2: -0.481874863761 +/- 0.019844 Ha (sampling SE)
LOCAL sequential/COBYLA/rep1: -0.448455311003 +/- 0.019739 Ha (sampling SE)
LOCAL sequential/COBYLA/rep2: -0.480818913778 +/- 0.019785 Ha (sampling SE)
LOCAL sequential/SPSA/rep1: -0.468814779009 +/- 0.019929 Ha (sampling SE)
LOCAL sequential/SPSA/rep2: -0.439763655890 +/- 0.019985 Ha (sampling SE)
LOCAL sequential_level3/COBYLA/rep1: -0.425467344012 +/- 0.019866 Ha (sampling SE)
LOCAL sequential_level3/COBYLA/rep2: -0.470131665184 +/- 0.019861 Ha (sampling SE)
LOCAL sequential_level3/SPSA/rep1: -0.464087637038 +/- 0.019835 Ha (sampling SE)
LOCAL sequential_level3/SPSA/rep2: -0.427716620309 +/- 0.019878 Ha (sampling SE)
LOCAL sequential_calibrated/COBYLA/rep1: -0.487835933140 +/- 0.019838 Ha (sampling SE)
LOCAL sequential_calibrated/COBYLA/rep2: -0.464116710481 +/- 0.019753 Ha (sampling SE)
LOCAL sequential_calibrated/SPSA/rep1: -0.473212870643 +/- 0.019999 Ha (sampling SE)
LOCAL sequential_calibrated/SPSA/rep2: -0.463617720831 +/- 0.020018 Ha (sampling SE)
Local acceptance: PASS. Corruption controls so far: [{'control': 'sequential-preparation/x/q4', 'detected': True}, {'control': 'wrong-basis/group16/q0/X', 'detected': True}, {'control': 'classical-bit-swap/group16/c0/c4', 'detected': True}]
qiskit_runtime_service._discover_account:WARNING:2026-10-13 21:08:29,575: Loading account with the given token. A saved account will not be used.
qiskit_runtime_service.__init__:WARNING:2026-10-13 21:08:32,019: Instance was not set at service instantiation. Free and trial plan instances will be prioritized. Based on the following filters: (tags: None, region: us-east, eu-de), and available plans: (open), the available account instances are: open-instance. If you need a specific instance set it explicitly either by using a saved account with a saved default instance or passing it in
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
directly to QiskitRuntimeService(). Alternatively, pass instance='auto' or save it to your account for auto-selection without warning.
qiskit_runtime_service.backends:WARNING:2026-10-13 21:08:32,020: Using instance: open-instance, plan: open
=== BACKEND AND COMPILED ACCEPTANCE ===
Backend=ibm_fez; logical qubits=5; physical width=156
Baseline layout: [3, 4, 5, 6, 7]
Calibrated layout: [142, 143, 144, 145, 146]
Calibration heuristic score: 0.05521615837797252
Calibration timestamp: 2026-10-13 20:41:52+00:00
Compact compiled-validator self-test error: 1.6653345369377348e-16
All compiled measurement probabilities: PASS
All negative controls: [{'control': 'sequential-preparation/x/q4', 'detected': True}, {'control': 'wrong-basis/group16/q0/X', 'detected': True}, {'control': 'classical-bit-swap/group16/c0/c4', 'detected': True}, {'control': 'compiled-output-X/group16/physical7', 'detected': True}]

PREPARATION-ONLY METRICS; HISTORICAL 46-GATE COMPARISON:
baseline              COBYLA: depth=158, two-qubit gates=46, count-46=+0
baseline              SPSA  : depth=160, two-qubit gates=46, count-46=+0
sequential             COBYLA: depth=41, two-qubit gates=10, count-46=-36
sequential             SPSA  : depth=41, two-qubit gates=10, count-46=-36
sequential_level3       COBYLA: depth=40, two-qubit gates=10, count-46=-36
sequential_level3       SPSA  : depth=40, two-qubit gates=10, count-46=-36
sequential_calibrated    COBYLA: depth=41, two-qubit gates=10, count-46=-36
sequential_calibrated    SPSA  : depth=41, two-qubit gates=10, count-46=-36
Historical 46-gate settings/counting scope were not supplied. The actual freshly compiled baseline above is the controlled comparator; a mismatch is reported, not repaired or silently relabeled.

COMPLETE MEASUREMENT-CIRCUIT METRICS AND MAPPINGS:
baseline              COBYLA g000 YXXXY depth= 160 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline              COBYLA g001 XYXXY depth= 160 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline              COBYLA g002 YXXYX depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline              COBYLA g003 YXYYY depth= 160 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline              COBYLA g004 YYYXY depth= 160 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline              COBYLA g005 XXYXY depth= 160 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline              COBYLA g006 XYXYX depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline              COBYLA g007 XYYYY depth= 160 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline              COBYLA g008 YXYXX depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline              COBYLA g009 YYXYY depth= 160 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline              COBYLA g010 YYYYX depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
baseline          COBYLA g011 XXXYY depth= 160 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline          COBYLA g012 XXYYX depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline          COBYLA g013 XYYXX depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline          COBYLA g014 YYXXX depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline          COBYLA g015 XXXXX depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline          COBYLA g016 ZZZZZ depth= 159 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline          SPSA   g000 YXXXY depth= 160 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline          SPSA   g001 XYXXY depth= 160 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline          SPSA   g002 YXXYX depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline          SPSA   g003 YXYYY depth= 160 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline          SPSA   g004 YYYXY depth= 160 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline          SPSA   g005 XXYXY depth= 160 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline          SPSA   g006 XYXYX depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline          SPSA   g007 XYYYY depth= 160 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline          SPSA   g008 YXYXX depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline          SPSA   g009 YYXYY depth= 160 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline          SPSA   g010 YYYYX depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline          SPSA   g011 XXXYY depth= 160 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline          SPSA   g012 XXYYX depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline          SPSA   g013 XYYXX depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline          SPSA   g014 YYXXX depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline          SPSA   g015 XXXXX depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
baseline          SPSA   g016 ZZZZZ depth= 161 2q=  46 initial=[3, 4, 5, 6, 7] final=[5, 4, 6, 3, 7] c->physical={0: 5, 1: 4, 2: 6, 3: 3, 4: 7}
sequential         COBYLA g000 YXXXY depth=  43 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c->physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential         COBYLA g001 XYXXY depth=  43 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c->physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential         COBYLA g002 YXXYX depth=  43 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential          COBYLA g003 YXYYY depth=  43 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential          COBYLA g004 YYYXY depth=  42 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential          COBYLA g005 XXYXY depth=  43 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential          COBYLA g006 XYXYX depth=  43 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential          COBYLA g007 XYYYY depth=  43 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential          COBYLA g008 YXYXX depth=  43 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential          COBYLA g009 YYXYY depth=  42 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential          COBYLA g010 YYYYX depth=  42 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential          COBYLA g011 XXXYY depth=  43 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential          COBYLA g012 XXYYX depth=  43 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential          COBYLA g013 XYYXX depth=  43 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential          COBYLA g014 YYXXX depth=  42 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential          COBYLA g015 XXXXX depth=  43 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential          COBYLA g016 ZZZZZ depth=  42 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential          SPSA   g000 YXXXY depth=  43 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential          SPSA   g001 XYXXY depth=  43 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential          SPSA   g002 YXXYX depth=  43 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential          SPSA   g003 YXYYY depth=  43 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential          SPSA   g004 YYYXY depth=  42 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential          SPSA   g005 XXYXY depth=  43 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential          SPSA   g006 XYXYX depth=  43 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential          SPSA   g007 XYYYY depth=  43 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential          SPSA   g008 YXYXX depth=  43 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential          SPSA   g009 YYXYY depth=  42 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential          SPSA   g010 YYYYX depth=  42 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
sequential          SPSA   g011 XXXYY depth=  43 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential          SPSA   g012 XXYYX depth=  43 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential          SPSA   g013 XYYXX depth=  43 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential          SPSA   g014 YYXXX depth=  42 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential          SPSA   g015 XXXXX depth=  43 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential          SPSA   g016 ZZZZZ depth=  42 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential_level3       COBYLA g000 YXXXY depth=  41 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential_level3       COBYLA g001 XYXXY depth=  41 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential_level3       COBYLA g002 YXXYX depth=  42 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential_level3       COBYLA g003 YXYYY depth=  40 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential_level3       COBYLA g004 YYYXY depth=  40 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential_level3       COBYLA g005 XXYXY depth=  41 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential_level3       COBYLA g006 XYXYX depth=  42 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential_level3       COBYLA g007 XYYYY depth=  40 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential_level3       COBYLA g008 YXYXX depth=  41 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential_level3       COBYLA g009 YYXYY depth=  40 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential_level3       COBYLA g010 YYYYX depth=  42 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential_level3       COBYLA g011 XXXYY depth=  41 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential_level3       COBYLA g012 XXYYX depth=  42 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential_level3       COBYLA g013 XYYXX depth=  43 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential_level3       COBYLA g014 YYXXX depth=  41 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential_level3       COBYLA g015 XXXXX depth=  42 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential_level3       COBYLA g016 ZZZZZ depth=  41 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential_level3       SPSA   g000 YXXXY depth=  41 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential_level3       SPSA   g001 XYXXY depth=  41 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential_level3       SPSA   g002 YXXYX depth=  42 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential_level3       SPSA   g003 YXYYY depth=  40 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential_level3       SPSA   g004 YYYXY depth=  39 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential_level3       SPSA   g005 XXYXY depth=  42 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential_level3       SPSA   g006 XYXYX depth=  42 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential_level3       SPSA   g007 XYYYY depth=  42 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential_level3       SPSA   g008 YXYXX depth=  41 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential_level3       SPSA   g009 YYXYY depth=  40 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential_level3       SPSA   g010 YYYYX depth=  42 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential_level3       SPSA   g011 XXXYY depth=  41 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential_level3       SPSA   g012 XXYYX depth=  43 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential_level3       SPSA   g013 XYYXX depth=  43 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential_level3       SPSA   g014 YYXXX depth=  41 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential_level3       SPSA   g015 XXXXX depth=  42 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential_level3       SPSA   g016 ZZZZZ depth=  41 2q=  10 initial=[3, 4, 5, 6, 7] final=[3, 4, 5, 6, 7] c-
>physical={0: 3, 1: 4, 2: 5, 3: 6, 4: 7}
sequential_calibrated    COBYLA g000 YXXXY depth=  43 2q=  10 initial=[142, 143, 144, 145, 146]
final=[142, 143, 144, 145, 146] c->physical={0: 142, 1: 143, 2: 144, 3: 145, 4: 146}
sequential_calibrated    COBYLA g001 XYXXY depth=  43 2q=  10 initial=[142, 143, 144, 145, 146]
final=[142, 143, 144, 145, 146] c->physical={0: 142, 1: 143, 2: 144, 3: 145, 4: 146}
sequential_calibrated    COBYLA g002 YXXYX depth=  43 2q=  10 initial=[142, 143, 144, 145, 146]
final=[142, 143, 144, 145, 146] c->physical={0: 142, 1: 143, 2: 144, 3: 145, 4: 146}
sequential_calibrated    COBYLA g003 YXYYY depth=  43 2q=  10 initial=[142, 143, 144, 145, 146]
final=[142, 143, 144, 145, 146] c->physical={0: 142, 1: 143, 2: 144, 3: 145, 4: 146}
sequential_calibrated    COBYLA g004 YYYXY depth=  42 2q=  10 initial=[142, 143, 144, 145, 146]
final=[142, 143, 144, 145, 146] c->physical={0: 142, 1: 143, 2: 144, 3: 145, 4: 146}
sequential_calibrated    COBYLA g005 XXYXY depth=  43 2q=  10 initial=[142, 143, 144, 145, 146]
final=[142, 143, 144, 145, 146] c->physical={0: 142, 1: 143, 2: 144, 3: 145, 4: 146}
sequential_calibrated    COBYLA g006 XYXYX depth=  43 2q=  10 initial=[142, 143, 144, 145, 146]
final=[142, 143, 144, 145, 146] c->physical={0: 142, 1: 143, 2: 144, 3: 145, 4: 146}
sequential_calibrated    COBYLA g007 XYYYY depth=  43 2q=  10 initial=[142, 143, 144, 145, 146]
final=[142, 143, 144, 145, 146] c->physical={0: 142, 1: 143, 2: 144, 3: 145, 4: 146}
sequential_calibrated    COBYLA g008 YXYXX depth=  43 2q=  10 initial=[142, 143, 144, 145, 146]
final=[142, 143, 144, 145, 146] c->physical={0: 142, 1: 143, 2: 144, 3: 145, 4: 146}
sequential_calibrated    COBYLA g009 YYXYY depth=  42 2q=  10 initial=[142, 143, 144, 145, 146]
final=[142, 143, 144, 145, 146] c->physical={0: 142, 1: 143, 2: 144, 3: 145, 4: 146}
sequential_calibrated    COBYLA g010 YYYYX depth=  42 2q=  10 initial=[142, 143, 144, 145, 146]
final=[142, 143, 144, 145, 146] c->physical={0: 142, 1: 143, 2: 144, 3: 145, 4: 146}
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
sequential_calibrated    COBYLA g011 XXXYY depth=  43 2q=  10 initial=[142, 143, 144, 145, 146]
final=[142, 143, 144, 145, 146] c->physical={0: 142, 1: 143, 2: 144, 3: 145, 4: 146}
sequential_calibrated    COBYLA g012 XXYYX depth=  43 2q=  10 initial=[142, 143, 144, 145, 146]
final=[142, 143, 144, 145, 146] c->physical={0: 142, 1: 143, 2: 144, 3: 145, 4: 146}
sequential_calibrated    COBYLA g013 XYYXX depth=  43 2q=  10 initial=[142, 143, 144, 145, 146]
final=[142, 143, 144, 145, 146] c->physical={0: 142, 1: 143, 2: 144, 3: 145, 4: 146}
sequential_calibrated    COBYLA g014 YYXXX depth=  42 2q=  10 initial=[142, 143, 144, 145, 146]
final=[142, 143, 144, 145, 146] c->physical={0: 142, 1: 143, 2: 144, 3: 145, 4: 146}
sequential_calibrated    COBYLA g015 XXXXX depth=  43 2q=  10 initial=[142, 143, 144, 145, 146]
final=[142, 143, 144, 145, 146] c->physical={0: 142, 1: 143, 2: 144, 3: 145, 4: 146}
sequential_calibrated    COBYLA g016 ZZZZZ depth=  42 2q=  10 initial=[142, 143, 144, 145, 146] final=[142,
143, 144, 145, 146] c->physical={0: 142, 1: 143, 2: 144, 3: 145, 4: 146}
sequential_calibrated    SPSA   g000 YXXXY depth=  43 2q=  10 initial=[142, 143, 144, 145, 146] final=[142,
143, 144, 145, 146] c->physical={0: 142, 1: 143, 2: 144, 3: 145, 4: 146}
sequential_calibrated    SPSA   g001 XYXXY depth=  43 2q=  10 initial=[142, 143, 144, 145, 146] final=[142,
143, 144, 145, 146] c->physical={0: 142, 1: 143, 2: 144, 3: 145, 4: 146}
sequential_calibrated    SPSA   g002 YXXYX depth=  43 2q=  10 initial=[142, 143, 144, 145, 146] final=[142,
143, 144, 145, 146] c->physical={0: 142, 1: 143, 2: 144, 3: 145, 4: 146}
sequential_calibrated    SPSA   g003 YXYYY depth=  43 2q=  10 initial=[142, 143, 144, 145, 146] final=[142,
143, 144, 145, 146] c->physical={0: 142, 1: 143, 2: 144, 3: 145, 4: 146}
sequential_calibrated    SPSA   g004 YYYXY depth=  42 2q=  10 initial=[142, 143, 144, 145, 146] final=[142,
143, 144, 145, 146] c->physical={0: 142, 1: 143, 2: 144, 3: 145, 4: 146}
sequential_calibrated    SPSA   g005 XXYXY depth=  43 2q=  10 initial=[142, 143, 144, 145, 146] final=[142,
143, 144, 145, 146] c->physical={0: 142, 1: 143, 2: 144, 3: 145, 4: 146}
sequential_calibrated    SPSA   g006 XYXYX depth=  43 2q=  10 initial=[142, 143, 144, 145, 146] final=[142,
143, 144, 145, 146] c->physical={0: 142, 1: 143, 2: 144, 3: 145, 4: 146}
sequential_calibrated    SPSA   g007 XYYYY depth=  43 2q=  10 initial=[142, 143, 144, 145, 146] final=[142,
143, 144, 145, 146] c->physical={0: 142, 1: 143, 2: 144, 3: 145, 4: 146}
sequential_calibrated    SPSA   g008 YXYXX depth=  43 2q=  10 initial=[142, 143, 144, 145, 146] final=[142,
143, 144, 145, 146] c->physical={0: 142, 1: 143, 2: 144, 3: 145, 4: 146}
sequential_calibrated    SPSA   g009 YYXYY depth=  42 2q=  10 initial=[142, 143, 144, 145, 146] final=[142,
143, 144, 145, 146] c->physical={0: 142, 1: 143, 2: 144, 3: 145, 4: 146}
sequential_calibrated    SPSA   g010 YYYYX depth=  42 2q=  10 initial=[142, 143, 144, 145, 146] final=[142,
143, 144, 145, 146] c->physical={0: 142, 1: 143, 2: 144, 3: 145, 4: 146}
sequential_calibrated    SPSA   g011 XXXYY depth=  43 2q=  10 initial=[142, 143, 144, 145, 146] final=[142,
143, 144, 145, 146] c->physical={0: 142, 1: 143, 2: 144, 3: 145, 4: 146}
sequential_calibrated    SPSA   g012 XXYYX depth=  43 2q=  10 initial=[142, 143, 144, 145, 146] final=[142,
143, 144, 145, 146] c->physical={0: 142, 1: 143, 2: 144, 3: 145, 4: 146}
sequential_calibrated    SPSA   g013 XYYXX depth=  43 2q=  10 initial=[142, 143, 144, 145, 146] final=[142,
143, 144, 145, 146] c->physical={0: 142, 1: 143, 2: 144, 3: 145, 4: 146}
sequential_calibrated    SPSA   g014 YYXXX depth=  42 2q=  10 initial=[142, 143, 144, 145, 146] final=[142,
143, 144, 145, 146] c->physical={0: 142, 1: 143, 2: 144, 3: 145, 4: 146}
sequential_calibrated    SPSA   g015 XXXXX depth=  43 2q=  10 initial=[142, 143, 144, 145, 146] final=[142,
143, 144, 145, 146] c->physical={0: 142, 1: 143, 2: 144, 3: 145, 4: 146}
sequential_calibrated    SPSA   g016 ZZZZZ depth=  42 2q=  10 initial=[142, 143, 144, 145, 146] final=[142,
143, 144, 145, 146] c->physical={0: 142, 1: 143, 2: 144, 3: 145, 4: 146}

MEAN VERSUS SUMMED METRICS (SUMMED DEPTH IS NOT EXECUTION LATENCY):
{'variant': 'baseline', 'state': 'COBYLA', 'measurement_circuit_count': 17,
'mean_depth_per_measurement_circuit': 160.41176470588235, 'summed_depth_across_measurement_circuits':
2727, 'mean_two_qubit_gates_per_measurement_circuit': 46.0,
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

'summed_two_qubit_gates_across_measurement_circuits': 782, 'maximum_compiled_probability_discrepancy': 6.38378239159465e-15}
{'variant': 'baseline', 'state': 'SPSA', 'measurement_circuit_count': 17, 'mean_depth_per_measurement_circuit': 160.52941176470588, 'summed_depth_across_measurement_circuits': 2729,
'mean_two_qubit_gates_per_measurement_circuit': 46.0,
'summed_two_qubit_gates_across_measurement_circuits': 782, 'maximum_compiled_probability_discrepancy': 4.6074255521944e-15}
{'variant': 'sequential', 'state': 'COBYLA', 'measurement_circuit_count': 17,
'mean_depth_per_measurement_circuit': 42.705882352941174, 'summed_depth_across_measurement_circuits': 726, 'mean_two_qubit_gates_per_measurement_circuit': 10.0,
'summed_two_qubit_gates_across_measurement_circuits': 170, 'maximum_compiled_probability_discrepancy': 1.2753686995381486e-14}
{'variant': 'sequential', 'state': 'SPSA', 'measurement_circuit_count': 17, 'mean_depth_per_measurement_circuit': 42.705882352941174, 'summed_depth_across_measurement_circuits': 726,
'mean_two_qubit_gates_per_measurement_circuit': 10.0,
'summed_two_qubit_gates_across_measurement_circuits': 170, 'maximum_compiled_probability_discrepancy': 1.5765166949677223e-14}
{'variant': 'sequential_level3', 'state': 'COBYLA', 'measurement_circuit_count': 17,
'mean_depth_per_measurement_circuit': 41.1764705882353, 'summed_depth_across_measurement_circuits': 700, 'mean_two_qubit_gates_per_measurement_circuit': 10.0,
'summed_two_qubit_gates_across_measurement_circuits': 170, 'maximum_compiled_probability_discrepancy': 1.2906342661267445e-14}
{'variant': 'sequential_level3', 'state': 'SPSA', 'measurement_circuit_count': 17,
'mean_depth_per_measurement_circuit': 41.35294117647059, 'summed_depth_across_measurement_circuits': 703, 'mean_two_qubit_gates_per_measurement_circuit': 10.0,
'summed_two_qubit_gates_across_measurement_circuits': 170, 'maximum_compiled_probability_discrepancy': 1.587618925213974e-14}
{'variant': 'sequential_calibrated', 'state': 'COBYLA', 'measurement_circuit_count': 17,
'mean_depth_per_measurement_circuit': 42.705882352941174, 'summed_depth_across_measurement_circuits': 726, 'mean_two_qubit_gates_per_measurement_circuit': 10.0,
'summed_two_qubit_gates_across_measurement_circuits': 170, 'maximum_compiled_probability_discrepancy': 1.2753686995381486e-14}
{'variant': 'sequential_calibrated', 'state': 'SPSA', 'measurement_circuit_count': 17,
'mean_depth_per_measurement_circuit': 42.705882352941174, 'summed_depth_across_measurement_circuits': 726, 'mean_two_qubit_gates_per_measurement_circuit': 10.0,
'summed_two_qubit_gates_across_measurement_circuits': 170, 'maximum_compiled_probability_discrepancy': 1.5654144647214707e-14}

ONE-FACTOR COMPARISONS:

{'state': 'COBYLA', 'variant': 'sequential', 'comparator': 'baseline', 'changed_factor': 'synthesis',
'mean_depth_difference': -117.70588235294117, 'summed_two_qubit_gate_difference': -612, 'interpretation': 'Negative differences are measured compiled-cost reductions.'}
{'state': 'SPSA', 'variant': 'sequential', 'comparator': 'baseline', 'changed_factor': 'synthesis',
'mean_depth_difference': -117.82352941176471, 'summed_two_qubit_gate_difference': -612, 'interpretation': 'Negative differences are measured compiled-cost reductions.'}
{'state': 'COBYLA', 'variant': 'sequential_level3', 'comparator': 'sequential', 'changed_factor': 'compiler optimization level', 'mean_depth_difference': -1.529411764705877, 'summed_two_qubit_gate_difference': 0,
'interpretation': 'Negative differences are measured compiled-cost reductions.'}
{'state': 'SPSA', 'variant': 'sequential_level3', 'comparator': 'sequential', 'changed_factor': 'compiler optimization level', 'mean_depth_difference': -1.352941176470587, 'summed_two_qubit_gate_difference': 0, 'interpretation': 'Negative differences are measured compiled-cost reductions.'}

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

{'state': 'COBYLA', 'variant': 'sequential_calibrated', 'comparator': 'sequential', 'changed_factor': 'placement', 'mean_depth_difference': 0.0, 'summed_two_qubit_gate_difference': 0, 'interpretation': 'Negative differences are measured compiled-cost reductions.'}
{'state': 'SPSA', 'variant': 'sequential_calibrated', 'comparator': 'sequential', 'changed_factor': 'placement', 'mean_depth_difference': 0.0, 'summed_two_qubit_gate_difference': 0, 'interpretation': 'Negative differences are measured compiled-cost reductions.'}

RELEVANT CALIBRATION; FULL SNAPSHOT SAVED:
q3: readout_error=0.0218505859375, {'T1_seconds': 0.0001801762372280652 8, 'T2_seconds': 0.0002005986924270112, 'frequency_Hz': None}
q4: readout_error=0.0037841796875, {'T1_seconds': 0.0001692953634448355 3, 'T2_seconds': 0.0001362759070836563 3, 'frequency_Hz': None}
q5: readout_error=0.0076904296875, {'T1_seconds': 0.0002189301544354873, 'T2_seconds': 0.0001694474964624521 8, 'frequency_Hz': None}
q6: readout_error=0.0030517578125, {'T1_seconds': 0.0001802924921549945 2, 'T2_seconds': 0.0001518867989153694, 'frequency_Hz': None}
q7: readout_error=0.0140380859375, {'T1_seconds': 0.0001425651911368348, 'T2_seconds': 4.139721640027132e-05, 'frequency_Hz': None}
q142: readout_error=0.00390625, {'T1_seconds': 0.0002125274057346072 7, 'T2_seconds': 0.0002350959650578019 5, 'frequency_Hz': None}
q143: readout_error=0.0047607421875, {'T1_seconds': 0.0001306443164365860 7, 'T2_seconds': 0.0001423239549008882, 'frequency_Hz': None}
q144: readout_error=0.0035400390625, {'T1_seconds': 0.0001567077387331029 4, 'T2_seconds': 0.0001842341513463269 6, 'frequency_Hz': None}
q145: readout_error=0.013671875, {'T1_seconds': 0.0001917963308775889 4, 'T2_seconds': 0.0001344646149647542 7, 'frequency_Hz': None}
q146: readout_error=0.0054931640625, {'T1_seconds': 0.0001679874275570017 1, 'T2_seconds': 1.176939324925980 6e-05, 'frequency_Hz': None}
{'operation': 'cz', 'qubits': [3, 4], 'error': 0.0017361226971474575, 'duration_seconds': 6.8e-08}
{'operation': 'cz', 'qubits': [4, 3], 'error': 0.0017361226971474575, 'duration_seconds': 6.8e-08}
{'operation': 'cz', 'qubits': [4, 5], 'error': 0.0029133327733613656, 'duration_seconds': 6.8e-08}
{'operation': 'cz', 'qubits': [5, 4], 'error': 0.0029133327733613656, 'duration_seconds': 6.8e-08}
{'operation': 'cz', 'qubits': [5, 6], 'error': 0.010851730034133006, 'duration_seconds': 6.8e-08}
{'operation': 'cz', 'qubits': [6, 5], 'error': 0.010851730034133006, 'duration_seconds': 6.8e-08}
{'operation': 'cz', 'qubits': [6, 7], 'error': 0.0092442258827872, 'duration_seconds': 6.8e-08}
{'operation': 'cz', 'qubits': [7, 6], 'error': 0.0092442258827872, 'duration_seconds': 6.8e-08}
{'operation': 'cz', 'qubits': [142, 143], 'error': 0.0023260137908588385, 'duration_seconds': 6.8e-08}
{'operation': 'cz', 'qubits': [143, 142], 'error': 0.0023260137908588385, 'duration_seconds': 6.8e-08}
{'operation': 'cz', 'qubits': [143, 144], 'error': 0.0028646026040956007, 'duration_seconds': 6.8e-08}
{'operation': 'cz', 'qubits': [144, 143], 'error': 0.0028646026040956007, 'duration_seconds': 6.8e-08}
{'operation': 'cz', 'qubits': [144, 145], 'error': 0.002347944592682394, 'duration_seconds': 6.8e-08}
{'operation': 'cz', 'qubits': [145, 144], 'error': 0.002347944592682394, 'duration_seconds': 6.8e-08}
{'operation': 'cz', 'qubits': [145, 146], 'error': 0.0014201453490203797, 'duration_seconds': 6.8e-08}
{'operation': 'cz', 'qubits': [146, 145], 'error': 0.0014201453490203797, 'duration_seconds': 6.8e-08}
Calibration proxies are not evidence of energy improvement or a diagnosis.

=== FULL PROPOSED QPU WORKLOAD — NO JOBS SUBMITTED ===
Backend=ibm_fez; logical qubits=5; physical width=156; Pauli terms=63; measurement groups=17
Variants=4; states=2; complete repetitions/state/variant=2; shots/circuit=4096
Job 00, block 1: baseline COBYLA, 17 circuits x 4096 shots = 69,632 requested shots
Job 01, block 1: sequential COBYLA, 17 circuits x 4096 shots = 69,632 requested shots

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
Job 02, block 1: sequential_level3       COBYLA, 17 circuits x 4096 shots = 69,632 requested shots
Job 03, block 1: sequential_calibrated   COBYLA, 17 circuits x 4096 shots = 69,632 requested shots
Job 04, block 1: baseline                SPSA , 17 circuits x 4096 shots = 69,632 requested shots
Job 05, block 1: sequential              SPSA , 17 circuits x 4096 shots = 69,632 requested shots
Job 06, block 1: sequential_level3       SPSA , 17 circuits x 4096 shots = 69,632 requested shots
Job 07, block 1: sequential_calibrated   SPSA , 17 circuits x 4096 shots = 69,632 requested shots
Job 08, block 2: sequential_calibrated   SPSA , 17 circuits x 4096 shots = 69,632 requested shots
Job 09, block 2: sequential_level3       SPSA , 17 circuits x 4096 shots = 69,632 requested shots
Job 10, block 2: sequential              SPSA , 17 circuits x 4096 shots = 69,632 requested shots
Job 11, block 2: baseline                SPSA , 17 circuits x 4096 shots = 69,632 requested shots
Job 12, block 2: sequential_calibrated   COBYLA, 17 circuits x 4096 shots = 69,632 requested shots
Job 13, block 2: sequential_level3       COBYLA, 17 circuits x 4096 shots = 69,632 requested shots
Job 14, block 2: sequential              COBYLA, 17 circuits x 4096 shots = 69,632 requested shots
Job 15, block 2: baseline                COBYLA, 17 circuits x 4096 shots = 69,632 requested shots
TOTAL: 16 job-mode Sampler jobs; 272 circuit executions; 1,114,112 requested shots.
Every group is executed in every complete repetition, including identity.
No Session, Estimator, mitigation, postselection, or additional optimization.
This is a new controlled benchmark. Historical repetitions are NOT counted as acceptance evidence or merged
with these repetitions.
/tmp/ipykernel_5433/4030956173.py:1374: DeprecationWarning: The SamplerV2 class is deprecated as of
qiskit-ibm-runtime 0.50.0 and will be removed no sooner than 3 months after the release date.
`qiskit_ibm_runtime.executor_sampler.Sampler` provides a new Sampler implementation that runs all the pre-
and post-processing in the client side. The top-level `qiskit_ibm_runtime.Sampler` import will be updated to
point to the client-side sampler in an upcoming release, with the legacy Sampler being available until the
deprecation is completed.
  hardware_sampler = SamplerV2(mode=backend, options=options)
Submitted job 00: db79rn2raffc73fa343g
COMPLETE baseline/COBYLA/rep1: 0.240876989142 +/- 0.025161 Ha (sampling SE)
Submitted job 01: db79vqmr6oqs73873qrg
COMPLETE sequential/COBYLA/rep1: -0.019427597987 +/- 0.022330 Ha (sampling SE)
Submitted job 02: db7a0hgr1c5c738k39jg
COMPLETE sequential_level3/COBYLA/rep1: -0.182088863266 +/- 0.022239 Ha (sampling SE)
Submitted job 03: db7a18mr6oqs73873ul0
WARNING:urllib3.connectionpool:Retrying (PostForcelistRetry(total=4, connect=3, read=None, redirect=None,
status=None)) after connection broken by 'ConnectionResetError(104, 'Connection reset by peer')':
/api/v1/jobs/db7a18mr6oqs73873ul0?exclude_params=true
COMPLETE sequential_calibrated/COBYLA/rep1: -0.304247907405 +/- 0.020953 Ha (sampling SE)
Submitted job 04: db7a1t2raffc73fa3i40
COMPLETE baseline/SPSA/rep1: 0.126823597392 +/- 0.024186 Ha (sampling SE)
Submitted job 05: db7a24mil99s73fnlodg
COMPLETE sequential/SPSA/rep1: -0.155971658972 +/- 0.022535 Ha (sampling SE)
Submitted job 06: db7a2hor1c5c738k3e2g
COMPLETE sequential_level3/SPSA/rep1: -0.187637447849 +/- 0.022346 Ha (sampling SE)
Submitted job 07: db7a2omil99s73fnlpog
COMPLETE sequential_calibrated/SPSA/rep1: -0.305996363706 +/- 0.021147 Ha (sampling SE)
Submitted job 08: db7a3hmil99s73fnlrhg
COMPLETE sequential_calibrated/SPSA/rep2: -0.327068459495 +/- 0.020869 Ha (sampling SE)
Submitted job 09: db7a4for1c5c738k3i80
COMPLETE sequential_level3/SPSA/rep2: -0.204873592773 +/- 0.022248 Ha (sampling SE)
Submitted job 10: db7a5c6r6oqs7387485g
COMPLETE sequential/SPSA/rep2: -0.144469023720 +/- 0.022729 Ha (sampling SE)
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
Submitted job 11: db7a6p0r1c5c738k3n4g
COMPLETE baseline/SPSA/rep2: 0.067220099757 +/- 0.023941 Ha (sampling SE)
Submitted job 12: db7a838r1c5c738k3q8g
COMPLETE sequential_calibrated/COBYLA/rep2: -0.323506821926 +/- 0.021163 Ha (sampling SE)
Submitted job 13: db7aa6uil99s73fnmalg
COMPLETE sequential_level3/COBYLA/rep2: -0.197840716670 +/- 0.022186 Ha (sampling SE)
Submitted job 14: db7aaliraffc73fa46ag
COMPLETE sequential/COBYLA/rep2: -0.101557934335 +/- 0.022608 Ha (sampling SE)
Submitted job 15: db7ab32raffc73fa478g
COMPLETE baseline/COBYLA/rep2: 0.024905399015 +/- 0.023592 Ha (sampling SE)

=== FINAL SCIENTIFIC COMPARISON ===
Sampling SE excludes systematic device error. Between-repetition SD and SE are separate descriptive estimates
from only two repeats.

COBYLA/baseline
  state: COBYLA
  variant: baseline
  mitigation: NONE
  E_exact: -0.5
  E0_fd: -0.46097392795978287
  classical_variational_energy: -0.46097392795976344
  exact_pauli_energy: -0.4609739279597633
  exact_measurement_energy: -0.4609739279597616
  local_mean_energy: -0.467112633841378
  local_mean_sampling_standard_error: 0.01398262958485123
  hardware_repetition_energies: [0.24087698914197958, 0.024905399015422547]
  hardware_repetition_sampling_standard_errors: [0.025160902439112124, 0.023592274570993492]
  hardware_mean_energy: 0.13289119407870106
  hardware_mean_sampling_standard_error: 0.017245770720553957
  between_repetition_standard_deviation: 0.1527149759221301
  between_repetition_standard_error_of_mean: 0.10798579506327852
  finite_representation_difference: 0.03902607204021713
  classical_variational_residual: 1.942890293094024e-14
  quantum_implementation_difference: 1.8318679906315083e-15
  hardware_difference: 0.5938651220384626
  job_ids: ['db79rn2raffc73fa343g', 'db7ab32raffc73fa478g']
  Residual cancellation: signed=+0.593865, sum|residual|=0.620373, positive=+0.607119, negative=-0.013254,
cancellation fraction=0.04272968635255581
  Dominant group residuals:
    g002 YXXYX residual=+0.242810, sampling SE=0.009133, TV=0.15830
    g008 YXYXX residual=+0.161357, sampling SE=0.005435, TV=0.14688
    g014 YYXXX residual=+0.091179, sampling SE=0.003760, TV=0.19843
    g016 ZZZZZ residual=+0.043132, sampling SE=0.005730, TV=0.09524
    g003 YXYYY residual=+0.024318, sampling SE=0.007501, TV=0.07949
    g010 YYYYX residual=+0.017585, sampling SE=0.003853, TV=0.11542
    g004 YYYXY residual=+0.012136, sampling SE=0.003858, TV=0.12366
    g009 YYXYY residual=-0.007969, sampling SE=0.003887, TV=0.11747
    g015 XXXXX residual=+0.005243, sampling SE=0.002156, TV=0.18876
    g013 XYYXX residual=+0.004015, sampling SE=0.002030, TV=0.15644
  Rep2-Rep1=-0.215972 +/- 0.034492 (sampling SE), sampling-only z=-6.26158127769716
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

COBYLA/sequential
  state: COBYLA
  variant: sequential
  mitigation: NONE
  E_exact: -0.5
  E0_fd: -0.46097392795978287
  classical_variational_energy: -0.46097392795976344
  exact_pauli_energy: -0.4609739279597633
  exact_measurement_energy: -0.4609739279597616
  local_mean_energy: -0.46463711239042715
  local_mean_sampling_standard_error: 0.013973924600154255
  hardware_repetition_energies: [-0.01942759798716793, -0.10155793433549554]
  hardware_repetition_sampling_standard_errors: [0.022330018029108552, 0.022608196942596834]
  hardware_mean_energy: -0.06049276616133173
  hardware_mean_sampling_standard_error: 0.015888362676622456
  between_repetition_standard_deviation: 0.05807491777303444
  between_repetition_standard_error_of_mean: 0.0410651681741638
  finite_representation_difference: 0.03902607204021713
  classical_variational_residual: 1.942890293094024e-14
  quantum_implementation_difference: 1.8318679906315083e-15
  hardware_difference: 0.40048116179842985
  job_ids: ['db79vqmr6oqs73873qrg', 'db7aaliraffc73fa46ag']
  Residual cancellation: signed=+0.400481, sum|residual|=0.405104, positive=+0.402792, negative=-0.002311, cancellation fraction=0.011410759043636753
  Dominant group residuals:
    g002 YXXYX residual=+0.110540, sampling SE=0.006846, TV=0.07188
    g008 YXYXX residual=+0.093540, sampling SE=0.004626, TV=0.07768
    g016 ZZZZZ residual=+0.078395, sampling SE=0.005797, TV=0.12602
    g003 YXYYY residual=+0.050759, sampling SE=0.007552, TV=0.10107
    g014 YYXXX residual=+0.036821, sampling SE=0.003545, TV=0.09031
    g010 YYYYX residual=+0.009754, sampling SE=0.003860, TV=0.05807
    g004 YYYXY residual=+0.009561, sampling SE=0.003859, TV=0.11648
    g006 XYXYX residual=+0.003451, sampling SE=0.002120, TV=0.06800
    g007 XYYYY residual=+0.003427, sampling SE=0.002124, TV=0.10638
    g013 XYYXX residual=+0.001897, sampling SE=0.002131, TV=0.08320
  Rep2-Rep1=-0.082130 +/- 0.031777 (sampling SE), sampling-only z=-2.584606671560032

COBYLA/sequential_level3
  state: COBYLA
  variant: sequential_level3
  mitigation: NONE
  E_exact: -0.5
  E0_fd: -0.46097392795978287
  classical_variational_energy: -0.46097392795976344
  exact_pauli_energy: -0.4609739279597633
  exact_measurement_energy: -0.4609739279597616
  local_mean_energy: -0.44779950459798146
  local_mean_sampling_standard_error: 0.014045702180928682
  hardware_repetition_energies: [-0.18208886326581783, -0.19784071666961528]
  hardware_repetition_sampling_standard_errors: [0.022239145401466563, 0.022185613531160935]

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
  hardware_mean_energy: -0.18996478996771654
  hardware_mean_sampling_standard_error: 0.01570653555006325
  between_repetition_standard_deviation: 0.011138242358081578
  between_repetition_standard_error_of_mean: 0.007875926701898725
  finite_representation_difference: 0.03902607204021713
  classical_variational_residual: 1.942890293094024e-14
  quantum_implementation_difference: 1.8318679906315083e-15
  hardware_difference: 0.27100913799204507
  job_ids: ['db7a0hgr1c5c738k39jg', 'db7aa6uil99s73fnmalg']
  Residual cancellation: signed=+0.271009, sum|residual|=0.286739, positive=+0.278874, negative=-0.007865,
cancellation fraction=0.05485695230843857
  Dominant group residuals:
    g002 YXXYX residual=+0.108795, sampling SE=0.006818, TV=0.07075
    g008 YXYXX residual=+0.083673, sampling SE=0.004441, TV=0.08384
    g016 ZZZZZ residual=+0.042975, sampling SE=0.005766, TV=0.08848
    g014 YYXXX residual=+0.023133, sampling SE=0.003491, TV=0.07264
    g010 YYYYX residual=+0.006119, sampling SE=0.003845, TV=0.05229
    g004 YYYXY residual=+0.004929, sampling SE=0.003843, TV=0.10375
    g007 XYYYY residual=+0.004653, sampling SE=0.002036, TV=0.08572
    g003 YXYYY residual=-0.004554, sampling SE=0.007463, TV=0.08079
    g012 XXYYX residual=-0.002328, sampling SE=0.002065, TV=0.07873
    g006 XYXYX residual=+0.001540, sampling SE=0.002036, TV=0.07932
  Rep2-Rep1=-0.015752 +/- 0.031413 (sampling SE), sampling-only z=-0.5014426432102024

COBYLA/sequential_calibrated
  state: COBYLA
  variant: sequential_calibrated
  mitigation: NONE
  E_exact: -0.5
  E0_fd: -0.46097392795978287
  classical_variational_energy: -0.46097392795976344
  exact_pauli_energy: -0.4609739279597633
  exact_measurement_energy: -0.4609739279597616
  local_mean_energy: -0.4759763218107679
  local_mean_sampling_standard_error: 0.01399736510145443
  hardware_repetition_energies: [-0.30424790740473073, -0.32350682192583424]
  hardware_repetition_sampling_standard_errors: [0.020952676506255142, 0.021163033585745412]
  hardware_mean_energy: -0.3138773646652825
  hardware_mean_sampling_standard_error: 0.014890337834709833
  between_repetition_standard_deviation: 0.013618109056164361
  between_repetition_standard_error_of_mean: 0.009629457260551753
  finite_representation_difference: 0.03902607204021713
  classical_variational_residual: 1.942890293094024e-14
  quantum_implementation_difference: 1.8318679906315083e-15
  hardware_difference: 0.14709656329447912
  job_ids: ['db7a18mr6oqs73873ul0', 'db7a838r1c5c738k3q8g']
  Residual cancellation: signed=+0.147097, sum|residual|=0.162618, positive=+0.154857, negative=-0.007761,
cancellation fraction=0.09544547645799828
  Dominant group residuals:
    g008 YXYXX residual=+0.053763, sampling SE=0.003995, TV=0.07704
    g002 YXXYX residual=+0.041518, sampling SE=0.005074, TV=0.06920
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    g014 YYXXX residual=+0.032189, sampling SE=0.003518, TV=0.06396
    g010 YYYYX residual=+0.010398, sampling SE=0.003844, TV=0.07392
    g003 YXYYY residual=+0.007431, sampling SE=0.007470, TV=0.07875
    g009 YYXYY residual=-0.006307, sampling SE=0.003875, TV=0.08876
    g006 XYXYX residual=+0.003202, sampling SE=0.002083, TV=0.07692
    g001 XYXXY residual=+0.002557, sampling SE=0.002079, TV=0.08223
    g016 ZZZZZ residual=+0.001938, sampling SE=0.005548, TV=0.05413
    g007 XYYYY residual=+0.000997, sampling SE=0.002088, TV=0.08586
  Rep2-Rep1=-0.019259 +/- 0.029781 (sampling SE), sampling-only z=-0.6466916578685807

SPSA/baseline
  state: SPSA
  variant: baseline
  mitigation: NONE
  E_exact: -0.5
  E0_fd: -0.46097392795978287
  classical_variational_energy: -0.46077921338321903
  exact_pauli_energy: -0.4607792133832186
  exact_measurement_energy: -0.4607792133832168
  local_mean_energy: -0.4758490880330404
  local_mean_sampling_standard_error: 0.014119419258267076
  hardware_repetition_energies: [0.12682359739186114, 0.06722009975718009]
  hardware_repetition_sampling_standard_errors: [0.02418637604789294, 0.023941358903345082]
  hardware_mean_energy: 0.09702184857452062
  hardware_mean_sampling_standard_error: 0.01701594437923495
  between_repetition_standard_deviation: 0.042146037359919314
  between_repetition_standard_error_of_mean: 0.02980174881734052
  finite_representation_difference: 0.03902607204021713
  classical_variational_residual: 0.00019471457656383695
  quantum_implementation_difference: 2.220446049250313e-15
  hardware_difference: 0.5578010619577374
  job_ids: ['db7a1t2raffc73fa3i40', 'db7a6p0r1c5c738k3n4g']
  Residual cancellation: signed=+0.557801, sum|residual|=0.567658, positive=+0.562730, negative=-0.004929,
cancellation fraction=0.01736503346409446
  Dominant group residuals:
    g002 YXXYX residual=+0.214224, sampling SE=0.008727, TV=0.13264
    g008 YXYXX residual=+0.150127, sampling SE=0.005368, TV=0.16415
    g016 ZZZZZ residual=+0.072107, sampling SE=0.005862, TV=0.12304
    g003 YXYYY residual=+0.051238, sampling SE=0.007510, TV=0.10165
    g014 YYXXX residual=+0.036141, sampling SE=0.003587, TV=0.14607
    g010 YYYYX residual=+0.020798, sampling SE=0.003858, TV=0.13763
    g015 XXXXX residual=+0.008966, sampling SE=0.002154, TV=0.13938
    g005 XXYXY residual=-0.004048, sampling SE=0.002164, TV=0.11047
    g009 YYXYY residual=+0.002195, sampling SE=0.003880, TV=0.09975
    g001 XYXXY residual=+0.002177, sampling SE=0.002035, TV=0.11536
  Rep2-Rep1=-0.059603 +/- 0.034032 (sampling SE), sampling-only z=-1.7514014005422152

SPSA/sequential
  state: SPSA
  variant: sequential
  mitigation: NONE
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
  E_exact: -0.5
  E0_fd: -0.46097392795978287
  classical_variational_energy: -0.46077921338321903
  exact_pauli_energy: -0.4607792133832186
  exact_measurement_energy: -0.4607792133832168
  local_mean_energy: -0.45428921744936984
  local_mean_sampling_standard_error: 0.01411186307200074
  hardware_repetition_energies: [-0.1559716589716642, -0.14446902372048254]
  hardware_repetition_sampling_standard_errors: [0.02253461156570452, 0.02272850227841528]
  hardware_mean_energy: -0.15022034134607337
  hardware_mean_sampling_standard_error: 0.01600307419089464
  between_repetition_standard_deviation: 0.008133591387625984
  between_repetition_standard_error_of_mean: 0.005751317625590834
  finite_representation_difference: 0.03902607204021713
  classical_variational_residual: 0.00019471457656383695
  quantum_implementation_difference: 2.220446049250313e-15
  hardware_difference: 0.31055887203714344
  job_ids: ['db7a24mil99s73fnlodg', 'db7a5c6r6oqs7387485g']
  Residual cancellation: signed=+0.310559, sum|residual|=0.335548, positive=+0.323053, negative=-0.012495,
cancellation fraction=0.0744723486276595
  Dominant group residuals:
    g002 YXXYX residual=+0.128087, sampling SE=0.007189, TV=0.07088
    g008 YXYXX residual=+0.083535, sampling SE=0.004521, TV=0.07925
    g016 ZZZZZ residual=+0.041714, sampling SE=0.005793, TV=0.08258
    g014 YYXXX residual=+0.027355, sampling SE=0.003576, TV=0.09003
    g003 YXYYY residual=+0.022345, sampling SE=0.007516, TV=0.12251
    g010 YYYYX residual=+0.013881, sampling SE=0.003852, TV=0.09302
    g009 YYXYY residual=-0.003932, sampling SE=0.003904, TV=0.11854
    g015 XXXXX residual=-0.003725, sampling SE=0.002059, TV=0.07435
    g001 XYXXY residual=-0.001998, sampling SE=0.002154, TV=0.13276
    g007 XYYYY residual=+0.001849, sampling SE=0.002149, TV=0.11712
  Rep2-Rep1=+0.011503 +/- 0.032006 (sampling SE), sampling-only z=0.3593882998344902

SPSA/sequential_level3
  state: SPSA
  variant: sequential_level3
  mitigation: NONE
  E_exact: -0.5
  E0_fd: -0.46097392795978287
  classical_variational_energy: -0.46077921338321903
  exact_pauli_energy: -0.4607792133832186
  exact_measurement_energy: -0.4607792133832168
  local_mean_energy: -0.445902128673735
  local_mean_sampling_standard_error: 0.014040620517251438
  hardware_repetition_energies: [-0.18763744784915481, -0.2048735927728211]
  hardware_repetition_sampling_standard_errors: [0.0223458871616051, 0.022248396595218267]
  hardware_mean_energy: -0.19625552031098797
  hardware_mean_sampling_standard_error: 0.015766497899797748
  between_repetition_standard_deviation: 0.012187794957038521
  between_repetition_standard_error_of_mean: 0.008618072461833144
  finite_representation_difference: 0.03902607204021713
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
  classical_variational_residual: 0.00019471457656383695
  quantum_implementation_difference: 2.220446049250313e-15
  hardware_difference: 0.26452369307222884
  job_ids: ['db7a2hor1c5c738k3e2g', 'db7a4for1c5c738k3i80']
  Residual cancellation: signed=+0.264524, sum|residual|=0.292998, positive=+0.278761, negative=-0.014237,
cancellation fraction=0.09718210657528115
  Dominant group residuals:
    g002 YXXYX residual=+0.109497, sampling SE=0.006809, TV=0.06410
    g008 YXYXX residual=+0.089724, sampling SE=0.004580, TV=0.09070
    g016 ZZZZZ residual=+0.038695, sampling SE=0.005690, TV=0.07518
    g014 YYXXX residual=+0.020209, sampling SE=0.003483, TV=0.07541
    g010 YYYYX residual=+0.008460, sampling SE=0.003857, TV=0.08672
    g006 XYXYX residual=+0.005206, sampling SE=0.002136, TV=0.07421
    g012 XXYYX residual=+0.003847, sampling SE=0.002067, TV=0.06893
    g001 XYXXY residual=-0.003473, sampling SE=0.002129, TV=0.11348
    g003 YXYYY residual=-0.003431, sampling SE=0.007455, TV=0.09620
    g005 XXYXY residual=-0.002864, sampling SE=0.002044, TV=0.10211
  Rep2-Rep1=-0.017236 +/- 0.031533 (sampling SE), sampling-only z=-0.5466066412848536

SPSA/sequential_calibrated
  state: SPSA
  variant: sequential_calibrated
  mitigation: NONE
  E_exact: -0.5
  E0_fd: -0.46097392795978287
  classical_variational_energy: -0.46077921338321903
  exact_pauli_energy: -0.4607792133832186
  exact_measurement_energy: -0.4607792133832168
  local_mean_energy: -0.4684152957371302
  local_mean_sampling_standard_error: 0.014148095599615076
  hardware_repetition_energies: [-0.30599636370632544, -0.3270684594948832]
  hardware_repetition_sampling_standard_errors: [0.021146664908673663, 0.020869370707030616]
  hardware_mean_energy: -0.3165324116006043
  hardware_mean_sampling_standard_error: 0.01485523536053197
  between_repetition_standard_deviation: 0.014900221825901701
  between_repetition_standard_error_of_mean: 0.010536047894278894
  finite_representation_difference: 0.03902607204021713
  classical_variational_residual: 0.00019471457656383695
  quantum_implementation_difference: 2.220446049250313e-15
  hardware_difference: 0.1442468017826125
  job_ids: ['db7a2omil99s73fnlpog', 'db7a3hmil99s73fnlrhg']
  Residual cancellation: signed=+0.144247, sum|residual|=0.168676, positive=+0.156462, negative=-0.012215,
cancellation fraction=0.14483061525777818
  Dominant group residuals:
    g008 YXYXX residual=+0.054767, sampling SE=0.004027, TV=0.06048
    g002 YXXYX residual=+0.036445, sampling SE=0.004913, TV=0.06572
    g014 YYXXX residual=+0.022827, sampling SE=0.003520, TV=0.06114
    g010 YYYYX residual=+0.013590, sampling SE=0.003857, TV=0.06560
    g003 YXYYY residual=+0.006352, sampling SE=0.007480, TV=0.08185
    g004 YYYXY residual=-0.005782, sampling SE=0.003880, TV=0.07687
    g013 XYYXX residual=+0.005218, sampling SE=0.002086, TV=0.05950
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

g016 ZZZZZ residual=+0.004888, sampling SE=0.005539, TV=0.04522
g015 XXXXX residual=+0.004770, sampling SE=0.002105, TV=0.05930
g006 XYXYX residual=-0.003518, sampling SE=0.002114, TV=0.07305
Rep2-Rep1=-0.021072 +/- 0.029710 (sampling SE), sampling-only z=-0.7092481296036225
CALIBRATED SPSA REPETITION DISCREPANCY:
{'group': 16, 'basis': 'ZZZZZ', 'repetition_2_minus_1': 0.018517480932633168, 'sampling_variance_of_difference': 0.00012273926813785571, 'sampling_standard_error_of_difference': 0.011078775570335186}
{'group': 8, 'basis': 'YXYXX', 'repetition_2_minus_1': -0.018070793151855447, 'sampling_variance_of_difference': 6.485469229226088e-05, 'sampling_standard_error_of_difference': 0.008053241104813693}
{'group': 10, 'basis': 'YYYYX', 'repetition_2_minus_1': 0.011424064636230465, 'sampling_variance_of_difference': 5.9515686781208945e-05, 'sampling_standard_error_of_difference': 0.007714641066258945}
{'group': 14, 'basis': 'YYXXX', 'repetition_2_minus_1': -0.010468597412109343, 'sampling_variance_of_difference': 4.9573364920810544e-05, 'sampling_standard_error_of_difference': 0.007040835527180744}
{'group': 9, 'basis': 'YYXYY', 'repetition_2_minus_1': -0.010053176879882804, 'sampling_variance_of_difference': 6.041279584058729e-05, 'sampling_standard_error_of_difference': 0.007772568759160435}
{'group': 4, 'basis': 'YYYXY', 'repetition_2_minus_1': -0.008682289123535171, 'sampling_variance_of_difference': 6.0208395901810904e-05, 'sampling_standard_error_of_difference': 0.007759406929773106 5}
{'group': 2, 'basis': 'YXXYX', 'repetition_2_minus_1': -0.007976074218749796, 'sampling_variance_of_difference': 9.656381677973699e-05, 'sampling_standard_error_of_difference': 0.009826689003918716}
{'group': 15, 'basis': 'XXXXX', 'repetition_2_minus_1': -0.006771354675292976, 'sampling_variance_of_difference': 1.7728895046169335e-05, 'sampling_standard_error_of_difference': 0.004210569444406461}
{'group': 11, 'basis': 'XXXYY', 'repetition_2_minus_1': 0.005857429504394524, 'sampling_variance_of_difference': 1.7982008957229217e-05, 'sampling_standard_error_of_difference': 0.00424051989232797}
{'group': 5, 'basis': 'XXYXY', 'repetition_2_minus_1': 0.0050265884399414065, 'sampling_variance_of_difference': 1.788865789420234e-05, 'sampling_standard_error_of_difference': 0.004229498539330915}
Cause remains unresolved; no causal label is assigned.

=== ONE-FACTOR HARDWARE COMPARISONS ===
{'state': 'COBYLA', 'variant': 'sequential', 'comparator': 'baseline', 'changed_factor': 'synthesis', 'mean_energy_difference': -0.1933839602400328, 'sampling_standard_error_of_energy_difference': 0.023449022928254508, 'absolute_energy_residual_reduction': 0.1933839602400328, 'interpretation': 'Positive absolute-residual reduction is an observed point estimate of improvement, not proof of drift-robust superiority. No adjustment or mitigation has been applied.'}
{'state': 'SPSA', 'variant': 'sequential', 'comparator': 'baseline', 'changed_factor': 'synthesis', 'mean_energy_difference': -0.247242189920594, 'sampling_standard_error_of_energy_difference': 0.023358954314705437, 'absolute_energy_residual_reduction': 0.2472421899205939 4, 'interpretation': 'Positive absolute-residual reduction is an observed point estimate of improvement, not proof of drift-robust superiority. No adjustment or mitigation has been applied.'}
{'state': 'COBYLA', 'variant': 'sequential_level3', 'comparator': 'sequential', 'changed_factor': 'compiler optimization level', 'mean_energy_difference': -0.1294720238063848, 'sampling_standard_error_of_energy_difference': 0.022341336744458467, 'absolute_energy_residual_reduction':

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

0.12947202380638478, 'interpretation': 'Positive absolute-residual reduction is an observed point estimate of improvement, not proof of drift-robust superiority. No adjustment or mitigation has been applied.'}
{'state': 'SPSA', 'variant': 'sequential_level3', 'comparator': 'sequential', 'changed_factor': 'compiler optimization level', 'mean_energy_difference': -0.0460351789649146, 'sampling_standard_error_of_energy_difference': 0.02246510270583255, 'absolute_energy_residual_reduction': 0.0460351789649146, 'interpretation': 'Positive absolute-residual reduction is an observed point estimate of improvement, not proof of drift-robust superiority. No adjustment or mitigation has been applied.'}
{'state': 'COBYLA', 'variant': 'sequential_calibrated', 'comparator': 'sequential', 'changed_factor': 'placement', 'mean_energy_difference': -0.2533845985039508, 'sampling_standard_error_of_energy_difference': 0.021775266459349713, 'absolute_energy_residual_reduction': 0.25338459850395073, 'interpretation': 'Positive absolute-residual reduction is an observed point estimate of improvement, not proof of drift-robust superiority. No adjustment or mitigation has been applied.'}
{'state': 'SPSA', 'variant': 'sequential_calibrated', 'comparator': 'sequential', 'changed_factor': 'placement', 'mean_energy_difference': -0.16631207025453093, 'sampling_standard_error_of_energy_difference': 0.021835210124385742, 'absolute_energy_residual_reduction': 0.16631207025453093, 'interpretation': 'Positive absolute-residual reduction is an observed point estimate of improvement, not proof of drift-robust superiority. No adjustment or mitigation has been applied.'}

Job IDs: ['db79rn2raffc73fa343g', 'db79vqmr6oqs73873qrg', 'db7a0hgr1c5c738k39jg', 'db7a18mr6oqs73873ul0', 'db7a1t2raffc73fa3i40', 'db7a24mil99s73fnlodg', 'db7a2hor1c5c738k3e2g', 'db7a2omil99s73fnlpog', 'db7a3hmil99s73fnlrhg', 'db7a4for1c5c738k3i80', 'db7a5c6r6oqs7387485g', 'db7a6p0r1c5c738k3n4g', 'db7a838r1c5c738k3q8g', 'db7aa6uil99s73fnmalg', 'db7aaliraffc73fa46ag', 'db7ab32raffc73fa478g']
Evidence directory: /content/hydrogen_part_ii_20261013T210806_159257Z_a657370a
Historical calibrated SPSA repetitions were not supplied numerically. They have not been reinterpreted as new evidence; the analysis above applies only to the newly authorized executions.

Third-run conclusion

Run 3 provides the strongest positive hardware comparison in this study. Quantum Flow implemented the requested structure-aware sequential preparation of the unchanged states, reducing two-qubit gates per full measurement circuit from 46 to 10 (78.3%) and mean depth by about 73.4% relative to the same-run baseline. The reported maximum compiled probability discrepancy remained below 1.6×10^{-14}. On the **ibm_fez** backend, sequential synthesis with calibration-informed placement gave raw means of -0.313877 ± 0.014890 Ha and -0.316532 ± 0.014855 Ha for the COBYLA- and SPSA-derived states. These sampling-only estimates reduced the observed absolute discrepancies by 75.2% and 74.1%, respectively, compared with that run's generic-preparation baseline, but still differed from their targets by about 0.147 Ha and 0.144 Ha. This is evidence of useful problem-specific circuit optimization and improved point estimates within this experiment, not of global optimality or robustness across devices and calibrations. Only two repetitions were recorded, and the baseline itself varied between them.

Why the sequential construction is scientifically meaningful

The low bond dimension follows directly from the preserved trial-state structure. Let $q = \exp(-\alpha h)$, ignore the overall normalization, and write the computational-basis index as $k = a + Lb$, where $L = 2^m$ at a contiguous binary cut. Then $\psi_k \propto (k + 1)q^k$ can be expressed as $(a + 1)q^a q^{Lb} + q^a(Lb)q^{Lb}$. This is a sum of two products of functions of a and b, so the coefficient matrix has rank at most two across each such cut. The recorded numerical Schmidt-rank checks and state reconstruction test the corresponding finite vectors.

Sequential matrix-product-state generation is an established construction [26]. Quantum Flow applied that construction to the supplied benchmark under explicit revision guidance; the study does not claim to invent MPS synthesis or prove that ten native entanglers is the minimum possible. Exploiting low-rank structure can remove

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

unnecessary generic state-preparation work while leaving the physical target unchanged. This makes the compiled-resource reduction more informative than merely replacing the problem with an easier ansatz.

Prompt sent to Quantum Flow to refine the third code version

Extend the Part II to improve the remaining hardware-energy discrepancy. Use the successful 10-two-qubit-gate sequential_calibrated implementation as the new baseline, retaining the original 46-gate results as historical context.
Preserve Part I, the complete Hamiltonian, five-qubit encoding, both fixed optimized states, all measurement groups, and existing validation and evidence safeguards. Keep the current SDK environment and counts-based Sampler job mode on ibm_fez; no new VQE loop, backend substitution, or reference-energy adjustment.
First optimize locally. Test compiler level 3 on the calibrated layout. Separately investigate the freedom in the sequential isometry's unused unitary columns, real-valued structure, and equivalent decompositions to reduce native entangling gates, single-qubit gates, or execution duration further. Change one factor per comparison, preserve exact state equivalence within existing tolerances, and report computed improvements or failures—not assumed benefits.
Investigate measurement error separately. Add an optional readout-calibration experiment on the actual measured physical qubits, with independently validated correction and clearly separated raw versus mitigated results. Include calibration uncertainty, shared-calibration covariance, and conditioning checks. Never fit corrections to the hydrogen reference, clip results toward it, or assume readout error explains the remaining discrepancy. Preserve analysis of dominant groups 2, 8, and 14 and residual cancellation.
Keep the hardware experiment focused. Select at most one new circuit candidate using predefined local criteria, then compare it with the current best baseline in at least four balanced, interleaved execution blocks. Count all calibration and diagnostic overhead; propose a total shot budget no larger than the preceding 1,114,112 shots unless explicitly authorized. Keep sampling, calibration, and between-block uncertainties distinct.
Return one complete runnable cell and a brief rationale. Preserve negative controls, validation of the actual submitted circuits, circuit reuse, hashes, raw counts, and job associations. Print local acceptance results, circuit comparisons, and the complete workload before explicit QPU confirmation. Report improvement only when supported by the new measurements.

Note: The prompt also included the results of the third run.

Fourth code version (verbatim)

```
# Initial code produced by Stellora.AI Quantum Flow

def _hydrogen_marrakesh_part_ii(
    H, r, h, n_interior, R_max, E_exact, E0_fd,
    alpha_cobyla, alpha_spsa, normalized_trial_state,
):
    import os
    import json
    import math
    import pickle
    import marshal
    import hashlib
    import inspect
    import getpass
    import uuid
    from pathlib import Path
    from datetime import datetime, timezone
    from collections import Counter
    from importlib.metadata import version
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
import numpy as np
import scipy
import qiskit
from scipy.stats import chi2, t as student_t
from qiskit import (
    QuantumCircuit, QuantumRegister, ClassicalRegister, qpy,
)
from qiskit.circuit.library import UnitaryGate, CXGate
from qiskit.quantum_info import Statevector, Operator, SparsePauliOp
from qiskit.primitives import StatevectorSampler
from qiskit.synthesis import TwoQubitBasisDecomposer
from qiskit.transpiler.preset_passmanagers import (
    generate_preset_pass_manager,
)
from qiskit_ibm_runtime import QiskitRuntimeService, SamplerV2
from qiskit_ibm_runtime.accounts.exceptions import AccountNotFoundError

# ---------------------------- Configuration ----------------------------
ENABLE_READOUT = True
BACKEND_NAME = "ibm_marrakesh"
CHANNEL = "ibm_quantum_platform"
REGISTER = "hydrogen"

NQ, DIM = 5, 32
BLOCKS = 4
SCIENCE_SHOTS = 2048
CALIBRATION_SHOTS = 1024
VALIDATION_SHOTS = 512
LOCAL_SHOTS = 4096
LOCAL_REPETITIONS = 2
MAX_TOTAL_SHOTS = 1_114_112

TRANSPILE_SEED = 731
LOCAL_SEED = 20261014
MATRIX_TOL = 1e-12
ENERGY_TOL = 1e-10
PROB_TOL = 1e-10
SCHMIDT_TOL = 1e-12
MAX_ACTIVE_QUBITS = 20

# Unchanged conditioning safeguards.
MAX_ASSIGNMENT_CONDITION = 10.0
MIN_ASSIGNMENT_SINGULAR_VALUE = 0.10
READOUT_VALIDATION_FAMILY_ALPHA = 0.01

# Explicitly request non-approximate synthesis at every preset compilation.
APPROXIMATION_DEGREE = 1.0

def require(condition, message):
    if not bool(condition):
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        raise RuntimeError(message)

def real(value, label):
    value = complex(value)
    require(
        np.isfinite(value.real)
        and np.isfinite(value.imag)
        and abs(value.imag) <= MATRIX_TOL,
        f"{label}: nonfinite or unexpectedly complex.",
    )
    return float(value.real)

def utc():
    return datetime.now(timezone.utc).isoformat()

def sha(data):
    return hashlib.sha256(data).hexdigest()

def jsonable(value):
    if isinstance(value, np.ndarray):
        return jsonable(value.tolist())
    if isinstance(value, np.generic):
        return jsonable(value.item())
    if isinstance(value, complex):
        return {"real": value.real, "imag": value.imag}
    if isinstance(value, datetime):
        return value.isoformat()
    if isinstance(value, dict):
        return {str(k): jsonable(v) for k, v in value.items()}
    if isinstance(value, (tuple, list)):
        return [jsonable(v) for v in value]
    if isinstance(value, float) and not math.isfinite(value):
        return None
    if value is None or isinstance(value, (str, int, float, bool)):
        return value
    return repr(value)

def nonnegative_variance(value, label):
    value = float(value)
    require(
        np.isfinite(value) and value >= -1e-12,
        f"{label}: invalid covariance quadratic form.",
    )
    # Only roundoff in covariance quadratic forms is cleaned up.
    # Energies and corrected probabilities are never clipped.
    return max(0.0, value)

# ---------------------------- Part I checks ----------------------------
matrix = np.array(H, dtype=complex, copy=True)
grid = np.array(r, dtype=float, copy=True)
require(matrix.shape == (DIM, DIM), "H must have shape (32,32).")
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
require(grid.shape == (DIM,), "r must contain exactly 32 elements.")
require(np.all(np.isfinite(matrix)), "H contains nonfinite values.")
require(np.all(np.isfinite(grid)), "r contains nonfinite values.")
hermitian_error = float(np.max(np.abs(matrix - matrix.conj().T)))
require(hermitian_error <= MATRIX_TOL, "H fails Hermiticity validation.")
require(int(n_interior) == DIM, "n_interior must be 32.")
require(np.isfinite(h) and float(h) > 0, "Invalid h.")
require(np.isfinite(R_max) and float(R_max) > 0, "Invalid R_max.")
require(callable(normalized_trial_state), "Invalid trial-state function.")

analytical_reference = real(E_exact, "E_exact")
finite_reference = real(E0_fd, "E0_fd")
parameters = {"COBYLA": alpha_cobyla, "SPSA": alpha_spsa}
states, classical = {}, {}

for name, alpha in parameters.items():
    require(
        np.ndim(alpha) == 0 and np.isfinite(alpha) and float(alpha) > 0,
        f"{name}: alpha must be finite and positive.",
    )
    vector = np.array(
        normalized_trial_state(alpha, grid.copy()),
        dtype=complex, copy=True,
    )
    require(vector.shape == (DIM,), f"{name}: incorrect state dimension.")
    require(np.all(np.isfinite(vector)), f"{name}: nonfinite amplitudes.")
    require(
        abs(real(np.vdot(vector, vector), "State norm") - 1) <= MATRIX_TOL,
        f"{name}: state is not normalized.",
    )
    states[name] = vector
    classical[name] = real(np.vdot(vector, matrix @ vector), name)
names = list(states)

versions = {
    "numpy": np.__version__,
    "scipy": scipy.__version__,
    "qiskit": qiskit.__version__,
    "qiskit-ibm-runtime": version("qiskit-ibm-runtime"),
}
require(versions["qiskit"] == "2.5.2", "Requires Qiskit 2.5.2.")
require(
    versions["qiskit-ibm-runtime"] == "0.50.0",
    "Requires qiskit-ibm-runtime 0.50.0.",
)

# ---------------------------- Evidence saving ----------------------------
stamp = datetime.now(timezone.utc).strftime("%Y%m%dT%H%M%S_%fZ")
evidence = Path.cwd() / (
    f"hydrogen_marrakesh_{stamp}_{uuid.uuid4().hex[:8]}"
)
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
evidence.mkdir(mode=0o700, exist_ok=False)

def save_json(filename, content):
    temporary = evidence / f".{filename}.{uuid.uuid4().hex}.tmp"
    with temporary.open("x", encoding="utf-8") as handle:
        os.chmod(temporary, 0o600)
        json.dump(jsonable(content), handle, indent=2, allow_nan=False)
        handle.flush()
        os.fsync(handle.fileno())
    os.replace(temporary, evidence / filename)

def save_pickle(filename, content):
    path = evidence / filename
    with path.open("xb") as handle:
        os.chmod(path, 0o600)
        pickle.dump(content, handle, protocol=pickle.HIGHEST_PROTOCOL)
        handle.flush()
        os.fsync(handle.fileno())

def save_qpy(filename, circuits):
    path = evidence / filename
    with path.open("xb") as handle:
        os.chmod(path, 0o600)
        qpy.dump(circuits, handle)
    return sha(path.read_bytes())

def circuit_hash(circuit):
    payload = {
        "num_qubits": circuit.num_qubits,
        "num_clbits": circuit.num_clbits,
        "global_phase": circuit.global_phase,
        "classical_registers": [(c.name, len(c)) for c in circuit.cregs],
        "instructions": [
            {
                "operation": item.operation.name,
                "qubits": [circuit.find_bit(q).index for q in item.qubits],
                "clbits": [circuit.find_bit(c).index for c in item.clbits],
                "parameters": list(item.operation.params),
            }
            for item in circuit.data
        ],
    }
    return sha(json.dumps(
        jsonable(payload), sort_keys=True,
        separators=(",", ":"), allow_nan=False,
    ).encode())

frame = inspect.currentframe()
own_code = frame.f_code
del frame
source_hashes = {
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    "part_ii_code_object_sha256": sha(marshal.dumps(own_code)),
    "trial_code_object_sha256": (
        sha(marshal.dumps(normalized_trial_state.__code__))
        if hasattr(normalized_trial_state, "__code__") else None
    ),
    "part_ii_source_sha256": None,
    "trial_source_sha256": None,
    "qualification": (
        "No prior notebook history or previous run is read. Source hashes "
        "are null if inspect cannot retrieve source. Code-object hashes "
        "are Python-version-specific."
    ),
}
for label, obj in (
    ("part_ii", own_code),
    ("trial", normalized_trial_state),
):
    try:
        source = inspect.getsource(obj)
    except (OSError, TypeError):
        source = None
    if source is not None:
        source_hashes[f"{label}_source_sha256"] = sha(source.encode())
        path = evidence / f"{label}_source.py"
        path.write_text(source, encoding="utf-8")
        os.chmod(path, 0o600)
save_json("source_hashes.json", source_hashes)

# Static user-supplied historical context, never new validation evidence.
historical = {
    "provenance": "User-pasted results from the previous backend.",
    "previous_generic_preparation_two_qubit_gates": 46,
    "previous_sequential_preparation_two_qubit_gates": 10,
    "previous_sequential_preparation_depth": 41,
    "previous_layout": [142, 143, 144, 145, 146],
    "previous_compiler_level": 2,
    "previous_seed": 731,
    "COBYLA": {
        "raw_mean": -0.3138773646652825,
        "sampling_se_of_mean": 0.014890337834709833,
        "residual": 0.14709656329447912,
        "groups_2_8_14_residuals": [0.041518, 0.053763, 0.032189],
    },
    "SPSA": {
        "raw_mean": -0.3165324116006043,
        "sampling_se_of_mean": 0.01485523536053197,
        "residual": 0.1442468017826125,
        "repetitions": [-0.30599636370632544, -0.3270684594948832],
        "rep2_minus_rep1": -0.02107209578855776,
        "difference_sampling_se": 0.02971047072106394,
        "sampling_only_z": -0.7092481296036225,
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        "groups_2_8_14_residuals": [0.036445, 0.054767, 0.022827],
    },
    "qualification": (
        "No historical circuit artifacts or counts are loaded. Previous "
        "10/46 gate counts are context, not acceptance requirements on a "
        "different backend with different native gates and calibration."
    ),
}
save_json("historical_context.json", historical)

manifest = {
    "created_utc": utc(),
    "versions": versions,
    "backend": BACKEND_NAME,
    "channel": CHANNEL,
    "logical_qubits": NQ,
    "blocks": BLOCKS,
    "science_shots_per_circuit": SCIENCE_SHOTS,
    "readout_enabled": ENABLE_READOUT,
    "calibration_shots_per_prepared_state": CALIBRATION_SHOTS,
    "heldout_shots_per_prepared_state": VALIDATION_SHOTS,
    "budget_cap": MAX_TOTAL_SHOTS,
    "transpiler_seed": TRANSPILE_SEED,
    "approximation_degree": APPROXIMATION_DEGREE,
    "local_seed": LOCAL_SEED,
    "hamiltonian_sha256": sha(matrix.tobytes()),
    "state_sha256": {k: sha(v.tobytes()) for k, v in states.items()},
    "parameters": {k: float(v) for k, v in parameters.items()},
    "references": {
        "E_exact": analytical_reference, "E0_fd": finite_reference,
    },
    "grid": {
        "h": float(h), "n_interior": int(n_interior), "R_max": float(R_max),
    },
    "source_hashes": source_hashes,
    "tolerances": {
        "matrix": MATRIX_TOL, "energy": ENERGY_TOL,
        "probability": PROB_TOL, "Schmidt_tail": SCHMIDT_TOL,
    },
    "conditioning_limits": {
        "maximum_condition_number": MAX_ASSIGNMENT_CONDITION,
        "minimum_singular_value": MIN_ASSIGNMENT_SINGULAR_VALUE,
    },
    "candidate_failure_policy": (
        "A failed optional local candidate is explicitly rejected in full, "
        "not repaired, recompiled, substituted, or submitted. Baseline "
        "failure stops the experiment. All submitted circuits must pass."
    ),
    "raw_processing": "Unadjusted counts; no postselection.",
    "readout_processing": (
        "Optional full assignment inversion; no clipping, pseudoinverse, "
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        "projection, regularization, or hydrogen-reference fitting."
    ),
    "uncertainty_scope": (
        "Science sampling, first-order calibration with shared covariance, "
        "and between-block spread are distinct. Systematic hardware errors "
        "are not fully quantified."
    ),
    "jobs": [],
    "status": "local_validation",
}
save_json("manifest.json", manifest)
np.savez_compressed(
    evidence / "scientific_inputs.npz",
    H=matrix, r=grid, COBYLA=states["COBYLA"], SPSA=states["SPSA"],
)
os.chmod(evidence / "scientific_inputs.npz", 0o600)

# ---------------------- Hamiltonian and measurement ----------------------
pauli = SparsePauliOp.from_operator(Operator(matrix), atol=0, rtol=0)
original = pauli.to_list()
labels = [label for label, coefficient in original]
require(len(labels) == len(set(labels)), "Duplicate original Pauli terms.")
for label, coefficient in original:
    real(coefficient, label)
reconstruction_error = float(np.max(np.abs(pauli.to_matrix() - matrix)))
require(reconstruction_error <= MATRIX_TOL, "Pauli reconstruction failed.")

operators = pauli.group_commuting(qubit_wise=True)
grouped = [term for operator in operators for term in operator.to_list()]
require(
    Counter(label for label, coefficient in grouped) == Counter(labels),
    "Missing or duplicated grouped terms.",
)
original_coefficients = dict(original)
require(
    all(c == original_coefficients[label] for label, c in grouped),
    "Grouping changed a coefficient.",
)
require(
    labels.count("IIIII") == 1
    and sum(label == "IIIII" for label, c in grouped) == 1,
    "Identity must occur exactly once.",
)
groups = []
for gi, operator in enumerate(operators):
    axes = ["I"] * NQ
    weights = np.zeros(DIM)
    for label, coefficient in operator.to_list():
        mask = 0
        for q in range(NQ):
            axis = label[NQ - 1 - q]
```

```
            if axis != "I":
                require(axes[q] in ("I", axis), "Non-QWC measurement group.")
                axes[q] = axis
                mask |= 1 << q
        weights += real(coefficient, label) * np.array([
            -1.0 if (k & mask).bit_count() % 2 else 1.0
            for k in range(DIM)
        ])
    axes = ["Z" if axis == "I" else axis for axis in axes]
    groups.append({
        "index": gi, "axes": axes, "basis": "".join(reversed(axes)),
        "weights": weights, "terms": operator.to_list(),
    })

documented_bases = [
    "YXXXY", "XYXXY", "YXXYX", "YXYYY", "YYYXY", "XXYXY",
    "XYXYX", "XYYYY", "YXYXX", "YYXYY", "YYYYX", "XXXYY",
    "XXYYX", "XYYXX", "YYXXX", "XXXXX", "ZZZZZ",
]
require(len(original) == 63, "Expected the complete 63-term Hamiltonian.")
require(
    [g["basis"] for g in groups] == documented_bases,
    "Group ordering differs from the authoritative benchmark.",
)

def check_state(actual, intended, label):
    overlap = np.vdot(intended, actual)
    require(abs(overlap) > 0, f"{label}: zero overlap.")
    error = float(np.linalg.norm(
        actual - overlap / abs(overlap) * intended
    ))
    require(error <= MATRIX_TOL, f"{label}: state error {error:.3e}.")
    return error

def check_probabilities(actual, intended, label):
    actual = np.asarray(actual, dtype=float)
    require(
        actual.shape == (DIM,) and np.all(np.isfinite(actual))
        and np.min(actual) >= -MATRIX_TOL,
        f"{label}: invalid probabilities.",
    )
    require(
        abs(float(actual.sum()) - 1) <= PROB_TOL,
        f"{label}: invalid probability normalization.",
    )
    error = float(np.max(np.abs(actual - intended)))
    require(error <= PROB_TOL, f"{label}: probability error {error:.3e}.")
    return error

specifications = [
    {"name": "baseline", "level": 2, "mode": "complex", "theta": 0.0},
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    {"name": "compiler_level3", "level": 3, "mode": "complex", "theta": 0.0},
    {"name": "real_svd", "level": 2, "mode": "real", "theta": 0.0},
    {"name": "unused_columns_minus_pi4", "level": 2,
     "mode": "rotate", "theta": -np.pi / 4},
    {"name": "unused_columns_plus_pi4", "level": 2,
     "mode": "rotate", "theta": np.pi / 4},
    {"name": "unused_columns_plus_pi2", "level": 2,
     "mode": "rotate", "theta": np.pi / 2},
    {"name": "unused_columns_phase", "level": 2,
     "mode": "phase", "theta": np.pi / 2},
    {"name": "equivalent_explicit_cx", "level": 2,
     "mode": "explicit_cx", "theta": 0.0},
]
cx_decomposer = TwoQubitBasisDecomposer(CXGate())

# Exactly five wires; bond moves along q0,q1,...,q4.
# order="F" preserves amplitude index k = sum_j bit[j]*2**j.
def synthesize(vector, specification, name):
    if specification["mode"] == "real":
        require(np.max(np.abs(vector.imag)) <= MATRIX_TOL, "State is not real.")
        residual = vector.real.copy()
    else:
        residual = vector.copy()

    cuts = []
    for cut in range(1, NQ):
        s = np.linalg.svd(
            vector.reshape(1 << cut, 1 << (NQ - cut), order="F"),
            compute_uv=False,
        )
        tail = float(np.linalg.norm(s[2:]))
        require(tail <= SCHMIDT_TOL, "Bond-two Schmidt condition failed.")
        cuts.append({"cut": cut, "singular_values": s, "rank_two_tail": tail})

    tensors = [None] * NQ
    right_rank = 1
    tails = []
    for q in range(NQ - 1, 0, -1):
        unfolding = residual.reshape(1 << q, 2 * right_rank, order="F")
        u, s, vh = np.linalg.svd(unfolding, full_matrices=False)
        tail = float(np.linalg.norm(s[2:]))
        require(tail <= SCHMIDT_TOL, "Sequential rank-two factorization failed.")
        tails.append(tail)
        tensors[q] = vh[:2].reshape(2, 2, right_rank, order="F")
        residual = u[:, :2] * s[:2]
        right_rank = 2

    retained_norm = float(np.linalg.norm(residual))
    require(abs(retained_norm - 1) <= MATRIX_TOL, "Invalid retained MPS norm.")
    tensors[0] = (residual / retained_norm).reshape(1, 2, 2, order="F")
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    circuit = QuantumCircuit(NQ)
    independent = np.zeros(DIM, dtype=complex)
    independent[0] = 1
    gate_records = []

    for q, tensor in enumerate(tensors):
        left_rank, _, next_rank = tensor.shape
        isometry = tensor.transpose(1, 2, 0).reshape(
            2 * next_rank, left_rank, order="F"
        )
        require(
            np.max(np.abs(
                isometry.conj().T @ isometry - np.eye(left_rank)
            )) <= MATRIX_TOL,
            "Invalid sequential isometry.",
        )
        _, _, vh = np.linalg.svd(isometry.conj().T, full_matrices=True)
        complement = vh.conj().T[:, left_rank:]
        unused = complement.shape[1]
        if unused >= 2:
            theta = specification["theta"]
            if specification["mode"] == "rotate":
                rotation = np.eye(unused, dtype=complex)
                rotation[:2, :2] = [
                    [np.cos(theta), -np.sin(theta)],
                    [np.sin(theta), np.cos(theta)],
                ]
                complement = complement @ rotation
            elif specification["mode"] == "phase":
                phase = np.eye(unused, dtype=complex)
                phase[0, 0] = np.exp(1j * theta)
                complement = complement @ phase

        unitary = np.column_stack((isometry, complement)).astype(complex)
        require(
            np.max(np.abs(
                unitary.conj().T @ unitary - np.eye(len(unitary))
            )) <= MATRIX_TOL,
            "Unitary completion failed.",
        )
        require(
            np.max(np.abs(unitary[:, :left_rank] - isometry)) <= MATRIX_TOL,
            "Completion changed populated columns.",
        )
        wires = [q, q + 1] if q < NQ - 1 else [q]

        if specification["mode"] == "explicit_cx" and len(wires) == 2:
            decomposition = cx_decomposer(unitary, approximate=False)
            actual = Operator(decomposition).data
            overlap = np.vdot(unitary.ravel(), actual.ravel())
            require(abs(overlap) > 0, "Zero decomposition overlap.")
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
            require(
                np.max(np.abs(
                    actual - overlap / abs(overlap) * unitary
                )) <= MATRIX_TOL,
                "Explicit CX decomposition fails exact equivalence.",
            )
            circuit.compose(decomposition, qubits=wires, inplace=True)
        else:
            circuit.append(UnitaryGate(unitary), wires)

        dimension = len(unitary)
        mask = (dimension - 1) << q
        output = np.zeros(DIM, dtype=complex)
        for k, amplitude in enumerate(independent):
            column = (k >> q) & (dimension - 1)
            base = k & ~mask
            for row in range(dimension):
                output[base | (row << q)] += unitary[row, column] * amplitude
        independent = output
        gate_records.append({
            "qargs": wires, "unused_columns": unused,
            "unitary_sha256": sha(unitary.tobytes()),
            "maximum_imaginary_entry": float(np.max(np.abs(unitary.imag))),
        })

    return circuit, {
        "variant": specification["name"], "state": name,
        "cuts": cuts, "factorization_tails": tails,
        "retained_norm_before_roundoff_cleanup": retained_norm,
        "independent_state_error": check_state(
            independent, vector, "Independent sequential synthesis"
        ),
        "qiskit_state_error": check_state(
            Statevector.from_instruction(circuit).data,
            vector, "Qiskit sequential synthesis",
        ),
        "gates": gate_records,
    }

def measured_circuit(preparation, group, axes_override=None):
    qr = QuantumRegister(NQ, "q")
    cr = ClassicalRegister(NQ, REGISTER)
    circuit = QuantumCircuit(qr, cr)
    circuit.compose(preparation, inplace=True)
    axes = group["axes"] if axes_override is None else axes_override
    for q, axis in enumerate(axes):
        if axis == "X":
            circuit.h(q)
        elif axis == "Y":
            circuit.sdg(q)
            circuit.h(q)
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        else:
            require(axis == "Z", "Invalid measurement axis.")
    for q in range(NQ):
        circuit.measure(qr[q], cr[q])
    return circuit

preparations, logical = {}, {}
synthesis_checks, logical_checks, rejected = [], [], []
active_specs = []

# Optional synthesis failure rejects that entire local candidate.
# There is no fallback to another preparation method.
for specification in specifications:
    temporary_preps, temporary_checks = {}, []
    try:
        for name, vector in states.items():
            prep, record = synthesize(vector, specification, name)
            temporary_preps[(specification["name"], name)] = prep
            temporary_checks.append(record)
    except RuntimeError as error:
        if specification["name"] == "baseline":
            raise
        rejected.append({
            "variant": specification["name"], "stage": "local_synthesis",
            "reason": str(error), "submitted": False,
        })
        continue
    preparations.update(temporary_preps)
    synthesis_checks.extend(temporary_checks)
    active_specs.append(specification)

ideal, exact = {}, {}
for name, vector in states.items():
    intended = Statevector(vector)
    direct = real(intended.expectation_value(pauli), "Direct Pauli energy")
    require(abs(direct - classical[name]) <= ENERGY_TOL, "Pauli/classical mismatch.")
    ideal[name], contributions = [], []
    for gi, group in enumerate(groups):
        measured = measured_circuit(preparations[("baseline", name)], group)
        p = Statevector.from_instruction(
            measured.remove_final_measurements(inplace=False)
        ).probabilities()
        a = real(intended.expectation_value(operators[gi]), "Direct group energy")
        b = float(p @ group["weights"])
        require(abs(a - b) <= ENERGY_TOL, "Independent group check failed.")
        ideal[name].append(p)
        contributions.append(b)
    exact[name] = {
        "direct_pauli_energy": direct,
        "measurement_energy": math.fsum(contributions),
        "group_contributions": contributions,
```

```
    }
    require(
        abs(exact[name]["measurement_energy"] - classical[name]) <= ENERGY_TOL,
        "Exact measurement sum failed.",
    )

logical_specs = []
for specification in active_specs:
    temporary, checks = {}, []
    try:
        for name in names:
            key = (specification["name"], name)
            temporary[key] = []
            for gi, group in enumerate(groups):
                circuit = measured_circuit(preparations[key], group)
                circuit.name = f"{key[0]}_{name}_g{gi:03d}"
                error = check_probabilities(
                    Statevector.from_instruction(
                        circuit.remove_final_measurements(inplace=False)
                    ).probabilities(),
                    ideal[name][gi], circuit.name,
                )
                temporary[key].append(circuit)
                checks.append({
                    "variant": key[0], "state": name, "group": gi,
                    "probability_error": error,
                    "logical_hash": circuit_hash(circuit),
                })
    except RuntimeError as error:
        if specification["name"] == "baseline":
            raise
        rejected.append({
            "variant": specification["name"], "stage": "logical_validation",
            "reason": str(error), "submitted": False,
        })
        continue
    logical.update(temporary)
    logical_checks.extend(checks)
    logical_specs.append(specification)

# ---------------------------- Negative controls -------------------------
negative_controls = []

def expect_failure(function, label):
    try:
        function()
    except RuntimeError:
        negative_controls.append({"label": label, "detected": True})
    else:
        raise RuntimeError(f"Negative control NOT detected: {label}")
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
bad_prep = preparations[("baseline", "COBYLA")].copy()
bad_prep.x(4)
expect_failure(
    lambda: check_state(
        Statevector.from_instruction(bad_prep).data,
        states["COBYLA"], "Corrupt preparation",
    ),
    "preparation-X-q4",
)
bad_axes = list(groups[16]["axes"])
bad_axes[0] = "X"
bad_basis = measured_circuit(
    preparations[("baseline", "COBYLA")], groups[16], bad_axes
)
expect_failure(
    lambda: check_probabilities(
        Statevector.from_instruction(
            bad_basis.remove_final_measurements(inplace=False)
        ).probabilities(),
        ideal["COBYLA"][16], "Wrong basis",
    ),
    "wrong-basis-group16",
)
swap = np.array([
    k ^ ((((k >> 0) ^ (k >> 4)) & 1) * ((1 << 0) | (1 << 4)))
    for k in range(DIM)
])
expect_failure(
    lambda: check_probabilities(
        ideal["COBYLA"][16][swap], ideal["COBYLA"][16], "Wrong bit order"
    ),
    "classical-bit-swap-0-4",
)

def returned_counts(publication):
    require(hasattr(publication.data, REGISTER), "Returned register missing.")
    bits = getattr(publication.data, REGISTER)
    require(bits.num_bits == NQ, "Returned register has wrong width.")
    return bits.get_counts()

def counts_vector(counts, shots):
    frequencies = np.zeros(DIM, dtype=np.int64)
    require(isinstance(counts, dict) and bool(counts), "Missing counts.")
    for bits, count in counts.items():
        require(
            isinstance(bits, str) and len(bits) == NQ
            and set(bits).issubset({"0", "1"}),
            "Invalid returned count string.",
        )
        require(
            isinstance(count, (int, np.integer)) and count >= 0,
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
            "Invalid count multiplicity.",
        )
        frequencies[int(bits, 2)] += int(count)
    require(int(frequencies.sum()) == shots, "Shot count mismatch.")
    return frequencies

def weighted_statistics(probability, weights, shots):
    require(shots > 1, "More than one shot is required.")
    mean = float(probability @ weights)
    variance = float(
        probability @ ((weights - mean) ** 2) / (shots - 1)
    )
    require(variance >= 0, "Negative science sampling variance.")
    return mean, variance

local_results, local_counts = [], []
spec_index = {s["name"]: i for i, s in enumerate(specifications)}
for specification in logical_specs:
    vi = spec_index[specification["name"]]
    for si, name in enumerate(names):
        for rep in range(LOCAL_REPETITIONS):
            energies, variances = [], []
            for gi, group in enumerate(groups):
                entropy = [LOCAL_SEED, vi, si, rep, gi]
                seed = int(np.random.SeedSequence(entropy).generate_state(1)[0])
                local_sampler = StatevectorSampler(
                    default_shots=LOCAL_SHOTS, seed=seed
                )
                publication = local_sampler.run(
                    [logical[(specification["name"], name)][gi]],
                    shots=LOCAL_SHOTS,
                ).result()[0]
                counts = returned_counts(publication)
                p = counts_vector(counts, LOCAL_SHOTS) / LOCAL_SHOTS
                mean, variance = weighted_statistics(p, group["weights"], LOCAL_SHOTS)
                energies.append(mean)
                variances.append(variance)
                local_counts.append({
                    "variant": specification["name"], "state": name,
                    "repetition": rep + 1, "group": gi,
                    "seed_entropy": entropy, "seed": seed, "counts": counts,
                })
            energy = math.fsum(energies)
            variance = math.fsum(variances)
            residual = energy - exact[name]["measurement_energy"]
            se = math.sqrt(variance)
            require(
                abs(residual) <= 6 * se
                if se > 0 else abs(residual) <= ENERGY_TOL,
                "Local finite-shot six-SE acceptance failed.",
            )
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
            local_results.append({
                "variant": specification["name"], "state": name,
                "repetition": rep + 1, "energy": energy,
                "sampling_se": se, "difference_from_exact": residual,
            })

# ------------------- Readout mathematics and local tests -----------------
def assignment_covariance(column, shots):
    return (np.diag(column) - np.outer(column, column)) / (shots - 1)

def assignment_checks(assignment):
    require(
        assignment.shape == (DIM, DIM)
        and np.all(np.isfinite(assignment))
        and np.min(assignment) >= 0
        and np.max(np.abs(assignment.sum(axis=0) - 1)) <= MATRIX_TOL,
        "Invalid column-stochastic assignment matrix.",
    )
    singular = np.linalg.svd(assignment, compute_uv=False)
    condition = (
        float(singular[0] / singular[-1])
        if singular[-1] > 0 else float("inf")
    )
    require(
        singular[-1] >= MIN_ASSIGNMENT_SINGULAR_VALUE
        and condition <= MAX_ASSIGNMENT_CONDITION,
        "Readout inversion rejected by predefined conditioning limits.",
    )
    return {
        "condition_number": condition,
        "smallest_singular_value": float(singular[-1]),
    }

def corrected_group(assignment, probability, weights):
    corrected_probability = np.linalg.solve(assignment, probability)
    corrected_weights = np.linalg.solve(assignment.T, weights)
    energy = float(weights @ corrected_probability)
    jacobian = -np.outer(corrected_weights, corrected_probability)
    return energy, corrected_weights, corrected_probability, jacobian

def calibration_covariance(jacobians, assignment, shots):
    number = len(jacobians)
    covariance = np.zeros((number, number))
    for j in range(DIM):
        covariance += (
            jacobians[:, :, j]
            @ assignment_covariance(assignment[:, j], shots)
            @ jacobians[:, :, j].T
        )
    return (covariance + covariance.T) / 2
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
# Fixed nonsingular positive fixture; no duplicated-column construction.
strengths = np.linspace(0.04, 0.12, DIM)
synthetic = np.diag(1 - strengths) + np.ones((DIM, 1)) * strengths[None, :] / DIM
synthetic_diagnostics = assignment_checks(synthetic)
require(
    np.max(np.abs(synthetic - synthetic.T)) > MATRIX_TOL,
    "Orientation fixture must be nonsymmetric.",
)
rng = np.random.default_rng(321930)
true_probability = rng.dirichlet(np.ones(DIM))
test_weights = rng.normal(size=DIM)
observed = synthetic @ true_probability
recovered, z, q, jac = corrected_group(synthetic, observed, test_weights)
require(np.max(np.abs(q - true_probability)) <= PROB_TOL, "Readout recovery failed.")
require(
    abs(recovered - test_weights @ true_probability) <= ENERGY_TOL,
    "Readout expectation recovery failed.",
)
direction = rng.normal(size=(DIM, DIM))
direction -= direction.mean(axis=0, keepdims=True)
epsilon = 1e-7
derivative_fd = (
    corrected_group(synthetic + epsilon * direction, observed, test_weights)[0]
    - corrected_group(synthetic - epsilon * direction, observed, test_weights)[0]
) / (2 * epsilon)
derivative_analytic = float(np.sum(jac * direction))
require(
    abs(derivative_fd - derivative_analytic) <= 1e-6,
    "Readout Jacobian validation failed.",
)
duplicate_covariance = calibration_covariance(
    np.stack([jac, jac]), synthetic, 10000
)
require(
    abs(duplicate_covariance[0, 0] + duplicate_covariance[1, 1]
        - 2 * duplicate_covariance[0, 1]) <= MATRIX_TOL,
    "Shared-calibration covariance validation failed.",
)
expect_failure(
    lambda: assignment_checks(np.ones((DIM, DIM)) / DIM),
    "singular-readout-assignment",
)
expect_failure(
    lambda: require(
        np.max(np.abs(
            np.linalg.solve(synthetic.T, observed) - true_probability
        )) <= PROB_TOL,
        "Transposed assignment detected.",
    ),
    "transposed-assignment",
)
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
column = synthetic[:, 7]
draws = rng.multinomial(5000, column, size=12000) / 5000
theoretical = (np.diag(column) - np.outer(column, column)) / 5000
covariance_relative_error = float(
    np.linalg.norm(np.cov(draws, rowvar=False, ddof=1) - theoretical)
    / np.linalg.norm(theoretical)
)
require(covariance_relative_error < 0.15, "Local covariance simulation failed.")

save_json("local_acceptance.json", {
    "hermitian_error": hermitian_error,
    "hamiltonian_reconstruction_error": reconstruction_error,
    "exact": exact, "synthesis_checks": synthesis_checks,
    "logical_checks": logical_checks,
    "negative_controls": negative_controls,
    "synthetic_readout_conditioning": synthetic_diagnostics,
    "readout_derivative_finite_difference": derivative_fd,
    "readout_derivative_analytic": derivative_analytic,
    "covariance_relative_error": covariance_relative_error,
})
save_json("local_sampling_results.json", local_results)
save_json("local_sampling_counts.json", local_counts)
save_json("measurement_groups.json", [
    {k: v for k, v in group.items() if k != "weights"} for group in groups
])
save_json("rejected_local_candidates.json", rejected)

print("=== CHANGE SUMMARY ===")
print("All QPU execution targets ibm_marrakesh; no backend substitution.")
print("Only Part I inputs are read. Historical results are static context.")
print("approximation_degree=1.0 explicitly requested; tolerance remains 1e-10.")
print("Optional invalid candidates are rejected, not submitted or silently repaired.")
print("Baseline failure stops. No generic state-preparation fallback is used.")
print("Readout fixture fixed; conditioning safeguards unchanged.")
print("\nVersions:", versions)
print("Evidence:", evidence.resolve())
print("\n=== LOCAL ACCEPTANCE: PASS ===")
print("Readout fixture:", synthetic_diagnostics)
print("Jacobian analytic / finite difference:", derivative_analytic, derivative_fd)
print("Covariance relative error:", covariance_relative_error)
print(
    f"Pauli terms={len(original)}, groups={len(groups)}, "
    f"H reconstruction error={reconstruction_error:.3e}"
)
for name in names:
    print(
        f"{name}: classical={classical[name]:.16f}; "
        f"Pauli={exact[name]['direct_pauli_energy']:.16f}; "
        f"measurement={exact[name]['measurement_energy']:.16f}"
    )
for item in local_results:
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    print(
        f"LOCAL {item['variant']}/{item['state']}/rep{item['repetition']}: "
        f"{item['energy']:.12f} +/- {item['sampling_se']:.6f} Ha (sampling SE)"
    )
print("Negative controls:", negative_controls)

# ------------------ Backend connection after local acceptance ------------
try:
    service = QiskitRuntimeService(channel=CHANNEL)
except AccountNotFoundError:
    token = getpass.getpass("IBM Quantum API token (hidden; not saved): ")
    try:
        require(bool(token.strip()), "Empty API token.")
        service = QiskitRuntimeService(channel=CHANNEL, token=token)
    finally:
        del token

backend = service.backend(BACKEND_NAME)
require(backend.name == BACKEND_NAME, "Backend substitution forbidden.")
require(backend.status().operational, f"{BACKEND_NAME} is not operational.")
width = int(backend.num_qubits)
manifest["physical_backend_width"] = width
properties = backend.properties()

target_snapshot = []
readout_errors = {q: None for q in range(width)}
oneq_errors = {q: [] for q in range(width)}
edge_errors = {}
adjacency = {q: set() for q in range(width)}

for operation_name in backend.target.operation_names:
    for qargs, props in backend.target[operation_name].items():
        if qargs is None:
            continue
        error = None if props is None else props.error
        duration = None if props is None else props.duration
        target_snapshot.append({
            "operation": operation_name, "qubits": list(qargs),
            "error": error, "duration_seconds": duration,
        })
        if len(qargs) == 2:
            a, b = qargs
            adjacency[a].add(b)
            adjacency[b].add(a)
        if error is None or not np.isfinite(error):
            continue
        if len(qargs) == 1 and operation_name == "measure":
            readout_errors[qargs[0]] = float(error)
        elif len(qargs) == 1 and operation_name not in ("delay", "reset"):
            oneq_errors[qargs[0]].append(float(error))
        elif len(qargs) == 2:
```

```
            edge = tuple(sorted(qargs))
            edge_errors[edge] = min(edge_errors.get(edge, float("inf")), float(error))

known_readout = [x for x in readout_errors.values() if x is not None]
require(
    known_readout and edge_errors,
    "Insufficient backend calibration/connectivity for declared placement policy.",
)
missing_readout = max(known_readout) + 0.1
missing_edge = max(edge_errors.values()) + 0.1

def qubit_cost(q):
    return (
        readout_errors[q] if readout_errors[q] is not None else missing_readout
    ) + (float(np.mean(oneq_errors[q])) if oneq_errors[q] else 0.0)

# Backend-specific placement: do not reuse the old backend's qubit numbers.
paths = [(qubit_cost(q), (q,)) for q in range(width)]
for length in range(1, NQ):
    extended = []
    for cost, path in paths:
        for neighbor in sorted(adjacency[path[-1]]):
            if neighbor not in path:
                extended.append((
                    cost + qubit_cost(neighbor)
                    + edge_errors.get(
                        tuple(sorted((path[-1], neighbor))), missing_edge
                    ),
                    path + (neighbor,),
                ))
    require(bool(extended), "No connected five-qubit path found.")
    paths = sorted(extended)[:5000]
placement_score, chosen_path = min(paths)
layout = list(chosen_path)
manifest["placement"] = {
    "initial_layout": layout,
    "heuristic_score": placement_score,
    "method": (
        "Current backend target; five-site connected-path beam search, "
        "beam width 5000. Same selected layout for all compiler/synthesis variants."
    ),
    "qualification": "Calibration score is not a measured energy improvement.",
}
save_json("calibration_before.json", {
    "retrieved_utc": utc(),
    "properties": None if properties is None else properties.to_dict(),
    "target_instructions": target_snapshot,
    "chosen_layout": layout,
})

def make_manager(level, initial_layout):
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    return generate_preset_pass_manager(
        backend=backend,
        optimization_level=level,
        initial_layout=list(initial_layout),
        seed_transpiler=TRANSPILE_SEED,
        approximation_degree=APPROXIMATION_DEGREE,
        unitary_synthesis_method="default",
    )

def compact_validation(circuit, corrupt_output=None):
    require(
        circuit.num_clbits == NQ and len(circuit.cregs) == 1
        and circuit.cregs[0].name == REGISTER
        and len(circuit.cregs[0]) == NQ,
        "Compiled classical register changed.",
    )
    mapping, active = {}, set()
    for item in circuit.data:
        physical = [circuit.find_bit(q).index for q in item.qubits]
        if item.operation.name == "measure":
            require(
                len(physical) == 1 and len(item.clbits) == 1,
                "Invalid compiled measurement.",
            )
            c = circuit.find_bit(item.clbits[0]).index
            require(c not in mapping, "Repeated measurement destination.")
            mapping[c] = physical[0]
        if item.operation.name not in ("barrier", "delay"):
            active.update(physical)
    require(
        set(mapping) == set(range(NQ))
        and len(set(mapping.values())) == NQ,
        "Invalid compiled measurement mapping.",
    )
    require(len(active) <= MAX_ACTIVE_QUBITS, "Compact validation exceeds cap.")
    require(circuit.layout is not None, "Missing compiled layout.")
    final = list(circuit.layout.final_index_layout(filter_ancillas=True))
    if len(final) == NQ:
        require(
            all(mapping[c] == final[c] for c in range(NQ)),
            "Layout/classical mapping mismatch.",
        )
    wires = sorted(active)
    index = {q: j for j, q in enumerate(wires)}
    compact = QuantumCircuit(len(wires))
    compact.global_phase = circuit.global_phase
    measured = set()
    for item in circuit.data:
        op = item.operation
        physical = [circuit.find_bit(q).index for q in item.qubits]
        if op.name == "barrier":
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
            continue
        if op.name == "measure":
            measured.update(physical)
            continue
        require(
            not item.clbits and getattr(op, "condition", None) is None,
            "Dynamic control unsupported by exact validator.",
        )
        require(not measured.intersection(physical), "Nonterminal measurement.")
        if op.name == "delay":
            continue
        require(op.name != "reset", "Unexpected reset in exact synthesis.")
        compact.append(op.copy(), [index[q] for q in physical])
    if corrupt_output is not None:
        compact.x(index[corrupt_output])
    p = Statevector.from_instruction(compact).probabilities(
        qargs=[index[mapping[c]] for c in range(NQ)]
    )
    return p, mapping, wires

def compiled_metrics(circuit):
    clocks = np.zeros(circuit.num_qubits)
    oneq, twoq, pulse_oneq = 0, 0, 0
    missing = []
    twoq_types = Counter()
    for item in circuit.data:
        op = item.operation
        physical = tuple(circuit.find_bit(q).index for q in item.qubits)
        if op.name == "barrier":
            if physical:
                clocks[list(physical)] = max(clocks[list(physical)])
            continue
        require(
            backend.target.instruction_supported(
                operation_name=op.name, qargs=physical
            ),
            f"Unsupported backend instruction {op.name} {physical}.",
        )
        props = backend.target[op.name].get(physical)
        duration = None if props is None else props.duration
        if duration is None:
            missing.append({"operation": op.name, "qubits": physical})
            duration = 0.0
        if len(physical) == 2:
            twoq += 1
            twoq_types[op.name] += 1
        elif len(physical) == 1 and op.name not in ("measure", "delay", "reset"):
            oneq += 1
            if duration > 0:
                pulse_oneq += 1
        if physical:
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
            clocks[list(physical)] = max(clocks[list(physical)]) + float(duration)
    return {
        "depth": circuit.depth(),
        "two_qubit_gates": twoq,
        "two_qubit_gate_types": dict(twoq_types),
        "single_qubit_gates": oneq,
        "positive_duration_single_qubit_gates": pulse_oneq,
        "ideal_ASAP_seconds": None if missing else float(clocks.max()),
        "missing_duration_entries": missing,
        "duration_qualification": (
            "Target-based ideal ASAP critical path including measurement; "
            "not a scheduled pulse artifact or measured runtime."
        ),
    }

# Compact validator independently tested on a nonsymmetric circuit.
probe = QuantumCircuit(
    QuantumRegister(NQ, "q"), ClassicalRegister(NQ, REGISTER)
)
probe.ry(0.37, 0)
probe.h(2)
probe.cx(0, 4)
probe.rz(-0.29, 4)
probe.cx(4, 2)
probe.sdg(2)
for q in range(NQ):
    probe.measure(q, q)
compiled_probe = make_manager(2, layout).run(probe)
p, mapping, _ = compact_validation(compiled_probe)
probe_error = check_probabilities(
    p,
    Statevector.from_instruction(
        probe.remove_final_measurements(inplace=False)
    ).probabilities(),
    "Compact-validator self-test",
)
save_json("compact_validator_self_test.json", {
    "probability_error": probe_error, "mapping": mapping,
})

compiled, records, accepted_specs = {}, [], []
for specification in logical_specs:
    variant = specification["name"]
    temporary_compiled, temporary_records = {}, []
    failure_context = {}
    try:
        manager = make_manager(specification["level"], layout)
        for name in names:
            circuits = manager.run(logical[(variant, name)])
            temporary_compiled[(variant, name)] = circuits
            for gi, circuit in enumerate(circuits):
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
            failure_context = {
                "state": name, "group": gi,
                "compiled_hash": circuit_hash(circuit),
            }
            p, mapping, active = compact_validation(circuit)
            discrepancy = float(np.max(np.abs(p - ideal[name][gi])))
            failure_context["probability_discrepancy"] = discrepancy
            error = check_probabilities(
                p, ideal[name][gi], f"{variant}/{name}/compiled/group{gi}"
            )
            temporary_records.append({
                "variant": variant, "state": name,
                "group": gi, "basis": groups[gi]["basis"],
                "logical_hash": circuit_hash(logical[(variant, name)][gi]),
                "compiled_hash": circuit_hash(circuit),
                "measurement_map": mapping,
                "active_physical_qubits": active,
                "initial_layout": list(
                    circuit.layout.initial_index_layout(filter_ancillas=True)
                ),
                "final_layout": list(
                    circuit.layout.final_index_layout(filter_ancillas=True)
                ),
                "layout_repr": str(circuit.layout),
                "probability_error": error,
                **compiled_metrics(circuit),
            })
    except RuntimeError as error:
        # Save failed artifacts as rejected evidence, never submission data.
        failed_circuits = [
            c for cs in temporary_compiled.values() for c in cs
        ]
        failed_qpy_hash = (
            save_qpy(f"REJECTED_{variant}.qpy", failed_circuits)
            if failed_circuits else None
        )
        failure = {
            "variant": variant, "stage": "compiled_validation",
            "reason": str(error), **failure_context,
            "rejected_qpy_sha256": failed_qpy_hash,
            "submitted": False,
        }
        rejected.append(failure)
        save_json("rejected_local_candidates.json", rejected)
        print("REJECTED LOCAL CANDIDATE:", failure)
        if variant == "baseline":
            manifest["status"] = "baseline_validation_failed"
            save_json("manifest.json", manifest)
            raise
        continue
    compiled.update(temporary_compiled)
```

```
    records.extend(temporary_records)
    accepted_specs.append(specification)

require(("baseline", "COBYLA") in compiled, "No validated baseline.")
control = compiled[("baseline", "COBYLA")][16]
_, control_map, _ = compact_validation(control)
bad, _, _ = compact_validation(control, corrupt_output=control_map[0])
expect_failure(
    lambda: check_probabilities(bad, ideal["COBYLA"][16], "Corrupt compiled output"),
    "compiled-output-X",
)

# All candidate comparisons use the same layout and approximation setting.
# No hardware energy, reference energy, or prior result enters selection.
scores, aggregate = {}, []
for specification in accepted_specs:
    variant = specification["name"]
    vr = [record for record in records if record["variant"] == variant]
    durations = [record["ideal_ASAP_seconds"] for record in vr]
    scores[variant] = (
        sum(record["two_qubit_gates"] for record in vr),
        sum(record["positive_duration_single_qubit_gates"] for record in vr),
        sum(durations) if all(x is not None for x in durations) else float("inf"),
        sum(record["depth"] for record in vr),
    )
    for name in names:
        sr = [record for record in vr if record["state"] == name]
        aggregate.append({
            "variant": variant, "state": name,
            "measurement_circuit_count": len(sr),
            "mean_depth_per_circuit": float(np.mean([x["depth"] for x in sr])),
            "summed_depth_across_circuits": sum(x["depth"] for x in sr),
            "mean_two_qubit_gates_per_circuit": float(np.mean([
                x["two_qubit_gates"] for x in sr
            ])),
            "summed_two_qubit_gates": sum(x["two_qubit_gates"] for x in sr),
            "summed_single_qubit_gates": sum(x["single_qubit_gates"] for x in sr),
            "summed_positive_duration_single_qubit_gates": sum(
                x["positive_duration_single_qubit_gates"] for x in sr
            ),
            "summed_ideal_ASAP_seconds": (
                sum(x["ideal_ASAP_seconds"] for x in sr)
                if all(x["ideal_ASAP_seconds"] is not None for x in sr) else None
            ),
            "maximum_compiled_probability_error": max(
                x["probability_error"] for x in sr
            ),
        })

eligible = []
for specification in accepted_specs:
```

```
    variant = specification["name"]
    if variant == "baseline":
        continue
    no_twoq_increase = all(
        sum(x["two_qubit_gates"] for x in records
            if x["variant"] == variant and x["state"] == name)
        <= sum(x["two_qubit_gates"] for x in records
               if x["variant"] == "baseline" and x["state"] == name)
        for name in names
    )
    if no_twoq_increase and scores[variant] < scores["baseline"]:
        eligible.append(variant)
candidate = min(eligible, key=lambda name: (scores[name], name)) if eligible else None
selected = ["baseline"] + ([candidate] if candidate else [])

factor_comparisons = []
for row in aggregate:
    base = next(
        x for x in aggregate
        if x["variant"] == "baseline" and x["state"] == row["state"]
    )
    factor_comparisons.append({
        "variant": row["variant"], "state": row["state"],
        "comparator": "baseline",
        "changed_factor": (
            "none" if row["variant"] == "baseline" else
            "compiler level only" if row["variant"] == "compiler_level3" else
            "synthesis only"
        ),
        "mean_depth_difference": (
            row["mean_depth_per_circuit"] - base["mean_depth_per_circuit"]
        ),
        "summed_two_qubit_difference": (
            row["summed_two_qubit_gates"] - base["summed_two_qubit_gates"]
        ),
        "summed_single_qubit_difference": (
            row["summed_single_qubit_gates"] - base["summed_single_qubit_gates"]
        ),
    })
manifest["selection"] = {
    "criteria": (
        "Validated exact circuits first; no per-state 2q increase; "
        "lexicographic reduction in summed 2q, positive-duration 1q, "
        "ideal ASAP seconds, then depth; names break ties."
    ),
    "scores": scores,
    "candidate": candidate,
    "selected_variants": selected,
    "rejected_candidates": rejected,
    "no_candidate_policy": "Four baseline-only blocks; no invented improvement.",
}
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
save_json("all_accepted_compiled_records.json", records)
save_json("compiled_aggregate_metrics.json", aggregate)
save_json("one_factor_compiled_comparisons.json", factor_comparisons)
save_json("negative_controls.json", negative_controls)

selected_records = [record for record in records if record["variant"] in selected]
physical_sets = sorted({
    tuple(sorted(record["measurement_map"].values()))
    for record in selected_records
})
require(all(len(s) == NQ for s in physical_sets), "Invalid physical readout set.")

def canonical_permutation(mapping, physical_set):
    position = {q: j for j, q in enumerate(physical_set)}
    permutation = np.zeros(DIM, dtype=int)
    for logical_index in range(DIM):
        canonical_index = 0
        for c in range(NQ):
            canonical_index |= (
                ((logical_index >> c) & 1) << position[mapping[c]]
            )
        permutation[logical_index] = canonical_index
    require(len(set(permutation.tolist())) == DIM, "Nonbijective explicit permutation.")
    return permutation

for record in selected_records:
    physical_set = tuple(sorted(record["measurement_map"].values()))
    record["physical_set_id"] = physical_sets.index(physical_set)
    record["logical_to_canonical_physical_index"] = canonical_permutation(
        record["measurement_map"], physical_set
    ).tolist()

# ------------------------ Readout circuit construction -------------------
calibration_circuits, calibration_associations = {}, {}
if ENABLE_READOUT:
    physical_manager = make_manager(0, list(range(width)))
    for set_id, physical_set in enumerate(physical_sets):
        circuits, associations = [], []
        for prepared in range(DIM):
            circuit = QuantumCircuit(
                QuantumRegister(width, "physical"),
                ClassicalRegister(NQ, REGISTER),
            )
            for c, physical in enumerate(physical_set):
                if (prepared >> c) & 1:
                    circuit.x(physical)
            for c, physical in enumerate(physical_set):
                circuit.measure(physical, c)
            circuit = physical_manager.run(circuit)
            circuit.name = f"readout_set{set_id}_prepare{prepared:02d}"
            p, mapping, _ = compact_validation(circuit)
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
            require(
                all(mapping[c] == physical_set[c] for c in range(NQ)),
                "Calibration physical measurement mapping changed.",
            )
            expected = np.zeros(DIM)
            expected[prepared] = 1
            discrepancy = check_probabilities(
                p, expected, "Compiled readout preparation"
            )
            circuits.append(circuit)
            associations.append({
                "publication_index": prepared,
                "prepared_canonical_index": prepared,
                "prepared_canonical_bitstring": format(prepared, "05b"),
                "physical_set": physical_set,
                "measurement_map": mapping,
                "compiled_hash": circuit_hash(circuit),
                "probability_error": discrepancy,
                "metrics": compiled_metrics(circuit),
            })
        calibration_circuits[set_id] = circuits
        calibration_associations[set_id] = associations

submission_circuits = [
    circuit for variant in selected for name in names
    for circuit in compiled[(variant, name)]
]
for set_id in calibration_circuits:
    submission_circuits.extend(calibration_circuits[set_id])
manifest["submission_qpy_sha256"] = save_qpy(
    "validated_submission_circuits.qpy", submission_circuits
)
save_json("selected_circuit_associations.json", selected_records)
save_json("readout_circuit_associations.json", calibration_associations)

print("\n=== BACKEND / COMPILED ACCEPTANCE ===")
print(
    f"Backend={BACKEND_NAME}; logical qubits={NQ}; physical width={width}; "
    f"chosen initial layout={layout}"
)
print("Placement score:", placement_score, "(heuristic, not measured benefit)")
print("Compact validator self-test:", probe_error)
print("Approximation degree:", APPROXIMATION_DEGREE)
print("Probability tolerance:", PROB_TOL)
for row in aggregate:
    print(row)
print("\nONE-FACTOR COMPARISONS:")
for row in factor_comparisons:
    print(row)
print("Rejected candidates:", rejected)
print("Selected candidate:", candidate)
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
print("Summed depth is not execution latency. ASAP durations are target-based proxies.")
print(
    "Historical 46/10 counts are not imposed on this backend. "
    "The new native-gate counts above are the actual comparisons."
)
for record in selected_records:
    print(
        f"{record['variant']}/{record['state']}/g{record['group']:02d}: "
        f"depth={record['depth']}, 2q={record['two_qubit_gates']}, "
        f"initial={record['initial_layout']}, final={record['final_layout']}, "
        f"c->physical={record['measurement_map']}"
    )
print("All negative controls:", negative_controls)
used_physical = {q for s in physical_sets for q in s}
print(
    "Backend calibration timestamp:",
    None if properties is None else properties.last_update_date,
)
print("Relevant calibration (full snapshot saved):")
for item in target_snapshot:
    if set(item["qubits"]).issubset(used_physical):
        print(item)
if properties is not None:
    for physical in sorted(used_physical):
        values = {}
        for label, getter in (
            ("T1_seconds", properties.t1),
            ("T2_seconds", properties.t2),
            ("frequency_Hz", properties.frequency),
        ):
            try:
                values[label] = getter(physical)
            except Exception:
                values[label] = None
        print("Physical qubit", physical, values)

# ------------------------ Balanced execution schedule --------------------
cells = [(variant, name) for name in names for variant in selected]
if len(cells) == 4:
    orders = [
        [0, 1, 3, 2], [1, 2, 0, 3],
        [2, 3, 1, 0], [3, 0, 2, 1],
    ]
else:
    orders = [[0, 1], [1, 0], [1, 0], [0, 1]]

schedule = []
for block in range(BLOCKS):
    if ENABLE_READOUT:
        for set_id in range(len(physical_sets)):
            schedule.append({
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
                "kind": "readout_training", "block": block + 1,
                "set_id": set_id, "shots": CALIBRATION_SHOTS,
                "circuit_count": DIM,
            })
    for cell_index in orders[block]:
        variant, name = cells[cell_index]
        schedule.append({
            "kind": "science", "block": block + 1,
            "variant": variant, "state": name,
            "shots": SCIENCE_SHOTS, "circuit_count": len(groups),
        })
    if ENABLE_READOUT:
        for set_id in range(len(physical_sets)):
            schedule.append({
                "kind": "readout_heldout", "block": block + 1,
                "set_id": set_id, "shots": VALIDATION_SHOTS,
                "circuit_count": DIM,
            })
for index, row in enumerate(schedule):
    row["job_index"] = index
    row["requested_shots"] = row["shots"] * row["circuit_count"]
total_shots = sum(row["requested_shots"] for row in schedule)
require(
    total_shots <= MAX_TOTAL_SHOTS,
    "All-overhead workload exceeds the authorized cap; no submission.",
)
manifest.update({
    "schedule": schedule,
    "job_count": len(schedule),
    "circuit_executions": sum(row["circuit_count"] for row in schedule),
    "total_requested_shots": total_shots,
    "physical_readout_sets": physical_sets,
    "status": "validated_awaiting_authorization",
    "budget_policy": (
        "YES approves exactly this schedule; no automatic extra jobs, "
        "adaptive shots, application-level retries, or extra repetitions."
    ),
})

constructor_signature = inspect.signature(SamplerV2)
run_signature = inspect.signature(SamplerV2.run)
require("mode" in constructor_signature.parameters, "Sampler job mode missing.")
require("shots" in run_signature.parameters, "Sampler shots API missing.")
save_json("installed_sampler_api.json", {
    "constructor_signature": str(constructor_signature),
    "run_signature": str(run_signature),
    "class_docstring": inspect.getdoc(SamplerV2),
    "run_docstring": inspect.getdoc(SamplerV2.run),
    "result_schema": f"publication.data.{REGISTER}.get_counts()",
    "qualification": (
        "Installed SDK introspection. No external documentation retrieval "
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        "is claimed. SamplerV2 may emit a deprecation warning in runtime 0.50."
    ),
})
options = {
    "default_shots": SCIENCE_SHOTS,
    "dynamical_decoupling": {"enable": False},
    "twirling": {"enable_gates": False, "enable_measure": False},
}
hardware_sampler = SamplerV2(mode=backend, options=options)
manifest["sampler_options"] = options
save_json("manifest.json", manifest)

print("\n=== COMPLETE WORKLOAD — NO JOBS SUBMITTED ===")
print("Backend:", BACKEND_NAME)
for row in schedule:
    label = (
        f"{row['variant']}/{row['state']}"
        if row["kind"] == "science" else f"physical-set-{row['set_id']}"
    )
    print(
        f"Job {row['job_index']:02d}, block {row['block']}: "
        f"{row['kind']:16s} {label}; {row['circuit_count']} circuits "
        f"x {row['shots']} = {row['requested_shots']:,} shots"
    )
print(
    f"TOTAL: {len(schedule)} jobs; {manifest['circuit_executions']} "
    f"circuit executions; {total_shots:,} shots; cap={MAX_TOTAL_SHOTS:,}."
)
print(
    "One candidate + one physical readout set gives 753,664 total shots: "
    "557,056 science + 131,072 calibration + 65,536 held-out validation."
)
print("Readout enabled:", ENABLE_READOUT)
print("All corruption/synthetic tests are local: zero QPU shots.")
print("No Estimator, Session, new optimization, postselection, or reference fitting.")
print(
    "Failed readout checks make correction unavailable, not silently raw, "
    "synthetic, clipped, or regularized."
)

if input("Type YES to authorize exactly this complete QPU workload: ").strip() != "YES":
    manifest["status"] = "not_submitted"
    save_json("manifest.json", manifest)
    print("No QPU jobs submitted. Hardware results/uncertainties unavailable.")
    return {
        "manifest": manifest,
        "evidence_directory": str(evidence.resolve()),
        "local_results": local_results,
        "compiled_comparisons": factor_comparisons,
        "hardware": None,
    }
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
# ---------------------------- QPU execution -----------------------------
raw_data = {}
stage = "authorized"
try:
    manifest["status"] = "authorized"
    manifest["authorized_utc"] = utc()
    save_json("manifest.json", manifest)

    for row in schedule:
        stage = f"job-{row['job_index']}-submission"
        if row["kind"] == "science":
            circuits = compiled[(row["variant"], row["state"])]
            associations = sorted(
                [
                    record for record in selected_records
                    if record["variant"] == row["variant"]
                    and record["state"] == row["state"]
                ],
                key=lambda record: record["group"],
            )
        else:
            circuits = calibration_circuits[row["set_id"]]
            associations = calibration_associations[row["set_id"]]

        require(len(circuits) == row["circuit_count"], "Workload changed.")
        expected_hashes = [a["compiled_hash"] for a in associations]
        actual_hashes = [circuit_hash(c) for c in circuits]
        require(actual_hashes == expected_hashes, "Validated circuit mutated.")

        # These exact objects were validated and QPY-saved above.
        job = hardware_sampler.run(circuits, shots=row["shots"])
        job_record = {
            **row, "job_id": job.job_id(),
            "submitted_utc": utc(), "status": "submitted",
            "publication_associations": associations,
            "submitted_circuit_hashes": actual_hashes,
        }
        manifest["jobs"].append(job_record)
        save_json("manifest.json", manifest)
        print("Submitted", row["job_index"], job.job_id())

        stage = f"job-{row['job_index']}-retrieval"
        result = job.result()
        save_pickle(f"job_{row['job_index']:02d}_raw_result.pkl", result)
        require(len(result) == len(circuits), "Returned publication count mismatch.")
        counts = [returned_counts(publication) for publication in result]
        save_json(
            f"job_{row['job_index']:02d}_raw_counts.json",
            {
                "job_id": job.job_id(), "schedule": row,
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
            "publication_associations": associations,
            "circuit_hashes": actual_hashes,
            "classical_register": REGISTER,
            "counts": counts,
            "publication_metadata": [pub.metadata for pub in result],
            "result_metadata": getattr(result, "metadata", None),
        },
    )
    frequencies = np.stack([
        counts_vector(c, row["shots"]) for c in counts
    ])
    if row["kind"] == "science":
        key = ("science", row["block"], row["variant"], row["state"])
    else:
        key = (row["kind"], row["block"], row["set_id"])
    raw_data[key] = frequencies

    job_record["status"] = "completed"
    job_record["completed_utc"] = utc()
    try:
        job_record["metrics"] = job.metrics()
    except Exception as error:
        job_record["metrics_unavailable_exception_type"] = type(error).__name__
    save_json("manifest.json", manifest)

# ----------------- Readout conditioning and held-out checks ------------
readout_models, readout_reports = {}, []
if ENABLE_READOUT:
    threshold = READOUT_VALIDATION_FAMILY_ALPHA / (
        BLOCKS * len(physical_sets) * DIM
    )
    for block in range(1, BLOCKS + 1):
        for set_id in range(len(physical_sets)):
            training = raw_data[("readout_training", block, set_id)]
            heldout = raw_data[("readout_heldout", block, set_id)]
            assignment = training.T / CALIBRATION_SHOTS
            report = {
                "block": block, "set_id": set_id,
                "assignment_matrix": assignment,
            }
            try:
                report.update(assignment_checks(assignment))
                inverse = np.linalg.solve(assignment, np.eye(DIM))
                failures, validation = [], []
                for prepared in range(DIM):
                    observed_probability = heldout[prepared] / VALIDATION_SHOTS
                    corrected = inverse @ observed_probability
                    expected = np.zeros(DIM)
                    expected[prepared] = 1

                    shot_covariance = (
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
                    inverse
                    @ assignment_covariance(
                        observed_probability, VALIDATION_SHOTS
                    )
                    @ inverse.T
                )
                cal_covariance = np.zeros((DIM, DIM))
                for j in range(DIM):
                    cal_covariance += (
                        corrected[j] ** 2
                        * inverse
                        @ assignment_covariance(
                            assignment[:, j], CALIBRATION_SHOTS
                        )
                        @ inverse.T
                    )
                covariance = shot_covariance + cal_covariance
                covariance = (covariance + covariance.T) / 2
                residual = corrected - expected
                eig, vectors = np.linalg.eigh(covariance)
                require(eig.min() >= -1e-12, "Invalid validation covariance.")
                keep = eig > max(1e-14, float(eig.max()) * 1e-9)
                null_residual = float(np.linalg.norm(
                    vectors[:, ~keep].T @ residual
                ))
                statistic = float(np.sum(
                    (vectors[:, keep].T @ residual) ** 2 / eig[keep]
                ))
                degrees = int(np.count_nonzero(keep))
                pvalue = (
                    float(chi2.sf(statistic, degrees))
                    if degrees else (
                        1.0 if np.linalg.norm(residual) < 1e-8 else 0.0
                    )
                )
                passed = null_residual <= 1e-6 and pvalue >= threshold
                if not passed:
                    failures.append(prepared)
                validation.append({
                    "prepared": prepared,
                    "corrected_vector": corrected,
                    "chi_square": statistic,
                    "degrees_of_freedom": degrees,
                    "pvalue": pvalue,
                    "threshold": threshold,
                    "nullspace_residual": null_residual,
                    "passed": passed,
                })
            report["validation"] = validation
            report["failed_preparations"] = failures
            report["accepted"] = not failures
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
                report["qualification"] = (
                    "Conservative asymptotic Wald/chi-square diagnostic "
                    "with first-order calibration propagation. Passing "
                    "does not establish transferability or absence of "
                    "preparation errors/drift."
                )
                if report["accepted"]:
                    readout_models[(block, set_id)] = assignment
            except RuntimeError as error:
                report["accepted"] = False
                report["reason"] = str(error)
            readout_reports.append(report)
            print(
                f"READOUT block={block}, set={set_id}: "
                f"accepted={report['accepted']}, "
                f"condition={report.get('condition_number')}"
            )
    save_json("readout_conditioning_and_validation.json", readout_reports)

# ------------------- Group / complete energy reconstruction ------------
block_estimates, group_estimates = [], []
shared_calibration_covariances = {}

for block in range(1, BLOCKS + 1):
    correction_available = ENABLE_READOUT and all(
        (block, set_id) in readout_models
        for set_id in range(len(physical_sets))
    )
    cell_results, cell_jacobians = [], {}
    for variant, name in cells:
        frequencies = raw_data[("science", block, variant, name)]
        raw_energy, raw_variance = 0.0, 0.0
        corrected_energy, corrected_shot_variance = 0.0, 0.0
        jacobians = {
            set_id: np.zeros((DIM, DIM))
            for set_id in range(len(physical_sets))
        }
        for gi, group in enumerate(groups):
            record = next(
                x for x in selected_records
                if x["variant"] == variant and x["state"] == name
                and x["group"] == gi
            )
            p_logical = frequencies[gi] / SCIENCE_SHOTS
            mean, variance = weighted_statistics(
                p_logical, group["weights"], SCIENCE_SHOTS
            )
            raw_energy += mean
            raw_variance += variance
            intended = exact[name]["group_contributions"][gi]
            gr = {
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
                "block": block, "variant": variant, "state": name,
                "group": gi, "basis": group["basis"],
                "ideal_energy": intended,
                "raw_energy": mean,
                "raw_residual": mean - intended,
                "raw_sampling_variance": variance,
                "raw_probability_TV": float(
                    0.5 * np.abs(p_logical - ideal[name][gi]).sum()
                ),
                "mitigated_energy": None,
            }
            if correction_available:
                permutation = np.array(
                    record["logical_to_canonical_physical_index"]
                )
                p_physical = np.zeros(DIM)
                w_physical = np.zeros(DIM)
                p_physical[permutation] = p_logical
                w_physical[permutation] = group["weights"]
                set_id = record["physical_set_id"]
                assignment = readout_models[(block, set_id)]
                energy, corrected_weights, q, jacobian = corrected_group(
                    assignment, p_physical, w_physical
                )
                _, shot_var = weighted_statistics(
                    p_physical, corrected_weights, SCIENCE_SHOTS
                )
                cal_var = nonnegative_variance(
                    calibration_covariance(
                        np.stack([jacobian]), assignment, CALIBRATION_SHOTS
                    )[0, 0],
                    "Group calibration variance",
                )
                corrected_energy += energy
                corrected_shot_variance += shot_var
                jacobians[set_id] += jacobian
                gr.update({
                    "mitigated_energy": energy,
                    "mitigated_residual": energy - intended,
                    "mitigated_sampling_variance": shot_var,
                    "mitigated_calibration_variance": cal_var,
                    "corrected_quasiprobability_minimum": float(q.min()),
                    "corrected_quasiprobability_sum": float(q.sum()),
                    "calibration_jacobian": jacobian,
                })
            group_estimates.append(gr)

        cell_results.append({
            "block": block, "variant": variant, "state": name,
            "raw_energy": raw_energy,
            "raw_sampling_variance": raw_variance,
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
            "mitigated_energy": (
                corrected_energy if correction_available else None
            ),
            "mitigated_sampling_variance": (
                corrected_shot_variance if correction_available else None
            ),
            "mitigation_available": correction_available,
        })
        cell_jacobians[(variant, name)] = jacobians

    if correction_available:
        covariance = np.zeros((len(cells), len(cells)))
        for set_id in range(len(physical_sets)):
            jacobians = np.stack([
                cell_jacobians[cell][set_id] for cell in cells
            ])
            covariance += calibration_covariance(
                jacobians, readout_models[(block, set_id)], CALIBRATION_SHOTS
            )
        shared_calibration_covariances[block] = covariance
        for i, result in enumerate(cell_results):
            result["mitigated_calibration_variance"] = nonnegative_variance(
                covariance[i, i], "Complete energy calibration variance"
            )
    block_estimates.extend(cell_results)

save_json("hardware_group_estimates.json", group_estimates)
save_json("hardware_block_estimates.json", block_estimates)
save_json("shared_calibration_covariances.json", {
    "cell_order": cells,
    "block_covariances": shared_calibration_covariances,
    "qualification": (
        "First-order propagation. Group Jacobians summed before energy "
        "covariance; off-diagonal cell terms retain shared calibration "
        "across implementations/states. Calibrations in distinct blocks "
        "are treated as statistically independent, not drift-independent."
    ),
})

# ---------------------------- Final reporting ----------------------
summaries, residual_analysis, block_difference_analysis = [], [], []
print("\n=== FINAL SCIENTIFIC COMPARISON ===")
print("Raw and mitigated results remain separate.")
for variant, name in cells:
    rows = sorted(
        [
            x for x in block_estimates
            if x["variant"] == variant and x["state"] == name
        ],
        key=lambda x: x["block"],
    )
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        local_rows = [
            x for x in local_results
            if x["variant"] == variant and x["state"] == name
        ]
        for mode in ("raw", "mitigated"):
            if not all(row[f"{mode}_energy"] is not None for row in rows):
                summaries.append({
                    "variant": variant, "state": name, "mode": mode,
                    "available": False,
                    "reason": (
                        "Readout disabled or at least one block rejected; "
                        "no blocks silently discarded or replaced."
                    ),
                })
                print(variant, name, mode, "UNAVAILABLE")
                continue

            energies = np.array([row[f"{mode}_energy"] for row in rows])
            mean = float(energies.mean())
            shot_variance = math.fsum(
                row[f"{mode}_sampling_variance"] for row in rows
            ) / BLOCKS ** 2
            cal_variance = (
                math.fsum(
                    row["mitigated_calibration_variance"] for row in rows
                ) / BLOCKS ** 2 if mode == "mitigated" else None
            )
            block_sd = float(energies.std(ddof=1))
            summary = {
                "variant": variant, "state": name, "mode": mode,
                "available": True,
                "E_exact": analytical_reference,
                "E0_fd": finite_reference,
                "classical_energy": classical[name],
                "exact_pauli_energy": exact[name]["direct_pauli_energy"],
                "exact_measurement_energy": exact[name]["measurement_energy"],
                "local_repetitions": local_rows,
                "block_energies": energies,
                "block_sampling_standard_errors": [
                    math.sqrt(row[f"{mode}_sampling_variance"]) for row in rows
                ],
                "mean_energy": mean,
                "sampling_standard_error_of_mean": math.sqrt(shot_variance),
                "calibration_standard_error_of_mean": (
                    math.sqrt(cal_variance) if cal_variance is not None else None
                ),
                "between_block_standard_deviation": block_sd,
                "between_block_standard_error_of_mean": block_sd / math.sqrt(BLOCKS),
                "finite_representation_difference": (
                    finite_reference - analytical_reference
                ),
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    "classical_variational_residual": classical[name] - finite_reference,
    "quantum_implementation_difference": (
        exact[name]["measurement_energy"] - classical[name]
    ),
    "hardware_difference": mean - exact[name]["measurement_energy"],
    "uncertainty_qualification": (
        "Sampling, calibration, and between-block uncertainties "
        "are distinct. Between-block spread already includes "
        "stochastic variation; do not add it as another independent "
        "sampling variance. Systematic hardware error is not covered."
    ),
}
summaries.append(summary)
print(json.dumps(jsonable(summary), indent=2))

ranked = []
for gi, group in enumerate(groups):
    gr = [
        x for x in group_estimates
        if x["variant"] == variant and x["state"] == name
        and x["group"] == gi
    ]
    residual = float(np.mean([
        x[f"{mode}_energy"] - x["ideal_energy"] for x in gr
    ]))
    ranked.append({
        "group": gi, "basis": group["basis"],
        "mean_residual": residual,
        "sampling_se_of_mean": math.sqrt(math.fsum(
            x[f"{mode}_sampling_variance"] for x in gr
        ) / BLOCKS ** 2),
        "calibration_se_of_group_mean": (
            math.sqrt(math.fsum(
                x["mitigated_calibration_variance"] for x in gr
            ) / BLOCKS ** 2) if mode == "mitigated" else None
        ),
    })
signed = math.fsum(x["mean_residual"] for x in ranked)
absolute = math.fsum(abs(x["mean_residual"]) for x in ranked)
require(
    abs(signed - summary["hardware_difference"]) <= ENERGY_TOL,
    "Residual cancellation does not reconstruct total difference.",
)
analysis = {
    "variant": variant, "state": name, "mode": mode,
    "signed_residual_sum": signed,
    "absolute_residual_sum": absolute,
    "positive_residual_sum": math.fsum(
        max(0.0, x["mean_residual"]) for x in ranked
    ),
    "negative_residual_sum": math.fsum(
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
            min(0.0, x["mean_residual"]) for x in ranked
        ),
        "cancellation_fraction": (
            1 - abs(signed) / absolute if absolute > 0 else None
        ),
        "groups_ranked_by_absolute_residual": sorted(
            ranked, key=lambda x: abs(x["mean_residual"]), reverse=True
        ),
        "prespecified_groups_2_8_14": [
            x for x in ranked if x["group"] in (2, 8, 14)
        ],
    }
    residual_analysis.append(analysis)
    print(
        "Residual cancellation:", signed, absolute,
        analysis["cancellation_fraction"],
    )
    print("Groups 2,8,14:", analysis["prespecified_groups_2_8_14"])
    print("Top groups:", analysis["groups_ranked_by_absolute_residual"][:10])

    for earlier, later in zip(rows[:-1], rows[1:]):
        delta = later[f"{mode}_energy"] - earlier[f"{mode}_energy"]
        delta_shot_var = (
            earlier[f"{mode}_sampling_variance"]
            + later[f"{mode}_sampling_variance"]
        )
        delta_cal_var = (
            earlier["mitigated_calibration_variance"]
            + later["mitigated_calibration_variance"]
            if mode == "mitigated" else None
        )
        group_differences = []
        for gi in range(len(groups)):
            first = next(
                x for x in group_estimates
                if x["variant"] == variant and x["state"] == name
                and x["group"] == gi and x["block"] == earlier["block"]
            )
            second = next(
                x for x in group_estimates
                if x["variant"] == variant and x["state"] == name
                and x["group"] == gi and x["block"] == later["block"]
            )
            group_differences.append({
                "group": gi, "basis": groups[gi]["basis"],
                "later_minus_earlier": (
                    second[f"{mode}_energy"] - first[f"{mode}_energy"]
                ),
            })
        require(
            abs(math.fsum(
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
                    x["later_minus_earlier"] for x in group_differences
                ) - delta) <= ENERGY_TOL,
                "Group differences do not reconstruct block difference.",
            )
            block_difference_analysis.append({
                "variant": variant, "state": name, "mode": mode,
                "earlier_block": earlier["block"],
                "later_block": later["block"],
                "energy_difference": delta,
                "sampling_standard_error": math.sqrt(delta_shot_var),
                "calibration_standard_error": (
                    math.sqrt(delta_cal_var) if delta_cal_var is not None else None
                ),
                "dominant_group_differences": sorted(
                    group_differences,
                    key=lambda x: abs(x["later_minus_earlier"]),
                    reverse=True,
                ),
                "cause": "Unresolved; these differences alone do not identify a cause.",
            })

comparisons = []
if candidate is not None:
    print("\n=== PAIRED NEW-MEASUREMENT COMPARISONS ===")
    for name in names:
        for mode in ("raw", "mitigated"):
            relevant = [
                x for x in summaries
                if x["state"] == name and x["mode"] == mode and x["available"]
            ]
            if len(relevant) != 2:
                continue
            baseline_summary = next(
                x for x in relevant if x["variant"] == "baseline"
            )
            candidate_summary = next(
                x for x in relevant if x["variant"] == candidate
            )
            deltas = (
                np.asarray(candidate_summary["block_energies"])
                - np.asarray(baseline_summary["block_energies"])
            )
            mean_delta = float(deltas.mean())
            paired_se = float(deltas.std(ddof=1) / math.sqrt(BLOCKS))
            critical = float(student_t.ppf(0.975, BLOCKS - 1))
            interval = [
                mean_delta - critical * paired_se,
                mean_delta + critical * paired_se,
            ]
            shot_variance = 0.0
            for block in range(1, BLOCKS + 1):
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
            b = next(
                x for x in block_estimates
                if x["block"] == block and x["variant"] == "baseline"
                and x["state"] == name
            )
            a = next(
                x for x in block_estimates
                if x["block"] == block and x["variant"] == candidate
                and x["state"] == name
            )
            shot_variance += (
                b[f"{mode}_sampling_variance"]
                + a[f"{mode}_sampling_variance"]
            ) / BLOCKS ** 2
        cal_variance = None
        if mode == "mitigated":
            i = cells.index(("baseline", name))
            j = cells.index((candidate, name))
            cal_variance = nonnegative_variance(
                math.fsum(
                    c[i, i] + c[j, j] - 2 * c[i, j]
                    for c in shared_calibration_covariances.values()
                ) / BLOCKS ** 2,
                "Shared-calibration comparison variance",
            )
        comparison = {
            "state": name, "mode": mode, "candidate": candidate,
            "paired_block_differences": deltas,
            "mean_candidate_minus_baseline": mean_delta,
            "conditional_sampling_se": math.sqrt(shot_variance),
            "shared_calibration_se": (
                math.sqrt(cal_variance) if cal_variance is not None else None
            ),
            "between_block_se_of_paired_difference": paired_se,
            "descriptive_95_percent_paired_t_interval": interval,
            "absolute_residual_reduction_point_estimate": (
                abs(baseline_summary["hardware_difference"])
                - abs(candidate_summary["hardware_difference"])
            ),
            "interpretation": (
                "A positive residual-reduction point estimate is not "
                "an automatic improvement claim. Four blocks give "
                "limited drift inference. Compiled-cost reductions "
                "do not establish hardware-energy improvement."
            ),
        }
        comparisons.append(comparison)
        print(json.dumps(jsonable(comparison), indent=2))

save_json("final_numerical_summary.json", summaries)
save_json("dominant_groups_and_cancellation.json", residual_analysis)
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        save_json("block_difference_analysis.json", block_difference_analysis)
        save_json("new_measurement_comparisons.json", comparisons)
        try:
            after = backend.properties(refresh=True)
            save_json("calibration_after.json", {
                "retrieved_utc": utc(),
                "properties": None if after is None else after.to_dict(),
            })
        except Exception as error:
            save_json("calibration_after_unavailable.json", {
                "exception_type": type(error).__name__, "utc": utc(),
            })

        manifest["status"] = "completed"
        manifest["finished_utc"] = utc()
        save_json("manifest.json", manifest)
        print("\nJob IDs:", [x["job_id"] for x in manifest["jobs"]])
        print("Evidence:", evidence.resolve())
        print(
            "No readout-causation claim is made. Assignment calibration includes "
            "preparation error; transferability, drift, and gate error remain "
            "possible systematic limitations."
        )
        return {
            "manifest": manifest,
            "evidence_directory": str(evidence.resolve()),
            "summaries": summaries,
            "comparisons": comparisons,
            "residual_analysis": residual_analysis,
            "block_difference_analysis": block_difference_analysis,
        }

    except BaseException as error:
        manifest["status"] = "interrupted_or_failed"
        manifest["failure"] = {
            "stage": stage, "exception_type": type(error).__name__, "utc": utc(),
        }
        save_json("manifest.json", manifest)
        print(
            f"Stopped at {stage}. Raw results/job associations remain in "
            f"{evidence.resolve()}. Inspect the manifest before rerunning. "
            "No extra QPU jobs or application-level retries are authorized."
        )
        raise


# Strict dependency sweep:
# Only these ten pre-existing Part I definitions are read.
# All other objects, imports, helpers, circuits, and quantum variables are
# created inside this cell. No prior Part II output, service, backend,
# evidence directory, or notebook-history variable is required.
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
_part_ii_required = (
    "H", "r", "h", "n_interior", "R_max", "E_exact", "E0_fd",
    "alpha_cobyla", "alpha_spsa", "normalized_trial_state",
)
_part_ii_missing = [
    name for name in _part_ii_required if name not in globals()
]
if _part_ii_missing:
    raise NameError(
        "Run authoritative Part I first. Missing: " + ", ".join(_part_ii_missing)
    )

part_ii_marrakesh_results = _hydrogen_marrakesh_part_ii(
    **{name: globals()[name] for name in _part_ii_required}
)
```

Fourth-run results (verbatim)

```
=== CHANGE SUMMARY ===
All QPU execution targets ibm_marrakesh; no backend substitution.
Only Part I inputs are read. Historical results are static context.
approximation_degree=1.0 explicitly requested; tolerance remains 1e-10.
Optional invalid candidates are rejected, not submitted or silently repaired.
Baseline failure stops. No generic state-preparation fallback is used.
Readout fixture fixed; conditioning safeguards unchanged.

Versions: {'numpy': '2.1.3', 'scipy': '1.16.3', 'qiskit': '2.5.2', 'qiskit-ibm-runtime': '0.50.0'}
Evidence: /content/hydrogen_marrakesh_20261014T174449_879279Z_97381a6a

=== LOCAL ACCEPTANCE: PASS ===
Readout fixture: {'condition_number': 1.1378498452226022, 'smallest_singular_value': 0.8805583747220377}
Jacobian analytic / finite difference: 0.12304790038006863 0.12304789964723284
Covariance relative error: 0.015721664103721108
Pauli terms=63, groups=17, H reconstruction error=1.776e-15
COBYLA: classical=-0.4609739279597634; Pauli=-0.4609739279597633; measurement=-
0.4609739279597637
SPSA: classical=-0.4607792133832190; Pauli=-0.4607792133832186; measurement=-0.4607792133832206
LOCAL baseline/COBYLA/rep1: -0.437731072562 +/- 0.019792 Ha (sampling SE)
LOCAL baseline/COBYLA/rep2: -0.445599085490 +/- 0.019756 Ha (sampling SE)
LOCAL baseline/SPSA/rep1: -0.464978917440 +/- 0.019771 Ha (sampling SE)
LOCAL baseline/SPSA/rep2: -0.444868253072 +/- 0.019974 Ha (sampling SE)
LOCAL compiler_level3/COBYLA/rep1: -0.455134399051 +/- 0.019932 Ha (sampling SE)
LOCAL compiler_level3/COBYLA/rep2: -0.476422493344 +/- 0.019595 Ha (sampling SE)
LOCAL compiler_level3/SPSA/rep1: -0.448241386414 +/- 0.020040 Ha (sampling SE)
LOCAL compiler_level3/SPSA/rep2: -0.454695136661 +/- 0.019763 Ha (sampling SE)
LOCAL real_svd/COBYLA/rep1: -0.448415442806 +/- 0.019889 Ha (sampling SE)
LOCAL real_svd/COBYLA/rep2: -0.432301855541 +/- 0.019924 Ha (sampling SE)
LOCAL real_svd/SPSA/rep1: -0.481315828959 +/- 0.019825 Ha (sampling SE)
LOCAL real_svd/SPSA/rep2: -0.472051896595 +/- 0.019791 Ha (sampling SE)
LOCAL unused_columns_minus_pi4/COBYLA/rep1: -0.459685987675 +/- 0.019845 Ha (sampling SE)
LOCAL unused_columns_minus_pi4/COBYLA/rep2: -0.456761109349 +/- 0.019851 Ha (sampling SE)
LOCAL unused_columns_minus_pi4/SPSA/rep1: -0.466615202768 +/- 0.019795 Ha (sampling SE)
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
LOCAL unused_columns_minus_pi4/SPSA/rep2: -0.475453158787 +/- 0.019658 Ha (sampling SE)
LOCAL unused_columns_plus_pi4/COBYLA/rep1: -0.453298147292 +/- 0.019818 Ha (sampling SE)
LOCAL unused_columns_plus_pi4/COBYLA/rep2: -0.464571794782 +/- 0.019649 Ha (sampling SE)
LOCAL unused_columns_plus_pi4/SPSA/rep1: -0.436835610526 +/- 0.019904 Ha (sampling SE)
LOCAL unused_columns_plus_pi4/SPSA/rep2: -0.459903459095 +/- 0.019751 Ha (sampling SE)
LOCAL unused_columns_plus_pi2/COBYLA/rep1: -0.465193646749 +/- 0.019855 Ha (sampling SE)
LOCAL unused_columns_plus_pi2/COBYLA/rep2: -0.515638587588 +/- 0.019605 Ha (sampling SE)
LOCAL unused_columns_plus_pi2/SPSA/rep1: -0.441494235084 +/- 0.019857 Ha (sampling SE)
LOCAL unused_columns_plus_pi2/SPSA/rep2: -0.474597811018 +/- 0.019905 Ha (sampling SE)
LOCAL unused_columns_phase/COBYLA/rep1: -0.459928625198 +/- 0.019735 Ha (sampling SE)
LOCAL unused_columns_phase/COBYLA/rep2: -0.438291140965 +/- 0.019907 Ha (sampling SE)
LOCAL unused_columns_phase/SPSA/rep1: -0.452761063349 +/- 0.019921 Ha (sampling SE)
LOCAL unused_columns_phase/SPSA/rep2: -0.419896398272 +/- 0.019813 Ha (sampling SE)
LOCAL equivalent_explicit_cx/COBYLA/rep1: -0.458102159046 +/- 0.019760 Ha (sampling SE)
LOCAL equivalent_explicit_cx/COBYLA/rep2: -0.429216145107 +/- 0.019929 Ha (sampling SE)
LOCAL equivalent_explicit_cx/SPSA/rep1: -0.466945381165 +/- 0.019850 Ha (sampling SE)
LOCAL equivalent_explicit_cx/SPSA/rep2: -0.501491205125 +/- 0.019613 Ha (sampling SE)
Negative controls: [{'label': 'preparation-X-q4', 'detected': True}, {'label': 'wrong-basis-group16', 'detected':
True}, {'label': 'classical-bit-swap-0-4', 'detected': True}, {'label': 'singular-readout-assignment', 'detected': True},
{'label': 'transposed-assignment', 'detected': True}]
IBM Quantum API token (hidden; not saved): ··········
qiskit_runtime_service._discover_account:WARNING:2026-10-14 17:45:55,469: Loading account with the
given token. A saved account will not be used.
qiskit_runtime_service.__init__:WARNING:2026-10-14 17:45:57,643: Instance was not set at service
instantiation. Free and trial plan instances will be prioritized. Based on the following filters: (tags: None, region:
us-east, eu-de), and available plans: (open), the available account instances are: open-instance. If you need a
specific instance set it explicitly either by using a saved account with a saved default instance or passing it in
directly to QiskitRuntimeService(). Alternatively, pass instance='auto' or save it to your account for auto-
selection without warning.
qiskit_runtime_service.backends:WARNING:2026-10-14 17:45:57,645: Using instance: open-instance, plan:
open
REJECTED LOCAL CANDIDATE: {'variant': 'unused_columns_plus_pi4', 'stage': 'compiled_validation',
'reason': 'unused_columns_plus_pi4/SPSA/compiled/group0: probability error 3.914e-07.', 'state': 'SPSA',
'group': 0, 'compiled_hash': '933d04932baecefb41d471533b3097b9830e0e788edd2039da58d55eaeeda943',
'probability_discrepancy': 3.9136746793344557e-07, 'rejected_qpy_sha256':
'f643e77406d7db177cff7ef5348c5b1e520c454aa02abd2426332c4c6ba9c739', 'submitted': False}
REJECTED LOCAL CANDIDATE: {'variant': 'unused_columns_phase', 'stage': 'compiled_validation', 'reason':
'unused_columns_phase/COBYLA/compiled/group0: probability error 3.245e-06.', 'state': 'COBYLA', 'group': 0,
'compiled_hash': 'ef656adb15a3826bd9824067963c8c8d7b3db11087e7455b931f376beb4137f6',
'probability_discrepancy': 3.245157901121676e-06, 'rejected_qpy_sha256':
'5328e90ebdaec9339d199e7e9cb0a7830e3725efdc7f7e247482c0687ffeeef3', 'submitted': False}

=== BACKEND / COMPILED ACCEPTANCE ===
Backend=ibm_marrakesh; logical qubits=5; physical width=156; chosen initial layout=[104, 105, 117, 125, 124]
Placement score: 0.06312511683595579 (heuristic, not measured benefit)
Compact validator self-test: 3.3306690738754696e-16
Approximation degree: 1.0
Probability tolerance: 1e-10
{'variant': 'baseline', 'state': 'COBYLA', 'measurement_circuit_count': 17, 'mean_depth_per_circuit':
42.705882352941174, 'summed_depth_across_circuits': 726, 'mean_two_qubit_gates_per_circuit': 10.0,
'summed_two_qubit_gates': 170, 'summed_single_qubit_gates': 1096,
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

'summed_positive_duration_single_qubit_gates': 596, 'summed_ideal_ASAP_seconds': 6.8204e-05, 'maximum_compiled_probability_error': 1.1879386363489175e-14}
{'variant': 'baseline', 'state': 'SPSA', 'measurement_circuit_count': 17, 'mean_depth_per_circuit': 42.705882352941174, 'summed_depth_across_circuits': 726, 'mean_two_qubit_gates_per_circuit': 10.0, 'summed_two_qubit_gates': 170, 'summed_single_qubit_gates': 1079, 'summed_positive_duration_single_qubit_gates': 596, 'summed_ideal_ASAP_seconds': 6.8204e-05, 'maximum_compiled_probability_error': 1.5432100042289676e-14}
{'variant': 'compiler_level3', 'state': 'COBYLA', 'measurement_circuit_count': 17, 'mean_depth_per_circuit': 41.529411764705884, 'summed_depth_across_circuits': 706, 'mean_two_qubit_gates_per_circuit': 10.0, 'summed_two_qubit_gates': 170, 'summed_single_qubit_gates': 1064, 'summed_positive_duration_single_qubit_gates': 572, 'summed_ideal_ASAP_seconds': 6.7628e-05, 'maximum_compiled_probability_error': 1.1879386363489175e-14}
{'variant': 'compiler_level3', 'state': 'SPSA', 'measurement_circuit_count': 17, 'mean_depth_per_circuit': 41.529411764705884, 'summed_depth_across_circuits': 706, 'mean_two_qubit_gates_per_circuit': 10.0, 'summed_two_qubit_gates': 170, 'summed_single_qubit_gates': 1058, 'summed_positive_duration_single_qubit_gates': 572, 'summed_ideal_ASAP_seconds': 6.748399999999999e-05, 'maximum_compiled_probability_error': 1.5432100042289676e-14}
{'variant': 'real_svd', 'state': 'COBYLA', 'measurement_circuit_count': 17, 'mean_depth_per_circuit': 42.705882352941174, 'summed_depth_across_circuits': 726, 'mean_two_qubit_gates_per_circuit': 10.0, 'summed_two_qubit_gates': 170, 'summed_single_qubit_gates': 1088, 'summed_positive_duration_single_qubit_gates': 596, 'summed_ideal_ASAP_seconds': 6.8204e-05, 'maximum_compiled_probability_error': 1.1823875212257917e-14}
{'variant': 'real_svd', 'state': 'SPSA', 'measurement_circuit_count': 17, 'mean_depth_per_circuit': 42.23529411764706, 'summed_depth_across_circuits': 718, 'mean_two_qubit_gates_per_circuit': 10.0, 'summed_two_qubit_gates': 170, 'summed_single_qubit_gates': 1071, 'summed_positive_duration_single_qubit_gates': 588, 'summed_ideal_ASAP_seconds': 6.791599999999999e-05, 'maximum_compiled_probability_error': 1.4307999229856705e-14}
{'variant': 'unused_columns_minus_pi4', 'state': 'COBYLA', 'measurement_circuit_count': 17, 'mean_depth_per_circuit': 42.705882352941174, 'summed_depth_across_circuits': 726, 'mean_two_qubit_gates_per_circuit': 10.0, 'summed_two_qubit_gates': 170, 'summed_single_qubit_gates': 1063, 'summed_positive_duration_single_qubit_gates': 588, 'summed_ideal_ASAP_seconds': 6.8024e-05, 'maximum_compiled_probability_error': 1.790234627208065e-15}
{'variant': 'unused_columns_minus_pi4', 'state': 'SPSA', 'measurement_circuit_count': 17, 'mean_depth_per_circuit': 40.76470588235294, 'summed_depth_across_circuits': 693, 'mean_two_qubit_gates_per_circuit': 10.0, 'summed_two_qubit_gates': 170, 'summed_single_qubit_gates': 1015, 'summed_positive_duration_single_qubit_gates': 572, 'summed_ideal_ASAP_seconds': 6.7448e-05, 'maximum_compiled_probability_error': 3.774758283725532e-15}
{'variant': 'unused_columns_plus_pi2', 'state': 'COBYLA', 'measurement_circuit_count': 17, 'mean_depth_per_circuit': 43.64705882352941, 'summed_depth_across_circuits': 742, 'mean_two_qubit_gates_per_circuit': 10.0, 'summed_two_qubit_gates': 170, 'summed_single_qubit_gates': 1113, 'summed_positive_duration_single_qubit_gates': 612, 'summed_ideal_ASAP_seconds': 6.8168e-05, 'maximum_compiled_probability_error': 1.3877787807814457e-15}
{'variant': 'unused_columns_plus_pi2', 'state': 'SPSA', 'measurement_circuit_count': 17, 'mean_depth_per_circuit': 43.64705882352941, 'summed_depth_across_circuits': 742, 'mean_two_qubit_gates_per_circuit': 10.0, 'summed_two_qubit_gates': 170, 'summed_single_qubit_gates': 1113, 'summed_positive_duration_single_qubit_gates': 612, 'summed_ideal_ASAP_seconds': 6.8168e-05, 'maximum_compiled_probability_error': 2.4147350785597155e-15}
{'variant': 'equivalent_explicit_cx', 'state': 'COBYLA', 'measurement_circuit_count': 17, 'mean_depth_per_circuit': 41.588235294117645, 'summed_depth_across_circuits': 707, 'mean_two_qubit_gates_per_circuit': 10.0, 'summed_two_qubit_gates': 170, 'summed_single_qubit_gates': 1016, 'summed_positive_duration_single_qubit_gates': 572, 'summed_ideal_ASAP_seconds': 6.7628e-05,

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

'maximum_compiled_probability_error': 1.9484414082171497e-14}
{'variant': 'equivalent_explicit_cx', 'state': 'SPSA', 'measurement_circuit_count': 17, 'mean_depth_per_circuit': 41.588235294117645, 'summed_depth_across_circuits': 707, 'mean_two_qubit_gates_per_circuit': 10.0, 'summed_two_qubit_gates': 170, 'summed_single_qubit_gates': 1015, 'summed_positive_duration_single_qubit_gates': 572, 'summed_ideal_ASAP_seconds': 6.7628e-05, 'maximum_compiled_probability_error': 2.2676305277968822e-14}

ONE-FACTOR COMPARISONS:
{'variant': 'baseline', 'state': 'COBYLA', 'comparator': 'baseline', 'changed_factor': 'none', 'mean_depth_difference': 0.0, 'summed_two_qubit_difference': 0, 'summed_single_qubit_difference': 0}
{'variant': 'baseline', 'state': 'SPSA', 'comparator': 'baseline', 'changed_factor': 'none', 'mean_depth_difference': 0.0, 'summed_two_qubit_difference': 0, 'summed_single_qubit_difference': 0}
{'variant': 'compiler_level3', 'state': 'COBYLA', 'comparator': 'baseline', 'changed_factor': 'compiler level only', 'mean_depth_difference': -1.17647058823529, 'summed_two_qubit_difference': 0, 'summed_single_qubit_difference': -32}
{'variant': 'compiler_level3', 'state': 'SPSA', 'comparator': 'baseline', 'changed_factor': 'compiler level only', 'mean_depth_difference': -1.17647058823529, 'summed_two_qubit_difference': 0, 'summed_single_qubit_difference': -21}
{'variant': 'real_svd', 'state': 'COBYLA', 'comparator': 'baseline', 'changed_factor': 'synthesis only', 'mean_depth_difference': 0.0, 'summed_two_qubit_difference': 0, 'summed_single_qubit_difference': -8}
{'variant': 'real_svd', 'state': 'SPSA', 'comparator': 'baseline', 'changed_factor': 'synthesis only', 'mean_depth_difference': -0.470588235294116, 'summed_two_qubit_difference': 0, 'summed_single_qubit_difference': -8}
{'variant': 'unused_columns_minus_pi4', 'state': 'COBYLA', 'comparator': 'baseline', 'changed_factor': 'synthesis only', 'mean_depth_difference': 0.0, 'summed_two_qubit_difference': 0, 'summed_single_qubit_difference': -33}
{'variant': 'unused_columns_minus_pi4', 'state': 'SPSA', 'comparator': 'baseline', 'changed_factor': 'synthesis only', 'mean_depth_difference': -1.941176470588232, 'summed_two_qubit_difference': 0, 'summed_single_qubit_difference': -64}
{'variant': 'unused_columns_plus_pi2', 'state': 'COBYLA', 'comparator': 'baseline', 'changed_factor': 'synthesis only', 'mean_depth_difference': 0.9411764705882391, 'summed_two_qubit_difference': 0, 'summed_single_qubit_difference': 17}
{'variant': 'unused_columns_plus_pi2', 'state': 'SPSA', 'comparator': 'baseline', 'changed_factor': 'synthesis only', 'mean_depth_difference': 0.9411764705882391, 'summed_two_qubit_difference': 0, 'summed_single_qubit_difference': 34}
{'variant': 'equivalent_explicit_cx', 'state': 'COBYLA', 'comparator': 'baseline', 'changed_factor': 'synthesis only', 'mean_depth_difference': -1.117647058823529, 'summed_two_qubit_difference': 0, 'summed_single_qubit_difference': -80}
{'variant': 'equivalent_explicit_cx', 'state': 'SPSA', 'comparator': 'baseline', 'changed_factor': 'synthesis only', 'mean_depth_difference': -1.117647058823529, 'summed_two_qubit_difference': 0, 'summed_single_qubit_difference': -64}
Rejected candidates: [{'variant': 'unused_columns_plus_pi4', 'stage': 'compiled_validation', 'reason': 'unused_columns_plus_pi4/SPSA/compiled/group0: probability error 3.914e-07.', 'state': 'SPSA', 'group': 0, 'compiled_hash': '933d04932baecefb41d471533b3097b9830e0e788edd2039da58d55eaeeda943', 'probability_discrepancy': 3.9136746793344557e-07, 'rejected_qpy_sha256': 'f643e77406d7db177cff7ef5348c5b1e520c454aa02abd2426332c4c6ba9c739', 'submitted': False}, {'variant': 'unused_columns_phase', 'stage': 'compiled_validation', 'reason': 'unused_columns_phase/COBYLA/compiled/group0: probability error 3.245e-06.', 'state': 'COBYLA', 'group': 0, 'compiled_hash': 'ef656adb15a3826bd9824067963c8c8d7b3db11087e7455b931f376beb4137f6', 'probability_discrepancy': 3.245157901121676e-06, 'rejected_qpy_sha256': '5328e90ebdaec9339d199e7e9cb0a7830e3725efdc7f7e247482c0687ffeeef3', 'submitted': False}]
Selected candidate: compiler_level3

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

Summed depth is not execution latency. ASAP durations are target-based proxies.
Historical 46/10 counts are not imposed on this backend. The new native-gate counts above are the actual comparisons.
baseline/COBYLA/g00: depth=43, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
baseline/COBYLA/g01: depth=43, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
baseline/COBYLA/g02: depth=43, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
baseline/COBYLA/g03: depth=43, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
baseline/COBYLA/g04: depth=42, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
baseline/COBYLA/g05: depth=43, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
baseline/COBYLA/g06: depth=43, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
baseline/COBYLA/g07: depth=43, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
baseline/COBYLA/g08: depth=43, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
baseline/COBYLA/g09: depth=42, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
baseline/COBYLA/g10: depth=42, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
baseline/COBYLA/g11: depth=43, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
baseline/COBYLA/g12: depth=43, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
baseline/COBYLA/g13: depth=43, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
baseline/COBYLA/g14: depth=42, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
baseline/COBYLA/g15: depth=43, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
baseline/COBYLA/g16: depth=42, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
baseline/SPSA/g00: depth=43, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
baseline/SPSA/g01: depth=43, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
baseline/SPSA/g02: depth=43, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
baseline/SPSA/g03: depth=43, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
baseline/SPSA/g04: depth=42, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
baseline/SPSA/g05: depth=43, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
baseline/SPSA/g06: depth=43, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

baseline/SPSA/g07: depth=43, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
baseline/SPSA/g08: depth=43, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
baseline/SPSA/g09: depth=42, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
baseline/SPSA/g10: depth=42, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
baseline/SPSA/g11: depth=43, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
baseline/SPSA/g12: depth=43, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
baseline/SPSA/g13: depth=43, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
baseline/SPSA/g14: depth=42, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
baseline/SPSA/g15: depth=43, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
baseline/SPSA/g16: depth=42, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
compiler_level3/COBYLA/g00: depth=42, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
compiler_level3/COBYLA/g01: depth=40, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
compiler_level3/COBYLA/g02: depth=43, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
compiler_level3/COBYLA/g03: depth=42, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
compiler_level3/COBYLA/g04: depth=40, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
compiler_level3/COBYLA/g05: depth=42, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
compiler_level3/COBYLA/g06: depth=41, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
compiler_level3/COBYLA/g07: depth=40, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
compiler_level3/COBYLA/g08: depth=43, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
compiler_level3/COBYLA/g09: depth=40, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
compiler_level3/COBYLA/g10: depth=41, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
compiler_level3/COBYLA/g11: depth=42, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
compiler_level3/COBYLA/g12: depth=43, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
compiler_level3/COBYLA/g13: depth=41, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
compiler_level3/COBYLA/g14: depth=41, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
compiler_level3/COBYLA/g15: depth=43, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125,

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
compiler_level3/COBYLA/g16: depth=42, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
compiler_level3/SPSA/g00: depth=41, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
compiler_level3/SPSA/g01: depth=41, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
compiler_level3/SPSA/g02: depth=42, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
compiler_level3/SPSA/g03: depth=42, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
compiler_level3/SPSA/g04: depth=39, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
compiler_level3/SPSA/g05: depth=42, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
compiler_level3/SPSA/g06: depth=42, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
compiler_level3/SPSA/g07: depth=42, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
compiler_level3/SPSA/g08: depth=43, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
compiler_level3/SPSA/g09: depth=40, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
compiler_level3/SPSA/g10: depth=40, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
compiler_level3/SPSA/g11: depth=41, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
compiler_level3/SPSA/g12: depth=43, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
compiler_level3/SPSA/g13: depth=43, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
compiler_level3/SPSA/g14: depth=42, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
compiler_level3/SPSA/g15: depth=42, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
compiler_level3/SPSA/g16: depth=41, 2q=10, initial=[104, 105, 117, 125, 124], final=[104, 105, 117, 125, 124], c->physical={0: 104, 1: 105, 2: 117, 3: 125, 4: 124}
All negative controls: [{'label': 'preparation-X-q4', 'detected': True}, {'label': 'wrong-basis-group16', 'detected': True}, {'label': 'classical-bit-swap-0-4', 'detected': True}, {'label': 'singular-readout-assignment', 'detected': True}, {'label': 'transposed-assignment', 'detected': True}, {'label': 'compiled-output-X', 'detected': True}]
Backend calibration timestamp: 2026-10-14 17:21:41+00:00
Relevant calibration (full snapshot saved):
{'operation': 'measure_reset', 'qubits': [104], 'error': None, 'duration_seconds': 2.72e-06}
{'operation': 'measure_reset', 'qubits': [105], 'error': None, 'duration_seconds': 2.72e-06}
{'operation': 'measure_reset', 'qubits': [117], 'error': None, 'duration_seconds': 2.72e-06}
{'operation': 'measure_reset', 'qubits': [124], 'error': None, 'duration_seconds': 2.72e-06}
{'operation': 'measure_reset', 'qubits': [125], 'error': None, 'duration_seconds': 2.72e-06}
{'operation': 'sx', 'qubits': [104], 'error': 0.0009222656780841014, 'duration_seconds': 3.6e-08}
{'operation': 'sx', 'qubits': [105], 'error': 0.00022322736788863725, 'duration_seconds': 3.6e-08}
{'operation': 'sx', 'qubits': [117], 'error': 0.0004841694696129765, 'duration_seconds': 3.6e-08}
{'operation': 'sx', 'qubits': [124], 'error': 0.00032324573235972023, 'duration_seconds': 3.6e-08}

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

{'operation': 'sx', 'qubits': [125], 'error': 0.00042127930454535794, 'duration_seconds': 3.6e-08}
{'operation': 'id', 'qubits': [104], 'error': 0.0009222656780841014, 'duration_seconds': 3.6e-08}
{'operation': 'id', 'qubits': [105], 'error': 0.00022322736788863725, 'duration_seconds': 3.6e-08}
{'operation': 'id', 'qubits': [117], 'error': 0.0004841694696129765, 'duration_seconds': 3.6e-08}
{'operation': 'id', 'qubits': [124], 'error': 0.00032324573235972023, 'duration_seconds': 3.6e-08}
{'operation': 'id', 'qubits': [125], 'error': 0.00042127930454535794, 'duration_seconds': 3.6e-08}
{'operation': 'reset', 'qubits': [104], 'error': None, 'duration_seconds': 2.72e-06}
{'operation': 'reset', 'qubits': [105], 'error': None, 'duration_seconds': 2.72e-06}
{'operation': 'reset', 'qubits': [117], 'error': None, 'duration_seconds': 2.72e-06}
{'operation': 'reset', 'qubits': [124], 'error': None, 'duration_seconds': 2.72e-06}
{'operation': 'reset', 'qubits': [125], 'error': None, 'duration_seconds': 2.72e-06}
{'operation': 'cz', 'qubits': [104, 105], 'error': 0.0023261621418437517, 'duration_seconds': 6.8e-08}
{'operation': 'cz', 'qubits': [105, 104], 'error': 0.0023261621418437517, 'duration_seconds': 6.8e-08}
{'operation': 'cz', 'qubits': [105, 117], 'error': 0.004037101627000467, 'duration_seconds': 6.8e-08}
{'operation': 'cz', 'qubits': [117, 105], 'error': 0.004037101627000467, 'duration_seconds': 6.8e-08}
{'operation': 'cz', 'qubits': [117, 125], 'error': 0.0032198784356635146, 'duration_seconds': 6.8e-08}
{'operation': 'cz', 'qubits': [124, 125], 'error': 0.00211380532562086, 'duration_seconds': 6.8e-08}
{'operation': 'cz', 'qubits': [125, 117], 'error': 0.0032198784356635146, 'duration_seconds': 6.8e-08}
{'operation': 'cz', 'qubits': [125, 124], 'error': 0.00211380532562086, 'duration_seconds': 6.8e-08}
{'operation': 'rz', 'qubits': [104], 'error': 0.0, 'duration_seconds': 0.0}
{'operation': 'rz', 'qubits': [105], 'error': 0.0, 'duration_seconds': 0.0}
{'operation': 'rz', 'qubits': [117], 'error': 0.0, 'duration_seconds': 0.0}
{'operation': 'rz', 'qubits': [124], 'error': 0.0, 'duration_seconds': 0.0}
{'operation': 'rz', 'qubits': [125], 'error': 0.0, 'duration_seconds': 0.0}
{'operation': 'delay', 'qubits': [104], 'error': None, 'duration_seconds': None}
{'operation': 'delay', 'qubits': [105], 'error': None, 'duration_seconds': None}
{'operation': 'delay', 'qubits': [117], 'error': None, 'duration_seconds': None}
{'operation': 'delay', 'qubits': [124], 'error': None, 'duration_seconds': None}
{'operation': 'delay', 'qubits': [125], 'error': None, 'duration_seconds': None}
{'operation': 'xslow', 'qubits': [104], 'error': 0.0009222656780841014, 'duration_seconds': 1e-06}
{'operation': 'xslow', 'qubits': [105], 'error': 0.00022322736788863725, 'duration_seconds': 1e-06}
{'operation': 'xslow', 'qubits': [117], 'error': 0.0004841694696129765, 'duration_seconds': 1e-06}
{'operation': 'xslow', 'qubits': [124], 'error': 0.00032324573235972023, 'duration_seconds': 1e-06}
{'operation': 'xslow', 'qubits': [125], 'error': 0.00042127930454535794, 'duration_seconds': 1e-06}
{'operation': 'measure', 'qubits': [104], 'error': 0.0042724609375, 'duration_seconds': 2.684e-06}
{'operation': 'measure', 'qubits': [105], 'error': 0.0059814453125, 'duration_seconds': 2.684e-06}
{'operation': 'measure', 'qubits': [117], 'error': 0.0059814453125, 'duration_seconds': 2.684e-06}
{'operation': 'measure', 'qubits': [124], 'error': 0.0079345703125, 'duration_seconds': 2.684e-06}
{'operation': 'measure', 'qubits': [125], 'error': 0.011962890625, 'duration_seconds': 2.684e-06}
{'operation': 'measure_2', 'qubits': [104], 'error': 0.007568359375, 'duration_seconds': 1.884e-06}
{'operation': 'measure_2', 'qubits': [105], 'error': 0.014892578125, 'duration_seconds': 1.884e-06}
{'operation': 'measure_2', 'qubits': [117], 'error': 0.01025390625, 'duration_seconds': 1.884e-06}
{'operation': 'measure_2', 'qubits': [124], 'error': 0.010498046875, 'duration_seconds': 1.884e-06}
{'operation': 'measure_2', 'qubits': [125], 'error': 0.0390625, 'duration_seconds': 1.884e-06}
{'operation': 'reset_2', 'qubits': [104], 'error': None, 'duration_seconds': 1.92e-06}
{'operation': 'reset_2', 'qubits': [105], 'error': None, 'duration_seconds': 1.92e-06}
{'operation': 'reset_2', 'qubits': [117], 'error': None, 'duration_seconds': 1.92e-06}
{'operation': 'reset_2', 'qubits': [124], 'error': None, 'duration_seconds': 1.92e-06}
{'operation': 'reset_2', 'qubits': [125], 'error': None, 'duration_seconds': 1.92e-06}
{'operation': 'measure_reset_2', 'qubits': [104], 'error': None, 'duration_seconds': 1.92e-06}
{'operation': 'measure_reset_2', 'qubits': [105], 'error': None, 'duration_seconds': 1.92e-06}

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

{'operation': 'measure_reset_2', 'qubits': [117], 'error': None, 'duration_seconds': 1.92e-06}
{'operation': 'measure_reset_2', 'qubits': [124], 'error': None, 'duration_seconds': 1.92e-06}
{'operation': 'measure_reset_2', 'qubits': [125], 'error': None, 'duration_seconds': 1.92e-06}
{'operation': 'x', 'qubits': [104], 'error': 0.0009222656780841014, 'duration_seconds': 3.6e-08}
{'operation': 'x', 'qubits': [105], 'error': 0.00022322736788863725, 'duration_seconds': 3.6e-08}
{'operation': 'x', 'qubits': [117], 'error': 0.0004841694696129765, 'duration_seconds': 3.6e-08}
{'operation': 'x', 'qubits': [124], 'error': 0.00032324573235972023, 'duration_seconds': 3.6e-08}
{'operation': 'x', 'qubits': [125], 'error': 0.00042127930454535794, 'duration_seconds': 3.6e-08}
Physical qubit 104 {'T1_seconds': 0.0001923968688758789, 'T2_seconds': 7.999573896350571e-05, 'frequency_Hz': None}
Physical qubit 105 {'T1_seconds': 0.00012102093289914497, 'T2_seconds': 9.785709616895256e-05, 'frequency_Hz': None}
Physical qubit 117 {'T1_seconds': 0.0001645938503030854, 'T2_seconds': 7.237503769052485e-06, 'frequency_Hz': None}
Physical qubit 124 {'T1_seconds': 8.992313333646118e-05, 'T2_seconds': 2.0017032973450083e-05, 'frequency_Hz': None}
Physical qubit 125 {'T1_seconds': 4.350935608306467e-05, 'T2_seconds': 3.6729416541854015e-05, 'frequency_Hz': None}

=== COMPLETE WORKLOAD — NO JOBS SUBMITTED ===
Backend: ibm_marrakesh
Job 00, block 1: readout_training physical-set-0; 32 circuits x 1024 = 32,768 shots
Job 01, block 1: science baseline/COBYLA; 17 circuits x 2048 = 34,816 shots
Job 02, block 1: science compiler_level3/COBYLA; 17 circuits x 2048 = 34,816 shots
Job 03, block 1: science compiler_level3/SPSA; 17 circuits x 2048 = 34,816 shots
Job 04, block 1: science baseline/SPSA; 17 circuits x 2048 = 34,816 shots
Job 05, block 1: readout_heldout physical-set-0; 32 circuits x 512 = 16,384 shots
Job 06, block 2: readout_training physical-set-0; 32 circuits x 1024 = 32,768 shots
Job 07, block 2: science compiler_level3/COBYLA; 17 circuits x 2048 = 34,816 shots
Job 08, block 2: science baseline/SPSA; 17 circuits x 2048 = 34,816 shots
Job 09, block 2: science baseline/COBYLA; 17 circuits x 2048 = 34,816 shots
Job 10, block 2: science compiler_level3/SPSA; 17 circuits x 2048 = 34,816 shots
Job 11, block 2: readout_heldout physical-set-0; 32 circuits x 512 = 16,384 shots
Job 12, block 3: readout_training physical-set-0; 32 circuits x 1024 = 32,768 shots
Job 13, block 3: science baseline/SPSA; 17 circuits x 2048 = 34,816 shots
Job 14, block 3: science compiler_level3/SPSA; 17 circuits x 2048 = 34,816 shots
Job 15, block 3: science compiler_level3/COBYLA; 17 circuits x 2048 = 34,816 shots
Job 16, block 3: science baseline/COBYLA; 17 circuits x 2048 = 34,816 shots
Job 17, block 3: readout_heldout physical-set-0; 32 circuits x 512 = 16,384 shots
Job 18, block 4: readout_training physical-set-0; 32 circuits x 1024 = 32,768 shots
Job 19, block 4: science compiler_level3/SPSA; 17 circuits x 2048 = 34,816 shots
Job 20, block 4: science baseline/COBYLA; 17 circuits x 2048 = 34,816 shots
Job 21, block 4: science baseline/SPSA; 17 circuits x 2048 = 34,816 shots
Job 22, block 4: science compiler_level3/COBYLA; 17 circuits x 2048 = 34,816 shots
Job 23, block 4: readout_heldout physical-set-0; 32 circuits x 512 = 16,384 shots
TOTAL: 24 jobs; 528 circuit executions; 753,664 shots; cap=1,114,112.
One candidate + one physical readout set gives 753,664 total shots: 557,056 science + 131,072 calibration + 65,536 held-out validation.
Readout enabled: True
All corruption/synthetic tests are local: zero QPU shots.
No Estimator, Session, new optimization, postselection, or reference fitting.

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
Failed readout checks make correction unavailable, not silently raw, synthetic, clipped, or regularized.
/tmp/ipykernel_1099/972478245.py:1558: DeprecationWarning: The SamplerV2 class is deprecated as of qiskit-ibm-runtime 0.50.0 and will be removed no sooner than 3 months after the release date. `qiskit_ibm_runtime.executor_sampler.Sampler` provides a new Sampler implementation that runs all the pre- and post-processing in the client side. The top-level `qiskit_ibm_runtime.Sampler` import will be updated to point to the client-side sampler in an upcoming release, with the legacy Sampler being available until the deprecation is completed.
  hardware_sampler = SamplerV2(mode=backend, options=options)
Type YES to authorize exactly this complete QPU workload: YES
Submitted 0 db7rvtmr6oqs73886pfg
Submitted 1 db7s212raffc73fb627g
Submitted 2 db7s546r6oqs73887590
Submitted 3 db7s6ber6oqs73887870
Submitted 4 db7s6o6r6oqs73887970
Submitted 5 db7s72ur6oqs73887a20
Submitted 6 db7s788r1c5c738l6qfg
Submitted 7 db7sejqraffc73fb6vb0
Submitted 8 db7slluil99s73foq4ng
Submitted 9 db7slqiraffc73fb7e3g
Submitted 10 db7sm2ur6oqs73888alg
Submitted 11 db7sm9uil99s73foq60g
Submitted 12 db7smdaraffc73fb7fb0
Submitted 13 db7smi6r6oqs73888bog
Submitted 14 db7smn6r6oqs73888c6g
Submitted 15 db7sms6il99s73foq760
Submitted 16 db7sn36il99s73foq7gg
Submitted 17 db7sna2raffc73fb7hd0
Submitted 18 db7snl0r1c5c738l7v50
Submitted 19 db7snq6il99s73foq94g
Submitted 20 db7so0or1c5c738l7vsg
Submitted 21 db7soomr6oqs73888gkg
Submitted 22 db7sotmil99s73foqbig
Submitted 23 db7sp4gr1c5c738l829g
READOUT block=1, set=0: accepted=True, condition=1.1013640636346567
READOUT block=2, set=0: accepted=True, condition=1.145092917457411
READOUT block=3, set=0: accepted=True, condition=1.1304346730742614
READOUT block=4, set=0: accepted=True, condition=1.178856136637715

=== FINAL SCIENTIFIC COMPARISON ===
Raw and mitigated results remain separate.
{
  "variant": "baseline",
  "state": "COBYLA",
  "mode": "raw",
  "available": true,
  "E_exact": -0.5,
  "E0_fd": -0.46097392795978287,
  "classical_energy": -0.46097392795976344,
  "exact_pauli_energy": -0.4609739279597633,
  "exact_measurement_energy": -0.4609739279597637,
  "local_repetitions": [
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

{
"variant": "baseline",
"state": "COBYLA",
"repetition": 1,
"energy": -0.4377310725620817,
"sampling_se": 0.019792082541606536,
"difference_from_exact": 0.023242855397682005
},
{
"variant": "baseline",
"state": "COBYLA",
"repetition": 2,
"energy": -0.44559908548990845,
"sampling_se": 0.01975615554566369,
"difference_from_exact": 0.015374842469855265
}
],
"block_energies": [
-0.22125652254115624,
-0.17055859489984837,
-0.2374866031254832,
-0.17211174692775916
],
"block_sampling_standard_errors": [
0.03057580869033737,
0.03057784628783949,
0.030111611937266164,
0.03060114405480395
],
"mean_energy": -0.20035336687356176,
"sampling_standard_error_of_mean": 0.015233646870485049,
"calibration_standard_error_of_mean": null,
"between_block_standard_deviation": 0.03416204856041975,
"between_block_standard_error_of_mean": 0.017081024280209876,
"finite_representation_difference": 0.03902607204021713,
"classical_variational_residual": 1.942890293094024e-14,
"quantum_implementation_difference": -2.7755575615628914e-16,
"hardware_difference": 0.26062056108620196,
"uncertainty_qualification": "Sampling, calibration, and between-block uncertainties are distinct. Between-block spread already includes stochastic variation; do not add it as another independent sampling variance. Systematic hardware error is not covered."
}
Residual cancellation: 0.2606205610862021 0.28888486779885475 0.09783934661587257
Groups 2,8,14: [{'group': 2, 'basis': 'YXXYX', 'mean_residual': 0.07336004436245636, 'sampling_se_of_mean': 0.005930310852819033, 'calibration_se_of_group_mean': None}, {'group': 8, 'basis': 'YXYXX', 'mean_residual': 0.05430317752749875, 'sampling_se_of_mean': 0.0039627655679159105, 'calibration_se_of_group_mean': None}, {'group': 14, 'basis': 'YYXXX', 'mean_residual': 0.04159807125519892, 'sampling_se_of_mean': 0.0035395322932463315, 'calibration_se_of_group_mean': None}]
Top groups: [{'group': 2, 'basis': 'YXXYX', 'mean_residual': 0.07336004436245636, 'sampling_se_of_mean': 0.005930310852819033, 'calibration_se_of_group_mean': None}, {'group': 16, 'basis': 'ZZZZZ', 'mean_residual': 0.05850828813555625, 'sampling_se_of_mean': 0.005583906389454, 'calibration_se_of_group_mean': None},

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

{'group': 8, 'basis': 'YXYXX', 'mean_residual': 0.05430317752749875, 'sampling_se_of_mean': 0.0039627655679159105, 'calibration_se_of_group_mean': None}, {'group': 14, 'basis': 'YYXXX', 'mean_residual': 0.04159807125519892, 'sampling_se_of_mean': 0.0035395322932463315, 'calibration_se_of_group_mean': None}, {'group': 3, 'basis': 'YXYYY', 'mean_residual': 0.026436448121156298, 'sampling_se_of_mean': 0.007495885249211925, 'calibration_se_of_group_mean': None}, {'group': 10, 'basis': 'YYYYX', 'mean_residual': 0.012495602321779556, 'sampling_se_of_mean': 0.0038884657962143586, 'calibration_se_of_group_mean': None}, {'group': 9, 'basis': 'YYXYY', 'mean_residual': -0.006307060408630877, 'sampling_se_of_mean': 0.0038540295155408293, 'calibration_se_of_group_mean': None}, {'group': 1, 'basis': 'XYXXY', 'mean_residual': 0.003865082495438134, 'sampling_se_of_mean': 0.0021122560148583076, 'calibration_se_of_group_mean': None}, {'group': 15, 'basis': 'XXXXX', 'mean_residual': -0.0032939304286407917, 'sampling_se_of_mean': 0.002092676276154997, 'calibration_se_of_group_mean': None}, {'group': 0, 'basis': 'YXXXY', 'mean_residual': -0.0019731491012635605, 'sampling_se_of_mean': 0.0009398576224981835, 'calibration_se_of_group_mean': None}]
{
 "variant": "baseline",
 "state": "COBYLA",
 "mode": "mitigated",
 "available": true,
 "E_exact": -0.5,
 "E0_fd": -0.46097392795978287,
 "classical_energy": -0.46097392795976344,
 "exact_pauli_energy": -0.4609739279597633,
 "exact_measurement_energy": -0.4609739279597637,
 "local_repetitions": [
  {
   "variant": "baseline",
   "state": "COBYLA",
   "repetition": 1,
   "energy": -0.4377310725620817,
   "sampling_se": 0.019792082541606536,
   "difference_from_exact": 0.023242855397682005
  },
  {
   "variant": "baseline",
   "state": "COBYLA",
   "repetition": 2,
   "energy": -0.44559908548990845,
   "sampling_se": 0.01975615554566369,
   "difference_from_exact": 0.015374842469855265
  }
 ],
 "block_energies": [
  -0.23457431833705394,
  -0.23954865218801957,
  -0.2580436584420428,
  -0.18916965621691306
 ],
 "block_sampling_standard_errors": [
  0.03129860768077014,
  0.03247815091587274,

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

0.03146216842979507,
0.03210005504647169
],
"mean_energy": -0.23033407129600733,
"sampling_standard_error_of_mean": 0.015919160440253037,
"calibration_standard_error_of_mean": 0.002426467090618514,
"between_block_standard_deviation": 0.029241628343808303,
"between_block_standard_error_of_mean": 0.014620814171904152,
"finite_representation_difference": 0.03902607204021713,
"classical_variational_residual": 1.942890293094024e-14,
"quantum_implementation_difference": -2.7755575615628914e-16,
"hardware_difference": 0.2306398566637564,
"uncertainty_qualification": "Sampling, calibration, and between-block uncertainties are distinct. Between-block spread already includes stochastic variation; do not add it as another independent sampling variance. Systematic hardware error is not covered."
}
Residual cancellation: 0.23063985666375625 0.2574798330508548 0.10424108198717608
Groups 2,8,14: [{'group': 2, 'basis': 'YXXYX', 'mean_residual': 0.06681888238167338, 'sampling_se_of_mean': 0.005995794564769931, 'calibration_se_of_group_mean': 0.0006311596132580906}, {'group': 8, 'basis': 'YXYXX', 'mean_residual': 0.04365912176895423, 'sampling_se_of_mean': 0.004100167268386095, 'calibration_se_of_group_mean': 0.000739512975196128}, {'group': 14, 'basis': 'YYXXX', 'mean_residual': 0.03956748416861021, 'sampling_se_of_mean': 0.0036980472915268655, 'calibration_se_of_group_mean': 0.0005343874451132734}]
Top groups: [{'group': 2, 'basis': 'YXXYX', 'mean_residual': 0.06681888238167338, 'sampling_se_of_mean': 0.005995794564769931, 'calibration_se_of_group_mean': 0.0006311596132580906}, {'group': 16, 'basis': 'ZZZZZ', 'mean_residual': 0.052479077850352746, 'sampling_se_of_mean': 0.005785323288135165, 'calibration_se_of_group_mean': 0.0010823256240365129}, {'group': 8, 'basis': 'YXYXX', 'mean_residual': 0.04365912176895423, 'sampling_se_of_mean': 0.004100167268386095, 'calibration_se_of_group_mean': 0.000739512975196128}, {'group': 14, 'basis': 'YYXXX', 'mean_residual': 0.03956748416861021, 'sampling_se_of_mean': 0.0036980472915268655, 'calibration_se_of_group_mean': 0.0005343874451132734}, {'group': 3, 'basis': 'YXYYY', 'mean_residual': 0.023076566640997668, 'sampling_se_of_mean': 0.007777828511505737, 'calibration_se_of_group_mean': 0.0005491734597434004}, {'group': 10, 'basis': 'YYYYX', 'mean_residual': 0.010987444388631027, 'sampling_se_of_mean': 0.004113156626936459, 'calibration_se_of_group_mean': 0.00046505491358183687}, {'group': 9, 'basis': 'YYXYY', 'mean_residual': -0.006419537523837502, 'sampling_se_of_mean': 0.004070437236281636, 'calibration_se_of_group_mean': 0.0003585853670387483}, {'group': 1, 'basis': 'XYXXY', 'mean_residual': 0.004176301357466312, 'sampling_se_of_mean': 0.002294942045161018, 'calibration_se_of_group_mean': 0.0003219499512689186}, {'group': 15, 'basis': 'XXXXX', 'mean_residual': -0.0020266646094769056, 'sampling_se_of_mean': 0.0022507018189542273, 'calibration_se_of_group_mean': 0.00040366277843038254}, {'group': 0, 'basis': 'YXXXY', 'mean_residual': -0.0020054889299935263, 'sampling_se_of_mean': 0.0010246136670915217, 'calibration_se_of_group_mean': 0.00014677401585597695}]
{
"variant": "compiler_level3",
"state": "COBYLA",
"mode": "raw",
"available": true,
"E_exact": -0.5,
"E0_fd": -0.46097392795978287,
"classical_energy": -0.46097392795976344,
"exact_pauli_energy": -0.4609739279597633,
"exact_measurement_energy": -0.4609739279597637,

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
  "local_repetitions": [
    {
      "variant": "compiler_level3",
      "state": "COBYLA",
      "repetition": 1,
      "energy": -0.45513439905075764,
      "sampling_se": 0.01993212363937249,
      "difference_from_exact": 0.005839528909006075
    },
    {
      "variant": "compiler_level3",
      "state": "COBYLA",
      "repetition": 2,
      "energy": -0.4764224933442617,
      "sampling_se": 0.01959528587848649,
      "difference_from_exact": -0.015448565384497959
    }
  ],
  "block_energies": [
    -0.17053064026308828,
    -0.15013820974192565,
    -0.23339664625291257,
    -0.22896121063386432
  ],
  "block_sampling_standard_errors": [
    0.030320535819156152,
    0.03070476580997886,
    0.030288648618673066,
    0.030888341586211475
  ],
  "mean_energy": -0.19575667672294772,
  "sampling_standard_error_of_mean": 0.015275817115825395,
  "calibration_standard_error_of_mean": null,
  "between_block_standard_deviation": 0.04178000248775583,
  "between_block_standard_error_of_mean": 0.020890001243877916,
  "finite_representation_difference": 0.03902607204021713,
  "classical_variational_residual": 1.942890293094024e-14,
  "quantum_implementation_difference": -2.7755575615628914e-16,
  "hardware_difference": 0.265217251236816,
  "uncertainty_qualification": "Sampling, calibration, and between-block uncertainties are distinct. Between-block spread already includes stochastic variation; do not add it as another independent sampling variance. Systematic hardware error is not covered."
}
Residual cancellation: 0.265217251236816 0.28504732969146557 0.0695676696081089
Groups 2,8,14: [{'group': 2, 'basis': 'YXXYX', 'mean_residual': 0.07502172649136263, 'sampling_se_of_mean': 0.005956224039889831, 'calibration_se_of_group_mean': None}, {'group': 8, 'basis': 'YXYXX', 'mean_residual': 0.05768885486514522, 'sampling_se_of_mean': 0.004009187370892764, 'calibration_se_of_group_mean': None}, {'group': 14, 'basis': 'YYXXX', 'mean_residual': 0.04733087459992549, 'sampling_se_of_mean': 0.0036113291179722395, 'calibration_se_of_group_mean': None}]
Top groups: [{'group': 2, 'basis': 'YXXYX', 'mean_residual': 0.07502172649136263, 'sampling_se_of_mean': 0.005956224039889831, 'calibration_se_of_group_mean': None}, {'group': 8, 'basis': 'YXYXX', 'mean_residual':
```

0.05768885486514522, 'sampling_se_of_mean': 0.004009187370892764, 'calibration_se_of_group_mean': None}, {'group': 16, 'basis': 'ZZZZZ', 'mean_residual': 0.05118240901126789, 'sampling_se_of_mean': 0.005710623294834652, 'calibration_se_of_group_mean': None}, {'group': 14, 'basis': 'YYXXX', 'mean_residual': 0.04733087459992549, 'sampling_se_of_mean': 0.003613291179722395, 'calibration_se_of_group_mean': None}, {'group': 10, 'basis': 'YYYYX', 'mean_residual': 0.011457050991213157, 'sampling_se_of_mean': 0.003851229050008145, 'calibration_se_of_group_mean': None}, {'group': 3, 'basis': 'YXYYY', 'mean_residual': 0.008656449341859426, 'sampling_se_of_mean': 0.00747788162377026, 'calibration_se_of_group_mean': None}, {'group': 15, 'basis': 'XXXXX', 'mean_residual': -0.007219654458181801, 'sampling_se_of_mean': 0.002078163062370921, 'calibration_se_of_group_mean': None}, {'group': 4, 'basis': 'YYYXY', 'mean_residual': 0.006902088499146614, 'sampling_se_of_mean': 0.003869977525528741 8, 'calibration_se_of_group_mean': None}, {'group': 6, 'basis': 'XYXYX', 'mean_residual': 0.004344487653229788, 'sampling_se_of_mean': 0.002053731421379872, 'calibration_se_of_group_mean': None}, {'group': 12, 'basis': 'XXYYX', 'mean_residual': 0.0035500755779942187, 'sampling_se_of_mean': 0.002091691097664819, 'calibration_se_of_group_mean': None}]
{
 "variant": "compiler_level3",
 "state": "COBYLA",
 "mode": "mitigated",
 "available": true,
 "E_exact": -0.5,
 "E0_fd": -0.46097392795978287,
 "classical_energy": -0.46097392795976344,
 "exact_pauli_energy": -0.4609739279597633,
 "exact_measurement_energy": -0.4609739279597637,
 "local_repetitions": [
  {
   "variant": "compiler_level3",
   "state": "COBYLA",
   "repetition": 1,
   "energy": -0.45513439905075764,
   "sampling_se": 0.01993212363937249,
   "difference_from_exact": 0.005839528909006075
  },
  {
   "variant": "compiler_level3",
   "state": "COBYLA",
   "repetition": 2,
   "energy": -0.4764224933442617,
   "sampling_se": 0.01959528587848649,
   "difference_from_exact": -0.015448565384497959
  }
 ],
 "block_energies": [
  -0.18189396441489303,
  -0.2166175483327364,
  -0.25517585593451086,
  -0.2511364139768715
 ],
 "block_sampling_standard_errors": [
  0.031055355935870294,

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

0.03258364028476102,
0.03164895128899012,
0.03238674290930555
],
"mean_energy": -0.22620594566475294,
"sampling_standard_error_of_mean": 0.01596223273166347,
"calibration_standard_error_of_mean": 0.0024499375042469733,
"between_block_standard_deviation": 0.03423581456595773,
"between_block_standard_error_of_mean": 0.017117907282978864,
"finite_representation_difference": 0.03902607204021713,
"classical_variational_residual": 1.942890293094024e-14,
"quantum_implementation_difference": -2.7755575615628914e-16,
"hardware_difference": 0.23476798229501078,
"uncertainty_qualification": "Sampling, calibration, and between-block uncertainties are distinct. Between-block spread already includes stochastic variation; do not add it as another independent sampling variance. Systematic hardware error is not covered."
}
Residual cancellation: 0.2347679822950108 0.25275219662631127 0.07115354315946754
Groups 2,8,14: [{'group': 2, 'basis': 'YXXYX', 'mean_residual': 0.0685748648502289, 'sampling_se_of_mean': 0.006023785248573574, 'calibration_se_of_group_mean': 0.0006245155983872929}, {'group': 8, 'basis': 'YXYXX', 'mean_residual': 0.047017975716696664, 'sampling_se_of_mean': 0.004137514597193581, 'calibration_se_of_group_mean': 0.0007332834293406096}, {'group': 14, 'basis': 'YYXXX', 'mean_residual': 0.0457055100369071, 'sampling_se_of_mean': 0.0037768662208931035, 'calibration_se_of_group_mean': 0.0005311965109107301}]
Top groups: [{'group': 2, 'basis': 'YXXYX', 'mean_residual': 0.0685748648502289, 'sampling_se_of_mean': 0.006023785248573574, 'calibration_se_of_group_mean': 0.0006245155983872929}, {'group': 8, 'basis': 'YXYXX', 'mean_residual': 0.047017975716696664, 'sampling_se_of_mean': 0.004137514597193581, 'calibration_se_of_group_mean': 0.0007332834293406096}, {'group': 14, 'basis': 'YYXXX', 'mean_residual': 0.0457055100369071, 'sampling_se_of_mean': 0.0037768662208931035, 'calibration_se_of_group_mean': 0.0005311965109107301}, {'group': 16, 'basis': 'ZZZZZ', 'mean_residual': 0.04489429290987032, 'sampling_se_of_mean': 0.005916720801925411, 'calibration_se_of_group_mean': 0.0011132273446777892}, {'group': 10, 'basis': 'YYYYX', 'mean_residual': 0.00988439234172839, 'sampling_se_of_mean': 0.004070753196914623, 'calibration_se_of_group_mean': 0.0004555420444075000084}, {'group': 15, 'basis': 'XXXXX', 'mean_residual': -0.006008689171399339, 'sampling_se_of_mean': 0.002235895245243792, 'calibration_se_of_group_mean': 0.0004104481894668183}, {'group': 4, 'basis': 'YYYXY', 'mean_residual': 0.005974223053978274, 'sampling_se_of_mean': 0.0040633267380586514, 'calibration_se_of_group_mean': 0.0004179129813140349}, {'group': 6, 'basis': 'XYXYX', 'mean_residual': 0.004697570726307969, 'sampling_se_of_mean': 0.002243525432576218, 'calibration_se_of_group_mean': 0.00032085940494390507}, {'group': 12, 'basis': 'XXYYX', 'mean_residual': 0.0036850497714655643, 'sampling_se_of_mean': 0.002278550884299761, 'calibration_se_of_group_mean': 0.00031427181123284276}, {'group': 3, 'basis': 'YXYYY', 'mean_residual': 0.00344703339086767, 'sampling_se_of_mean': 0.0077678944199864935, 'calibration_se_of_group_mean': 0.0005758813508625509}]
{
"variant": "baseline",
"state": "SPSA",
"mode": "raw",
"available": true,
"E_exact": -0.5,
"E0_fd": -0.46097392795978287,
"classical_energy": -0.46077921338321903,
"exact_pauli_energy": -0.4607792133832186,

"exact_measurement_energy": -0.4607792133832206,
"local_repetitions": [
{
"variant": "baseline",
"state": "SPSA",
"repetition": 1,
"energy": -0.464978917439779,
"sampling_se": 0.019770508254946202,
"difference_from_exact": -0.004199704056558395
},
{
"variant": "baseline",
"state": "SPSA",
"repetition": 2,
"energy": -0.44486825307210265,
"sampling_se": 0.01997357916483739,
"difference_from_exact": 0.01591096031111794
}
],
"block_energies": [
-0.0587915018153925,
-0.2204023724970567,
-0.18428191339198818,
-0.18467350762659446
],
"block_sampling_standard_errors": [
0.031474073245634754,
0.03049017261742383,
0.031059495442736892,
0.030678503029607406
],
"mean_energy": -0.16203732383275796,
"sampling_standard_error_of_mean": 0.015463931255923628,
"calibration_standard_error_of_mean": null,
"between_block_standard_deviation": 0.07088346598710314,
"between_block_standard_error_of_mean": 0.03544173299355157,
"finite_representation_difference": 0.03902607204021713,
"classical_variational_residual": 0.00019471457656383695,
"quantum_implementation_difference": -1.5543122344752192e-15,
"hardware_difference": 0.29874188955046266,
"uncertainty_qualification": "Sampling, calibration, and between-block uncertainties are distinct. Between-block spread already includes stochastic variation; do not add it as another independent sampling variance. Systematic hardware error is not covered."
}
Residual cancellation: 0.29874188955046255 0.31264619891093565 0.0444729838677298
Groups 2,8,14: [{'group': 2, 'basis': 'YXXYX', 'mean_residual': 0.08064556625468694, 'sampling_se_of_mean': 0.006145114320153221, 'calibration_se_of_group_mean': None}, {'group': 8, 'basis': 'YXYXX', 'mean_residual': 0.06754094576712524, 'sampling_se_of_mean': 0.00426595870258336, 'calibration_se_of_group_mean': None}, {'group': 14, 'basis': 'YYXXX', 'mean_residual': 0.037220907632278546, 'sampling_se_of_mean': 0.003605872076749402, 'calibration_se_of_group_mean': None}]
Top groups: [{'group': 2, 'basis': 'YXXYX', 'mean_residual': 0.08064556625468694, 'sampling_se_of_mean':

0.006145114320153221, 'calibration_se_of_group_mean': None}, {'group': 8, 'basis': 'YXYXX', 'mean_residual': 0.06754094576712524, 'sampling_se_of_mean': 0.00426595870258336, 'calibration_se_of_group_mean': None}, {'group': 16, 'basis': 'ZZZZZ', 'mean_residual': 0.05062633451686516, 'sampling_se_of_mean': 0.005692244468048114, 'calibration_se_of_group_mean': None}, {'group': 3, 'basis': 'YXYYY', 'mean_residual': 0.037612140167181035, 'sampling_se_of_mean': 0.007534851800758758, 'calibration_se_of_group_mean': None}, {'group': 14, 'basis': 'YYXXX', 'mean_residual': 0.037220907632278546, 'sampling_se_of_mean': 0.003605872076749402, 'calibration_se_of_group_mean': None}, {'group': 10, 'basis': 'YYYYX', 'mean_residual': 0.009082667173906028, 'sampling_se_of_mean': 0.003838403989164187, 'calibration_se_of_group_mean': None}, {'group': 4, 'basis': 'YYYXY', 'mean_residual': 0.006223786742470666, 'sampling_se_of_mean': 0.0038683934062831195, 'calibration_se_of_group_mean': None}, {'group': 9, 'basis': 'YYXYY', 'mean_residual': -0.003994662002216664, 'sampling_se_of_mean': 0.003876809164796657, 'calibration_se_of_group_mean': None}, {'group': 1, 'basis': 'XYXXY', 'mean_residual': 0.003797094420267418, 'sampling_se_of_mean': 0.0021326418555555247, 'calibration_se_of_group_mean': None}, {'group': 12, 'basis': 'XXYYX', 'mean_residual': 0.003348070104252114, 'sampling_se_of_mean': 0.0020816934438894525, 'calibration_se_of_group_mean': None}]
{
 "variant": "baseline",
 "state": "SPSA",
 "mode": "mitigated",
 "available": true,
 "E_exact": -0.5,
 "E0_fd": -0.46097392795978287,
 "classical_energy": -0.46077921338321903,
 "exact_pauli_energy": -0.4607792133832186,
 "exact_measurement_energy": -0.4607792133832206,
 "local_repetitions": [
  {
   "variant": "baseline",
   "state": "SPSA",
   "repetition": 1,
   "energy": -0.464978917439779,
   "sampling_se": 0.019770508254946202,
   "difference_from_exact": -0.004199704056558395
  },
  {
   "variant": "baseline",
   "state": "SPSA",
   "repetition": 2,
   "energy": -0.44486825307210265,
   "sampling_se": 0.01997357916483739,
   "difference_from_exact": 0.01591096031111794
  }
 ],
 "block_energies": [
  -0.06929145803040915,
  -0.2889834752485734,
  -0.20584844226988375,
  -0.2038821043147762
 ],
 "block_sampling_standard_errors": [
  0.0322117537101092,

0.032401792753108234,
0.0324097783359621,
0.03218346356005146
],
"mean_energy": -0.1920013699659106,
"sampling_standard_error_of_mean": 0.01615093323387899,
"calibration_standard_error_of_mean": 0.002431783113910915,
"between_block_standard_deviation": 0.0909141435500839,
"between_block_standard_error_of_mean": 0.04545707177504195,
"finite_representation_difference": 0.03902607204021713,
"classical_variational_residual": 0.00019471457656383695,
"quantum_implementation_difference": -1.5543122344752192e-15,
"hardware_difference": 0.26877784341731,
"uncertainty_qualification": "Sampling, calibration, and between-block uncertainties are distinct. Between-block spread already includes stochastic variation; do not add it as another independent sampling variance. Systematic hardware error is not covered."
}
Residual cancellation: 0.26877784341730987 0.28344134511460595 0.05173381353862494
Groups 2,8,14: [{'group': 2, 'basis': 'YXXYX', 'mean_residual': 0.07398081000435386, 'sampling_se_of_mean': 0.006215594049696341, 'calibration_se_of_group_mean': 0.0006268506837494003}, {'group': 8, 'basis': 'YXYXX', 'mean_residual': 0.05691420501830935, 'sampling_se_of_mean': 0.004401439255722977, 'calibration_se_of_group_mean': 0.0007356691546293436}, {'group': 14, 'basis': 'YYXXX', 'mean_residual': 0.03511867591610311, 'sampling_se_of_mean': 0.0037689158091151905, 'calibration_se_of_group_mean': 0.0005386420750186987}]
Top groups: [{'group': 2, 'basis': 'YXXYX', 'mean_residual': 0.07398081000435386, 'sampling_se_of_mean': 0.006215594049696341, 'calibration_se_of_group_mean': 0.0006268506837494003}, {'group': 8, 'basis': 'YXYXX', 'mean_residual': 0.05691420501830935, 'sampling_se_of_mean': 0.004401439255722977, 'calibration_se_of_group_mean': 0.0007356691546293436}, {'group': 16, 'basis': 'ZZZZZ', 'mean_residual': 0.04355578061366941, 'sampling_se_of_mean': 0.005895726498433965, 'calibration_se_of_group_mean': 0.001130099199896268}, {'group': 14, 'basis': 'YYXXX', 'mean_residual': 0.03511867591610311, 'sampling_se_of_mean': 0.0037689158091151905, 'calibration_se_of_group_mean': 0.0005386420750186987}, {'group': 3, 'basis': 'YXYYY', 'mean_residual': 0.03432822457170992, 'sampling_se_of_mean': 0.007819594494123132, 'calibration_se_of_group_mean': 0.0005423437113482449}, {'group': 10, 'basis': 'YYYYX', 'mean_residual': 0.007487226269308455, 'sampling_se_of_mean': 0.004058342273889408, 'calibration_se_of_group_mean': 0.0004517898015308386 3}, {'group': 4, 'basis': 'YYYXY', 'mean_residual': 0.004916820368240948, 'sampling_se_of_mean': 0.004062917693309629, 'calibration_se_of_group_mean': 0.0004121165607029570 7}, {'group': 1, 'basis': 'XYXXY', 'mean_residual': 0.004204044747275145, 'sampling_se_of_mean': 0.0023144035250727154, 'calibration_se_of_group_mean': 0.0003200537124416088 5}, {'group': 9, 'basis': 'YYXYY', 'mean_residual': -0.0041059297710849835, 'sampling_se_of_mean': 0.004092494629945439, 'calibration_se_of_group_mean': 0.0003577546979574547}, {'group': 12, 'basis': 'XXYYX', 'mean_residual': 0.00335296553710812, 'sampling_se_of_mean': 0.0022706630628417726, 'calibration_se_of_group_mean': 0.0003117714998485897 3}]
{
"variant": "compiler_level3",
"state": "SPSA",
"mode": "raw",
"available": true,
"E_exact": -0.5,
"E0_fd": -0.46097392795978287,
"classical_energy": -0.46077921338321903,
"exact_pauli_energy": -0.4607792133832186,

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
  "exact_measurement_energy": -0.4607792133832206,
  "local_repetitions": [
    {
      "variant": "compiler_level3",
      "state": "SPSA",
      "repetition": 1,
      "energy": -0.4482413864135743,
      "sampling_se": 0.02003986623149179,
      "difference_from_exact": 0.012537826969646282
    },
    {
      "variant": "compiler_level3",
      "state": "SPSA",
      "repetition": 2,
      "energy": -0.45469513666062134,
      "sampling_se": 0.01976262101531043,
      "difference_from_exact": 0.006084076722599252
    }
  ],
  "block_energies": [
    -0.24851362258897647,
    -0.20402215468768392,
    -0.3192032110076405,
    -0.20893093422673686
  ],
  "block_sampling_standard_errors": [
    0.03047438223902361,
    0.03048755485568827,
    0.030103821350548805,
    0.030372223526051737
  ],
  "mean_energy": -0.24516748062775945,
  "sampling_standard_error_of_mean": 0.01517994358469204,
  "calibration_standard_error_of_mean": null,
  "between_block_standard_deviation": 0.05322440286402335,
  "between_block_standard_error_of_mean": 0.026612201432011675,
  "finite_representation_difference": 0.03902607204021713,
  "classical_variational_residual": 0.00019471457656383695,
  "quantum_implementation_difference": -1.5543122344752192e-15,
  "hardware_difference": 0.21561173275546114,
  "uncertainty_qualification": "Sampling, calibration, and between-block uncertainties are distinct. Between-block spread already includes stochastic variation; do not add it as another independent sampling variance. Systematic hardware error is not covered."
}
Residual cancellation: 0.21561173275546103 0.23340891614026046 0.07624894403821625
Groups 2,8,14: [{'group': 2, 'basis': 'YXXYX', 'mean_residual': 0.062491688996386185, 'sampling_se_of_mean': 0.005709671310966311, 'calibration_se_of_group_mean': None}, {'group': 8, 'basis': 'YXYXX', 'mean_residual': 0.05379052615042601, 'sampling_se_of_mean': 0.004021879237949935, 'calibration_se_of_group_mean': None}, {'group': 14, 'basis': 'YYXXX', 'mean_residual': 0.044968500558303925, 'sampling_se_of_mean': 0.003615068310390414, 'calibration_se_of_group_mean': None}]
Top groups: [{'group': 2, 'basis': 'YXXYX', 'mean_residual': 0.062491688996386185, 'sampling_se_of_mean':
```

0.005709671310966311, 'calibration_se_of_group_mean': None}, {'group': 8, 'basis': 'YXYXX', 'mean_residual': 0.05379052615042601, 'sampling_se_of_mean': 0.004021879237949935, 'calibration_se_of_group_mean': None}, {'group': 14, 'basis': 'YYXXX', 'mean_residual': 0.044968500558303925, 'sampling_se_of_mean': 0.003615068310390414, 'calibration_se_of_group_mean': None}, {'group': 16, 'basis': 'ZZZZZ', 'mean_residual': 0.04358044819915846, 'sampling_se_of_mean': 0.005661016459065476, 'calibration_se_of_group_mean': None}, {'group': 9, 'basis': 'YYXYY', 'mean_residual': -0.007484194472919787, 'sampling_se_of_mean': 0.0038642586269960854, 'calibration_se_of_group_mean': None}, {'group': 10, 'basis': 'YYYYX', 'mean_residual': 0.0059047001023728235, 'sampling_se_of_mean': 0.0038474499636219736, 'calibration_se_of_group_mean': None}, {'group': 12, 'basis': 'XXYYX', 'mean_residual': 0.003119588811527504, 'sampling_se_of_mean': 0.002083686798356564, 'calibration_se_of_group_mean': None}, {'group': 13, 'basis': 'XYYXX', 'mean_residual': 0.002746532740884082, 'sampling_se_of_mean': 0.002110011726715656, 'calibration_se_of_group_mean': None}, {'group': 6, 'basis': 'XYXYX', 'mean_residual': 0.0025468293739820853, 'sampling_se_of_mean': 0.002131665705325002, 'calibration_se_of_group_mean': None}, {'group': 15, 'basis': 'XXXXX', 'mean_residual': 0.002464292351138651, 'sampling_se_of_mean': 0.002082074755039789, 'calibration_se_of_group_mean': None}]
{
 "variant": "compiler_level3",
 "state": "SPSA",
 "mode": "mitigated",
 "available": true,
 "E_exact": -0.5,
 "E0_fd": -0.46097392795978287,
 "classical_energy": -0.46077921338321903,
 "exact_pauli_energy": -0.4607792133832186,
 "exact_measurement_energy": -0.4607792133832206,
 "local_repetitions": [
  {
   "variant": "compiler_level3",
   "state": "SPSA",
   "repetition": 1,
   "energy": -0.4482413864135743,
   "sampling_se": 0.02003986623149179,
   "difference_from_exact": 0.012537826969646282
  },
  {
   "variant": "compiler_level3",
   "state": "SPSA",
   "repetition": 2,
   "energy": -0.45469513666062134,
   "sampling_se": 0.01976262101531043,
   "difference_from_exact": 0.006084076722599252
  }
 ],
 "block_energies": [
  -0.26134851363022993,
  -0.2705838659530507,
  -0.34672524034704044,
  -0.23134055070266019
 ],
 "block_sampling_standard_errors": [
  0.0312241114064849,

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

0.03238719428610269,
0.03148574592900895,
0.031911975040415944
],
"mean_energy": -0.27749954265824534,
"sampling_standard_error_of_mean": 0.015877660868608326,
"calibration_standard_error_of_mean": 0.00244374821495783,
"between_block_standard_deviation": 0.049096951982143776,
"between_block_standard_error_of_mean": 0.024548475991071888,
"finite_representation_difference": 0.03902607204021713,
"classical_variational_residual": 0.00019471457656383695,
"quantum_implementation_difference": -1.5543122344752192e-15,
"hardware_difference": 0.18327967072497525,
"uncertainty_qualification": "Sampling, calibration, and between-block uncertainties are distinct. Between-block spread already includes stochastic variation; do not add it as another independent sampling variance. Systematic hardware error is not covered."
}
Residual cancellation: 0.1832796707249753 0.21352881760006862 0.1416630655059815
Groups 2,8,14: [{'group': 2, 'basis': 'YXXYX', 'mean_residual': 0.055571553085146, 'sampling_se_of_mean': 0.005777931263099405, 'calibration_se_of_group_mean': 0.000633926029085782}, {'group': 8, 'basis': 'YXYXX', 'mean_residual': 0.04312519518725394, 'sampling_se_of_mean': 0.004156817904452863, 'calibration_se_of_group_mean': 0.0007341200394433359}, {'group': 14, 'basis': 'YYXXX', 'mean_residual': 0.043532104232413965, 'sampling_se_of_mean': 0.0037810364486264235, 'calibration_se_of_group_mean': 0.0005302086326745006}]
Top groups: [{'group': 2, 'basis': 'YXXYX', 'mean_residual': 0.055571553085146, 'sampling_se_of_mean': 0.005777931263099405, 'calibration_se_of_group_mean': 0.000633926029085782}, {'group': 14, 'basis': 'YYXXX', 'mean_residual': 0.043532104232413965, 'sampling_se_of_mean': 0.0037810364486264235, 'calibration_se_of_group_mean': 0.0005302086326745006}, {'group': 8, 'basis': 'YXYXX', 'mean_residual': 0.04312519518725394, 'sampling_se_of_mean': 0.004156817904452863, 'calibration_se_of_group_mean': 0.0007341200394433359}, {'group': 16, 'basis': 'ZZZZZ', 'mean_residual': 0.036503275273372326, 'sampling_se_of_mean': 0.005864274099070767, 'calibration_se_of_group_mean': 0.0011330923081842856}, {'group': 9, 'basis': 'YYXYY', 'mean_residual': -0.008110291580347082, 'sampling_se_of_mean': 0.004080684759017046, 'calibration_se_of_group_mean': 0.0003560386790099163 6}, {'group': 3, 'basis': 'YXYYY', 'mean_residual': -0.005093137385383448, 'sampling_se_of_mean': 0.007745797671117758, 'calibration_se_of_group_mean': 0.0005673336512753718}, {'group': 10, 'basis': 'YYYYX', 'mean_residual': 0.004208123193678075, 'sampling_se_of_mean': 0.0040688156483467105, 'calibration_se_of_group_mean': 0.0004527786544676511}, {'group': 15, 'basis': 'XXXXX', 'mean_residual': 0.004120329290853076, 'sampling_se_of_mean': 0.002242290364218988, 'calibration_se_of_group_mean': 0.000411984997280407}, {'group': 13, 'basis': 'XYYXX', 'mean_residual': 0.0033163113764961153, 'sampling_se_of_mean': 0.002291507210785728 6, 'calibration_se_of_group_mean': 0.00040081503580709845}, {'group': 12, 'basis': 'XXYYX', 'mean_residual': 0.002999920055400798, 'sampling_se_of_mean': 0.002275423514091953, 'calibration_se_of_group_mean': 0.0003141183234099633}]

=== PAIRED NEW-MEASUREMENT COMPARISONS ===
{
"state": "COBYLA",
"mode": "raw",
"candidate": "compiler_level3",
"paired_block_differences": [
0.05072588227806796,
0.020420385157922716,

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    0.0040899568725706326,
    -0.05684946370610516
  ],
  "mean_candidate_minus_baseline": 0.004596690150614036,
  "conditional_sampling_se": 0.02157346948292701,
  "shared_calibration_se": null,
  "between_block_se_of_paired_difference": 0.022646151793065134,
  "descriptive_95_percent_paired_t_interval": [
    -0.06747347195213269,
    0.07666685225336076
  ],
  "absolute_residual_reduction_point_estimate": -0.004596690150614036,
  "interpretation": "A positive residual-reduction point estimate is not an automatic improvement claim. Four blocks give limited drift inference. Compiled-cost reductions do not establish hardware-energy improvement."
}
{
  "state": "COBYLA",
  "mode": "mitigated",
  "candidate": "compiler_level3",
  "paired_block_differences": [
    0.052680353922160916,
    0.022931103855283186,
    0.0028678025075319358,
    -0.06196675775995841
  ],
  "mean_candidate_minus_baseline": 0.004128125631254406,
  "conditional_sampling_se": 0.022543569879287215,
  "shared_calibration_se": 0.00032907168008095196,
  "between_block_se_of_paired_difference": 0.02429161873119309,
  "descriptive_95_percent_paired_t_interval": [
    -0.07317864664920504,
    0.08143489791171385
  ],
  "absolute_residual_reduction_point_estimate": -0.004128125631254392,
  "interpretation": "A positive residual-reduction point estimate is not an automatic improvement claim. Four blocks give limited drift inference. Compiled-cost reductions do not establish hardware-energy improvement."
}
{
  "state": "SPSA",
  "mode": "raw",
  "candidate": "compiler_level3",
  "paired_block_differences": [
    -0.18972212077358397,
    0.016380217809372766,
    -0.1349212976156523,
    -0.024257426600142407
  ],
  "mean_candidate_minus_baseline": -0.08313015679500148,
  "conditional_sampling_se": 0.021669422168631186,
  "shared_calibration_se": null,
  "between_block_se_of_paired_difference": 0.047795120102035064,
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
  "descriptive_95_percent_paired_t_interval": [
    -0.23523556017434055,
    0.06897524658433761
  ],
  "absolute_residual_reduction_point_estimate": 0.08313015679500152,
  "interpretation": "A positive residual-reduction point estimate is not an automatic improvement claim. Four blocks give limited drift inference. Compiled-cost reductions do not establish hardware-energy improvement."
}
{
  "state": "SPSA",
  "mode": "mitigated",
  "candidate": "compiler_level3",
  "paired_block_differences": [
    -0.19205705559982078,
    0.0183996092955227,
    -0.1408767980771567,
    -0.027458446387883972
  ],
  "mean_candidate_minus_baseline": -0.08549817269233469,
  "conditional_sampling_se": 0.022648460410892236,
  "shared_calibration_se": 0.00024829705949509437,
  "between_block_se_of_paired_difference": 0.04880654550322091,
  "descriptive_95_percent_paired_t_interval": [
    -0.24082238310274834,
    0.06982603771807895
  ],
  "absolute_residual_reduction_point_estimate": 0.08549817269233473,
  "interpretation": "A positive residual-reduction point estimate is not an automatic improvement claim. Four blocks give limited drift inference. Compiled-cost reductions do not establish hardware-energy improvement."
}

Job IDs: ['db7rvtmr6oqs73886pfg', 'db7s212raffc73fb627g', 'db7s546r6oqs73887590', 'db7s6ber6oqs73887870', 'db7s6o6r6oqs73887970', 'db7s72ur6oqs73887a20', 'db7s788r1c5c738l6qfg', 'db7sejqraffc73fb6vb0', 'db7slluil99s73foq4ng', 'db7slqiraffc73fb7e3g', 'db7sm2ur6oqs73888alg', 'db7sm9uil99s73foq60g', 'db7smdaraffc73fb7fb0', 'db7smi6r6oqs73888bog', 'db7smn6r6oqs73888c6g', 'db7sms6il99s73foq760', 'db7sn36il99s73foq7gg', 'db7sna2raffc73fb7hd0', 'db7snl0r1c5c738l7v50', 'db7snq6il99s73foq94g', 'db7so0or1c5c738l7vsg', 'db7soomr6oqs73888gkg', 'db7sotmil99s73foqbig', 'db7sp4gr1c5c738l829g']
Evidence: /content/hydrogen_marrakesh_20261014T174449_879279Z_97381a6a
No readout-causation claim is made. Assignment calibration includes preparation error; transferability, drift, and gate error remain possible systematic limitations.
```


Fourth-run conclusion

Run 4 tested the ten-entangling-gate baseline against compiler level 3 on the **ibm_marrakesh** backend, using four execution blocks, separate readout training and held-out validation, 24 jobs, and 753,664 requested shots. The submitted circuits passed the reported ideal checks; two optional candidates that failed the tolerance were excluded. Compiler level 3 reduced mean depth by approximately 2.75% without reducing the ten two-qubit gates, but paired intervals spanning zero did not establish a reproducible energy benefit. Readout correction shifted the observed means toward their references by about 0.030–0.032 Ha, leaving discrepancies of 0.183–0.269 Ha. This is a modest correction in this dataset, not proof that readout error explains the full physical discrepancy. The change from the **ibm_fez** to the **ibm_marrakesh** backend prevents attributing cross-run differences solely to compilation.

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

The failure to establish a general level-3 advantage is consistent with the distinction between compiler cost and physical accuracy. Native-gate count, depth, and target-derived timing proxies summarize resources; they do not fully describe coherent errors, relaxation, readout, or run-to-run variation [29, 30]. Likewise, fewer single-qubit operations need not compensate for an unfavorable physical placement. The study reports the observed comparisons rather than ranking devices or compiler settings universally.

Prompt sent to Quantum Flow for a further hardware-accuracy revision

Status of this extension: The following request led to the Run 5 implementation and partial execution recorded below. Approximate noisy density-matrix selection, randomized compiling, and folded circuits are present in that implementation; the large hardware experiment was cancelled before a complete result was obtained. Run 6 subsequently completed a smaller screening design. Implemented methods, acquired measurements, and accepted mitigated estimates are distinguished throughout.

Implement the next hardware-accuracy revision of the attached Part II. Return one complete executable Jupyter cell, not another audit, proposal, or collection of partial fixes. The objective is to reproduce the fixed Part I state energies on a real QPU with materially smaller error and defensible uncertainty.

1. Keep Part I authoritative and immutable

Read only these existing Part I definitions:

H, r, h, n_interior, R_max, E_exact, E0_fd, alpha_cobyla, alpha_spsa, normalized_trial_state.

Preserve the exact Hamiltonian, 32-dimensional representation, five-qubit encoding, amplitude ordering, normalization, and both full-precision optimized parameters. Generate each target vector through the supplied normalized_trial_state function. Calculate its reference energy independently as real(vdot(psi, H @ psi)); do not substitute rounded constants.

The intended fixed-state energies are approximately −0.460973927960 Ha and −0.460779213383 Ha, not −0.5 Ha. Keep continuum/discretization error separate from hardware error. Do not introduce a new VQE loop, change the ansatz, truncate the state, remove Hamiltonian terms, infer unmeasured coherences from ideal amplitudes, or fit corrections to the reference energies.

Retain the exact bond-dimension-two sequential preparation and the complete 63-term, 17-group measurement workflow. Keep the current ten-two-qubit-gate implementation as the experimental baseline.

2. Replace the placement policy with circuit-aware selection

Keep ibm_marrakesh as the explicit backend unless I change the configuration. Never substitute another backend automatically.

Replace qubit_cost() and the single-path selection with a bounded search over connected five-qubit paths and both path orientations. Evaluate a shortlist using the actual compiled and scheduled circuits, not averages over every operation exposed by the backend.

Use the error and duration of each native instruction actually executed on its actual physical qubits, together with current \(T_1/T_2\), readout characteristics, idle intervals, and calibration timestamps. Do not count virtual zero-duration rotations as equivalent to physical pulses. Report missing calibration data explicitly.

Test both directions of the sequential preparation while preserving the original logical basis and measurement mapping. Compare ASAP and ALAP scheduling using supported passes and backend timing constraints. Report each qubit’s premeasurement active and idle exposure, separately from readout duration. A low-\(T_2\) qubit such as the previously selected qubit 117 must be assessed through these calculations, not automatically reused or permanently blacklisted.

Add compact, calibration-informed noisy density-matrix simulation of the active circuit, including gate errors, relaxation/dephasing during relevant operations and delays, and readout error without double-counting noise. Preserve the existing noiseless checks separately. Use these approximate simulations to compare candidate configurations and test sensitivity to calibration uncertainty—not to manufacture hardware results or claim an exact device model.

Select one candidate configuration before hardware execution using predefined criteria that consider both states and group-level errors, not just total-energy cancellation or circuit depth.

3. Implement error suppression and gate-noise mitigation

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

Implement explicit, reproducible Pauli-randomized compilations around the native entangling gates. Start with 16 randomizations per measurement circuit, splitting the allocated shots among them rather than multiplying the shot budget. Track all Pauli-frame corrections and validate every resulting circuit against its intended classical-output probabilities.

Implement scheduled dynamical decoupling only in eligible idle intervals, with valid pulse spacing and backend alignment. Compare its added pulse cost and predicted benefit before including it in the selected candidate. Do not insert pulses indiscriminately or assume they always help.

Prefer explicit client-side circuit construction so the actual randomized and scheduled circuits can be validated, hashed, saved, and submitted. Disable duplicate server-side transformations. Preserve an unsuppressed baseline.

Also implement an explicitly labelled zero-noise extrapolation experiment for the selected candidate, using nominal entangling-gate folding factors 1, 3, and 5. Construct folds from the actual native entanglers and their inverses. Prevent subsequent optimization from cancelling the inserted identities. Record the realized gate counts, durations, and scaling factors; validate ideal equivalence after every transformation.

Implement this through the existing counts-based Sampler workflow; do not invent Estimator-only resilience options for Sampler. Keep randomization, scheduling policy, measurement decoding, and calibration handling consistent across noise factors.

Predefine the extrapolation model before viewing hardware results. Report every measured noise-factor energy, fit residuals, uncertainty amplification, and sensitivity to the fit model. Mark extrapolation unreliable or unavailable when the data do not support it. Entangling-gate folding does not scale every physical error equally; do not describe its intercept as proven fully noiseless execution.

4. Preserve and strengthen readout correction

Retain independently acquired assignment calibration and held-out validation on every physical readout set actually used. Export the matrices, conditioning statistics, numerical validation errors, and acceptance thresholds. Reject invalid corrections without discarding the raw science measurements. Do not use clipping, postselection, reference-dependent offsets, or reference-driven model selection.

Propagate science-shot, shared-calibration, randomization, and extrapolation uncertainties with their covariance. Include paired raw-versus-readout-corrected comparisons because they reuse the same science counts. Preserve between-block variation separately rather than adding it twice.

Investigate the previously observed large block-2 correction through the actual calibration data; do not assume either drift or a decoder defect.

5. Run one focused, budgeted comparison

Compare the unchanged baseline against the single selected hardware-aware candidate in at least four balanced, interleaved blocks, with noise factors distributed throughout each block rather than executed in separate long batches.

Derive all reported processing modes from their corresponding measured data: baseline raw, candidate raw at factor 1, readout-corrected estimates, and readout-corrected plus extrapolated estimates. These are different processing levels—not interchangeable evidence of improved physical state preparation.

Keep the complete workload within 1,114,112 shots, including all randomizations, noise factors, calibration, held-out validation, and diagnostics. Use covariance-aware group shot allocation with a minimum allocation for every nonconstant group; freeze the allocation before the confirmation run. Do not increase shots uniformly as a substitute for addressing bias.

Print the complete budget before submission. If the proposed configuration cannot fit, reduce exploratory breadth or stop for authorization; do not silently exceed the cap. Preserve successful jobs and support retrieval/resumption without duplicate submissions.

6. Define success before executing

Use a configurable initial engineering target of 0.05 Ha absolute error for each fixed state. This is a proposed milestone, not a claim of chemical accuracy or guaranteed hardware performance.

Report the target as achieved only when the prespecified final estimator’s statistical 95% interval lies entirely inside the corresponding reference energy ±0.05 Ha, the calibration/extrapolation validity checks pass, and the result is supported across the confirmation blocks. State the interval’s assumptions and any unquantified systematic limitations.

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

Report improvement over baseline separately from reaching this accuracy target. Do not select different processing methods retrospectively for COBYLA and SPSA simply because they give preferable answers.
Keep group residuals and cancellation diagnostics, especially groups 2, 8, 14, and 16. Preserve negative controls, independent local random streams, approximation_degree=1.0, the existing \(10^{-10}\) compiled-probability tolerance, circuit reuse, source/circuit hashes, raw counts, calibration records, and job associations.
Deliver the complete implementation now, with required dependencies and a short explanation of the actual changes. Run all available local checks before requesting explicit authorization of the backend and complete QPU workload. Report a missed target honestly; never force agreement with Part I.

Fifth code version (verbatim)

```
# Initial code produced by Stellora.AI Quantum Flow
#
# Required: qiskit==2.5.2, qiskit-ibm-runtime==0.50.0, numpy, scipy.
#
# Changes:
# - Circuit-aware placement, orientation, preparation-direction, ASAP/ALAP,
#   and eligible-idle DD comparison using actual compiled/scheduled circuits.
# - Explicit native-gate Pauli randomization and entangler folding 1/3/5.
# - Counts-only readout correction, shared-calibration covariance, independent
#   local validation, component-wise uncertainty, and prespecified ZNE checks.
# - Frozen shot allocation, four interleaved blocks, immutable circuit evidence,
#   and working explicit resumption without automatic duplicate submissions.
#
# No hardware improvement or successful execution is claimed in advance.
# The supplied source permits reconstruction of the sequential method, but
# historical circuit bytes are unavailable. Historical 10/46 counts are context.
# A changed backend/native gate set cannot honestly be forced to reproduce 10.
#
# Fresh execution needs ONLY the ten permitted Part I definitions.
# To resume THIS implementation's saved experiment, change ONLY RESUME_DIR.
RESUME_DIR = None


def _run_hydrogen_accuracy_revision(
    H, r, h, n_interior, R_max, E_exact, E0_fd,
    alpha_cobyla, alpha_spsa, normalized_trial_state, resume_dir,
):
    import os
    import io
    import json
    import math
    import uuid
    import pickle
    import hashlib
    import getpass
    import itertools
    import warnings
    from pathlib import Path
    from datetime import datetime, timezone
    from collections import Counter, defaultdict
    from importlib.metadata import version
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
import numpy as np
import scipy
from scipy.stats import chi2
import qiskit
from IPython import get_ipython

from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister, qpy
from qiskit.circuit.library import UnitaryGate, XGate
from qiskit.quantum_info import Operator, SparsePauliOp, Statevector
from qiskit.primitives import StatevectorSampler
from qiskit.primitives.containers import SamplerPub
from qiskit.transpiler import PassManager
from qiskit.transpiler.preset_passmanagers import (
    generate_preset_pass_manager,
)
from qiskit.transpiler.passes import (
    TimeUnitConversion, ASAPScheduleAnalysis, ALAPScheduleAnalysis,
    ConstrainedReschedule, PadDelay, PadDynamicalDecoupling,
)
from qiskit_ibm_runtime import QiskitRuntimeService, SamplerV2
from qiskit_ibm_runtime.accounts.exceptions import AccountNotFoundError

# All configuration is defined here; no previous Part II symbols are read.
CFG = {
    "backend": "ibm_marrakesh",
    "channel": "ibm_quantum_platform",
    "register": "hydrogen",
    "blocks": 4,
    "randomizations": 16,
    "shot_cap": 1_114_112,
    "engineering_target_Ha": 0.05,
    "seed": 392781,
    "transpiler_seed": 731,
    "approximation_degree": 1.0,
    "matrix_tolerance": 1e-12,
    "probability_tolerance": 1e-10,
    "energy_tolerance": 1e-10,
    "Schmidt_tail_tolerance": 1e-12,
    "path_limit": 32,
    "path_shortlist": 2,
    "path_beam_width": 3000,
    "calibration_shots_per_state_per_epoch": 1024,
    "heldout_shots_per_state_per_block": 128,
    "diagnostic_shots": 256,
    "local_shots": 4096,
    "bootstrap_replicates": 600,
    "job_circuit_limit": 64,
    "readout_condition_limit": 10.0,
    "readout_minimum_singular_value": 0.10,
    "heldout_TV_limit": 0.15,
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    "heldout_block_TV_limit": 0.30,
    "corrected_heldout_TV_limit": 0.30,
    "ZNE_lack_of_fit_p_min": 0.01,
    "ZNE_amplification_limit": 5.0,
    "model_sensitivity_floor_Ha": 0.025,
    "block_heterogeneity_p_min": 0.01,
}
NQ, DIM = 5, 32
NAMES = ["COBYLA", "SPSA"]
B = CFG["blocks"]
R = CFG["randomizations"]
TOL = CFG["probability_tolerance"]
MTOL = CFG["matrix_tolerance"]
ETOL = CFG["energy_tolerance"]
LINEAR = np.array([13 / 12, 1 / 3, -5 / 12], dtype=float)
QUADRATIC = np.array([15 / 8, -5 / 4, 3 / 8], dtype=float)

def require(ok, message):
    if not bool(ok):
        raise RuntimeError(message)

def utc():
    return datetime.now(timezone.utc).isoformat()

def sha(payload):
    return hashlib.sha256(payload).hexdigest()

def array_hash(value):
    a = np.ascontiguousarray(value)
    return sha(str(a.dtype).encode() + str(a.shape).encode() + a.tobytes())

def js(value):
    if isinstance(value, np.ndarray):
        return js(value.tolist())
    if isinstance(value, np.generic):
        return js(value.item())
    if isinstance(value, complex):
        return {"real": value.real, "imag": value.imag}
    if isinstance(value, dict):
        return {str(k): js(v) for k, v in value.items()}
    if isinstance(value, (list, tuple)):
        return [js(v) for v in value]
    if isinstance(value, (datetime, Path)):
        return str(value)
    if isinstance(value, float) and not math.isfinite(value):
        return None
    if value is None or isinstance(value, (bool, str, int, float)):
        return value
    return repr(value)

def real(value, label):
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    z = complex(value)
    require(
        np.isfinite(z.real) and np.isfinite(z.imag) and abs(z.imag) <= MTOL,
        f"{label}: nonfinite or unexpectedly complex value.",
    )
    return float(z.real)

def rng(*coordinates):
    return np.random.default_rng(
        np.random.SeedSequence([CFG["seed"], *map(int, coordinates)])
    )

# -------------------- Authoritative Part I reconstruction -----------------
Hc = np.array(H, dtype=complex, copy=True)
rc = np.array(r, dtype=float, copy=True)
require(Hc.shape == (32, 32), "H must have shape (32,32).")
require(rc.shape == (32,), "r must contain exactly 32 elements.")
require(np.all(np.isfinite(Hc)), "H has nonfinite entries.")
require(np.all(np.isfinite(rc)), "r has nonfinite entries.")
require(np.max(np.abs(Hc - Hc.conj().T)) <= MTOL, "H is not Hermitian.")
require(int(n_interior) == 32, "n_interior must be 32.")
require(np.isfinite(h) and float(h) > 0, "Invalid h.")
require(np.isfinite(R_max) and float(R_max) > 0, "Invalid R_max.")
require(callable(normalized_trial_state), "Invalid trial-state function.")

alpha = [alpha_cobyla, alpha_spsa]
vectors, references = [], []
for s in range(2):
    require(
        np.ndim(alpha[s]) == 0 and np.isfinite(alpha[s]) and alpha[s] > 0,
        f"{NAMES[s]} alpha must be finite and positive.",
    )
    v = np.array(normalized_trial_state(alpha[s], rc.copy()), complex, copy=True)
    require(v.shape == (32,) and np.all(np.isfinite(v)), "Invalid target vector.")
    require(abs(real(np.vdot(v, v), "Norm") - 1) <= MTOL, "Unnormalized vector.")
    vectors.append(v)
    references.append(real(np.vdot(v, Hc @ v), "Classical target energy"))
references = np.array(references)
continuum = real(E_exact, "E_exact")
fd_reference = real(E0_fd, "E0_fd")

versions = {
    "numpy": np.__version__, "scipy": scipy.__version__,
    "qiskit": qiskit.__version__,
    "qiskit-ibm-runtime": version("qiskit-ibm-runtime"),
}
require(versions["qiskit"] == "2.5.2", "Requires Qiskit 2.5.2.")
require(
    versions["qiskit-ibm-runtime"] == "0.50.0",
    "Requires qiskit-ibm-runtime 0.50.0.",
)
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
shell = get_ipython()
require(shell is not None, "Run as one complete Jupyter cell.")
source = shell.history_manager.input_hist_raw[-1]
require(
    source.lstrip().startswith("# Initial code produced by Stellora.AI Quantum Flow"),
    "Cannot identify this current complete cell's source.",
)

def canonical_source(text):
    # Only this explicit resumption configuration line may change.
    return "\n".join(
        "RESUME_DIR = <EXPLICIT_RESUME_DIRECTORY>"
        if line.startswith("RESUME_DIR = ") else line
        for line in text.splitlines()
    )

source_hash = sha(source.encode())
canonical_hash = sha(canonical_source(source).encode())
authority = {
    "H": array_hash(Hc), "r": array_hash(rc),
    "vectors": [array_hash(v) for v in vectors],
    "alpha": [float(a).hex() for a in alpha],
    "h": float(h).hex(), "n_interior": int(n_interior),
    "R_max": float(R_max).hex(),
    "E_exact": continuum.hex(), "E0_fd": fd_reference.hex(),
}
authority_hash = sha(json.dumps(authority, sort_keys=True).encode())

if resume_dir is None:
    directory = Path.cwd() / (
        "hydrogen_accuracy_" +
        datetime.now(timezone.utc).strftime("%Y%m%dT%H%M%S_%fZ") +
        "_" + uuid.uuid4().hex[:8]
    )
    directory.mkdir(mode=0o700, exist_ok=False)
    (directory / "circuits").mkdir(mode=0o700)
    (directory / "jobs").mkdir(mode=0o700)
else:
    directory = Path(resume_dir).expanduser().resolve()
    require((directory / "manifest.json").exists(), "Resume manifest missing.")

def save_json(relative, value):
    path = directory / relative
    temporary = path.with_name("." + path.name + "." + uuid.uuid4().hex + ".tmp")
    with temporary.open("x", encoding="utf-8") as handle:
        os.chmod(temporary, 0o600)
        json.dump(js(value), handle, indent=2, allow_nan=False)
        handle.flush()
        os.fsync(handle.fileno())
    os.replace(temporary, path)
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
def save_binary(relative, payload):
    path = directory / relative
    with path.open("xb") as handle:
        os.chmod(path, 0o600)
        handle.write(payload)
        handle.flush()
        os.fsync(handle.fileno())
    return sha(payload)

if resume_dir is None:
    save_binary("source.py", source.encode())
    save_json("authority.json", {
        "authority": authority, "authority_hash": authority_hash,
        "references": references, "continuum": continuum,
        "finite_reference": fd_reference,
    })
    np.savez_compressed(
        directory / "inputs.npz", H=Hc, r=rc,
        COBYLA=vectors[0], SPSA=vectors[1],
    )
    os.chmod(directory / "inputs.npz", 0o600)
    manifest = {
        "schema": 3, "created_utc": utc(), "configuration": CFG,
        "versions": versions, "authority_hash": authority_hash,
        "source_sha256": source_hash,
        "canonical_source_sha256": canonical_hash,
        "status": "building", "registry": {}, "plan": [], "jobs": [],
        "baseline_provenance": (
            "Reconstructed from the supplied complex-SVD sequential source: "
            "forward direction, level 2, seed 731, no RC or DD. Historical "
            "native count 10 versus generic 46 is static context; original "
            "artifact identity is not asserted."
        ),
        "historical_context": {
            "native_two_qubit_counts": {"sequential": 10, "generic": 46},
            "calibrated_COBYLA_raw_mean": -0.3138773646652825,
            "calibrated_SPSA_raw_mean": -0.3165324116006043,
            "calibrated_SPSA_rep2_minus_rep1": -0.02107209578855776,
            "calibrated_SPSA_difference_sampling_se": 0.02971047072106394,
            "historical_counts_and_calibrations_available": False,
            "qualification": "Not new evidence and not merged with this run.",
        },
        "estimator": {
            "primary": "Readout-corrected candidate linear OLS ZNE intercept",
            "factors": [1, 3, 5],
            "linear_weights": LINEAR,
            "quadratic_sensitivity_weights": QUADRATIC,
            "processing_selection": "Same prespecified method for both states.",
        },
    }
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    save_json("manifest.json", manifest)
else:
    manifest = json.loads((directory / "manifest.json").read_text())
    require(manifest["schema"] == 3, "Unsupported resume schema.")
    require(manifest["authority_hash"] == authority_hash, "Part I resume mismatch.")
    require(
        manifest["canonical_source_sha256"] == canonical_hash,
        "Source mismatch: change only RESUME_DIR when resuming.",
    )
    require(manifest["configuration"] == CFG, "Resume configuration mismatch.")
    require(manifest["versions"] == versions, "Resume package-version mismatch.")
    require(
        manifest["status"] != "building",
        "This directory contains an unfinished build, not a frozen experiment.",
    )
    save_json("last_resume.json", {
        "utc": utc(), "source_sha256": source_hash,
        "canonical_source_sha256": canonical_hash,
    })

# ------------------------ Full Pauli measurement model --------------------
pauli = SparsePauliOp.from_operator(Operator(Hc), atol=0, rtol=0)
original = pauli.to_list()
require(len(original) == 63, "Expected the full 63-term Hamiltonian.")
require(
    len(set(label for label, coefficient in original)) == len(original),
    "Duplicate original Pauli label.",
)
for label, coefficient in original:
    real(coefficient, label)
reconstruction_error = float(np.max(np.abs(pauli.to_matrix() - Hc)))
require(reconstruction_error <= MTOL, "Hamiltonian reconstruction failed.")

operators = list(pauli.group_commuting(qubit_wise=True))
require(len(operators) == 17, "Expected 17 complete QWC groups.")
grouped = [term for op in operators for term in op.to_list()]
require(
    Counter(label for label, coefficient in grouped) ==
    Counter(label for label, coefficient in original),
    "Grouped terms missing or duplicated.",
)
coefficients = dict(original)
require(
    all(c == coefficients[label] for label, c in grouped),
    "Grouping changed coefficients.",
)
require(
    sum(label == "IIIII" for label, c in grouped) == 1,
    "Identity must occur exactly once.",
)
groups = []
```

```
for g, op in enumerate(operators):
    basis = ["I"] * NQ
    values = np.zeros(DIM)
    for label, coefficient in op.to_list():
        mask = 0
        for q in range(NQ):
            p = label[NQ - 1 - q]
            if p != "I":
                require(basis[q] in ("I", p), "Non-QWC group.")
                basis[q] = p
                mask |= 1 << q
        values += real(coefficient, label) * np.array([
            -1.0 if (b & mask).bit_count() % 2 else 1.0 for b in range(DIM)
        ])
    basis = ["Z" if p == "I" else p for p in basis]
    groups.append({
        "index_zero_based": g, "basis_by_logical_qubit": basis,
        "qiskit_basis_label": "".join(reversed(basis)),
        "terms": op.to_list(), "values": values,
    })
G = len(groups)
documented_bases = [
    "YXXXY", "XYXXY", "YXXYX", "YXYYY", "YYYXY", "XXYXY", "XYXYX",
    "XYYYY", "YYXXX", "YYXYY", "YYYYX", "XXXYY", "XXYYX", "XYYXX",
    "YYXXX", "XXXXX", "ZZZZZ",
]
require(
    [g["qiskit_basis_label"] for g in groups] == documented_bases,
    "Historical group numbering/order changed.",
)

def check_vector(actual, expected, label):
    overlap = np.vdot(expected, actual)
    require(abs(overlap) > 0, label + ": zero overlap.")
    error = float(np.linalg.norm(actual - overlap / abs(overlap) * expected))
    require(error <= MTOL, f"{label}: vector error {error:.3e}.")
    return error

def check_probabilities(actual, expected, label):
    actual = np.asarray(actual, float)
    require(
        actual.shape == (DIM,) and np.all(np.isfinite(actual))
        and actual.min() >= -MTOL
        and abs(float(actual.sum()) - 1) <= TOL,
        label + ": invalid probabilities.",
    )
    error = float(np.max(np.abs(actual - expected)))
    require(error <= TOL, f"{label}: probability error {error:.3e}.")
    return error

# Exact supplied forward synthesis generalized by an explicit wire permutation.
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
# No StatePreparation or Initialize is used.
def sequential(vector, direction):
    order = list(range(NQ)) if direction == 1 else list(reversed(range(NQ)))
    perm = np.array([
        sum(((k >> j) & 1) << order[j] for j in range(NQ))
        for k in range(DIM)
    ])
    working = vector[perm]
    tensors = [None] * NQ
    carry = working.copy()
    right_rank = 1
    cuts = []
    for cut in range(1, NQ):
        singular = np.linalg.svd(
            working.reshape(1 << cut, 1 << (NQ - cut), order="F"),
            compute_uv=False,
        )
        tail = float(np.linalg.norm(singular[2:]))
        require(tail <= CFG["Schmidt_tail_tolerance"], "Bond-two cut failed.")
        cuts.append({"cut": cut, "singular_values": singular, "tail": tail})
    for q in range(NQ - 1, 0, -1):
        mat = carry.reshape(1 << q, 2 * right_rank, order="F")
        u, singular, vh = np.linalg.svd(mat, full_matrices=False)
        require(
            np.linalg.norm(singular[2:]) <= CFG["Schmidt_tail_tolerance"],
            "Sequential factorization would truncate a nonzero bond.",
        )
        tensors[q] = vh[:2].reshape(2, 2, right_rank, order="F")
        carry = u[:, :2] * singular[:2]
        right_rank = 2
    norm = float(np.linalg.norm(carry))
    require(abs(norm - 1) <= MTOL, "Sequential norm failed.")
    tensors[0] = (carry / norm).reshape(1, 2, 2, order="F")
    qc = QuantumCircuit(NQ)
    independent = np.zeros(DIM, complex)
    independent[0] = 1
    for q, tensor in enumerate(tensors):
        left_rank, _, next_rank = tensor.shape
        iso = tensor.transpose(1, 2, 0).reshape(
            2 * next_rank, left_rank, order="F"
        )
        require(
            np.max(np.abs(iso.conj().T @ iso - np.eye(left_rank))) <= MTOL,
            "Isometry failed.",
        )
        _, _, vh = np.linalg.svd(iso.conj().T, full_matrices=True)
        unitary = np.column_stack((iso, vh.conj().T[:, left_rank:]))
        require(
            np.max(np.abs(
                unitary.conj().T @ unitary - np.eye(len(unitary))
            )) <= MTOL,
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
            "Unitary completion failed.",
        )
        local_wires = [q, q + 1] if q < 4 else [q]
        qc.append(UnitaryGate(unitary), [order[j] for j in local_wires])
        d = len(unitary)
        output = np.zeros(DIM, complex)
        for k, amplitude in enumerate(independent):
            column = (k >> q) & (d - 1)
            base = k & ~((d - 1) << q)
            for row in range(d):
                output[base | (row << q)] += unitary[row, column] * amplitude
        independent = output
    physical_vector = np.zeros(DIM, complex)
    physical_vector[perm] = independent
    independent_error = check_vector(
        physical_vector, vector, "Independent sequential vector"
    )
    qiskit_error = check_vector(
        Statevector.from_instruction(qc).data, vector, "Qiskit sequential vector"
    )
    return qc, {
        "direction": direction, "cuts": cuts,
        "independent_error": independent_error, "qiskit_error": qiskit_error,
    }

preparations, prep_checks = {}, []
for s in range(2):
    for direction in (1, -1):
        preparations[s, direction], record = sequential(vectors[s], direction)
        prep_checks.append({"state": NAMES[s], **record})

def rotate(qc, basis):
    for q, axis in enumerate(basis):
        if axis == "X":
            qc.h(q)
        elif axis == "Y":
            qc.sdg(q)
            qc.h(q)

def logical_measurement(s, g, direction):
    qc = QuantumCircuit(
        QuantumRegister(NQ, "q"), ClassicalRegister(NQ, CFG["register"])
    )
    qc.compose(preparations[s, direction], inplace=True)
    rotate(qc, groups[g]["basis_by_logical_qubit"])
    for q in range(NQ):
        qc.measure(q, q)
    return qc

ideal = np.empty((2, G, DIM))
ideal_group = np.empty((2, G))
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    ideal_variance = np.empty((2, G))
    direct_pauli = np.empty(2)
    for s in range(2):
        direct_pauli[s] = real(Statevector(vectors[s]).expectation_value(pauli), "Pauli")
        require(abs(direct_pauli[s] - references[s]) <= ETOL, "Pauli/reference mismatch.")
        for g in range(G):
            direct = real(
                Statevector(vectors[s]).expectation_value(operators[g]), "Group"
            )
            for direction in (1, -1):
                p = Statevector.from_instruction(
                    logical_measurement(s, g, direction)
                    .remove_final_measurements(inplace=False)
                ).probabilities()
                require(abs(float(p @ groups[g]["values"]) - direct) <= ETOL,
                        "Independent deterministic measurement validation failed.")
                if direction == 1:
                    ideal[s, g] = p
                else:
                    check_probabilities(p, ideal[s, g], "Reverse-direction measurement")
            f = groups[g]["values"]
            ideal_group[s, g] = ideal[s, g] @ f
            ideal_variance[s, g] = ideal[s, g] @ ((f - ideal_group[s, g]) ** 2)
        require(abs(ideal_group[s].sum() - references[s]) <= ETOL,
                "Exact group sum failed.")

    def decode(counts, shots):
        out = np.zeros(DIM, np.int64)
        require(isinstance(counts, dict) and bool(counts), "Missing counts.")
        for bits, number in counts.items():
            require(
                isinstance(bits, str) and len(bits) == NQ
                and set(bits).issubset({"0", "1"}),
                "Unexpected returned register bitstring.",
            )
            require(
                isinstance(number, (int, np.integer)) and number >= 0,
                "Invalid count multiplicity.",
            )
            out[int(bits, 2)] += int(number)
        require(int(out.sum()) == shots, "Shot count mismatch.")
        return out

    def get_counts(pub):
        require(hasattr(pub.data, CFG["register"]), "Expected classical register missing.")
        bits = getattr(pub.data, CFG["register"])
        require(bits.num_bits == NQ, "Returned register width changed.")
        return bits.get_counts()

    controls = []
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
def expect_failure(function, label):
    try:
        function()
    except RuntimeError:
        controls.append({"label": label, "detected": True})
    else:
        raise RuntimeError("Negative control not detected: " + label)

corrupted = preparations[0, 1].copy()
corrupted.x(4)
expect_failure(
    lambda: check_vector(
        Statevector.from_instruction(corrupted).data, vectors[0], "Corrupt prep"
    ), "corrupted-preparation-X-q4",
)
bad_basis = QuantumCircuit(NQ)
bad_basis.compose(preparations[0, 1], inplace=True)
bad_basis.h(0)
expect_failure(
    lambda: check_probabilities(
        Statevector.from_instruction(bad_basis).probabilities(),
        ideal[0, 16], "Wrong measurement basis",
    ), "wrong-basis-group16",
)
reversed_indices = np.array([int(format(i, "05b")[::-1], 2) for i in range(DIM)])
expect_failure(
    lambda: check_probabilities(
        ideal[0, 16][reversed_indices], ideal[0, 16], "Wrong decoder"
    ), "reversed-bit-decoder",
)

def safe_inverse(A):
    require(
        A.shape == (DIM, DIM) and np.all(np.isfinite(A))
        and A.min() >= 0 and np.max(np.abs(A.sum(axis=0) - 1)) <= MTOL,
        "Invalid assignment matrix.",
    )
    singular = np.linalg.svd(A, compute_uv=False)
    condition = float(singular[0] / singular[-1]) if singular[-1] > 0 else float("inf")
    require(
        condition <= CFG["readout_condition_limit"]
        and singular[-1] >= CFG["readout_minimum_singular_value"],
        "Assignment conditioning rejected.",
    )
    inverse = np.linalg.solve(A, np.eye(DIM))
    error = float(np.max(np.abs(inverse @ A - np.eye(DIM))))
    require(error <= ETOL, "Assignment numerical inverse validation failed.")
    return inverse, condition, float(singular[-1]), error

strengths = np.linspace(0.04, 0.12, DIM)
fixture = np.diag(1 - strengths) + np.ones((DIM, 1)) * strengths[None, :] / DIM
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
fixture_inverse, fixture_condition, _, fixture_error = safe_inverse(fixture)
fixture_p = rng(1).dirichlet(np.arange(1, DIM + 1))
require(np.max(np.abs(fixture_inverse @ (fixture @ fixture_p) - fixture_p)) <= MTOL,
        "Synthetic readout recovery failed.")
expect_failure(
    lambda: safe_inverse(np.ones((DIM, DIM)) / DIM),
    "singular-assignment",
)
expect_failure(
    lambda: require(
        np.max(np.abs(
            np.linalg.solve(fixture.T, fixture @ fixture_p) - fixture_p
        )) <= TOL,
        "Transposed calibration detected.",
    ), "transposed-assignment",
)

# Independent local sampling streams, using the actual counts decoder.
local_rows = []
for s in range(2):
    for direction in (1, -1):
        for rep in range(2):
            group_rows = []
            for g in range(G):
                entropy = [CFG["seed"], 2, s, int(direction == 1), rep, g]
                seed = int(np.random.SeedSequence(entropy).generate_state(1)[0])
                sampler = StatevectorSampler(default_shots=CFG["local_shots"], seed=seed)
                pub = sampler.run(
                    [logical_measurement(s, g, direction)],
                    shots=CFG["local_shots"],
                ).result()[0]
                counts = get_counts(pub)
                p = decode(counts, CFG["local_shots"]) / CFG["local_shots"]
                f = groups[g]["values"]
                mean = float(p @ f)
                variance = float(p @ ((f - mean) ** 2) / (CFG["local_shots"] - 1))
                group_rows.append({
                    "group": g, "counts": counts, "seed_entropy": entropy,
                    "seed": seed, "energy": mean, "sampling_variance": variance,
                })
            energy = math.fsum(x["energy"] for x in group_rows)
            se = math.sqrt(math.fsum(x["sampling_variance"] for x in group_rows))
            require(abs(energy - references[s]) <= 6 * se,
                    "Local finite-shot statistical acceptance failed.")
            local_rows.append({
                "state": NAMES[s], "direction": direction, "repetition": rep + 1,
                "energy": energy, "sampling_se": se, "groups": group_rows,
            })

if resume_dir is None:
    save_json("local_validation.json", {
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        "references": references, "direct_pauli": direct_pauli,
        "exact_group_sum": ideal_group.sum(axis=1),
        "reconstruction_error": reconstruction_error,
        "preparations": prep_checks, "negative_controls": controls,
        "fixture_condition": fixture_condition, "fixture_inverse_error": fixture_error,
        "local_sampling": local_rows,
        "groups": groups,
    })

print("LOCAL ACCEPTANCE: PASS")
print("Versions:", versions)
print("Fixed-state references:", dict(zip(NAMES, references)))
print("Continuum / finite representation references:", continuum, fd_reference)
print("Pauli terms / groups:", len(pauli), G)
print("Reconstruction error:", reconstruction_error)
print("Synthetic conditioning:", fixture_condition)
print("Negative controls:", controls)
for row in local_rows:
    print("LOCAL", row["state"], row["direction"], row["repetition"],
          row["energy"], "+/-", row["sampling_se"], "(sampling SE)")
print("Evidence:", directory)

# ---------- IBM access after local validation ----------
try:
    service = QiskitRuntimeService(channel=CFG["channel"])
except AccountNotFoundError:
    token = getpass.getpass("IBM Quantum API token (hidden; never saved): ")
    try:
        require(bool(token.strip()), "Empty token.")
        service = QiskitRuntimeService(channel=CFG["channel"], token=token)
    finally:
        del token
backend = service.backend(CFG["backend"])
require(backend.name == CFG["backend"], "Backend substitution forbidden.")
require(backend.status().operational, "Requested backend is not operational.")
target = backend.target
width = int(backend.num_qubits)
dt = float(backend.dt)
require(dt > 0, "Backend dt unavailable.")
pulse_alignment = int(getattr(target, "pulse_alignment", 1) or 1)
acquire_alignment = int(getattr(target, "acquire_alignment", 1) or 1)
properties = backend.properties()
require(properties is not None, "Backend calibration properties unavailable.")
properties_dict = properties.to_dict()
native_two = {
    name for name in target.operation_names
    if getattr(target.operation_from_name(name), "num_qubits", None) == 2
    and name not in ("barrier", "delay")
}
require(bool(native_two), "No native entanglers available.")
missing, inconsistencies = set(), set()
```

```
    gate_dates = defaultdict(list)
    for gate in properties_dict.get("gates", []):
        gate_dates[gate["gate"], tuple(gate["qubits"])] = gate.get("parameters", [])

    def qvalue(q, name):
        try:
            item = properties.qubit_property(q).get(name)
        except Exception:
            item = None
        if item is None:
            missing.add(f"qubit:{q}:{name}")
            return None
        return float(item[0])

    def instruction_info(name, physical):
        physical = tuple(physical)
        require(
            target.instruction_supported(operation_name=name, qargs=physical),
            f"Unsupported instruction {name}{physical}.",
        )
        info = target[name].get(physical)
        duration = None if info is None else info.duration
        error = None if info is None else info.error
        if duration is None:
            missing.add(f"duration:{name}:{physical}")
        if error is None and name != "rz":
            missing.add(f"error:{name}:{physical}")
        return duration, error

    def duration_seconds(op, physical):
        if op.name == "barrier":
            return 0.0
        if op.name == "delay":
            factors = {"dt": dt, "s": 1., "ms": 1e-3, "us": 1e-6,
                       "ns": 1e-9, "ps": 1e-12}
            require(op.unit in factors, "Unsupported delay unit.")
            return float(op.duration) * factors[op.unit]
        duration, _ = instruction_info(op.name, physical)
        require(duration is not None, f"Missing executed duration {op.name}{physical}.")
        return float(duration)

    def readout_one(q):
        p10, p01 = qvalue(q, "prob_meas1_prep0"), qvalue(q, "prob_meas0_prep1")
        if p10 is None or p01 is None:
            _, average = instruction_info("measure", (q,))
            require(average is not None, "No usable readout calibration.")
            missing.add(f"asymmetric_readout:{q}:symmetric_model_assumption")
            p10 = p01 = float(average)
        require(0 <= p10 <= 1 and 0 <= p01 <= 1, "Invalid readout calibration.")
        return np.array([[1 - p10, p01], [p10, 1 - p01]])
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
def manager(level, layout):
    return generate_preset_pass_manager(
        backend=backend, optimization_level=level,
        initial_layout=list(layout), approximation_degree=1.0,
        unitary_synthesis_method="default",
        seed_transpiler=CFG["transpiler_seed"],
    )

def mapping(qc):
    require(qc.num_clbits == NQ and len(qc.cregs) == 1
            and qc.cregs[0].name == CFG["register"], "Classical register changed.")
    out = {}
    for item in qc.data:
        if item.operation.name == "measure":
            c = qc.find_bit(item.clbits[0]).index
            require(c not in out, "Duplicate measurement destination.")
            out[c] = qc.find_bit(item.qubits[0]).index
    require(set(out) == set(range(NQ)) and len(set(out.values())) == NQ,
            "Invalid measurement map.")
    return tuple(out[c] for c in range(NQ))

def actual_probabilities(qc):
    physical_for_c = mapping(qc)
    physical = sorted(physical_for_c)
    index = {q: j for j, q in enumerate(physical)}
    compact = QuantumCircuit(NQ)
    compact.global_phase = qc.global_phase
    measured = set()
    for item in qc.data:
        op = item.operation
        qs = [qc.find_bit(q).index for q in item.qubits]
        if op.name == "measure":
            measured.update(qs)
            continue
        if op.name in ("barrier", "delay"):
            continue
        require(set(qs) <= set(physical), "Active operation outside readout set.")
        require(not measured.intersection(qs), "Nonterminal measurement.")
        require(not item.clbits and getattr(op, "condition", None) is None,
                "Dynamic classical control unsupported.")
        require(op.name != "reset", "Unexpected reset.")
        compact.append(op.copy(), [index[q] for q in qs])
    local_p = Statevector.from_instruction(compact).probabilities()
    permutation = np.array([
        sum(((b >> index[physical_for_c[c]]) & 1) << c for c in range(NQ))
        for b in range(DIM)
    ])
    result = np.zeros(DIM)
    result[permutation] = local_p
    return result
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
def validate(qc, intended):
    return check_probabilities(actual_probabilities(qc), intended, "Actual circuit")

def schedule(qc, policy, dd):
    analysis = ASAPScheduleAnalysis if policy == "ASAP" else ALAPScheduleAnalysis
    passes = [
        TimeUnitConversion(target=target),
        analysis(target=target),
        ConstrainedReschedule(
            acquire_alignment=acquire_alignment, pulse_alignment=pulse_alignment
        ),
    ]
    if dd:
        passes.append(PadDynamicalDecoupling(
            durations=backend.instruction_durations,
            dd_sequence=[XGate(), XGate()],
            spacing=[0.25, 0.5, 0.25],
            skip_reset_qubits=True,
            pulse_alignment=pulse_alignment,
            target=target,
        ))
    else:
        passes.append(PadDelay(target=target))
    return PassManager(passes).run(qc)

def exposure(qc):
    selected = mapping(qc)
    clocks = defaultdict(float)
    measured = set()
    report = {
        q: {"active_s": 0., "idle_s": 0., "readout_s": 0.,
            "physical_pulses": 0, "virtual_zero_duration_operations": 0}
        for q in selected
    }
    counts = Counter()
    for item in qc.data:
        op = item.operation
        qs = tuple(qc.find_bit(q).index for q in item.qubits)
        if op.name == "barrier":
            if qs:
                start = max(clocks[q] for q in qs)
                for q in qs:
                    require(abs(clocks[q] - start) <= 1.1 * dt,
                            "Scheduled barrier has an unrepresented idle gap.")
            continue
        duration = duration_seconds(op, qs)
        start = max((clocks[q] for q in qs), default=0.)
        for q in qs:
            if q in report and q not in measured:
                require(abs(clocks[q] - start) <= 1.1 * dt,
                        "Scheduled operation has an unrepresented idle gap.")
```

```
            if op.name == "measure":
                report[q]["readout_s"] += duration
            elif op.name == "delay":
                report[q]["idle_s"] += duration
            else:
                report[q]["active_s"] += duration
                if duration == 0:
                    report[q]["virtual_zero_duration_operations"] += 1
                else:
                    report[q]["physical_pulses"] += 1
        clocks[q] = start + duration
    if op.name == "measure":
        measured.update(qs)
    if op.name not in ("barrier", "delay"):
        counts[op.name, qs] += 1
for q, item in report.items():
    item["T1_s"], item["T2_s"] = qvalue(q, "T1"), qvalue(q, "T2")
    item["premeasurement_s"] = item["active_s"] + item["idle_s"]
    item["idle_over_T2"] = (
        item["idle_s"] / item["T2_s"] if item["T2_s"] else None
    )
return {
    "qubits": report,
    "duration_s": max(clocks.values(), default=0.),
    "depth": qc.depth(),
    "native_two_qubit_count": sum(
        count for (name, qs), count in counts.items()
        if len(qs) == 2 and name in native_two
    ),
    "instruction_counts": [
        {
            "name": name, "physical": qs, "count": count,
            "duration_s": instruction_info(name, qs)[0],
            "error": instruction_info(name, qs)[1],
            "calibration_parameters": gate_dates.get((name, qs), []),
        }
        for (name, qs), count in sorted(counts.items())
    ],
}

# Explicit native Clifford Pauli randomization.
P = [
    np.eye(2, dtype=complex),
    np.array([[0, 1], [1, 0]], complex),
    np.array([[0, -1j], [1j, 0]], complex),
    np.diag([1, -1]).astype(complex),
]
PN = ["I", "X", "Y", "Z"]
pair_paulis = [np.kron(P[b], P[a]) for a in range(4) for b in range(4)]
twirl_cache = {}
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
def twirl_table(op):
    U = Operator(op).data
    key = array_hash(U)
    if key not in twirl_cache:
        table = []
        for a in range(4):
            for b in range(4):
                transformed = U @ np.kron(P[b], P[a]) @ U.conj().T
                overlaps = [
                    abs(np.trace(pa.conj().T @ transformed)) / 4
                    for pa in pair_paulis
                ]
                j = int(np.argmax(overlaps))
                require(overlaps[j] >= 1 - 1e-10,
                        f"{op.name} is not exactly Pauli-twirlable.")
                table.append((a, b, j // 4, j % 4))
        twirl_cache[key] = table
    return twirl_cache[key]

def append_pauli(qc, label, q):
    # Y differs from XZ only by global phase; explicit output validation
    # checks the complete circuit. No hidden classical correction is used.
    if label in (2, 3):
        require(target.instruction_supported(operation_name="rz", qargs=(q,)),
                "Native RZ unavailable.")
        qc.rz(np.pi, q)
    if label in (1, 2):
        require(target.instruction_supported(operation_name="x", qargs=(q,)),
                "Native X unavailable.")
        qc.x(q)

def fold_randomize(qc, factor, random, randomized):
    require(factor in (1, 3, 5), "Invalid fold factor.")
    out = qc.copy_empty_like()
    frames = []
    original_count = 0
    for item in qc.data:
        op = item.operation
        qs = [qc.find_bit(q).index for q in item.qubits]
        cs = [qc.find_bit(c).index for c in item.clbits]
        if len(qs) != 2 or op.name not in native_two:
            out.append(op.copy(), qs, cs)
            continue
        original_count += 1
        inv = op.inverse()
        if not target.instruction_supported(operation_name=inv.name, qargs=tuple(qs)):
            U = Operator(op).data
            phase = np.trace(U @ U) / 4
            require(np.max(np.abs(U @ U - phase * np.eye(4))) <= MTOL
                    and abs(abs(phase) - 1) <= MTOL,
                    "No supported exact native inverse for folding.")
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
            inv = op
        sequence = [op]
        for _ in range((factor - 1) // 2):
            sequence.extend([inv, op])
        for executed in sequence:
            instruction_info(executed.name, tuple(qs))
            out.barrier(*qs)
            if randomized:
                a, b, c, d = twirl_table(executed)[int(random.integers(16))]
                append_pauli(out, a, qs[0])
                append_pauli(out, b, qs[1])
                out.barrier(*qs)
                out.append(executed.copy(), qs)
                out.barrier(*qs)
                append_pauli(out, c, qs[0])
                append_pauli(out, d, qs[1])
                frames.append({
                    "native_gate": executed.name, "physical": qs,
                    "before": [PN[a], PN[b]], "after": [PN[c], PN[d]],
                    "classical_xor": 0,
                })
            else:
                out.append(executed.copy(), qs)
            out.barrier(*qs)
    realized = sum(
        len(item.qubits) == 2 and item.operation.name in native_two
        for item in out.data
    )
    require(realized == factor * original_count, "Realized fold count mismatch.")
    return out, frames, original_count, realized

def compile_measurement(s, g, path, direction):
    qc = manager(2, path).run(logical_measurement(s, g, direction))
    active = {
        qc.find_bit(q).index for item in qc.data
        if item.operation.name not in ("barrier", "delay")
        for q in item.qubits
    }
    require(active == set(path), "Routing changed the five-qubit active set.")
    validate(qc, ideal[s, g])
    return qc

# Compact noisy simulator with thermal channels plus residual depolarization.
# It is a selection model, not an exact device model.
def left_apply(U, rho, qubits):
    axes = list(reversed(qubits))
    remaining = [q for q in reversed(range(NQ)) if q not in axes]
    permutation = [4 - q for q in axes + remaining] + list(range(5, 10))
    inverse = np.argsort(permutation)
    tensor = rho.reshape([2] * 10).transpose(permutation)
    tensor = U @ tensor.reshape(2 ** len(qubits), -1)
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    return tensor.reshape([2] * 10).transpose(inverse).reshape(DIM, DIM)

def sandwich(U, rho, qubits):
    left = left_apply(U, rho, qubits)
    return left_apply(U, left.conj().T, qubits).conj().T

def thermal(q, duration, sensitivity):
    if duration == 0:
        return [P[0]]
    t1, t2 = qvalue(q, "T1"), qvalue(q, "T2")
    require(t1 is not None and t2 is not None and t1 > 0 and t2 > 0,
            f"Missing T1/T2 for executed qubit {q}.")
    t1, t2 = t1 * sensitivity, t2 * sensitivity
    if t2 > 2 * t1:
        inconsistencies.add(f"T2_above_2T1:{q}")
        t2 = 2 * t1
    gamma = -np.expm1(-duration / t1)
    extra = np.exp(-duration * max(0., 1 / t2 - 1 / (2 * t1)))
    phase_probability = (1 - extra) / 2
    amp = [
        np.diag([1, np.sqrt(1 - gamma)]).astype(complex),
        np.array([[0, np.sqrt(gamma)], [0, 0]], complex),
    ]
    return (
        [np.sqrt(1 - phase_probability) * k for k in amp] +
        [np.sqrt(phase_probability) * (P[3] @ k) for k in amp]
    )

def noisy_probabilities(qc, sensitivity):
    physical_for_c = mapping(qc)
    physical = sorted(physical_for_c)
    local = {q: j for j, q in enumerate(physical)}
    rho = np.zeros((DIM, DIM), complex)
    rho[0, 0] = 1
    for item in qc.data:
        op = item.operation
        qs = tuple(qc.find_bit(q).index for q in item.qubits)
        if op.name in ("barrier", "measure"):
            continue
        if not set(qs) <= set(physical):
            require(op.name == "delay", "Unexpected active spectator.")
            continue
        lqs = [local[q] for q in qs]
        duration = duration_seconds(op, qs)
        if op.name != "delay":
            rho = sandwich(Operator(op).data, rho, lqs)
        entanglement_fidelity = 1.
        for q, lq in zip(qs, lqs):
            kraus = thermal(q, duration, sensitivity)
            entanglement_fidelity *= sum(abs(np.trace(k)) ** 2 for k in kraus) / 4
            rho = sum(sandwich(k, rho, [lq]) for k in kraus)
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        if op.name == "delay":
            continue
        _, reported = instruction_info(op.name, qs)
        if reported is None:
            # Explicit modelling assumption only; never a hardware correction.
            reported = 0. if duration == 0 else (0.01 if len(qs) == 2 else 0.001)
            missing.add(f"selection_model_error_assumption:{op.name}:{qs}")
        d = 2 ** len(qs)
        thermal_error = 1 - (d * entanglement_fidelity + 1) / (d + 1)
        requested = float(reported) / sensitivity
        if requested < thermal_error:
            inconsistencies.add(f"reported_below_thermal:{op.name}:{qs}")
        denominator = (d - 1) / d - thermal_error
        residual = (
            max(0., requested - thermal_error) / denominator
            if denominator > 0 else 0.
        )
        residual = min(1., residual)
        if residual > 0:
            mixed = np.zeros_like(rho)
            for labels in itertools.product(range(4), repeat=len(qs)):
                U = np.array([[1]], complex)
                for label in reversed(labels):
                    U = np.kron(U, P[label])
                mixed += sandwich(U, rho, lqs) / (4 ** len(qs))
            rho = (1 - residual) * rho + residual * mixed
    p_local = rho.diagonal().real
    require(p_local.min() >= -1e-9 and abs(p_local.sum() - 1) <= 1e-8,
            "Noisy density-matrix simulation failed.")
    permutation = np.array([
        sum(((b >> local[physical_for_c[c]]) & 1) << c for c in range(NQ))
        for b in range(DIM)
    ])
    p = np.zeros(DIM)
    p[permutation] = p_local
    A = np.array([[1.]])
    for q in reversed(physical_for_c):
        A = np.kron(A, readout_one(q))
    # Readout applied once. Measurement duration is not also added as
    # premeasurement thermal evolution.
    return A @ p

if resume_dir is None:
    save_json("calibration_before.json", {
        "utc": utc(), "properties": properties_dict,
        "dt": dt, "pulse_alignment": pulse_alignment,
        "acquire_alignment": acquire_alignment,
        "target_granularity": getattr(target, "granularity", None),
        "target_min_length": getattr(target, "min_length", None),
    })
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
# Bounded connected-path beam search across all roots, not first-root DFS.
adjacency = defaultdict(set)
for name in native_two:
    for pair in target[name]:
        if pair is not None and len(pair) == 2:
            a, b = pair
            adjacency[a].add(b)
            adjacency[b].add(a)

def rough(path):
    score = 0.
    for q in path:
        t2 = qvalue(q, "T2")
        _, ro = instruction_info("measure", (q,))
        score += float(ro) if ro is not None else 0.1
        score += 1e-6 / t2 if t2 and t2 > 0 else 0.1
    for a, b in zip(path[:-1], path[1:]):
        possibilities = []
        for name in native_two:
            for pair in ((a, b), (b, a)):
                if target.instruction_supported(operation_name=name, qargs=pair):
                    duration, error = instruction_info(name, pair)
                    if duration is not None:
                        possibilities.append(
                            (float(error) if error is not None else 0.1)
                            + float(duration) / 1e-4
                        )
        score += min(possibilities, default=1.)
    return float(score)

beam = [(rough((q,)), (q,)) for q in sorted(adjacency)]
for length in range(1, NQ):
    extended = []
    for cost, path in beam:
        for q in sorted(adjacency[path[-1]]):
            if q not in path:
                new = path + (q,)
                extended.append((rough(new), new))
    require(bool(extended), "No connected five-qubit path.")
    beam = sorted(extended)[:CFG["path_beam_width"]]
unique = {}
for score, path in beam:
    canonical = min(path, tuple(reversed(path)))
    unique[canonical] = min(score, unique.get(canonical, float("inf")))
bounded = sorted(unique, key=lambda p: (unique[p], p))[:CFG["path_limit"]]
shortlist = bounded[:CFG["path_shortlist"]]

# Assess a path through 117 when it exists, without auto-selection or blacklist.
path117 = None
if 117 in adjacency:
    def extend117(path):
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        if len(path) == NQ:
            return path
        for q in sorted(adjacency[path[-1]], key=lambda x: (rough((x,)), x)):
            if q not in path:
                found = extend117(path + (q,))
                if found is not None:
                    return found
        return None
    found = extend117((117,))
    if found is not None:
        path117 = min(found, tuple(reversed(found)))
        if path117 not in shortlist:
            shortlist.append(path117)
save_json("path_screen.json", {
    "bounded_paths": [{"path": p, "rough_score": unique[p]} for p in bounded],
    "compiled_shortlist": shortlist,
    "path_through_117": path117,
    "qualification": (
        "Rough scores only screen paths. Final selection uses actual "
        "executed instructions, schedules, group bias, and both states."
    ),
})

# Self-test compact physical/classical decoding.
test = QuantumCircuit(
    QuantumRegister(NQ, "q"), ClassicalRegister(NQ, CFG["register"])
)
test.ry(0.37, 0)
test.h(2)
test.cx(0, 4)
test.rz(-0.29, 4)
test.cx(4, 2)
test.sdg(2)
for q in range(NQ):
    test.measure(q, q)
compiled_test = manager(2, shortlist[0]).run(test)
validate(compiled_test, Statevector.from_instruction(
    test.remove_final_measurements(inplace=False)
).probabilities())

candidate_records, candidates = [], []
print("Evaluating actual compiled/scheduled candidate configurations...")
candidate_number = 0
for path_index, path0 in enumerate(shortlist):
    for orientation in (1, -1):
        path = tuple(path0 if orientation == 1 else reversed(path0))
        for direction in (1, -1):
            try:
                compiled_base = {
                    (s, g): compile_measurement(s, g, path, direction)
                    for s in range(2) for g in range(G)
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
                    }
                except Exception as error:
                    candidate_records.append({
                        "path": path, "direction": direction, "valid": False,
                        "reason": f"{type(error).__name__}: {error}",
                    })
                    continue
                for policy in ("ASAP", "ALAP"):
                    for dd in (False, True):
                        descriptor = {
                            "path": path, "direction": direction,
                            "scheduling": policy, "dd": dd,
                        }
                        number = candidate_number
                        candidate_number += 1
                        try:
                            representative, reports = {}, {}
                            for s in range(2):
                                for g in range(G):
                                    transformed, _, _, _ = fold_randomize(
                                        compiled_base[s, g], 1,
                                        rng(10, path_index, int(orientation == 1),
                                            int(direction == 1), s, g),
                                        True,
                                    )
                                    validate(transformed, ideal[s, g])
                                    qc = schedule(transformed, policy, dd)
                                    validate(qc, ideal[s, g])
                                    representative[s, g] = qc
                                    reports[s, g] = exposure(qc)

                            sensitivity_records = []
                            central_p = None
                            for sensitivity in (0.8, 1., 1.2):
                                probabilities = np.array([
                                    [noisy_probabilities(representative[s, g], sensitivity)
                                     for g in range(G)]
                                    for s in range(2)
                                ])
                                energies = np.array([
                                    [probabilities[s, g] @ groups[g]["values"]
                                     for g in range(G)] for s in range(2)
                                ])
                                bias = energies - ideal_group
                                pulses = np.mean([
                                    sum(x["physical_pulses"]
                                        for x in report["qubits"].values())
                                    for report in reports.values()
                                ])
                                # No signed cancellation in this score.
                                score = float(np.max(
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
                        np.abs(bias).sum(axis=1)
                        + 2 * np.abs(bias).max(axis=1)
                    ) + 1e-5 * pulses)
                    sensitivity_records.append({
                        "sensitivity": sensitivity, "score": score,
                        "group_bias": bias,
                    })
                    if sensitivity == 1.:
                        central_p = probabilities
                robust_score = max(x["score"] for x in sensitivity_records)
                record = {
                    **descriptor, "valid": True, "robust_score": robust_score,
                    "sensitivity_results": sensitivity_records,
                    "mean_depth_per_circuit": float(np.mean([
                        x["depth"] for x in reports.values()
                    ])),
                    "summed_depth_across_circuits": sum(
                        x["depth"] for x in reports.values()
                    ),
                    "summed_native_entanglers": sum(
                        x["native_two_qubit_count"] for x in reports.values()
                    ),
                    "exposure": {
                        f"{s}:{g}": reports[s, g]
                        for s in range(2) for g in range(G)
                    },
                }
                candidate_records.append(record)
                candidates.append((robust_score, descriptor, central_p))
            except Exception as error:
                candidate_records.append({
                    **descriptor, "valid": False,
                    "reason": f"{type(error).__name__}: {error}",
                })
            save_json("candidate_search.json", candidate_records)
require(bool(candidates), "No candidate configuration passed local validation.")
candidates.sort(key=lambda x: (x[0], json.dumps(js(x[1]), sort_keys=True)))
_, selected, selected_model = candidates[0]
baseline = {
    "path": selected["path"], "direction": 1,
    "scheduling": "ASAP", "dd": False, "randomized": False,
}
# Placement is shared so the fresh unsuppressed control does not
# confound candidate suppression with a different physical readout set.
physical_set = tuple(sorted(selected["path"]))

baseline_compiled = {
    (s, g): compile_measurement(s, g, baseline["path"], 1)
    for s in range(2) for g in range(G)
}
candidate_compiled = {
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    (s, g): compile_measurement(s, g, selected["path"], selected["direction"])
    for s in range(2) for g in range(G)
}

baseline_metrics = []
for s in range(2):
    prep = manager(2, baseline["path"]).run(preparations[s, 1])
    prep_count = sum(
        len(item.qubits) == 2 and item.operation.name in native_two
        for item in prep.data
    )
    baseline_metrics.append({
        "state": NAMES[s], "preparation_only_depth": prep.depth(),
        "preparation_only_native_entanglers": prep_count,
        "historical_10_count_reproduced": prep_count == 10,
        "qualification": (
            "Same supplied sequential synthesis method; original artifact "
            "identity is unavailable. Any native-count mismatch is explicit."
        ),
    })

# Baseline noisy model, same actual timing model and sensitivities.
baseline_model_report = []
for s in range(2):
    reports, biases = [], []
    for g in range(G):
        qc = schedule(baseline_compiled[s, g], "ASAP", False)
        validate(qc, ideal[s, g])
        reports.append(exposure(qc))
        p = noisy_probabilities(qc, 1.)
        biases.append(float(p @ groups[g]["values"] - ideal_group[s, g]))
    baseline_model_report.append({
        "state": NAMES[s], "group_bias": biases,
        "absolute_group_bias_sum": float(np.abs(biases).sum()),
        "exposure": reports,
    })
save_json("baseline_model_and_metrics.json", {
    "preparation_only": baseline_metrics,
    "measurement_model": baseline_model_report,
})

# Training at both ends; held-out validation distributed across blocks.
cal_budget = 2 * DIM * CFG["calibration_shots_per_state_per_epoch"]
heldout_budget = B * DIM * CFG["heldout_shots_per_state_per_block"]
diagnostic_budget = 2 * CFG["diagnostic_shots"]
available = CFG["shot_cap"] - cal_budget - heldout_budget - diagnostic_budget

# Each unit contributes 256 shots/group/block. Candidate divides this
# among 16 randomizations, rather than multiplying the budget.
unit_per_block = R * 16
unit_total = B * unit_per_block
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    units = available // unit_total
    keys = [(s, mode, g) for s in range(2) for mode in range(4) for g in range(G)]
    require(units >= len(keys), "Minimum complete-group allocation cannot fit.")
    allocation = {key: 1 for key in keys}
    weights = {}
    for s, mode, g in keys:
        p = selected_model[s, g]
        f = groups[g]["values"]
        mean = float(p @ f)
        modeled_variance = float(p @ ((f - mean) ** 2))
        variance = max(modeled_variance, ideal_variance[s, g], 1e-10)
        importance = 1. if mode == 0 else max(0.25, abs(LINEAR[mode - 1]))
        weights[s, mode, g] = math.sqrt(variance) * importance
    for _ in range(int(units - len(keys))):
        key = max(
            keys,
            key=lambda k: weights[k] ** 2 /
            (allocation[k] * (allocation[k] + 1)),
        )
        allocation[key] += 1

    registry, plan, object_cache = {}, [], {}

    def register(qc, intended, metadata):
        error = validate(qc, intended)
        report = exposure(qc)
        buffer = io.BytesIO()
        qpy.dump(qc, buffer)
        payload = buffer.getvalue()
        cid = sha(payload)
        if cid not in registry:
            relative = "circuits/" + cid + ".qpy"
            save_binary(relative, payload)
            registry[cid] = {
                "file": relative, "sha256": cid,
                "probability_error": error, "exposure": report,
                "classical_to_physical": mapping(qc),
                "metadata": metadata,
            }
            object_cache[cid] = qc
        return cid

    def add(qc, shots, intended, metadata):
        cid = register(qc, intended, metadata)
        plan.append({
            "item_id": f"item_{len(plan):06d}", "circuit_id": cid,
            "shots": int(shots), **metadata,
        })

    # Physical calibration circuits have fixed canonical physical ordering.
    physical_manager = manager(0, list(range(width)))
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
def basis_circuit(value, corrupt=False):
    qc = QuantumCircuit(
        QuantumRegister(width, "physical"),
        ClassicalRegister(NQ, CFG["register"]),
    )
    for c, q in enumerate(physical_set):
        if (value >> c) & 1:
            qc.x(q)
    if corrupt:
        qc.x(physical_set[0])
    for c, q in enumerate(physical_set):
        qc.measure(q, c)
    qc = physical_manager.run(qc)
    require(mapping(qc) == physical_set, "Calibration mapping changed.")
    return schedule(qc, "ASAP", False)

basis_cache = {value: basis_circuit(value) for value in range(DIM)}
for value in range(DIM):
    add(
        basis_cache[value], CFG["calibration_shots_per_state_per_epoch"],
        np.eye(DIM)[value],
        {"kind": "assignment", "epoch": 0, "prepared": value, "block": -1},
    )
normal, deliberately_bad = basis_circuit(1), basis_circuit(1, True)
expect_failure(
    lambda: validate(deliberately_bad, np.eye(DIM)[1]),
    "physical-compiled-corruption",
)
for qc, prepared, label in (
    (normal, 1, "known-asymmetric-decoder-state"),
    (deliberately_bad, 0, "deliberately-corrupted-counterpart"),
):
    add(qc, CFG["diagnostic_shots"], np.eye(DIM)[prepared], {
        "kind": "diagnostic", "block": -1,
        "prepared": prepared, "label": label,
    })

# Four blocks, each containing both states, baseline, and all factors.
# Reverse/rotate state-group order; factors are interleaved at the
# randomization level rather than segregated into long factor batches.
for block in range(B):
    pairs = [(s, g) for s in range(2) for g in range(G)]
    order = rng(20, block).permutation(len(pairs))
    for pair_index in order:
        s, g = pairs[int(pair_index)]
        base_qc = schedule(baseline_compiled[s, g], "ASAP", False)
        base_meta = {
            "kind": "science", "block": block, "state": s, "group": g,
            "mode": 0, "factor": 1, "randomization": -1,
            "pauli_frame": [], "classical_xor": 0,
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
            }
            if (block + s + g) % 2 == 0:
                add(base_qc, unit_per_block * allocation[s, 0, g],
                    ideal[s, g], base_meta)
            for randomization in range(R):
                factor_order = np.roll(
                    [1, 3, 5], (block + s + g + randomization) % 3
                )
                for factor in factor_order:
                    factor = int(factor)
                    mode = {1: 1, 3: 2, 5: 3}[factor]
                    transformed, frames, n0, nreal = fold_randomize(
                        candidate_compiled[s, g], factor,
                        rng(30, block, s, g, randomization, factor), True,
                    )
                    validate(transformed, ideal[s, g])
                    qc = schedule(transformed, selected["scheduling"], selected["dd"])
                    add(qc, 16 * allocation[s, mode, g], ideal[s, g], {
                        "kind": "science", "block": block, "state": s, "group": g,
                        "mode": mode, "factor": factor,
                        "randomization": randomization,
                        "pauli_frame": frames, "classical_xor": 0,
                        "unfolded_native_entanglers": n0,
                        "realized_native_entanglers": nreal,
                        "realized_entangler_scale": nreal / n0 if n0 else None,
                    })
            if (block + s + g) % 2 != 0:
                add(base_qc, unit_per_block * allocation[s, 0, g],
                    ideal[s, g], base_meta)
    for value in rng(40, block).permutation(DIM):
        value = int(value)
        add(
            basis_cache[value], CFG["heldout_shots_per_state_per_block"],
            np.eye(DIM)[value],
            {"kind": "validation", "block": block, "prepared": value},
        )
for value in range(DIM):
    add(
        basis_cache[value], CFG["calibration_shots_per_state_per_epoch"],
        np.eye(DIM)[value],
        {"kind": "assignment", "epoch": 1, "prepared": value, "block": B},
    )

# Explicit logical-register to canonical physical index association.
for item in plan:
    if item["kind"] != "science":
        continue
    physical_for_c = registry[item["circuit_id"]]["classical_to_physical"]
    position = {q: j for j, q in enumerate(physical_set)}
    permutation = [
        sum(((k >> c) & 1) << position[physical_for_c[c]] for c in range(NQ))
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
            for k in range(DIM)
        ]
        require(len(set(permutation)) == DIM, "Canonical physical map not bijective.")
        item["logical_to_canonical_physical_index"] = permutation

    budget = Counter()
    for item in plan:
        budget[item["kind"]] += item["shots"]
    total_shots = int(sum(budget.values()))
    require(total_shots <= CFG["shot_cap"], "Frozen workload exceeds shot cap.")

    # Batch without crossing stage/block boundaries or changing item order.
    jobs = []
    current, current_stage = [], None
    for item in plan:
        stage_key = (
            item["kind"] if item["kind"] != "science" else "science",
            item.get("block"), item.get("epoch"),
        )
        if current and (
            stage_key != current_stage or len(current) >= CFG["job_circuit_limit"]
        ):
            jobs.append({
                "batch": len(jobs), "items": current,
                "status": "planned", "job_id": None,
            })
            current = []
        current_stage = stage_key
        current.append(item["item_id"])
    if current:
        jobs.append({
            "batch": len(jobs), "items": current,
            "status": "planned", "job_id": None,
        })

    manifest.update({
        "status": "frozen",
        "selected": selected, "baseline": baseline,
        "selection_rule": {
            "score": (
                "Worst over sensitivities and states of sum(abs(group bias)) "
                "+ 2*max(abs(group bias)) + 1e-5*mean physical-pulse count."
            ),
            "sensitivities": [0.8, 1., 1.2],
            "qualification": (
                "Approximate thermal/residual-depolarizing/readout model. "
                "One representative RC draw per group screens configurations; "
                "the actual experiment uses 16. No hardware results are used."
            ),
        },
        "baseline_preparation_metrics": baseline_metrics,
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        "physical_readout_set": physical_set,
        "registry": registry, "plan": plan, "jobs": jobs,
        "allocation": [
            {"state": s, "mode": mode, "group": g, "units": n}
            for (s, mode, g), n in allocation.items()
        ],
        "budget": dict(budget), "total_shots": total_shots,
        "missing_calibrations": sorted(missing),
        "model_calibration_inconsistencies": sorted(inconsistencies),
        "negative_controls": controls,
        "native_entangler_names": sorted(native_two),
        "timing": {
            "dt": dt, "pulse_alignment": pulse_alignment,
            "acquire_alignment": acquire_alignment,
        },
    })
    save_json("manifest.json", manifest)
else:
    selected = manifest["selected"]
    baseline = manifest["baseline"]
    physical_set = tuple(manifest["physical_readout_set"])
    registry = manifest["registry"]
    plan = manifest["plan"]
    jobs = manifest["jobs"]
    total_shots = manifest["total_shots"]
    budget = manifest["budget"]
    object_cache = {}

# Validate the saved circuit bytes, not an unchecked recompilation.
for cid, record in registry.items():
    payload = (directory / record["file"]).read_bytes()
    require(sha(payload) == record["sha256"] == cid, "Circuit artifact hash mismatch.")
    qc = qpy.load(io.BytesIO(payload))[0]
    require(mapping(qc) == tuple(record["classical_to_physical"]),
            "Saved circuit measurement association changed.")
    object_cache[cid] = qc
for item in plan:
    intended = (
        ideal[item["state"], item["group"]]
        if item["kind"] == "science" else np.eye(DIM)[item["prepared"]]
    )
    validate(object_cache[item["circuit_id"]], intended)

# Full associations, all shots, registry hashes, and exposures are frozen.
lookup = {item["item_id"]: item for item in plan}
require(len(lookup) == len(plan), "Duplicate plan item.")
print("\nFROZEN BACKEND:", CFG["backend"])
print("Physical width / logical qubits:", width, NQ)
print("Selected candidate:", selected)
print("Unsuppressed sequential baseline:", baseline)
print("Baseline preparation-only metrics:", manifest["baseline_preparation_metrics"])
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
print("Selected circuit exposures and native counts are in manifest.json.")
print("Mean depth and summed depth are different; summed depth is not runtime.")
print("Missing calibration:", manifest["missing_calibrations"])
print("Calibration/model inconsistencies:", manifest["model_calibration_inconsistencies"])

candidate_search_path = directory / "candidate_search.json"
if candidate_search_path.exists():
    candidate_report = json.loads(candidate_search_path.read_text())
    print("\nACTUAL COMPILED/SCHEDULED CANDIDATE COMPARISONS:")
    for record in candidate_report:
        print({
            k: record.get(k) for k in (
                "path", "direction", "scheduling", "dd", "valid",
                "robust_score", "mean_depth_per_circuit",
                "summed_depth_across_circuits", "summed_native_entanglers", "reason"
            )
        })

print("\nFULL WORKLOAD:")
for job_record in jobs:
    items = [lookup[key] for key in job_record["items"]]
    print({
        "batch": job_record["batch"],
        "status": job_record["status"],
        "job_id": job_record.get("job_id"),
        "publications": len(items),
        "shots": sum(x["shots"] for x in items),
        "kinds": sorted(set(x["kind"] for x in items)),
        "blocks_zero_based": sorted(set(x["block"] for x in items)),
    })
print({
    "backend": CFG["backend"], "blocks": B, "states": NAMES,
    "groups": G, "randomizations": R, "fold_factors": [1, 3, 5],
    "budget_by_kind": budget, "total_shots": total_shots,
    "shot_cap": CFG["shot_cap"], "circuit_executions": len(plan),
    "unique_saved_circuits": len(registry), "sampler_jobs": len(jobs),
    "evidence_directory": str(directory),
})
print("No optimizer or Estimator. Server DD/twirling will be disabled.")
print("Pulse additions, RC splits, folds, calibration, validation and diagnostics are counted.")
print("Within a job, exact device execution order is not asserted; factors are mixed in pubs.")

# Ambiguous submissions NEVER automatically resubmit.
for record in jobs:
    require(
        not (record["status"] == "submitting" and not record.get("job_id")),
        f'Batch {record['batch']} has ambiguous submission status. Inspect "
        "the account and explicitly attach its existing job ID in manifest.json; "
        "automatic resubmission is forbidden.",
    )
```

```
options = {
    "dynamical_decoupling": {"enable": False},
    "twirling": {"enable_gates": False, "enable_measure": False},
}
hardware_sampler = SamplerV2(mode=backend, options=options)
if resume_dir is None:
    save_json("sampler_options.json", options)

new_jobs = [j for j in jobs if j["status"] == "planned" and not j.get("job_id")]
new_shots = sum(
    lookup[key]["shots"] for job in new_jobs for key in job["items"]
)
if new_jobs:
    phrase = f"AUTHORIZE {CFG['backend']} {total_shots}"
    print("New submissions remaining:", len(new_jobs), "shots:", new_shots)
    if input("Type exactly: " + phrase + "\n> ").strip() != phrase:
        print("No new QPU submission authorized. Frozen evidence:", directory)
        return {"directory": str(directory), "manifest": manifest, "submitted": False}
    manifest.setdefault("authorizations", []).append({
        "utc": utc(), "new_jobs": len(new_jobs), "new_shots": new_shots,
        "total_frozen_shots": total_shots,
    })
    save_json("manifest.json", manifest)
else:
    print("No new submissions needed; retrieving saved/existing jobs only.")

# -------------------------- Execute/retrieve safely ----------------------
for record in jobs:
    result_relative = f"jobs/batch_{record['batch']:05d}.json"
    result_path = directory / result_relative
    if record["status"] == "complete":
        require(result_path.exists(), "Completed job raw-count file missing.")
        require(sha(result_path.read_bytes()) == record["raw_counts_sha256"],
                "Saved raw-count file hash mismatch.")
        continue
    items = [lookup[key] for key in record["items"]]
    if record.get("job_id"):
        job = service.job(record["job_id"])
    else:
        record["status"] = "submitting"
        record["submission_started_utc"] = utc()
        save_json("manifest.json", manifest)
        # If an exception occurs here, leave "submitting" intact.
        pubs = [
            SamplerPub(circuit=object_cache[item["circuit_id"]], shots=item["shots"])
            for item in items
        ]
        job = hardware_sampler.run(pubs)
        record["job_id"] = job.job_id()
        record["status"] = "submitted"
        save_json("manifest.json", manifest)
```

```
    result = job.result()
    binary_path = f"jobs/batch_{record['batch']:05d}_primitive_result.pkl"
    if not (directory / binary_path).exists():
        save_binary(binary_path, pickle.dumps(result, protocol=pickle.HIGHEST_PROTOCOL))
    require(len(result) == len(items), "Job/PUB count mismatch.")
    rows = []
    for item, pub in zip(items, result):
        counts = get_counts(pub)
        rows.append({
            "item_id": item["item_id"], "circuit_id": item["circuit_id"],
            "job_id": record["job_id"], "shots": item["shots"],
            "counts": counts, "metadata": getattr(pub, "metadata", {}),
        })
    # Save raw returned data before decoding/statistical validation.
    save_json(result_relative, {
        "job_id": record["job_id"], "batch": record["batch"],
        "result_metadata": getattr(result, "metadata", None), "rows": rows,
    })
    for row in rows:
        decode(row["counts"], row["shots"])
    record["raw_counts_sha256"] = sha(result_path.read_bytes())
    record["status"] = "complete"
    record["completed_utc"] = utc()
    save_json("manifest.json", manifest)
    print("Saved", record["job_id"], "batch", record["batch"])

try:
    after = backend.properties(refresh=True)
    save_json("calibration_after.json", {
        "utc": utc(), "properties": None if after is None else after.to_dict(),
    })
except Exception as error:
    save_json("calibration_after_unavailable.json", {
        "utc": utc(), "exception_type": type(error).__name__,
    })

# ---------------------------- Counts-only analysis ----------------------
observed = {}
for record in jobs:
    path = directory / f"jobs/batch_{record['batch']:05d}.json"
    require(sha(path.read_bytes()) == record["raw_counts_sha256"], "Raw hash mismatch.")
    content = json.loads(path.read_text())
    require(content["job_id"] == record["job_id"], "Raw job association mismatch.")
    require(
        [row["item_id"] for row in content["rows"]] == record["items"],
        "Raw publication ordering mismatch.",
    )
    for row in content["rows"]:
        require(row["item_id"] not in observed, "Duplicate returned item.")
        item = lookup[row["item_id"]]
        require(row["circuit_id"] == item["circuit_id"], "Circuit/result mismatch.")
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        observed[row["item_id"]] = decode(row["counts"], item["shots"])
require(len(observed) == len(plan), "Incomplete frozen experiment.")

assignment_counts = np.zeros((2, DIM, DIM), dtype=np.int64)
heldout_counts = np.zeros((B, DIM, DIM), dtype=np.int64)
science = defaultdict(list)
diagnostics = []
for item in plan:
    counts = observed[item["item_id"]]
    if item["kind"] == "assignment":
        assignment_counts[item["epoch"], :, item["prepared"]] += counts
    elif item["kind"] == "validation":
        heldout_counts[item["block"], :, item["prepared"]] += counts
    elif item["kind"] == "science":
        science[item["block"], item["state"], item["mode"], item["group"]].append(
            (item, counts)
        )
    else:
        diagnostics.append({
            **item, "counts": counts,
            "expected_fraction": float(counts[item["prepared"]] / counts.sum()),
        })
for key in science:
    science[key].sort(key=lambda pair: pair[0]["randomization"])

pooled_calibration = assignment_counts.sum(axis=0)
A = pooled_calibration / pooled_calibration.sum(axis=0, keepdims=True)
heldout = heldout_counts.sum(axis=0)
heldout = heldout / heldout.sum(axis=0, keepdims=True)
raw_TV = 0.5 * np.abs(A - heldout).sum(axis=0)
block_TV = [
    0.5 * np.abs(A - x / x.sum(axis=0, keepdims=True)).sum(axis=0)
    for x in heldout_counts
]
epochs = assignment_counts / assignment_counts.sum(axis=1, keepdims=True)
epoch_TV = 0.5 * np.abs(epochs[0] - epochs[1]).sum(axis=0)
readout_valid = False
inverse = None
readout_report = {
    "assignment_counts_by_epoch": assignment_counts,
    "pooled_assignment": A, "heldout_assignment": heldout,
    "heldout_TV": raw_TV, "heldout_block_TV": block_TV,
    "before_after_assignment_TV": epoch_TV,
    "thresholds": {
        "condition": CFG["readout_condition_limit"],
        "minimum_singular_value": CFG["readout_minimum_singular_value"],
        "heldout_TV": CFG["heldout_TV_limit"],
        "heldout_block_TV": CFG["heldout_block_TV_limit"],
        "corrected_heldout_TV": CFG["corrected_heldout_TV_limit"],
    },
}
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
try:
    inverse, condition, minimum, inverse_error = safe_inverse(A)
    corrected_heldout = inverse @ heldout
    recovery_TV = 0.5 * np.abs(corrected_heldout - np.eye(DIM)).sum(axis=0)
    readout_valid = bool(
        raw_TV.max() <= CFG["heldout_TV_limit"]
        and max(float(x.max()) for x in block_TV) <= CFG["heldout_block_TV_limit"]
        and recovery_TV.max() <= CFG["corrected_heldout_TV_limit"]
    )
    readout_report.update({
        "condition": condition, "minimum_singular_value": minimum,
        "inverse_identity_error": inverse_error,
        "corrected_heldout_assignment": corrected_heldout,
        "corrected_heldout_recovery_TV": recovery_TV,
    })
except RuntimeError as error:
    readout_report["reason"] = str(error)
readout_report["accepted"] = readout_valid
readout_report["qualification"] = (
    "Independent held-out empirical acceptance, not proof of transferability. "
    "Assignment calibration includes basis-state preparation errors. No "
    "probabilities/energies are clipped. Training is pooled across two epochs; "
    "epoch and per-block held-out discrepancies are exported, not called drift."
)
save_json("readout_validation.json", readout_report)
save_json("physical_diagnostics.json", diagnostics)

# Cache canonical observables and original per-randomization probabilities.
data = {}
for key, rows in science.items():
    block, s, mode, g = key
    probabilities, counts_list = [], []
    shots = []
    canonical_weights = None
    for item, counts in rows:
        permutation = np.array(item["logical_to_canonical_physical_index"])
        c = np.zeros(DIM, dtype=np.int64)
        c[permutation] = counts
        w = np.zeros(DIM)
        w[permutation] = groups[g]["values"]
        if canonical_weights is None:
            canonical_weights = w
        else:
            require(np.max(np.abs(w - canonical_weights)) <= MTOL,
                    "Randomization readout mapping unexpectedly changed.")
        counts_list.append(c)
        shots.append(int(c.sum()))
        probabilities.append(c / c.sum())
    require(len(set(shots)) == 1, "Unequal randomization shots within a group.")
    data[key] = {
        "p": np.stack(probabilities), "counts": np.stack(counts_list),
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        "shots": shots[0], "weights": canonical_weights,
    }

def evaluate(probabilities, inv=None):
    total = np.zeros((B, 2, 4))
    group_energy = np.zeros((B, 2, 4, G))
    conditional_variance = np.zeros((B, 2, 4))
    for key, item in data.items():
        block, s, mode, g = key
        p = probabilities[key]
        weights = item["weights"] if inv is None else item["weights"] @ inv
        values = p @ weights
        mean = float(values.mean())
        group_energy[block, s, mode, g] = mean
        total[block, s, mode] += mean
        # Complete weighted outcomes preserve within-group Pauli covariance.
        variance_per_draw = (
            p @ (weights ** 2) - values ** 2
        ) / (item["shots"] - 1)
        conditional_variance[block, s, mode] += (
            float(np.maximum(variance_per_draw, 0).sum()) / len(values) ** 2
        )
    return total, group_energy, conditional_variance

original_probabilities = {key: value["p"] for key, value in data.items()}
raw_blocks, raw_groups, raw_shot_var = evaluate(original_probabilities)
corrected_blocks = corrected_groups = corrected_shot_var = None
if readout_valid:
    corrected_blocks, corrected_groups, corrected_shot_var = evaluate(
        original_probabilities, inverse
    )

# Randomization covariance from paired factor/processing vectors.
# Subtract the estimated shot covariance before adding latent RC effects.
# PSD projection applies ONLY to a covariance estimate, not outcomes/energies.
latent_covariances = {}
latent_reports = []
if readout_valid:
    for block in range(B):
        for s in range(2):
            for g in range(G):
                columns, shot_covariance = [], np.zeros((6, 6))
                for mode in (1, 2, 3):
                    item = data[block, s, mode, g]
                    require(len(item["p"]) == R, "Missing randomizations.")
                    w = item["weights"]
                    z = w @ inverse
                    columns.append(item["p"] @ w)
                    columns.append(item["p"] @ z)
                    for p in item["p"]:
                        weights = np.stack([w, z])
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
                    mean = weights @ p
                    cov = (
                        (weights * p[None, :]) @ weights.T
                        - np.outer(mean, mean)
                    ) / (item["shots"] - 1)
                    ix = [(mode - 1) * 2, (mode - 1) * 2 + 1]
                    shot_covariance[np.ix_(ix, ix)] += cov / R
                samples = np.column_stack(columns)
                empirical = np.cov(samples, rowvar=False, ddof=1)
                proposed = empirical - shot_covariance
                eigenvalues, eigenvectors = np.linalg.eigh((proposed + proposed.T) / 2)
                psd = (eigenvectors * np.maximum(eigenvalues, 0)) @ eigenvectors.T
                latent_covariances[block, s, g] = psd / R
                latent_reports.append({
                    "block": block, "state": s, "group": g,
                    "raw_eigenvalues_before_PSD_projection": eigenvalues,
                    "covariance_of_randomization_mean": psd / R,
                    "column_order": [
                        "raw1", "corrected1", "raw3", "corrected3",
                        "raw5", "corrected5",
                    ],
                })
save_json("randomization_covariance_model.json", {
    "estimates": latent_reports,
    "qualification": (
        "Random-effects approximation from 16 paired randomization IDs. "
        "Shot covariance is subtracted before PSD projection; PSD cleanup "
        "does not alter counts, probabilities, or point energies."
    ),
})

# Component-wise bootstrap. The SAME calibration draw is used across every
# state/block/group/factor, retaining shared calibration covariance.
K = CFG["bootstrap_replicates"]
shape = (K, B, 2, 4)
raw_shot_boot = np.zeros(shape)
corr_shot_boot = np.zeros(shape) if readout_valid else None
corr_cal_boot = np.zeros(shape) if readout_valid else None
raw_random_boot = np.zeros(shape) if readout_valid else None
corr_random_boot = np.zeros(shape) if readout_valid else None
raw_joint_boot = np.zeros(shape)
corr_joint_boot = np.zeros(shape) if readout_valid else None
invalid_calibration_draws = 0

for iteration in range(K):
    shot_rng, cal_rng, random_rng = rng(100, iteration), rng(101, iteration), rng(102, iteration)
    sampled = {
        key: np.stack([
            shot_rng.multinomial(item["shots"], p) / item["shots"]
            for p in item["p"]
        ])
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        for key, item in data.items()
    }
    raw_shot_boot[iteration] = evaluate(sampled)[0]
    raw_joint_boot[iteration] = raw_shot_boot[iteration]

    if readout_valid:
        corr_shot_boot[iteration] = evaluate(sampled, inverse)[0]
        draw_counts = np.zeros((DIM, DIM), dtype=np.int64)
        for epoch in range(2):
            for column in range(DIM):
                counts = assignment_counts[epoch, :, column]
                draw_counts[:, column] += cal_rng.multinomial(
                    int(counts.sum()), counts / counts.sum()
                )
        draw_A = draw_counts / draw_counts.sum(axis=0, keepdims=True)
        try:
            draw_inverse, _, _, _ = safe_inverse(draw_A)
            corr_cal_boot[iteration] = evaluate(original_probabilities, draw_inverse)[0]
            corr_joint_boot[iteration] = evaluate(sampled, draw_inverse)[0]
        except RuntimeError:
            invalid_calibration_draws += 1
            corr_cal_boot[iteration] = np.nan
            corr_joint_boot[iteration] = np.nan

        raw_random_boot[iteration] = raw_blocks
        corr_random_boot[iteration] = corrected_blocks
        for (block, s, g), covariance in latent_covariances.items():
            delta = random_rng.multivariate_normal(np.zeros(6), covariance)
            raw_delta = delta[[0, 2, 4]]
            corr_delta = delta[[1, 3, 5]]
            raw_random_boot[iteration, block, s, 1:4] += raw_delta
            corr_random_boot[iteration, block, s, 1:4] += corr_delta
            raw_joint_boot[iteration, block, s, 1:4] += raw_delta
            corr_joint_boot[iteration, block, s, 1:4] += corr_delta

if readout_valid and invalid_calibration_draws:
    readout_valid = False
    warnings.warn(
        "Calibration bootstrap conditioning failed. No draws are silently "
        "discarded; corrected/ZNE inference is rejected. Raw data remain."
    )

def interval(samples):
    return np.quantile(samples, [0.025, 0.975], axis=0)

def summary(point, samples):
    return {
        "estimate": np.asarray(point),
        "standard_error": np.std(samples, axis=0, ddof=1),
        "statistical_95_percentile_interval": interval(samples),
    }
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
raw_mean = raw_blocks.mean(axis=0)
raw_boot_mean = raw_joint_boot.mean(axis=1)
report = {
    "backend": CFG["backend"], "references_Ha": dict(zip(NAMES, references)),
    "continuum_reference": continuum, "finite_reference": fd_reference,
    "finite_representation_difference": fd_reference - continuum,
    "classical_variational_residuals": references - fd_reference,
    "quantum_implementation_difference": ideal_group.sum(axis=1) - references,
    "local_sampling": local_rows,
    "baseline_raw": summary(raw_mean[:, 0], raw_boot_mean[:, :, 0]),
    "candidate_raw_factor1": summary(raw_mean[:, 1], raw_boot_mean[:, :, 1]),
    "raw_candidate_minus_baseline": summary(
        raw_mean[:, 1] - raw_mean[:, 0],
        raw_boot_mean[:, :, 1] - raw_boot_mean[:, :, 0],
    ),
    "raw_block_energies": raw_blocks,
    "raw_between_block_sd": raw_blocks.std(axis=0, ddof=1),
    "raw_between_block_se_of_mean": raw_blocks.std(axis=0, ddof=1) / math.sqrt(B),
    "readout_valid": readout_valid,
    "invalid_calibration_bootstrap_draws": invalid_calibration_draws,
    "uncertainty_assumptions": [
        "Independent multinomial samples conditional on each actual circuit.",
        "Shared calibration bootstrap matrices preserve cross-group/state/block/factor covariance.",
        "Two calibration epochs are independently resampled and pooled.",
        "Latent RC covariance approximates variation beyond science shot noise.",
        "No between-block variance is added a second time to shot/calibration/RC variance.",
        "Percentile intervals condition on frozen selection/allocation and the stated model.",
    ],
    "systematic_limitations": [
        "Calibration preparation error and transferability to science circuits.",
        "Correlated shots, leakage, crosstalk, coherent/non-Markovian errors, and drift.",
        "Approximate placement model is not an exact device model.",
        "Entangler folds do not uniformly scale all physical noise.",
        "ZNE intercept is not proven fully noiseless execution.",
        "Historical block-2 calibration/counts were not supplied; its cause cannot be reconstructed.",
    ],
    "processing_selection": "One prespecified corrected linear-ZNE estimator for both states.",
}

# Raw group residuals and current block-2 evidence are always exported.
raw_group_mean = raw_groups.mean(axis=0)
group_report = {
    "numbering": "Zero-based; unchanged supplied 17-group order.",
    "groups": groups,
    "ideal_group_energies": ideal_group,
    "raw_group_mean": raw_group_mean,
    "raw_group_residuals": raw_group_mean - ideal_group[:, None, :],
    "raw_block_group_energies": raw_groups,
    "raw_current_block2_group_energies": raw_groups[1],
}
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    achieved = np.zeros(2, bool)
    zne_valid = False

    if readout_valid:
        corr_mean = corrected_blocks.mean(axis=0)
        corr_boot_mean = corr_joint_boot.mean(axis=1)
        shot_mean = corr_shot_boot.mean(axis=1)
        cal_mean = corr_cal_boot.mean(axis=1)
        random_mean = corr_random_boot.mean(axis=1)

        factors = corr_mean[:, 1:4]
        factors_boot = corr_boot_mean[:, :, 1:4]
        final = factors @ LINEAR
        final_boot = factors_boot @ LINEAR
        quadratic = factors @ QUADRATIC
        quadratic_boot = factors_boot @ QUADRATIC

        design = np.column_stack([np.ones(3), [1., 3., 5.]])
        coefficients_fit = np.linalg.lstsq(design, factors.T, rcond=None)[0]
        fit_residuals = factors - (design @ coefficients_fit).T
        curvature_vector = np.array([1., -2., 1.])
        curvature = factors @ curvature_vector
        curvature_boot = factors_boot @ curvature_vector
        curvature_sd = curvature_boot.std(axis=0, ddof=1)
        fit_p = chi2.sf(curvature ** 2 / np.maximum(curvature_sd ** 2, 1e-30), 1)
        amplification = (
            final_boot.std(axis=0, ddof=1) /
            np.maximum(factors_boot[:, :, 0].std(axis=0, ddof=1), 1e-15)
        )
        sensitivity = quadratic - final
        sensitivity_boot = quadratic_boot - final_boot
        sensitivity_limit = np.maximum(
            CFG["model_sensitivity_floor_Ha"],
            2 * sensitivity_boot.std(axis=0, ddof=1),
        )
        valid_per_state = (
            (fit_p >= CFG["ZNE_lack_of_fit_p_min"])
            & (amplification <= CFG["ZNE_amplification_limit"])
            & (np.abs(sensitivity) <= sensitivity_limit)
        )
        zne_valid = bool(np.all(valid_per_state))

        block_final = corrected_blocks[:, :, 1:4] @ LINEAR
        block_final_boot = corr_joint_boot[:, :, :, 1:4] @ LINEAR
        block_final_intervals = interval(block_final_boot)
        confirmation = block_final[-2:].mean(axis=0)
        confirmation_boot = block_final_boot[:, -2:, :].mean(axis=1)
        confirmation_interval = interval(confirmation_boot)
        final_interval = interval(final_boot)

        heterogeneity_p = []
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
for s in range(2):
    contrast = np.eye(B) - np.ones((B, B)) / B
    covariance = np.cov(block_final_boot[:, :, s], rowvar=False)
    C = contrast @ covariance @ contrast.T
    delta = contrast @ block_final[:, s]
    eig, V = np.linalg.eigh((C + C.T) / 2)
    keep = eig > max(1e-16, float(eig.max()) * 1e-10)
    statistic = float(np.sum((V[:, keep].T @ delta) ** 2 / eig[keep]))
    pvalue = float(chi2.sf(statistic, int(keep.sum()))) if keep.any() else 0.
    heterogeneity_p.append(pvalue)
heterogeneity_p = np.array(heterogeneity_p)

for s in range(2):
    lower = references[s] - CFG["engineering_target_Ha"]
    upper = references[s] + CFG["engineering_target_Ha"]
    achieved[s] = bool(
        zne_valid
        and final_interval[0, s] >= lower and final_interval[1, s] <= upper
        and confirmation_interval[0, s] >= lower
        and confirmation_interval[1, s] <= upper
        and np.all(block_final[-2:, s] >= lower)
        and np.all(block_final[-2:, s] <= upper)
        and heterogeneity_p[s] >= CFG["block_heterogeneity_p_min"]
    )

improvement_point = (
    np.abs(raw_mean[:, 0] - references) - np.abs(final - references)
)
improvement_boot = (
    np.abs(raw_boot_mean[:, :, 0] - references)
    - np.abs(final_boot - references)
)
improvement_interval = interval(improvement_boot)

report.update({
    "baseline_readout_corrected": summary(corr_mean[:, 0], corr_boot_mean[:, :, 0]),
    "candidate_readout_corrected_factor1": summary(
        corr_mean[:, 1], corr_boot_mean[:, :, 1]
    ),
    "measured_corrected_factors_1_3_5": summary(factors, factors_boot),
    "corrected_linear_ZNE": summary(final, final_boot),
    "ZNE_error_against_fixed_state": final - references,
    "linear_fit_residuals_Ha": fit_residuals,
    "linear_lack_of_fit_p": fit_p,
    "ZNE_uncertainty_amplification": amplification,
    "quadratic_sensitivity_only": summary(quadratic, quadratic_boot),
    "quadratic_minus_linear": summary(sensitivity, sensitivity_boot),
    "quadratic_sensitivity_limit_Ha": sensitivity_limit,
    "ZNE_valid_per_state": valid_per_state,
    "ZNE_valid_for_both_states": zne_valid,
    "corrected_minus_raw_paired_baseline": summary(
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
            corr_mean[:, 0] - raw_mean[:, 0],
            corr_boot_mean[:, :, 0] - raw_boot_mean[:, :, 0],
        ),
        "corrected_minus_raw_paired_candidate_factor1": summary(
            corr_mean[:, 1] - raw_mean[:, 1],
            corr_boot_mean[:, :, 1] - raw_boot_mean[:, :, 1],
        ),
        "final_uncertainty_components": {
            "science_shot_se": (shot_mean[:, :, 1:4] @ LINEAR).std(axis=0, ddof=1),
            "shared_calibration_se": (cal_mean[:, :, 1:4] @ LINEAR).std(axis=0, ddof=1),
            "randomization_se": (random_mean[:, :, 1:4] @ LINEAR).std(axis=0, ddof=1),
            "joint_se": final_boot.std(axis=0, ddof=1),
            "qualification": (
                "Components are separately simulated; joint propagation is "
                "reported directly, not obtained by assuming component additivity."
            ),
        },
        "factor_covariance": [
            np.cov(factors_boot[:, s, :], rowvar=False) for s in range(2)
        ],
        "final_cross_state_covariance": np.cov(final_boot, rowvar=False),
        "block_final_estimates": block_final,
        "block_final_intervals": block_final_intervals,
        "between_block_final_sd": block_final.std(axis=0, ddof=1),
        "between_block_final_se_of_mean": block_final.std(axis=0, ddof=1) / math.sqrt(B),
        "block_heterogeneity_p": heterogeneity_p,
        "final_two_block_confirmation": summary(confirmation, confirmation_boot),
        "absolute_error_reduction_over_raw_baseline": summary(
            improvement_point, improvement_boot
        ),
        "statistically_supported_error_reduction": (
            (improvement_interval[0] > 0) & zne_valid
        ),
        "current_block2_corrected_minus_raw": (
            corrected_blocks[1] - raw_blocks[1]
        ),
        "current_block2_paired_correction_intervals": interval(
            corr_joint_boot[:, 1] - raw_joint_boot[:, 1]
        ),
        "block2_qualification": (
            "Current block-2 correction, shared-calibration uncertainty, "
            "held-out assignment differences, and training-epoch changes "
            "are measured/exported. Neither drift nor decoder error is assumed."
        ),
    })

    corr_group_mean = corrected_groups.mean(axis=0)
    zne_group = np.einsum("smg,m->sg", corr_group_mean[:, 1:4, :], LINEAR)
    residual = zne_group - ideal_group
    require(np.max(np.abs(residual.sum(axis=1) - (final - references))) <= ETOL,
            "ZNE group residual sum failed.")
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    signed = residual.sum(axis=1)
    absolute = np.abs(residual).sum(axis=1)
    group_report.update({
        "corrected_group_mean": corr_group_mean,
        "ZNE_group_energies": zne_group,
        "ZNE_group_residuals": residual,
        "signed_residual_sum": signed,
        "absolute_residual_sum": absolute,
        "positive_residual_sum": np.maximum(residual, 0).sum(axis=1),
        "negative_residual_sum": np.minimum(residual, 0).sum(axis=1),
        "cancellation_fraction": np.where(
            absolute > 0, 1 - np.abs(signed) / np.maximum(absolute, 1e-30), 0.
        ),
        "current_block2_group_corrections": corrected_groups[1] - raw_groups[1],
        "highlighted_groups": {
            str(g): {
                "basis": groups[g]["qiskit_basis_label"],
                "terms": groups[g]["terms"],
                "raw": raw_group_mean[:, :, g],
                "corrected": corr_group_mean[:, :, g],
                "ZNE_residual": residual[:, g],
                "current_block2_correction": corrected_groups[1, :, :, g] - raw_groups[1, :, :, g],
            }
            for g in (2, 8, 14, 16)
        },
    })
else:
    report["corrected_and_ZNE_inference"] = (
        "Unavailable/rejected under prespecified readout safeguards. "
        "Raw science results remain valid measured data. No alternative "
        "processing is selected retrospectively."
    )

report.update({
    "engineering_target_Ha": CFG["engineering_target_Ha"],
    "target_achieved": dict(zip(NAMES, map(bool, achieved))),
    "success_rule": (
        "Same corrected linear-ZNE estimator for both states. Final and "
        "final-two-block 95% intervals entirely inside reference +/- target; "
        "both final block point estimates inside; both-state ZNE validity; "
        "readout validity; heterogeneity p >= configured threshold."
    ),
    "improvement_vs_target": (
        "Improvement over baseline is reported separately from attaining "
        "the accuracy milestone. Processing levels are not interchangeable "
        "evidence of improved physical state preparation."
    ),
})
save_json("analysis.json", report)
save_json("group_diagnostics.json", group_report)
np.savez_compressed(
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        directory / "bootstrap_samples.npz",
        raw_shot=raw_shot_boot, raw_joint=raw_joint_boot,
        corrected_shot=corr_shot_boot if corr_shot_boot is not None else np.array([]),
        corrected_calibration=corr_cal_boot if corr_cal_boot is not None else np.array([]),
        raw_randomization=raw_random_boot if raw_random_boot is not None else np.array([]),
        corrected_randomization=corr_random_boot if corr_random_boot is not None else np.array([]),
        corrected_joint=corr_joint_boot if corr_joint_boot is not None else np.array([]),
    )
    os.chmod(directory / "bootstrap_samples.npz", 0o600)
    manifest["status"] = "complete"
    manifest["analysis_completed_utc"] = utc()
    manifest["target_achieved"] = report["target_achieved"]
    save_json("manifest.json", manifest)

    print("\nFINAL RESULTS — HARTREE")
    for s, name in enumerate(NAMES):
        print("\n", name, "fixed-state reference:", references[s])
        print("Baseline raw:", report["baseline_raw"]["estimate"][s],
              "95% interval:",
              report["baseline_raw"]["statistical_95_percentile_interval"][:, s])
        print("Candidate raw factor1:", report["candidate_raw_factor1"]["estimate"][s],
              "95% interval:",
              report["candidate_raw_factor1"]["statistical_95_percentile_interval"][:, s])
        if readout_valid:
            print("Corrected measured factors 1,3,5:",
                  report["measured_corrected_factors_1_3_5"]["estimate"][s])
            print("Prespecified corrected linear-ZNE estimate:",
                  report["corrected_linear_ZNE"]["estimate"][s],
                  "95% interval:",
                  report["corrected_linear_ZNE"]["statistical_95_percentile_interval"][:, s])
            print("ZNE accepted for both states:", zne_valid)
            print("Supported reduction vs baseline:",
                  bool(report["statistically_supported_error_reduction"][s]))
            print("Final uncertainty components:", {
                k: v[s] for k, v in report["final_uncertainty_components"].items()
                if isinstance(v, np.ndarray)
            })
        else:
            print("Readout-corrected/ZNE inference rejected or unavailable.")
        print("Engineering accuracy milestone:",
              "ACHIEVED" if achieved[s] else "NOT ACHIEVED")
    print("Group residual/cancellation diagnostics:", directory / "group_diagnostics.json")
    print("Job IDs:", [record["job_id"] for record in jobs])
    print("Evidence:", directory)
    print("No chemical-accuracy claim; statistical intervals do not cover all systematics.")
    return {"directory": str(directory), "manifest": manifest, "analysis": report}


# Strict dependency sweep: only these ten pre-existing Part I symbols are read.
# RESUME_DIR is defined in this cell; every other working symbol is imported or
# defined inside the function. No prior Part II variables or artifacts are
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
# required unless explicit RESUME_DIR opts into this cell's frozen experiment.
_part_ii_required = (
    "H", "r", "h", "n_interior", "R_max", "E_exact", "E0_fd",
    "alpha_cobyla", "alpha_spsa", "normalized_trial_state",
)
_part_ii_missing = [name for name in _part_ii_required if name not in globals()]
if _part_ii_missing:
    raise NameError(
        "Run authoritative Part I first. Missing: " + ", ".join(_part_ii_missing)
    )

part_ii_accuracy_results = _run_hydrogen_accuracy_revision(
    **{name: globals()[name] for name in _part_ii_required},
    resume_dir=RESUME_DIR,
)
```

Fifth-run results (verbatim)

```
XXXX
LOCAL ACCEPTANCE: PASS
Versions: {'numpy': '2.1.3', 'scipy': '1.16.3', 'qiskit': '2.5.2', 'qiskit-ibm-runtime': '0.50.0'}
Fixed-state references: {'COBYLA': np.float64(-0.46097392795976344), 'SPSA': np.float64(-
0.46077921338321903)}
Continuum / finite representation references: -0.5 -0.46097392795978287
Pauli terms / groups: 63 17
Reconstruction error: 1.7763568394002505e-15
Synthetic conditioning: 1.1378498452226022
Negative controls: [{'label': 'corrupted-preparation-X-q4', 'detected': True}, {'label': 'wrong-basis-group16',
'detected': True}, {'label': 'reversed-bit-decoder', 'detected': True}, {'label': 'singular-assignment', 'detected':
True}, {'label': 'transposed-assignment', 'detected': True}]
LOCAL COBYLA 1 1 -0.46778545379638664 +/- 0.019864914564208896 (sampling SE)
LOCAL COBYLA 1 2 -0.4500610188075476 +/- 0.019831798540152427 (sampling SE)
LOCAL COBYLA -1 1 -0.4669320478893465 +/- 0.019980887882022104 (sampling SE)
LOCAL COBYLA -1 2 -0.4761955170404343 +/- 0.01984334527357382 (sampling SE)
LOCAL SPSA 1 1 -0.4552678080967493 +/- 0.01997519562816146 (sampling SE)
LOCAL SPSA 1 2 -0.5386995370047432 +/- 0.019824904761228963 (sampling SE)
LOCAL SPSA -1 1 -0.4386052294441196 +/- 0.01988695283415537 (sampling SE)
LOCAL SPSA -1 2 -0.4574317151024229 +/- 0.019693278945804953 (sampling SE)
Evidence: /content/hydrogen_accuracy_20261015T174607_529559Z_fb1c49bf
IBM Quantum API token (hidden; never saved): ·····
qiskit_runtime_service._discover_account:WARNING:2026-10-15 17:46:23,180: Loading account with the
given token. A saved account will not be used.
qiskit_runtime_service.__init__:WARNING:2026-10-15 17:46:25,475: Instance was not set at service
instantiation. Free and trial plan instances will be prioritized. Based on the following filters: (tags: None, region:
us-east, eu-de), and available plans: (open), the available account instances are: open-instance. If you need a
specific instance set it explicitly either by using a saved account with a saved default instance or passing it in
directly to QiskitRuntimeService(). Alternatively, pass instance='auto' or save it to your account for auto-
selection without warning.
qiskit_runtime_service.backends:WARNING:2026-10-15 17:46:25,476: Using instance: open-instance, plan:
open
Evaluating actual compiled/scheduled candidate configurations...
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

FROZEN BACKEND: ibm_marrakesh
Physical width / logical qubits: 156 5
Selected candidate: {'path': (8, 7, 6, 5, 4), 'direction': 1, 'scheduling': 'ASAP', 'dd': False}
Unsuppressed sequential baseline: {'path': (8, 7, 6, 5, 4), 'direction': 1, 'scheduling': 'ASAP', 'dd': False, 'randomized': False}
Baseline preparation-only metrics: [{'state': 'COBYLA', 'preparation_only_depth': 41, 'preparation_only_native_entanglers': 10, 'historical_10_count_reproduced': True, 'qualification': 'Same supplied sequential synthesis method; original artifact identity is unavailable. Any native-count mismatch is explicit.'}, {'state': 'SPSA', 'preparation_only_depth': 41, 'preparation_only_native_entanglers': 10, 'historical_10_count_reproduced': True, 'qualification': 'Same supplied sequential synthesis method; original artifact identity is unavailable. Any native-count mismatch is explicit.'}]
Selected circuit exposures and native counts are in manifest.json.
Mean depth and summed depth are different; summed depth is not runtime.
Missing calibration: ['qubit:113:T2', 'qubit:130:T2', 'qubit:82:T2', 'qubit:94:T2']
Calibration/model inconsistencies: ['reported_below_thermal:sx:(102,)', 'reported_below_thermal:sx:(117,)', 'reported_below_thermal:x:(102,)', 'reported_below_thermal:x:(117,)']

ACTUAL COMPILED/SCHEDULED CANDIDATE COMPARISONS:
{'path': [4, 5, 6, 7, 8], 'direction': 1, 'scheduling': 'ASAP', 'dd': False, 'valid': True, 'robust_score': 0.40786581838487446, 'mean_depth_per_circuit': 73.29411764705883, 'summed_depth_across_circuits': 2492, 'summed_native_entanglers': 340, 'reason': None}
{'path': [4, 5, 6, 7, 8], 'direction': 1, 'scheduling': 'ASAP', 'dd': True, 'valid': False, 'robust_score': None, 'mean_depth_per_circuit': None, 'summed_depth_across_circuits': None, 'summed_native_entanglers': None, 'reason': 'RuntimeError: Nonterminal measurement.'}
{'path': [4, 5, 6, 7, 8], 'direction': 1, 'scheduling': 'ALAP', 'dd': False, 'valid': True, 'robust_score': 0.4167836918605403, 'mean_depth_per_circuit': 73.29411764705883, 'summed_depth_across_circuits': 2492, 'summed_native_entanglers': 340, 'reason': None}
{'path': [4, 5, 6, 7, 8], 'direction': 1, 'scheduling': 'ALAP', 'dd': True, 'valid': True, 'robust_score': 0.45455594563436946, 'mean_depth_per_circuit': 73.29411764705883, 'summed_depth_across_circuits': 2492, 'summed_native_entanglers': 340, 'reason': None}
{'path': [4, 5, 6, 7, 8], 'direction': -1, 'scheduling': 'ASAP', 'dd': False, 'valid': True, 'robust_score': 0.4535064674504879, 'mean_depth_per_circuit': 70.0, 'summed_depth_across_circuits': 2380, 'summed_native_entanglers': 340, 'reason': None}
{'path': [4, 5, 6, 7, 8], 'direction': -1, 'scheduling': 'ASAP', 'dd': True, 'valid': False, 'robust_score': None, 'mean_depth_per_circuit': None, 'summed_depth_across_circuits': None, 'summed_native_entanglers': None, 'reason': 'RuntimeError: Nonterminal measurement.'}
{'path': [4, 5, 6, 7, 8], 'direction': -1, 'scheduling': 'ALAP', 'dd': False, 'valid': True, 'robust_score': 0.4286944124968949, 'mean_depth_per_circuit': 70.0, 'summed_depth_across_circuits': 2380, 'summed_native_entanglers': 340, 'reason': None}
{'path': [4, 5, 6, 7, 8], 'direction': -1, 'scheduling': 'ALAP', 'dd': True, 'valid': True, 'robust_score': 0.4245676873888628, 'mean_depth_per_circuit': 70.0, 'summed_depth_across_circuits': 2380, 'summed_native_entanglers': 340, 'reason': None}
{'path': [8, 7, 6, 5, 4], 'direction': 1, 'scheduling': 'ASAP', 'dd': False, 'valid': True, 'robust_score': 0.2579694625648642, 'mean_depth_per_circuit': 74.20588235294117, 'summed_depth_across_circuits': 2523, 'summed_native_entanglers': 340, 'reason': None}
{'path': [8, 7, 6, 5, 4], 'direction': 1, 'scheduling': 'ASAP', 'dd': True, 'valid': False, 'robust_score': None, 'mean_depth_per_circuit': None, 'summed_depth_across_circuits': None, 'summed_native_entanglers': None, 'reason': 'RuntimeError: Nonterminal measurement.'}
{'path': [8, 7, 6, 5, 4], 'direction': 1, 'scheduling': 'ALAP', 'dd': False, 'valid': True, 'robust_score': 0.2887322424399685, 'mean_depth_per_circuit': 74.20588235294117, 'summed_depth_across_circuits': 2523, 'summed_native_entanglers': 340, 'reason': None}

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

{'path': [8, 7, 6, 5, 4], 'direction': 1, 'scheduling': 'ALAP', 'dd': True, 'valid': True, 'robust_score': 0.3505689055666758, 'mean_depth_per_circuit': 74.20588235294117, 'summed_depth_across_circuits': 2523, 'summed_native_entanglers': 340, 'reason': None}
{'path': [8, 7, 6, 5, 4], 'direction': -1, 'scheduling': 'ASAP', 'dd': False, 'valid': True, 'robust_score': 0.3605814448632275, 'mean_depth_per_circuit': 71.05882352941177, 'summed_depth_across_circuits': 2416, 'summed_native_entanglers': 340, 'reason': None}
{'path': [8, 7, 6, 5, 4], 'direction': -1, 'scheduling': 'ASAP', 'dd': True, 'valid': False, 'robust_score': None, 'mean_depth_per_circuit': None, 'summed_depth_across_circuits': None, 'summed_native_entanglers': None, 'reason': 'RuntimeError: Nonterminal measurement.'}
{'path': [8, 7, 6, 5, 4], 'direction': -1, 'scheduling': 'ALAP', 'dd': False, 'valid': True, 'robust_score': 0.29270218860501385, 'mean_depth_per_circuit': 71.05882352941177, 'summed_depth_across_circuits': 2416, 'summed_native_entanglers': 340, 'reason': None}
{'path': [8, 7, 6, 5, 4], 'direction': -1, 'scheduling': 'ALAP', 'dd': True, 'valid': True, 'robust_score': 0.2911121090162065, 'mean_depth_per_circuit': 71.05882352941177, 'summed_depth_across_circuits': 2416, 'summed_native_entanglers': 340, 'reason': None}
{'path': [2, 3, 4, 5, 6], 'direction': 1, 'scheduling': 'ASAP', 'dd': False, 'valid': True, 'robust_score': 0.34776278683553796, 'mean_depth_per_circuit': 73.23529411764706, 'summed_depth_across_circuits': 2490, 'summed_native_entanglers': 340, 'reason': None}
{'path': [2, 3, 4, 5, 6], 'direction': 1, 'scheduling': 'ASAP', 'dd': True, 'valid': False, 'robust_score': None, 'mean_depth_per_circuit': None, 'summed_depth_across_circuits': None, 'summed_native_entanglers': None, 'reason': 'RuntimeError: Nonterminal measurement.'}
{'path': [2, 3, 4, 5, 6], 'direction': 1, 'scheduling': 'ALAP', 'dd': False, 'valid': True, 'robust_score': 0.35843167036415025, 'mean_depth_per_circuit': 73.23529411764706, 'summed_depth_across_circuits': 2490, 'summed_native_entanglers': 340, 'reason': None}
{'path': [2, 3, 4, 5, 6], 'direction': 1, 'scheduling': 'ALAP', 'dd': True, 'valid': True, 'robust_score': 0.3736679279075456, 'mean_depth_per_circuit': 73.23529411764706, 'summed_depth_across_circuits': 2490, 'summed_native_entanglers': 340, 'reason': None}
{'path': [2, 3, 4, 5, 6], 'direction': -1, 'scheduling': 'ASAP', 'dd': False, 'valid': True, 'robust_score': 0.38834593861963523, 'mean_depth_per_circuit': 71.29411764705883, 'summed_depth_across_circuits': 2424, 'summed_native_entanglers': 340, 'reason': None}
{'path': [2, 3, 4, 5, 6], 'direction': -1, 'scheduling': 'ASAP', 'dd': True, 'valid': False, 'robust_score': None, 'mean_depth_per_circuit': None, 'summed_depth_across_circuits': None, 'summed_native_entanglers': None, 'reason': 'RuntimeError: Nonterminal measurement.'}
{'path': [2, 3, 4, 5, 6], 'direction': -1, 'scheduling': 'ALAP', 'dd': False, 'valid': True, 'robust_score': 0.3766888373802522, 'mean_depth_per_circuit': 71.29411764705883, 'summed_depth_across_circuits': 2424, 'summed_native_entanglers': 340, 'reason': None}
{'path': [2, 3, 4, 5, 6], 'direction': -1, 'scheduling': 'ALAP', 'dd': True, 'valid': True, 'robust_score': 0.37565510635347055, 'mean_depth_per_circuit': 71.29411764705883, 'summed_depth_across_circuits': 2424, 'summed_native_entanglers': 340, 'reason': None}
{'path': [6, 5, 4, 3, 2], 'direction': 1, 'scheduling': 'ASAP', 'dd': False, 'valid': True, 'robust_score': 0.27710725804475794, 'mean_depth_per_circuit': 73.97058823529412, 'summed_depth_across_circuits': 2515, 'summed_native_entanglers': 340, 'reason': None}
{'path': [6, 5, 4, 3, 2], 'direction': 1, 'scheduling': 'ASAP', 'dd': True, 'valid': False, 'robust_score': None, 'mean_depth_per_circuit': None, 'summed_depth_across_circuits': None, 'summed_native_entanglers': None, 'reason': 'RuntimeError: Nonterminal measurement.'}
{'path': [6, 5, 4, 3, 2], 'direction': 1, 'scheduling': 'ALAP', 'dd': False, 'valid': True, 'robust_score': 0.2822591720309563, 'mean_depth_per_circuit': 73.97058823529412, 'summed_depth_across_circuits': 2515, 'summed_native_entanglers': 340, 'reason': None}
{'path': [6, 5, 4, 3, 2], 'direction': 1, 'scheduling': 'ALAP', 'dd': True, 'valid': True, 'robust_score': 0.2963763858574192, 'mean_depth_per_circuit': 73.97058823529412, 'summed_depth_across_circuits': 2515, 'summed_native_entanglers': 340, 'reason': None}

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

{'path': [6, 5, 4, 3, 2], 'direction': -1, 'scheduling': 'ASAP', 'dd': False, 'valid': True, 'robust_score': 0.3181980266839739, 'mean_depth_per_circuit': 70.23529411764706, 'summed_depth_across_circuits': 2388, 'summed_native_entanglers': 340, 'reason': None}
{'path': [6, 5, 4, 3, 2], 'direction': -1, 'scheduling': 'ASAP', 'dd': True, 'valid': False, 'robust_score': None, 'mean_depth_per_circuit': None, 'summed_depth_across_circuits': None, 'summed_native_entanglers': None, 'reason': 'RuntimeError: Nonterminal measurement.'}
{'path': [6, 5, 4, 3, 2], 'direction': -1, 'scheduling': 'ALAP', 'dd': False, 'valid': True, 'robust_score': 0.29569583308859937, 'mean_depth_per_circuit': 70.23529411764706, 'summed_depth_across_circuits': 2388, 'summed_native_entanglers': 340, 'reason': None}
{'path': [6, 5, 4, 3, 2], 'direction': -1, 'scheduling': 'ALAP', 'dd': True, 'valid': True, 'robust_score': 0.29528166605555334, 'mean_depth_per_circuit': 70.23529411764706, 'summed_depth_across_circuits': 2388, 'summed_native_entanglers': 340, 'reason': None}
{'path': [102, 103, 104, 105, 117], 'direction': 1, 'scheduling': 'ASAP', 'dd': False, 'valid': True, 'robust_score': 0.7261840716295179, 'mean_depth_per_circuit': 74.17647058823529, 'summed_depth_across_circuits': 2522, 'summed_native_entanglers': 340, 'reason': None}
{'path': [102, 103, 104, 105, 117], 'direction': 1, 'scheduling': 'ASAP', 'dd': True, 'valid': False, 'robust_score': None, 'mean_depth_per_circuit': None, 'summed_depth_across_circuits': None, 'summed_native_entanglers': None, 'reason': 'RuntimeError: Nonterminal measurement.'}
{'path': [102, 103, 104, 105, 117], 'direction': 1, 'scheduling': 'ALAP', 'dd': False, 'valid': True, 'robust_score': 0.7788156543539245, 'mean_depth_per_circuit': 74.17647058823529, 'summed_depth_across_circuits': 2522, 'summed_native_entanglers': 340, 'reason': None}
{'path': [102, 103, 104, 105, 117], 'direction': 1, 'scheduling': 'ALAP', 'dd': True, 'valid': True, 'robust_score': 0.8434036513257084, 'mean_depth_per_circuit': 74.17647058823529, 'summed_depth_across_circuits': 2522, 'summed_native_entanglers': 340, 'reason': None}
{'path': [102, 103, 104, 105, 117], 'direction': -1, 'scheduling': 'ASAP', 'dd': False, 'valid': True, 'robust_score': 0.8827196509296913, 'mean_depth_per_circuit': 70.6470588235294, 'summed_depth_across_circuits': 2402, 'summed_native_entanglers': 340, 'reason': None}
{'path': [102, 103, 104, 105, 117], 'direction': -1, 'scheduling': 'ASAP', 'dd': True, 'valid': False, 'robust_score': None, 'mean_depth_per_circuit': None, 'summed_depth_across_circuits': None, 'summed_native_entanglers': None, 'reason': 'RuntimeError: Nonterminal measurement.'}
{'path': [102, 103, 104, 105, 117], 'direction': -1, 'scheduling': 'ALAP', 'dd': False, 'valid': True, 'robust_score': 0.7054887189643585, 'mean_depth_per_circuit': 70.6470588235294, 'summed_depth_across_circuits': 2402, 'summed_native_entanglers': 340, 'reason': None}
{'path': [102, 103, 104, 105, 117], 'direction': -1, 'scheduling': 'ALAP', 'dd': True, 'valid': True, 'robust_score': 0.704608076002429, 'mean_depth_per_circuit': 70.6470588235294, 'summed_depth_across_circuits': 2402, 'summed_native_entanglers': 340, 'reason': None}
{'path': [117, 105, 104, 103, 102], 'direction': 1, 'scheduling': 'ASAP', 'dd': False, 'valid': True, 'robust_score': 0.9181975877522687, 'mean_depth_per_circuit': 73.55882352941177, 'summed_depth_across_circuits': 2501, 'summed_native_entanglers': 340, 'reason': None}
{'path': [117, 105, 104, 103, 102], 'direction': 1, 'scheduling': 'ASAP', 'dd': True, 'valid': False, 'robust_score': None, 'mean_depth_per_circuit': None, 'summed_depth_across_circuits': None, 'summed_native_entanglers': None, 'reason': 'RuntimeError: Nonterminal measurement.'}
{'path': [117, 105, 104, 103, 102], 'direction': 1, 'scheduling': 'ALAP', 'dd': False, 'valid': True, 'robust_score': 1.1361758550655574, 'mean_depth_per_circuit': 73.55882352941177, 'summed_depth_across_circuits': 2501, 'summed_native_entanglers': 340, 'reason': None}
{'path': [117, 105, 104, 103, 102], 'direction': 1, 'scheduling': 'ALAP', 'dd': True, 'valid': True, 'robust_score': 1.1580038919362738, 'mean_depth_per_circuit': 73.55882352941177, 'summed_depth_across_circuits': 2501, 'summed_native_entanglers': 340, 'reason': None}
{'path': [117, 105, 104, 103, 102], 'direction': -1, 'scheduling': 'ASAP', 'dd': False, 'valid': True, 'robust_score': 1.3519343511401671, 'mean_depth_per_circuit': 70.11764705882354, 'summed_depth_across_circuits': 2384, 'summed_native_entanglers': 340, 'reason': None}

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

{'path': [117, 105, 104, 103, 102], 'direction': -1, 'scheduling': 'ASAP', 'dd': True, 'valid': False, 'robust_score': None, 'mean_depth_per_circuit': None, 'summed_depth_across_circuits': None, 'summed_native_entanglers': None, 'reason': 'RuntimeError: Nonterminal measurement.'}
{'path': [117, 105, 104, 103, 102], 'direction': -1, 'scheduling': 'ALAP', 'dd': False, 'valid': True, 'robust_score': 0.958585356900099, 'mean_depth_per_circuit': 70.11764705882354, 'summed_depth_across_circuits': 2384, 'summed_native_entanglers': 340, 'reason': None}
{'path': [117, 105, 104, 103, 102], 'direction': -1, 'scheduling': 'ALAP', 'dd': True, 'valid': True, 'robust_score': 0.9514119788672986, 'mean_depth_per_circuit': 70.11764705882354, 'summed_depth_across_circuits': 2384, 'summed_native_entanglers': 340, 'reason': None}

FULL WORKLOAD:

{'batch': 0, 'status': 'planned', 'job_id': None, 'publications': 32, 'shots': 32768, 'kinds': ['assignment'], 'blocks_zero_based': [-1]}
{'batch': 1, 'status': 'planned', 'job_id': None, 'publications': 2, 'shots': 512, 'kinds': ['diagnostic'], 'blocks_zero_based': [-1]}
{'batch': 2, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 6960, 'kinds': ['science'], 'blocks_zero_based': [0]}
{'batch': 3, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 14320, 'kinds': ['science'], 'blocks_zero_based': [0]}
{'batch': 4, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 7056, 'kinds': ['science'], 'blocks_zero_based': [0]}
{'batch': 5, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 22768, 'kinds': ['science'], 'blocks_zero_based': [0]}
{'batch': 6, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 6448, 'kinds': ['science'], 'blocks_zero_based': [0]}
{'batch': 7, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 7712, 'kinds': ['science'], 'blocks_zero_based': [0]}
{'batch': 8, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 10752, 'kinds': ['science'], 'blocks_zero_based': [0]}
{'batch': 9, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 13056, 'kinds': ['science'], 'blocks_zero_based': [0]}
{'batch': 10, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 7888, 'kinds': ['science'], 'blocks_zero_based': [0]}
{'batch': 11, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 7312, 'kinds': ['science'], 'blocks_zero_based': [0]}
{'batch': 12, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 13552, 'kinds': ['science'], 'blocks_zero_based': [0]}
{'batch': 13, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 11408, 'kinds': ['science'], 'blocks_zero_based': [0]}
{'batch': 14, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 18384, 'kinds': ['science'], 'blocks_zero_based': [0]}
{'batch': 15, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 4816, 'kinds': ['science'], 'blocks_zero_based': [0]}
{'batch': 16, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 8752, 'kinds': ['science'], 'blocks_zero_based': [0]}
{'batch': 17, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 7856, 'kinds': ['science'], 'blocks_zero_based': [0]}
{'batch': 18, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 14368, 'kinds': ['science'], 'blocks_zero_based': [0]}
{'batch': 19, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 4416, 'kinds': ['science'], 'blocks_zero_based': [0]}

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

{'batch': 20, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 5936, 'kinds': ['science'], 'blocks_zero_based': [0]}
{'batch': 21, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 12656, 'kinds': ['science'], 'blocks_zero_based': [0]}
{'batch': 22, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 6544, 'kinds': ['science'], 'blocks_zero_based': [0]}
{'batch': 23, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 11808, 'kinds': ['science'], 'blocks_zero_based': [0]}
{'batch': 24, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 11344, 'kinds': ['science'], 'blocks_zero_based': [0]}
{'batch': 25, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 9792, 'kinds': ['science'], 'blocks_zero_based': [0]}
{'batch': 26, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 5984, 'kinds': ['science'], 'blocks_zero_based': [0]}
{'batch': 27, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 4080, 'kinds': ['science'], 'blocks_zero_based': [0]}
{'batch': 28, 'status': 'planned', 'job_id': None, 'publications': 2, 'shots': 1824, 'kinds': ['science'], 'blocks_zero_based': [0]}
{'batch': 29, 'status': 'planned', 'job_id': None, 'publications': 32, 'shots': 4096, 'kinds': ['validation'], 'blocks_zero_based': [0]}
{'batch': 30, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 10176, 'kinds': ['science'], 'blocks_zero_based': [1]}
{'batch': 31, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 7504, 'kinds': ['science'], 'blocks_zero_based': [1]}
{'batch': 32, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 5904, 'kinds': ['science'], 'blocks_zero_based': [1]}
{'batch': 33, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 6608, 'kinds': ['science'], 'blocks_zero_based': [1]}
{'batch': 34, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 10304, 'kinds': ['science'], 'blocks_zero_based': [1]}
{'batch': 35, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 7456, 'kinds': ['science'], 'blocks_zero_based': [1]}
{'batch': 36, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 9008, 'kinds': ['science'], 'blocks_zero_based': [1]}
{'batch': 37, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 14128, 'kinds': ['science'], 'blocks_zero_based': [1]}
{'batch': 38, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 8896, 'kinds': ['science'], 'blocks_zero_based': [1]}
{'batch': 39, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 12384, 'kinds': ['science'], 'blocks_zero_based': [1]}
{'batch': 40, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 18352, 'kinds': ['science'], 'blocks_zero_based': [1]}
{'batch': 41, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 7760, 'kinds': ['science'], 'blocks_zero_based': [1]}
{'batch': 42, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 4400, 'kinds': ['science'], 'blocks_zero_based': [1]}
{'batch': 43, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 11744, 'kinds': ['science'], 'blocks_zero_based': [1]}
{'batch': 44, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 9504, 'kinds': ['science'], 'blocks_zero_based': [1]}
{'batch': 45, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 11616, 'kinds': ['science'],

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

'blocks_zero_based': [1]}
{'batch': 46, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 12304, 'kinds': ['science'],
'blocks_zero_based': [1]}
{'batch': 47, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 3392, 'kinds': ['science'],
'blocks_zero_based': [1]}
{'batch': 48, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 6016, 'kinds': ['science'],
'blocks_zero_based': [1]}
{'batch': 49, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 12736, 'kinds': ['science'],
'blocks_zero_based': [1]}
{'batch': 50, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 13424, 'kinds': ['science'],
'blocks_zero_based': [1]}
{'batch': 51, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 7328, 'kinds': ['science'],
'blocks_zero_based': [1]}
{'batch': 52, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 11952, 'kinds': ['science'],
'blocks_zero_based': [1]}
{'batch': 53, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 7808, 'kinds': ['science'],
'blocks_zero_based': [1]}
{'batch': 54, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 11072, 'kinds': ['science'],
'blocks_zero_based': [1]}
{'batch': 55, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 11088, 'kinds': ['science'],
'blocks_zero_based': [1]}
{'batch': 56, 'status': 'planned', 'job_id': None, 'publications': 2, 'shots': 4928, 'kinds': ['science'],
'blocks_zero_based': [1]}
{'batch': 57, 'status': 'planned', 'job_id': None, 'publications': 32, 'shots': 4096, 'kinds': ['validation'],
'blocks_zero_based': [1]}
{'batch': 58, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 10896, 'kinds': ['science'],
'blocks_zero_based': [2]}
{'batch': 59, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 8432, 'kinds': ['science'],
'blocks_zero_based': [2]}
{'batch': 60, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 6064, 'kinds': ['science'],
'blocks_zero_based': [2]}
{'batch': 61, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 7584, 'kinds': ['science'],
'blocks_zero_based': [2]}
{'batch': 62, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 7392, 'kinds': ['science'],
'blocks_zero_based': [2]}
{'batch': 63, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 17776, 'kinds': ['science'],
'blocks_zero_based': [2]}
{'batch': 64, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 7040, 'kinds': ['science'],
'blocks_zero_based': [2]}
{'batch': 65, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 15888, 'kinds': ['science'],
'blocks_zero_based': [2]}
{'batch': 66, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 8896, 'kinds': ['science'],
'blocks_zero_based': [2]}
{'batch': 67, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 13536, 'kinds': ['science'],
'blocks_zero_based': [2]}
{'batch': 68, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 6032, 'kinds': ['science'],
'blocks_zero_based': [2]}
{'batch': 69, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 12784, 'kinds': ['science'],
'blocks_zero_based': [2]}
{'batch': 70, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 15840, 'kinds': ['science'],
'blocks_zero_based': [2]}

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

{'batch': 71, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 10288, 'kinds': ['science'], 'blocks_zero_based': [2]}
{'batch': 72, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 7824, 'kinds': ['science'], 'blocks_zero_based': [2]}
{'batch': 73, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 6016, 'kinds': ['science'], 'blocks_zero_based': [2]}
{'batch': 74, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 4688, 'kinds': ['science'], 'blocks_zero_based': [2]}
{'batch': 75, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 8768, 'kinds': ['science'], 'blocks_zero_based': [2]}
{'batch': 76, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 12736, 'kinds': ['science'], 'blocks_zero_based': [2]}
{'batch': 77, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 19008, 'kinds': ['science'], 'blocks_zero_based': [2]}
{'batch': 78, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 4400, 'kinds': ['science'], 'blocks_zero_based': [2]}
{'batch': 79, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 10432, 'kinds': ['science'], 'blocks_zero_based': [2]}
{'batch': 80, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 3920, 'kinds': ['science'], 'blocks_zero_based': [2]}
{'batch': 81, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 15664, 'kinds': ['science'], 'blocks_zero_based': [2]}
{'batch': 82, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 6064, 'kinds': ['science'], 'blocks_zero_based': [2]}
{'batch': 83, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 6544, 'kinds': ['science'], 'blocks_zero_based': [2]}
{'batch': 84, 'status': 'planned', 'job_id': None, 'publications': 2, 'shots': 3280, 'kinds': ['science'], 'blocks_zero_based': [2]}
{'batch': 85, 'status': 'planned', 'job_id': None, 'publications': 32, 'shots': 4096, 'kinds': ['validation'], 'blocks_zero_based': [2]}
{'batch': 86, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 11792, 'kinds': ['science'], 'blocks_zero_based': [3]}
{'batch': 87, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 17520, 'kinds': ['science'], 'blocks_zero_based': [3]}
{'batch': 88, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 13104, 'kinds': ['science'], 'blocks_zero_based': [3]}
{'batch': 89, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 14208, 'kinds': ['science'], 'blocks_zero_based': [3]}
{'batch': 90, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 8096, 'kinds': ['science'], 'blocks_zero_based': [3]}
{'batch': 91, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 7488, 'kinds': ['science'], 'blocks_zero_based': [3]}
{'batch': 92, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 9136, 'kinds': ['science'], 'blocks_zero_based': [3]}
{'batch': 93, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 7808, 'kinds': ['science'], 'blocks_zero_based': [3]}
{'batch': 94, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 5824, 'kinds': ['science'], 'blocks_zero_based': [3]}
{'batch': 95, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 6608, 'kinds': ['science'], 'blocks_zero_based': [3]}
{'batch': 96, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 20096, 'kinds': ['science'],

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

'blocks_zero_based': [3]}
{'batch': 97, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 3568, 'kinds': ['science'],
'blocks_zero_based': [3]}
{'batch': 98, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 6336, 'kinds': ['science'],
'blocks_zero_based': [3]}
{'batch': 99, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 9808, 'kinds': ['science'],
'blocks_zero_based': [3]}
{'batch': 100, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 9472, 'kinds': ['science'],
'blocks_zero_based': [3]}
{'batch': 101, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 5840, 'kinds': ['science'],
'blocks_zero_based': [3]}
{'batch': 102, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 13376, 'kinds': ['science'],
'blocks_zero_based': [3]}
{'batch': 103, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 8768, 'kinds': ['science'],
'blocks_zero_based': [3]}
{'batch': 104, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 11536, 'kinds': ['science'],
'blocks_zero_based': [3]}
{'batch': 105, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 8144, 'kinds': ['science'],
'blocks_zero_based': [3]}
{'batch': 106, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 5520, 'kinds': ['science'],
'blocks_zero_based': [3]}
{'batch': 107, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 16032, 'kinds': ['science'],
'blocks_zero_based': [3]}
{'batch': 108, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 12656, 'kinds': ['science'],
'blocks_zero_based': [3]}
{'batch': 109, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 5248, 'kinds': ['science'],
'blocks_zero_based': [3]}
{'batch': 110, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 8304, 'kinds': ['science'],
'blocks_zero_based': [3]}
{'batch': 111, 'status': 'planned', 'job_id': None, 'publications': 64, 'shots': 11360, 'kinds': ['science'],
'blocks_zero_based': [3]}
{'batch': 112, 'status': 'planned', 'job_id': None, 'publications': 2, 'shots': 144, 'kinds': ['science'],
'blocks_zero_based': [3]}
{'batch': 113, 'status': 'planned', 'job_id': None, 'publications': 32, 'shots': 4096, 'kinds': ['validation'],
'blocks_zero_based': [3]}
{'batch': 114, 'status': 'planned', 'job_id': None, 'publications': 32, 'shots': 32768, 'kinds': ['assignment'],
'blocks_zero_based': [4]}
{'backend': 'ibm_marrakesh', 'blocks': 4, 'states': ['COBYLA', 'SPSA'], 'groups': 17, 'randomizations': 16,
'fold_factors': [1, 3, 5], 'budget_by_kind': Counter({'science': 1031168, 'assignment': 65536, 'validation': 16384,
'diagnostic': 512}), 'total_shots': 1113600, 'shot_cap': 1114112, 'circuit_executions': 6858,
'unique_saved_circuits': 6596, 'sampler_jobs': 115, 'evidence_directory':
'/content/hydrogen_accuracy_20261015T174607_529559Z_fb1c49bf'}
No optimizer or Estimator. Server DD/twirling will be disabled.
Pulse additions, RC splits, folds, calibration, validation and diagnostics are counted.
Within a job, exact device execution order is not asserted; factors are mixed in pubs.
New submissions remaining: 115 shots: 1113600
/tmp/ipykernel_8303/1911544134.py:1678: DeprecationWarning: The SamplerV2 class is deprecated as of qiskit-ibm-runtime 0.50.0 and will be removed no sooner than 3 months after the release date. `qiskit_ibm_runtime.executor_sampler.Sampler` provides a new Sampler implementation that runs all the pre- and post-processing in the client side. The top-level `qiskit_ibm_runtime.Sampler` import will be updated to point to the client-side sampler in an upcoming release, with the legacy Sampler being available until the

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
deprecation is completed.
  hardware_sampler = SamplerV2(mode=backend, options=options)
Type exactly: AUTHORIZE ibm_marrakesh 1113600
> AUTHORIZE ibm_marrakesh 1113600
Saved db8hdi6il99s73fpvc10 batch 0
Saved db8he9araffc73fccj20 batch 1
/tmp/ipykernel_8303/1911544134.py:1721: DeprecationWarning: Specifying different 'shots' across pubs is deprecated as of qiskit-ibm-runtime 0.48.0 and will be removed no sooner than 3 months after the release date. Submit one job for each desired shot count instead. To reduce overhead, Consider submitting these jobs inside a Batch execution mode.
  job = hardware_sampler.run(pubs)
Saved db8hf1peukqs73aqk2k0 batch 2
Saved db8hfpheukqs73aqk3n0 batch 3
Saved db8hgh5ckpjs73b10l20 batch 4
Saved db8hh8gb3g5c73eg9ke0 batch 5
Saved db8hi65ckpjs73b10o8g batch 6
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
```

Fifth-run conclusion

Run 5 passed its reported local checks and froze a large **ibm_marrakesh** backend workload of 115 Runtime jobs, 6,858 circuit instances, and 1,113,600 requested shots. It was cancelled for practical QPU-time constraints before obtaining a balanced four-block dataset or final estimator. The preserved output records seven saved-batch entries, numbered 0–6. The available log does not establish the eventual status of any additional, unlogged job. Its partial measurements are not treated as a completed accuracy test. The cancellation exposed an experimental-budget mismatch, not a demonstrated failure of the local mathematical checks; planned shots and job counts must not be presented as measured QPU usage [33].

The subsequent compact design reduced the planned workload from 115 to nine jobs, from 6,858 to 1,014 circuit instances, and from 1,113,600 to 249,984 requested shots: reductions of approximately 92.2%, 85.2%, and 77.6%, respectively. These are design/workload reductions, not measured wall-clock or QPU-time speedups. Reduced sampling and fewer randomizations also weaken statistical and repeatability evidence, which is why Run 6 is labelled screening rather than confirmation [33].

Prompt sent to Quantum Flow to reduce the QPU workload

The existing code was included with the request to keep the revision tied to this experiment rather than to parallel Quantum Flow work. The omission marker below is preserved as part of the supplied transcript, not as a claim that the embedded prompt contains a complete executable attachment.

```
Rewrite the Part II hardware-accuracy code below as a fresh compact experiment. Do not resume, reuse, or recover the cancelled 115-job run.
Preserve Part I exactly: same Hamiltonian, five-qubit encoding, COBYLA/SPSA fixed states, 63 Pauli terms, 17 QWC groups, sequential 10-entangler state preparation, deterministic validation, negative controls, readout correction, and real-QPU execution on ibm_marrakesh.
The previous workload of 115 Runtime jobs / 1,113,600 shots was operationally excessive and was cancelled.
Optimize explicitly for limited IBM QPU time.
Change the experiment to:
- blocks = 2
- randomizations = 4
- ZNE factors remain [1, 3, 5]
- total QPU shot cap = 250,000
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

- assignment calibration = 512 shots per prepared state per epoch
- held-out validation = 64 shots per prepared state per block
- diagnostics = 128 shots
- preserve covariance-aware science-shot allocation
- increase the per-job publication limit from 64 to approximately 256, provided the complete job satisfies IBM Runtime limits

With these settings the expected scale should be roughly:
- ~1,000 circuit executions rather than 6,858
- ~9 Runtime jobs rather than 115
- ≤250k total shots

Before authorization, print the exact calculated number of jobs, circuit executions and shots. Abort automatically if jobs exceed 12 or shots exceed 250,000.
Keep both states and all 17 groups. Do not prune Hamiltonian terms or reduce scientific scope merely to save shots.
Keep the raw baseline plus candidate factor-1/3/5 measurements. Preserve readout-corrected and raw estimates separately.
Use four randomized compilations per group/factor as the screening ensemble. Do not expand to 16 randomizations automatically.
Retain the linear ZNE estimator, but label this as a screening experiment. If three-factor ZNE diagnostics are statistically inconclusive because of the smaller dataset, report that honestly rather than increasing the workload automatically.
Use only two balanced, interleaved hardware blocks. Randomize/interleave state, group and fold-factor order to reduce time-order bias.
Preserve sampling, readout-calibration, randomization and between-block uncertainties separately.
Do not claim the ±0.05 Ha target unless the resulting confidence interval actually satisfies it.
Return one complete runnable cell. This is a new experiment; do not reference or retrieve the cancelled runtime evidence.

Previous code:
```
<code part ommited>
```

Sixth code version (verbatim)

```
# Initial code produced by Stellora.AI Quantum Flow
#
# FRESH COMPACT SCREENING EXPERIMENT.
# Does not read, resume, recover, or reference any cancelled experiment artifacts.
#
# Required environment:
#   qiskit==2.5.2
#   qiskit-ibm-runtime==0.50.0
#   numpy, scipy
#
# Design:
#   2 blocks; 4 independent randomized compilations/group/factor;
#   full Hamiltonian; baseline + candidate factors 1,3,5;
#   512 assignment shots/state/epoch; 64 held-out shots/state/block;
#   2 physical diagnostics x 128 shots.
#
# Expected, before additional conservative job-limit splitting:
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
#   1,014 circuit executions; 9 Runtime jobs; 249,984 shots.
# The exact constructed workload is printed and checked before authorization.
#
# No automatic retries, extra randomizations, extra shots, or retrospective
# estimator selection. A failed/inconclusive screening result is reported honestly.

def _fresh_compact_hydrogen_screen(
    H, r, h, n_interior, R_max, E_exact, E0_fd,
    alpha_cobyla, alpha_spsa, normalized_trial_state,
):
    import io
    import os
    import json
    import math
    import uuid
    import hashlib
    import inspect
    import marshal
    import getpass
    from pathlib import Path
    from datetime import datetime, timezone
    from collections import Counter, defaultdict
    from importlib.metadata import version

    import numpy as np
    import scipy
    from scipy.stats import chi2
    import qiskit
    from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister, qpy
    from qiskit.circuit.library import UnitaryGate, XGate
    from qiskit.quantum_info import Operator, SparsePauliOp, Statevector
    from qiskit.primitives import StatevectorSampler
    from qiskit.primitives.containers import SamplerPub
    from qiskit.transpiler import PassManager
    from qiskit.transpiler.preset_passmanagers import generate_preset_pass_manager
    from qiskit.transpiler.passes import (
        TimeUnitConversion,
        ASAPScheduleAnalysis,
        ALAPScheduleAnalysis,
        ConstrainedReschedule,
        PadDelay,
        PadDynamicalDecoupling,
    )
    from qiskit_ibm_runtime import QiskitRuntimeService, SamplerV2
    from qiskit_ibm_runtime.accounts.exceptions import AccountNotFoundError

    CFG = {
        "experiment": "fresh_compact_screening",
        "backend": "ibm_marrakesh",
        "channel": "ibm_quantum_platform",
        "register": "hydrogen",
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    "blocks": 2,
    "randomizations": 4,
    "factors": [1, 3, 5],
    "shot_cap": 250_000,
    "job_cap": 12,
    "pub_limit": 256,
    "assignment_shots": 512,
    "heldout_shots": 64,
    "diagnostic_shots": 128,
    "local_shots": 4096,
    "bootstrap_replicates": 600,
    "seed": 941037,
    "transpiler_seed": 731,
    "approximation_degree": 1.0,
    "matrix_tolerance": 1e-12,
    "probability_tolerance": 1e-10,
    "energy_tolerance": 1e-10,
    "Schmidt_tail_tolerance": 1e-12,
    "readout_condition_limit": 10.0,
    "readout_minimum_singular_value": 0.10,
    "heldout_TV_limit": 0.15,
    "heldout_block_TV_limit": 0.30,
    "corrected_heldout_TV_limit": 0.30,
    "ZNE_lack_of_fit_p_min": 0.01,
    "ZNE_amplification_limit": 5.0,
    "model_sensitivity_floor_Ha": 0.025,
    "diagnostic_precision_Ha": 0.05,
    "engineering_target_Ha": 0.05,
    "block_heterogeneity_p_min": 0.01,
    # Conservative client-side bounds, NOT claims of universal IBM limits.
    "client_job_QPY_bytes_limit": 16 * 1024 * 1024,
    "client_job_instruction_limit": 1_000_000,
    "client_job_circuit_shots_limit": 1_000_000,
    "path_beam_width": 2000,
    "compiled_path_shortlist": 2,
}
NQ, DIM, B, R = 5, 32, 2, 4
NAMES = ["COBYLA", "SPSA"]
MTOL = CFG["matrix_tolerance"]
PTOL = CFG["probability_tolerance"]
ETOL = CFG["energy_tolerance"]
LINEAR = np.array([13 / 12, 1 / 3, -5 / 12], float)
QUADRATIC = np.array([15 / 8, -5 / 4, 3 / 8], float)

def require(ok, message):
    if not bool(ok):
        raise RuntimeError(message)

def real(value, label):
    value = complex(value)
    require(
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        np.isfinite(value.real) and np.isfinite(value.imag)
        and abs(value.imag) <= MTOL,
        f"{label}: nonfinite or unexpectedly complex.",
    )
    return float(value.real)

def rng(*coordinates):
    return np.random.default_rng(
        np.random.SeedSequence([CFG["seed"], *map(int, coordinates)])
    )

def utc():
    return datetime.now(timezone.utc).isoformat()

def sha(payload):
    return hashlib.sha256(payload).hexdigest()

def array_hash(value):
    a = np.ascontiguousarray(value)
    return sha(str(a.dtype).encode() + str(a.shape).encode() + a.tobytes())

def js(value):
    if isinstance(value, np.ndarray):
        return js(value.tolist())
    if isinstance(value, np.generic):
        return js(value.item())
    if isinstance(value, complex):
        return {"real": value.real, "imag": value.imag}
    if isinstance(value, dict):
        return {str(k): js(v) for k, v in value.items()}
    if isinstance(value, (tuple, list)):
        return [js(v) for v in value]
    if isinstance(value, (Path, datetime)):
        return str(value)
    if isinstance(value, float) and not math.isfinite(value):
        return None
    if value is None or isinstance(value, (str, int, float, bool)):
        return value
    return repr(value)

# ---------------------- Immutable authoritative Part I ------------------
Hc = np.array(H, dtype=complex, copy=True)
rc = np.array(r, dtype=float, copy=True)
require(Hc.shape == (32, 32), "H must be 32 x 32.")
require(rc.shape == (32,), "r must have exactly 32 elements.")
require(np.all(np.isfinite(Hc)) and np.all(np.isfinite(rc)), "Nonfinite Part I data.")
require(np.max(np.abs(Hc - Hc.conj().T)) <= MTOL, "H is not Hermitian.")
require(int(n_interior) == 32, "n_interior must be 32.")
require(np.isfinite(h) and float(h) > 0, "Invalid h.")
require(np.isfinite(R_max) and float(R_max) > 0, "Invalid R_max.")
require(callable(normalized_trial_state), "Invalid trial-state function.")
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
alphas = [alpha_cobyla, alpha_spsa]
vectors, references = [], []
for s, alpha in enumerate(alphas):
    require(
        np.ndim(alpha) == 0 and np.isfinite(alpha) and alpha > 0,
        f"{NAMES[s]} alpha must be finite and positive.",
    )
    vector = np.array(
        normalized_trial_state(alpha, rc.copy()), dtype=complex, copy=True
    )
    require(vector.shape == (32,) and np.all(np.isfinite(vector)), "Invalid target.")
    require(abs(real(np.vdot(vector, vector), "Norm") - 1) <= MTOL, "Invalid norm.")
    vectors.append(vector)
    references.append(real(np.vdot(vector, Hc @ vector), "Reference energy"))
references = np.array(references)
continuum = real(E_exact, "E_exact")
fd_reference = real(E0_fd, "E0_fd")

versions = {
    "numpy": np.__version__, "scipy": scipy.__version__,
    "qiskit": qiskit.__version__,
    "qiskit-ibm-runtime": version("qiskit-ibm-runtime"),
}
require(versions["qiskit"] == "2.5.2", "Requires Qiskit 2.5.2.")
require(
    versions["qiskit-ibm-runtime"] == "0.50.0",
    "Requires qiskit-ibm-runtime 0.50.0.",
)

# Fresh directory only. No old directory is opened or searched.
directory = Path.cwd() / (
    "hydrogen_compact_screen_" +
    datetime.now(timezone.utc).strftime("%Y%m%dT%H%M%S_%fZ") +
    "_" + uuid.uuid4().hex[:8]
)
directory.mkdir(mode=0o700, exist_ok=False)
(directory / "circuits").mkdir(mode=0o700)
(directory / "jobs").mkdir(mode=0o700)

def save_json(relative, value):
    path = directory / relative
    temporary = path.with_name("." + path.name + "." + uuid.uuid4().hex)
    with temporary.open("x", encoding="utf-8") as handle:
        os.chmod(temporary, 0o600)
        json.dump(js(value), handle, indent=2, allow_nan=False)
        handle.flush()
        os.fsync(handle.fileno())
    os.replace(temporary, path)

def save_bytes(relative, payload):
    path = directory / relative
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    with path.open("xb") as handle:
        os.chmod(path, 0o600)
        handle.write(payload)
        handle.flush()
        os.fsync(handle.fileno())
    return sha(payload)

frame = inspect.currentframe()
own_code = frame.f_code
del frame
source_record = {
    "function_code_object_sha256": sha(marshal.dumps(own_code)),
    "trial_code_object_sha256": (
        sha(marshal.dumps(normalized_trial_state.__code__))
        if hasattr(normalized_trial_state, "__code__") else None
    ),
    "function_source_sha256": None,
    "qualification": (
        "Code-object hashes are Python-version-specific. No previous "
        "notebook history or experiment source is read."
    ),
}
try:
    own_source = inspect.getsource(own_code)
except (OSError, TypeError):
    own_source = None
if own_source:
    source_record["function_source_sha256"] = save_bytes(
        "fresh_function_source.py", own_source.encode()
    )

manifest = {
    "schema": "compact-screen-1",
    "created_utc": utc(),
    "configuration": CFG,
    "versions": versions,
    "source": source_record,
    "authority": {
        "H_sha256": array_hash(Hc), "r_sha256": array_hash(rc),
        "state_sha256": [array_hash(v) for v in vectors],
        "alpha_full_precision_hex": [float(a).hex() for a in alphas],
        "references_Ha": references,
        "continuum_reference_Ha": continuum,
        "finite_reference_Ha": fd_reference,
        "h": float(h), "R_max": float(R_max), "n_interior": int(n_interior),
    },
    "status": "local_validation",
    "registry": {}, "plan": [], "jobs": [],
    "primary_estimator": "Readout-corrected candidate linear OLS ZNE intercept.",
    "linear_weights": LINEAR,
    "quadratic_sensitivity_weights": QUADRATIC,
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    "scope": "SCREENING; two blocks and four randomizations only.",
    "no_automatic_expansion": True,
}
save_json("manifest.json", manifest)
np.savez_compressed(
    directory / "inputs.npz", H=Hc, r=rc,
    COBYLA=vectors[0], SPSA=vectors[1],
)
os.chmod(directory / "inputs.npz", 0o600)

# ---------- Complete Hamiltonian / fixed groups ----------
pauli = SparsePauliOp.from_operator(Operator(Hc), atol=0, rtol=0)
terms = pauli.to_list()
require(len(terms) == 63, "Expected the complete 63-term Hamiltonian.")
require(len(set(label for label, c in terms)) == 63, "Duplicate Pauli labels.")
for label, coefficient in terms:
    real(coefficient, label)
reconstruction_error = float(np.max(np.abs(pauli.to_matrix() - Hc)))
require(reconstruction_error <= MTOL, "Hamiltonian reconstruction failed.")
operators = list(pauli.group_commuting(qubit_wise=True))
require(len(operators) == 17, "Expected all 17 QWC groups.")
grouped = [term for op in operators for term in op.to_list()]
require(Counter(label for label, c in grouped) == Counter(label for label, c in terms),
        "Grouped terms missing or duplicated.")
coefficient_map = dict(terms)
require(all(c == coefficient_map[label] for label, c in grouped),
        "Grouping altered a coefficient.")
require(sum(label == "IIIII" for label, c in grouped) == 1,
        "Identity must occur exactly once.")

groups = []
for g, op in enumerate(operators):
    axes, values = ["I"] * NQ, np.zeros(DIM)
    for label, coefficient in op.to_list():
        mask = 0
        for q in range(NQ):
            axis = label[NQ - 1 - q]
            if axis != "I":
                require(axes[q] in ("I", axis), "Invalid QWC group.")
                axes[q] = axis
                mask |= 1 << q
        values += real(coefficient, label) * np.array([
            -1.0 if (k & mask).bit_count() % 2 else 1.0 for k in range(DIM)
        ])
    axes = ["Z" if axis == "I" else axis for axis in axes]
    groups.append({
        "index": g, "axes": axes, "basis": "".join(reversed(axes)),
        "terms": op.to_list(), "values": values,
    })
G = len(groups)
require(
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    [g["basis"] for g in groups] == [
        "YXXXY", "XYXXY", "YXXYX", "YXYYY", "YYYXY", "XXYXY",
        "XYXYX", "XYYYY", "YXYXX", "YYXYY", "YYYYX", "XXXYY",
        "XXYYX", "XYYXX", "YYXXX", "XXXXX", "ZZZZZ",
    ],
    "Required QWC group ordering changed.",
)

def check_state(actual, intended, label):
    overlap = np.vdot(intended, actual)
    require(abs(overlap) > 0, label + ": zero overlap.")
    error = float(np.linalg.norm(actual - overlap / abs(overlap) * intended))
    require(error <= MTOL, f"{label}: state error {error:.3e}.")
    return error

def check_p(actual, intended, label):
    actual = np.asarray(actual, float)
    require(
        actual.shape == (DIM,) and np.all(np.isfinite(actual))
        and actual.min() >= -MTOL and abs(actual.sum() - 1) <= PTOL,
        label + ": invalid probabilities.",
    )
    error = float(np.max(np.abs(actual - intended)))
    require(error <= PTOL, f"{label}: probability error {error:.3e}.")
    return error

# Exact supplied complex-SVD sequential recipe; four adjacent two-qubit
# isometry completions and one single-qubit completion. No new ansatz.
def sequential(vector, direction):
    wire_order = list(range(NQ)) if direction == 1 else list(reversed(range(NQ)))
    perm = np.array([
        sum(((k >> j) & 1) << wire_order[j] for j in range(NQ))
        for k in range(DIM)
    ])
    working = vector[perm]
    cut_records = []
    for cut in range(1, NQ):
        singular = np.linalg.svd(
            working.reshape(1 << cut, 1 << (NQ - cut), order="F"),
            compute_uv=False,
        )
        tail = float(np.linalg.norm(singular[2:]))
        require(tail <= CFG["Schmidt_tail_tolerance"], "Bond-two cut failed.")
        cut_records.append({"cut": cut, "singular_values": singular, "tail": tail})

    tensors, carry, right_rank = [None] * NQ, working.copy(), 1
    for q in range(4, 0, -1):
        u, singular, vh = np.linalg.svd(
            carry.reshape(1 << q, 2 * right_rank, order="F"),
            full_matrices=False,
        )
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        require(np.linalg.norm(singular[2:]) <= CFG["Schmidt_tail_tolerance"],
                "Sequential factorization would truncate the state.")
        tensors[q] = vh[:2].reshape(2, 2, right_rank, order="F")
        carry = u[:, :2] * singular[:2]
        right_rank = 2
    norm = float(np.linalg.norm(carry))
    require(abs(norm - 1) <= MTOL, "Sequential retained norm failed.")
    tensors[0] = (carry / norm).reshape(1, 2, 2, order="F")
    circuit = QuantumCircuit(NQ)
    independent = np.zeros(DIM, complex)
    independent[0] = 1
    for q, tensor in enumerate(tensors):
        left_rank, _, next_rank = tensor.shape
        iso = tensor.transpose(1, 2, 0).reshape(
            2 * next_rank, left_rank, order="F"
        )
        require(np.max(np.abs(iso.conj().T @ iso - np.eye(left_rank))) <= MTOL,
                "Sequential isometry failed.")
        _, _, vh = np.linalg.svd(iso.conj().T, full_matrices=True)
        unitary = np.column_stack((iso, vh.conj().T[:, left_rank:]))
        require(
            np.max(np.abs(unitary.conj().T @ unitary - np.eye(len(unitary)))) <= MTOL,
            "Unitary completion failed.",
        )
        wires = [q, q + 1] if q < 4 else [q]
        circuit.append(UnitaryGate(unitary), [wire_order[j] for j in wires])
        d = len(unitary)
        output = np.zeros(DIM, complex)
        for k, amplitude in enumerate(independent):
            column = (k >> q) & (d - 1)
            base = k & ~((d - 1) << q)
            for row in range(d):
                output[base | (row << q)] += unitary[row, column] * amplitude
        independent = output
    restored = np.zeros(DIM, complex)
    restored[perm] = independent
    record = {
        "direction": direction, "cuts": cut_records,
        "independent_error": check_state(restored, vector, "Independent preparation"),
        "circuit_error": check_state(
            Statevector.from_instruction(circuit).data, vector, "Circuit preparation"
        ),
    }
    return circuit, record

preparations, preparation_checks = {}, []
for s in range(2):
    for direction in (1, -1):
        preparations[s, direction], record = sequential(vectors[s], direction)
        preparation_checks.append({"state": NAMES[s], **record})
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
def logical_measurement(s, g, direction):
    circuit = QuantumCircuit(
        QuantumRegister(NQ, "q"), ClassicalRegister(NQ, CFG["register"])
    )
    circuit.compose(preparations[s, direction], inplace=True)
    for q, axis in enumerate(groups[g]["axes"]):
        if axis == "X":
            circuit.h(q)
        elif axis == "Y":
            circuit.sdg(q)
            circuit.h(q)
    for q in range(NQ):
        circuit.measure(q, q)
    return circuit

ideal = np.zeros((2, G, DIM))
ideal_group = np.zeros((2, G))
ideal_variance = np.zeros((2, G))
for s in range(2):
    require(
        abs(real(Statevector(vectors[s]).expectation_value(pauli), "Pauli energy")
            - references[s]) <= ETOL,
        "Pauli/reference mismatch.",
    )
    for g in range(G):
        direct = real(Statevector(vectors[s]).expectation_value(operators[g]), "Group")
        for direction in (1, -1):
            p = Statevector.from_instruction(
                logical_measurement(s, g, direction)
                .remove_final_measurements(inplace=False)
            ).probabilities()
            require(abs(float(p @ groups[g]["values"]) - direct) <= ETOL,
                    "Independent measurement implementation failed.")
            if direction == 1:
                ideal[s, g] = p
            else:
                check_p(p, ideal[s, g], "Reverse-direction output")
        ideal_group[s, g] = direct
        ideal_variance[s, g] = ideal[s, g] @ (
            (groups[g]["values"] - direct) ** 2
        )
    require(abs(ideal_group[s].sum() - references[s]) <= ETOL,
            "Exact group sum failed.")

def decode(counts, shots):
    frequencies = np.zeros(DIM, np.int64)
    require(isinstance(counts, dict) and bool(counts), "Missing counts.")
    for bits, number in counts.items():
        require(isinstance(bits, str) and len(bits) == NQ
                and set(bits) <= {"0", "1"}, "Unexpected returned bitstring.")
        require(isinstance(number, (int, np.integer)) and number >= 0,
```

```
            "Invalid count multiplicity.")
        frequencies[int(bits, 2)] += int(number)
    require(int(frequencies.sum()) == shots, "Actual shot count mismatch.")
    return frequencies

def returned_counts(pub):
    require(hasattr(pub.data, CFG["register"]), "Returned classical register missing.")
    bits = getattr(pub.data, CFG["register"])
    require(bits.num_bits == NQ, "Returned register width mismatch.")
    return bits.get_counts()

controls = []

def expect_failure(function, label):
    try:
        function()
    except RuntimeError:
        controls.append({"label": label, "detected": True})
    else:
        raise RuntimeError("Negative control was not detected: " + label)

bad = preparations[0, 1].copy()
bad.x(4)
expect_failure(
    lambda: check_state(Statevector.from_instruction(bad).data, vectors[0], "Bad prep"),
    "corrupted-preparation",
)
bad = preparations[0, 1].copy()
bad.h(0)
expect_failure(
    lambda: check_p(Statevector.from_instruction(bad).probabilities(),
                ideal[0, 16], "Bad basis"),
    "wrong-measurement-basis",
)
reverse_indices = np.array([int(format(k, "05b")[::-1], 2) for k in range(DIM)])
expect_failure(
    lambda: check_p(ideal[0, 16][reverse_indices], ideal[0, 16], "Bad decoder"),
    "reversed-count-decoder",
)

def safe_inverse(A):
    require(A.shape == (DIM, DIM) and np.all(np.isfinite(A))
            and A.min() >= 0
            and np.max(np.abs(A.sum(axis=0) - 1)) <= MTOL,
            "Invalid assignment matrix.")
    singular = np.linalg.svd(A, compute_uv=False)
    condition = (
        float(singular[0] / singular[-1])
        if singular[-1] > 0 else float("inf")
    )
    require(
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        condition <= CFG["readout_condition_limit"]
        and singular[-1] >= CFG["readout_minimum_singular_value"],
        "Assignment conditioning rejected.",
    )
    inverse = np.linalg.solve(A, np.eye(DIM))
    error = float(np.max(np.abs(inverse @ A - np.eye(DIM))))
    require(error <= ETOL, "Assignment numerical inverse failed.")
    return inverse, condition, float(singular[-1]), error

strengths = np.linspace(0.04, 0.12, DIM)
fixture = np.diag(1 - strengths) + np.ones((DIM, 1)) * strengths[None, :] / DIM
fixture_inverse, fixture_condition, _, _ = safe_inverse(fixture)
test_p = rng(1).dirichlet(np.arange(1, DIM + 1))
require(np.max(np.abs(fixture_inverse @ fixture @ test_p - test_p)) <= MTOL,
        "Independent readout recovery failed.")
expect_failure(lambda: safe_inverse(np.ones((DIM, DIM)) / DIM),
               "singular-assignment")
expect_failure(
    lambda: require(
        np.max(np.abs(np.linalg.solve(fixture.T, fixture @ test_p) - test_p)) <= PTOL,
        "Transposed assignment detected.",
    ), "transposed-assignment",
)

# Seeded local counts use exactly the hardware decoder and complete group scores.
local_rows = []
for s in range(2):
    energies, variances, records = [], [], []
    for g in range(G):
        seed = int(np.random.SeedSequence([CFG["seed"], 2, s, g]).generate_state(1)[0])
        sampler = StatevectorSampler(default_shots=CFG["local_shots"], seed=seed)
        pub = sampler.run(
            [logical_measurement(s, g, 1)], shots=CFG["local_shots"]
        ).result()[0]
        counts = returned_counts(pub)
        p = decode(counts, CFG["local_shots"]) / CFG["local_shots"]
        f = groups[g]["values"]
        energy = float(p @ f)
        variance = float(p @ ((f - energy) ** 2) / (CFG["local_shots"] - 1))
        energies.append(energy)
        variances.append(variance)
        records.append({"group": g, "seed": seed, "counts": counts})
    energy = math.fsum(energies)
    se = math.sqrt(math.fsum(variances))
    require(abs(energy - references[s]) <= 6 * se, "Local six-SE acceptance failed.")
    local_rows.append({
        "state": NAMES[s], "energy": energy, "sampling_se": se, "groups": records,
    })

save_json("local_validation.json", {
    "reconstruction_error": reconstruction_error,
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    "preparations": preparation_checks,
    "groups": groups, "exact_group_energies": ideal_group,
    "references": references, "local_sampling": local_rows,
    "negative_controls": controls, "synthetic_condition": fixture_condition,
})
print("SCREENING EXPERIMENT — FRESH, NOT A RESUMPTION")
print("LOCAL ACCEPTANCE: PASS")
print("Versions:", versions)
print("Fixed-state references:", dict(zip(NAMES, references)))
print("Pauli terms / groups:", len(terms), G)
print("Hamiltonian reconstruction error:", reconstruction_error)
for row in local_rows:
    print("LOCAL", row["state"], row["energy"], "+/-", row["sampling_se"], "Ha")
print("No cancelled-experiment artifacts have been opened.")

# ---------- IBM connection, after all deterministic/local checks ----
try:
    service = QiskitRuntimeService(channel=CFG["channel"])
except AccountNotFoundError:
    token = getpass.getpass("IBM token, hidden and never saved: ")
    try:
        require(bool(token.strip()), "Empty token.")
        service = QiskitRuntimeService(channel=CFG["channel"], token=token)
    finally:
        del token

backend = service.backend(CFG["backend"])
require(backend.name == CFG["backend"], "Backend substitution forbidden.")
require(backend.status().operational, "ibm_marrakesh is not operational.")
target = backend.target
width = int(backend.num_qubits)
dt = float(backend.dt)
require(dt > 0, "Missing backend timing resolution.")
properties = backend.properties()
require(properties is not None, "Current calibration properties unavailable.")
pulse_alignment = int(getattr(target, "pulse_alignment", 1) or 1)
acquire_alignment = int(getattr(target, "acquire_alignment", 1) or 1)
native_two = {
    name for name in target.operation_names
    if getattr(target.operation_from_name(name), "num_qubits", None) == 2
    and name not in ("barrier", "delay")
}
require(native_two, "Native entanglers unavailable.")
missing_calibration = set()

def qvalue(q, name):
    try:
        value = properties.qubit_property(q).get(name)
    except Exception:
        value = None
    if value is None:
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
            missing_calibration.add(f"qubit:{q}:{name}")
            return None
        return float(value[0])

    def info(name, qs):
        qs = tuple(qs)
        require(target.instruction_supported(operation_name=name, qargs=qs),
                f"Unsupported native instruction {name}{qs}.")
        prop = target[name].get(qs)
        duration = None if prop is None else prop.duration
        error = None if prop is None else prop.error
        if duration is None:
            missing_calibration.add(f"duration:{name}:{qs}")
        if error is None:
            missing_calibration.add(f"error:{name}:{qs}")
        return duration, error

    def duration(op, qs):
        if op.name == "barrier":
            return 0.0
        if op.name == "delay":
            units = {"dt": dt, "s": 1., "ms": 1e-3, "us": 1e-6, "ns": 1e-9, "ps": 1e-12}
            require(op.unit in units, "Unsupported delay unit.")
            return float(op.duration) * units[op.unit]
        d, _ = info(op.name, qs)
        require(d is not None, "Missing executed native duration.")
        return float(d)

    def manager(layout, level=2):
        return generate_preset_pass_manager(
            backend=backend, optimization_level=level, initial_layout=list(layout),
            approximation_degree=1.0, unitary_synthesis_method="default",
            seed_transpiler=CFG["transpiler_seed"],
        )

    def mapping(circuit):
        require(circuit.num_clbits == NQ and len(circuit.cregs) == 1
                and circuit.cregs[0].name == CFG["register"], "Classical register changed.")
        result = {}
        for item in circuit.data:
            if item.operation.name == "measure":
                c = circuit.find_bit(item.clbits[0]).index
                require(c not in result, "Duplicate measurement destination.")
                result[c] = circuit.find_bit(item.qubits[0]).index
        require(set(result) == set(range(NQ)) and len(set(result.values())) == NQ,
                "Invalid compiled measurement map.")
        return tuple(result[c] for c in range(NQ))

    def probabilities(circuit):
        measured_physical = mapping(circuit)
        active = sorted(measured_physical)
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    index = {q: j for j, q in enumerate(active)}
    compact = QuantumCircuit(NQ)
    compact.global_phase = circuit.global_phase
    measured = set()
    for item in circuit.data:
        op = item.operation
        qs = [circuit.find_bit(q).index for q in item.qubits]
        if op.name == "measure":
            measured.update(qs)
            continue
        if op.name in ("barrier", "delay"):
            continue
        require(set(qs) <= set(active), "Routing uses extra active physical wires.")
        require(not measured.intersection(qs), "Nonterminal active operation.")
        require(not item.clbits and getattr(op, "condition", None) is None,
                "Dynamic control unsupported.")
        require(op.name != "reset", "Unexpected reset.")
        compact.append(op.copy(), [index[q] for q in qs])
    local_p = Statevector.from_instruction(compact).probabilities()
    result = np.zeros(DIM)
    for k in range(DIM):
        logical_index = sum(
            ((k >> index[measured_physical[c]]) & 1) << c for c in range(NQ)
        )
        result[logical_index] = local_p[k]
    return result

def validate(circuit, intended):
    return check_p(probabilities(circuit), intended, "Actual submitted circuit")

def scheduled(circuit, policy, dd):
    analysis = ASAPScheduleAnalysis if policy == "ASAP" else ALAPScheduleAnalysis
    passes = [
        TimeUnitConversion(target=target),
        analysis(target=target),
        ConstrainedReschedule(
            acquire_alignment=acquire_alignment, pulse_alignment=pulse_alignment
        ),
    ]
    if dd:
        passes.append(PadDynamicalDecoupling(
            durations=backend.instruction_durations,
            dd_sequence=[XGate(), XGate()],
            spacing=[0.25, 0.5, 0.25],
            skip_reset_qubits=True,
            pulse_alignment=pulse_alignment,
            target=target,
        ))
    else:
        passes.append(PadDelay(target=target))
    return PassManager(passes).run(circuit)
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
def exposure(circuit):
    physical = mapping(circuit)
    clocks = defaultdict(float)
    measured = set()
    report = {
        q: {"active_s": 0., "idle_s": 0., "readout_s": 0.,
            "physical_pulses": 0, "virtual_operations": 0}
        for q in physical
    }
    instruction_records = Counter()
    risk = 0.0
    entanglers = 0
    for item in circuit.data:
        op = item.operation
        qs = tuple(circuit.find_bit(q).index for q in item.qubits)
        if op.name == "barrier":
            if qs:
                start = max(clocks[q] for q in qs)
                for q in qs:
                    if q in report and q not in measured:
                        report[q]["idle_s"] += max(0., start - clocks[q])
                    clocks[q] = start
            continue
        d = duration(op, qs)
        start = max((clocks[q] for q in qs), default=0.)
        for q in qs:
            if q in report and q not in measured:
                report[q]["idle_s"] += max(0., start - clocks[q])
                if op.name == "measure":
                    report[q]["readout_s"] += d
                elif op.name == "delay":
                    report[q]["idle_s"] += d
                else:
                    report[q]["active_s"] += d
                    report[q]["physical_pulses" if d > 0 else "virtual_operations"] += 1
            clocks[q] = start + d
        if op.name == "measure":
            measured.update(qs)
        if op.name not in ("delay", "barrier"):
            instruction_records[op.name, qs] += 1
            _, error = info(op.name, qs)
            require(error is not None or d == 0,
                    "Missing nonzero-duration executed-instruction error.")
            if error is not None:
                risk += float(error)
            if len(qs) == 2 and op.name in native_two:
                entanglers += 1
    for q, row in report.items():
        t1, t2 = qvalue(q, "T1"), qvalue(q, "T2")
        require(t1 is not None and t2 is not None and t1 > 0 and t2 > 0,
```

```
                "Missing T1/T2 for candidate path.")
        row.update({"T1_s": t1, "T2_s": t2})
        row["premeasurement_s"] = row["active_s"] + row["idle_s"]
        # Conservative descriptive exposure proxy, not a fitted device model.
        # Gate calibration errors and exposure are distinct score terms;
        # this is NOT represented as a physical noise probability.
        risk += row["premeasurement_s"] / t1 + row["premeasurement_s"] / t2
    return {
        "depth": circuit.depth(), "native_entanglers": entanglers,
        "duration_s": max(clocks.values(), default=0.),
        "qubits": report, "risk_proxy": risk,
        "executed_instructions": [
            {"name": name, "physical": qs, "count": count,
             "duration_s": info(name, qs)[0], "error": info(name, qs)[1]}
            for (name, qs), count in sorted(instruction_records.items())
        ],
        "qualification": (
            "Calibration/exposure proxy only; no density-matrix or exact-device "
            "prediction is claimed. Virtual operations are not physical pulses."
        ),
    }

# ---------------------- Explicit RC and exact native folds ----------------
P = [
    np.eye(2, dtype=complex),
    np.array([[0, 1], [1, 0]], complex),
    np.array([[0, -1j], [1j, 0]], complex),
    np.diag([1, -1]).astype(complex),
]
PN = ["I", "X", "Y", "Z"]
pair_paulis = [np.kron(P[b], P[a]) for a in range(4) for b in range(4)]
twirl_cache = {}

def twirl_table(op):
    U = Operator(op).data
    key = array_hash(U)
    if key not in twirl_cache:
        table = []
        for a in range(4):
            for b in range(4):
                transformed = U @ np.kron(P[b], P[a]) @ U.conj().T
                overlap = [abs(np.trace(pa.conj().T @ transformed)) / 4
                           for pa in pair_paulis]
                j = int(np.argmax(overlap))
                require(overlap[j] >= 1 - 1e-10,
                        f"Native {op.name} is not Pauli-twirlable.")
                table.append((a, b, j // 4, j % 4))
        twirl_cache[key] = table
    return twirl_cache[key]

def append_pauli(circuit, label, q):
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        if label in (2, 3):
            info("rz", (q,))
            circuit.rz(np.pi, q)
        if label in (1, 2):
            info("x", (q,))
            circuit.x(q)

    def transformed(circuit, factor, random):
        out = circuit.copy_empty_like()
        frames = []
        original_count = 0
        for item in circuit.data:
            op = item.operation
            qs = [circuit.find_bit(q).index for q in item.qubits]
            cs = [circuit.find_bit(c).index for c in item.clbits]
            if len(qs) != 2 or op.name not in native_two:
                out.append(op.copy(), qs, cs)
                continue
            original_count += 1
            inverse_op = op.inverse()
            if not target.instruction_supported(
                operation_name=inverse_op.name, qargs=tuple(qs)
            ):
                U = Operator(op).data
                phase = np.trace(U @ U) / 4
                require(abs(abs(phase) - 1) <= MTOL
                        and np.max(np.abs(U @ U - phase * np.eye(4))) <= MTOL,
                        "No exact native inverse available for folding.")
                inverse_op = op
            sequence = [op]
            for _ in range((factor - 1) // 2):
                sequence.extend([inverse_op, op])
            for executed in sequence:
                info(executed.name, tuple(qs))
                a, b, c, d = twirl_table(executed)[int(random.integers(16))]
                out.barrier(*qs)
                append_pauli(out, a, qs[0])
                append_pauli(out, b, qs[1])
                out.barrier(*qs)
                out.append(executed.copy(), qs)
                out.barrier(*qs)
                append_pauli(out, c, qs[0])
                append_pauli(out, d, qs[1])
                out.barrier(*qs)
                frames.append({
                    "native": executed.name, "physical": qs,
                    "before": [PN[a], PN[b]], "after": [PN[c], PN[d]],
                    "classical_xor": 0,
                })
        count = sum(
            len(item.qubits) == 2 and item.operation.name in native_two
```

```
        for item in out.data
    )
    require(count == factor * original_count, "Native fold count mismatch.")
    return out, frames, original_count, count

# ---------------------- Bounded fresh placement screening -----------------
adjacency = defaultdict(set)
edge_error = {}
for name in native_two:
    for pair, prop in target[name].items():
        if pair is not None and len(pair) == 2:
            a, b = pair
            adjacency[a].add(b)
            adjacency[b].add(a)
            if prop is not None and prop.error is not None:
                key = tuple(sorted(pair))
                edge_error[key] = min(edge_error.get(key, float("inf")),
                                      float(prop.error))

def rough(path):
    score = 0.
    for q in path:
        t2 = qvalue(q, "T2")
        _, readout_error = info("measure", (q,))
        if t2 is None or t2 <= 0 or readout_error is None:
            return float("inf")
        score += float(readout_error) + 1e-6 / t2
    for a, b in zip(path[:-1], path[1:]):
        score += edge_error.get(tuple(sorted((a, b))), 1.)
    return score

beam = [(rough((q,)), (q,)) for q in sorted(adjacency)]
for length in range(1, NQ):
    expanded = [
        (rough(path + (q,)), path + (q,))
        for _, path in beam
        for q in sorted(adjacency[path[-1]])
        if q not in path
    ]
    require(expanded, "No connected five-qubit path.")
    beam = sorted(expanded)[:CFG["path_beam_width"]]
unique_paths = {}
for score, path in beam:
    canonical = min(path, tuple(reversed(path)))
    unique_paths[canonical] = min(score, unique_paths.get(canonical, float("inf")))
shortlist = sorted(unique_paths, key=lambda p: (unique_paths[p], p))[
    :CFG["compiled_path_shortlist"]
]
require(shortlist and np.isfinite(unique_paths[shortlist[0]]),
        "No path with usable calibration.")
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
compile_cache, candidates, candidate_records = {}, [], []
print("Fresh local placement / orientation / schedule / DD screening...")
for path_index, canonical in enumerate(shortlist):
    for orientation in (1, -1):
        path = tuple(canonical if orientation == 1 else reversed(canonical))
        for direction in (1, -1):
            descriptor = {"path": path, "direction": direction}
            try:
                compiled = {}
                prep_counts = []
                pm = manager(path)
                for s in range(2):
                    prep = pm.run(preparations[s, direction])
                    prep_count = sum(
                        len(item.qubits) == 2 and item.operation.name in native_two
                        for item in prep.data
                    )
                    prep_counts.append(prep_count)
                    require(prep_count == 10,
                            f"Sequential preparation native count is {prep_count}, not 10.")
                    for g in range(G):
                        qc = pm.run(logical_measurement(s, g, direction))
                        validate(qc, ideal[s, g])
                        compiled[s, g] = qc
                compile_cache[path, direction] = compiled
            except Exception as error:
                candidate_records.append({
                    **descriptor, "valid": False,
                    "reason": f"{type(error).__name__}: {error}",
                })
                continue
            for policy in ("ASAP", "ALAP"):
                for dd in (False, True):
                    description = {**descriptor, "schedule": policy, "dd": dd}
                    try:
                        reports = []
                        for s in range(2):
                            for g in range(G):
                                rc_qc, _, _, _ = transformed(
                                    compiled[s, g], 1,
                                    rng(10, path_index, int(orientation == 1),
                                        int(direction == 1), s, g),
                                )
                                validate(rc_qc, ideal[s, g])
                                qc = scheduled(rc_qc, policy, dd)
                                validate(qc, ideal[s, g])
                                reports.append({"state": s, "group": g,
                                                **exposure(qc)})
                        # Group- and both-state-sensitive proxy; no signed
                        # energy cancellation or known-energy fit is used.
                        scores = []
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
                    for s in range(2):
                        state_reports = [x for x in reports if x["state"] == s]
                        weighted = [
                            x["risk_proxy"] *
                            max(0.05, float(np.ptp(groups[x["group"]]["values"])))
                            for x in state_reports
                        ]
                        scores.append(sum(weighted) + 2 * max(weighted))
                    score = max(scores)
                    record = {
                        **description, "valid": True, "score": score,
                        "preparation_native_counts": prep_counts,
                        "mean_depth_per_measurement_circuit": float(np.mean(
                            [x["depth"] for x in reports]
                        )),
                        "summed_depth_across_measurement_circuits": sum(
                            x["depth"] for x in reports
                        ),
                        "reports": reports,
                    }
                    candidate_records.append(record)
                    candidates.append((score, description))
                except Exception as error:
                    candidate_records.append({
                        **description, "valid": False,
                        "reason": f"{type(error).__name__}: {error}",
                    })
save_json("fresh_candidate_screen.json", candidate_records)
require(candidates, "No exact 10-entangler candidate passed all validations.")
candidates.sort(key=lambda item: (item[0], json.dumps(js(item[1]), sort_keys=True)))
selected = candidates[0][1]
path = tuple(selected["path"])
physical_set = tuple(sorted(path))
candidate_compiled = compile_cache[path, selected["direction"]]

# Unsuppressed baseline: forward supplied synthesis, same physical path,
# level 2, no RC/DD; require its documented 10-native-entangler preparation.
pm = manager(path)
baseline_compiled = {}
baseline_metrics = []
for s in range(2):
    prep = pm.run(preparations[s, 1])
    count = sum(len(item.qubits) == 2 and item.operation.name in native_two
                for item in prep.data)
    require(count == 10, "Fresh unsuppressed baseline is not the required 10-entangler circuit.")
    baseline_metrics.append({"state": NAMES[s], "preparation_depth": prep.depth(),
                             "preparation_native_entanglers": count})
    for g in range(G):
        qc = pm.run(logical_measurement(s, g, 1))
        validate(qc, ideal[s, g])
        baseline_compiled[s, g] = qc
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
save_json("calibration_before.json", {
    "retrieved_utc": utc(), "properties": properties.to_dict(),
    "dt": dt, "pulse_alignment": pulse_alignment,
    "acquire_alignment": acquire_alignment,
    "missing_calibration": sorted(missing_calibration),
    "selected": selected,
    "selection_qualification": (
        "Executed-native-instruction and coherence-exposure proxy, not an "
        "exact device model and not evidence of hardware energy improvement."
    ),
})

# ------------------ Frozen covariance-aware science allocation -------------
calibration_budget = 2 * DIM * CFG["assignment_shots"]
validation_budget = B * DIM * CFG["heldout_shots"]
diagnostic_budget = 2 * CFG["diagnostic_shots"]
available = CFG["shot_cap"] - calibration_budget - validation_budget - diagnostic_budget
unit_per_block = R * 16                    # 64 total shots/group/block
unit_total = B * unit_per_block              # 128 shots over both blocks
total_units = available // unit_total
keys = [(s, mode, g) for s in range(2) for mode in range(4) for g in range(G)]
require(total_units >= len(keys), "Minimum all-group allocation cannot fit.")
allocation = {key: 1 for key in keys}

# Complete weighted group-outcome variance includes covariance of jointly
# measured Pauli terms. Small predeclared variance floor guards low-probability
# groups; no term is removed and every group gets positive shots.
weights = {}
for s, mode, g in keys:
    group_range = float(np.ptp(groups[g]["values"]))
    floor = (0.05 * group_range) ** 2
    variance = max(ideal_variance[s, g], floor, 1e-10)
    importance = 1. if mode == 0 else max(0.25, abs(LINEAR[mode - 1]))
    weights[s, mode, g] = math.sqrt(variance) * importance
for _ in range(int(total_units - len(keys))):
    key = max(
        keys,
        key=lambda k: weights[k] ** 2 / (allocation[k] * (allocation[k] + 1)),
    )
    allocation[key] += 1

registry, circuits, plan = {}, {}, []

def register(qc, intended, metadata):
    error = validate(qc, intended)
    report = exposure(qc)
    buffer = io.BytesIO()
    qpy.dump(qc, buffer)
    payload = buffer.getvalue()
    cid = sha(payload)
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    if cid not in registry:
        relative = "circuits/" + cid + ".qpy"
        save_bytes(relative, payload)
        registry[cid] = {
            "file": relative, "sha256": cid,
            "serialized_bytes": len(payload),
            "instructions": len(qc.data),
            "measurement_map": mapping(qc),
            "probability_error": error,
            "exposure": report, "metadata": metadata,
        }
        circuits[cid] = qc
    return cid

def add(stage, qc, shots, intended, metadata):
    cid = register(qc, intended, metadata)
    plan.append({
        "item_id": f"item_{len(plan):05d}", "stage": stage,
        "circuit_id": cid, "shots": int(shots), **metadata,
    })

# Calibration uses actual measured physical set, canonical bit ordering.
physical_pm = manager(list(range(width)), level=0)
basis_cache = {}
for prepared in range(DIM):
    qc = QuantumCircuit(
        QuantumRegister(width, "physical"),
        ClassicalRegister(NQ, CFG["register"]),
    )
    for c, q in enumerate(physical_set):
        if (prepared >> c) & 1:
            qc.x(q)
    for c, q in enumerate(physical_set):
        qc.measure(q, c)
    qc = scheduled(physical_pm.run(qc), "ASAP", False)
    require(mapping(qc) == physical_set, "Readout calibration mapping changed.")
    validate(qc, np.eye(DIM)[prepared])
    basis_cache[prepared] = qc

for prepared in range(DIM):
    add("assignment_start", basis_cache[prepared], CFG["assignment_shots"],
        np.eye(DIM)[prepared],
        {"kind": "assignment", "epoch": 0, "prepared": prepared})

# Physical diagnostics: known asymmetric basis states, no hidden postselection.
for prepared, label in ((1, "asymmetric-basis"), (0, "known-flipped-counterpart")):
    add("diagnostics", basis_cache[prepared], CFG["diagnostic_shots"],
        np.eye(DIM)[prepared],
        {"kind": "diagnostic", "prepared": prepared, "label": label})
expect_failure(
    lambda: validate(basis_cache[0], np.eye(DIM)[1]),
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    "physical-compiled-corruption",
)

for block in range(B):
    pairs = [(s, g) for s in range(2) for g in range(G)]
    order = rng(20, block).permutation(len(pairs))
    for pair_index in order:
        s, g = pairs[int(pair_index)]
        base = scheduled(baseline_compiled[s, g], "ASAP", False)
        base_meta = {
            "kind": "science", "block": block, "state": s, "group": g,
            "mode": 0, "factor": 1, "randomization": -1,
            "pauli_frames": [], "classical_xor": 0,
        }
        if (block + s + g) % 2 == 0:
            add(f"science_{block}", base, unit_per_block * allocation[s, 0, g],
                ideal[s, g], base_meta)
        for randomization in rng(21, block, s, g).permutation(R):
            randomization = int(randomization)
            for factor in rng(22, block, s, g, randomization).permutation([1, 3, 5]):
                factor = int(factor)
                mode = {1: 1, 3: 2, 5: 3}[factor]
                qc, frames, n0, nreal = transformed(
                    candidate_compiled[s, g], factor,
                    rng(30, block, s, g, randomization, factor),
                )
                validate(qc, ideal[s, g])
                qc = scheduled(qc, selected["schedule"], selected["dd"])
                add(
                    f"science_{block}", qc, 16 * allocation[s, mode, g],
                    ideal[s, g], {
                        "kind": "science", "block": block,
                        "state": s, "group": g, "mode": mode, "factor": factor,
                        "randomization": randomization,
                        "pauli_frames": frames, "classical_xor": 0,
                        "unfolded_native_entanglers": n0,
                        "realized_native_entanglers": nreal,
                        "realized_entangler_scale": nreal / n0,
                    },
                )
        if (block + s + g) % 2 != 0:
            add(f"science_{block}", base, unit_per_block * allocation[s, 0, g],
                ideal[s, g], base_meta)
    for prepared in rng(40, block).permutation(DIM):
        prepared = int(prepared)
        add(f"heldout_{block}", basis_cache[prepared], CFG["heldout_shots"],
            np.eye(DIM)[prepared],
            {"kind": "validation", "block": block, "prepared": prepared})

for prepared in range(DIM):
    add("assignment_end", basis_cache[prepared], CFG["assignment_shots"],
```

```
        np.eye(DIM)[prepared],
        {"kind": "assignment", "epoch": 1, "prepared": prepared})

# Explicit logical-count to canonical-physical permutation; no blind reversal.
for item in plan:
    if item["kind"] != "science":
        continue
    physical_for_c = registry[item["circuit_id"]]["measurement_map"]
    require(set(physical_for_c) == set(physical_set), "Multiple readout sets unexpected.")
    position = {q: j for j, q in enumerate(physical_set)}
    permutation = [
        sum(((k >> c) & 1) << position[physical_for_c[c]] for c in range(NQ))
        for k in range(DIM)
    ]
    require(len(set(permutation)) == DIM, "Invalid canonical permutation.")
    item["logical_to_canonical_index"] = permutation

# Validate saved byte-identical circuits again before authorization.
for cid, record in registry.items():
    payload = (directory / record["file"]).read_bytes()
    require(sha(payload) == cid, "Saved QPY hash mismatch.")
    require(mapping(circuits[cid]) == record["measurement_map"], "Circuit mapping changed.")

# ---------------- Conservative batching and Runtime checks ----------------
pub_limit = CFG["pub_limit"]
exposed_backend_limit = getattr(backend, "max_circuits", None)
if isinstance(exposed_backend_limit, int) and exposed_backend_limit > 0:
    pub_limit = min(pub_limit, exposed_backend_limit)

def fits(items):
    return (
        len(items) <= pub_limit
        and sum(registry[x["circuit_id"]]["serialized_bytes"] for x in items)
        <= CFG["client_job_QPY_bytes_limit"]
        and sum(registry[x["circuit_id"]]["instructions"] for x in items)
        <= CFG["client_job_instruction_limit"]
        and sum(x["shots"] for x in items) <= CFG["client_job_circuit_shots_limit"]
    )

jobs, current, stage = [], [], None

def finish_job(items):
    jobs.append({
        "batch": len(jobs), "items": [x["item_id"] for x in items],
        "stage": items[0]["stage"], "status": "planned", "job_id": None,
        "publications": len(items), "shots": sum(x["shots"] for x in items),
        "QPY_bytes_sum": sum(
            registry[x["circuit_id"]]["serialized_bytes"] for x in items
        ),
        "instructions_sum": sum(
            registry[x["circuit_id"]]["instructions"] for x in items
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        ),
    })

for item in plan:
    require(fits([item]), "One publication exceeds conservative client-side job bounds.")
    if current and (item["stage"] != stage or not fits(current + [item])):
        finish_job(current)
        current = []
    stage = item["stage"]
    current.append(item)
if current:
    finish_job(current)

budget = Counter()
for item in plan:
    budget[item["kind"]] += item["shots"]
total_shots = int(sum(budget.values()))
manifest.update({
    "status": "frozen",
    "selected_candidate": selected,
    "baseline": {"direction": 1, "schedule": "ASAP", "dd": False, "RC": False,
                 "path": path, "preparation_metrics": baseline_metrics},
    "physical_readout_set": physical_set,
    "registry": registry, "plan": plan, "jobs": jobs,
    "allocation": [
        {"state": s, "mode": mode, "group": g, "units": n}
        for (s, mode, g), n in allocation.items()
    ],
    "budget_by_kind": dict(budget),
    "total_shots": total_shots,
    "circuit_executions": len(plan),
    "job_count": len(jobs),
    "effective_publication_limit": pub_limit,
    "negative_controls": controls,
    "job_limit_qualification": (
        "Backend-exposed circuit limit and explicit conservative local byte/"
        "instruction/shot limits checked. These are not a claim that all "
        "changing server-side IBM limits have been independently certified. "
        "Runtime may reject a job; no automatic split/retry expands the plan."
    ),
})
save_json("manifest.json", manifest)

print("\n=== EXACT FROZEN SCREENING WORKLOAD ===")
print("Backend:", CFG["backend"], "physical width:", width, "logical qubits:", NQ)
print("Selected configuration:", selected)
print("Baseline preparation metrics:", baseline_metrics)
print("Two blocks; four randomizations; complete 17 groups; factors 1,3,5.")
for record in candidate_records:
    print({
        k: record.get(k) for k in (
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
            "path", "direction", "schedule", "dd", "valid", "score",
            "mean_depth_per_measurement_circuit",
            "summed_depth_across_measurement_circuits", "reason"
        )
    })
print("Mean depth and summed depth are distinct; summed depth is not job latency.")
for job in jobs:
    print(job)
print({
    "jobs": len(jobs), "circuit_executions": len(plan),
    "unique_saved_circuits": len(registry),
    "shots_by_kind": dict(budget), "total_shots": total_shots,
    "shot_cap": CFG["shot_cap"], "job_cap": CFG["job_cap"],
    "publications_per_job_limit": pub_limit,
    "evidence": str(directory),
})
require(total_shots <= 250_000, "ABORT: total shots exceed 250,000.")
require(len(jobs) <= 12, "ABORT: Runtime job count exceeds 12.")
require(len(plan) == 1014, "Unexpected change in complete screening circuit count.")
print("All deterministic, negative-control, and actual-circuit checks: PASS.")
print("No cancelled experiment was accessed. No QPU job has been submitted.")
print("Mixed per-PUB shots use SamplerPub; runtime 0.50 may emit deprecation warnings.")
print("Server-side twirling and DD disabled; all client transformations are saved.")
print("No automatic workload expansion or retry.")
phrase = f"AUTHORIZE SCREEN {CFG['backend']} {len(jobs)} JOBS {total_shots} SHOTS"
if input("Type exactly:\n" + phrase + "\n> ").strip() != phrase:
    manifest["status"] = "not_authorized"
    save_json("manifest.json", manifest)
    print("No QPU jobs submitted.")
    return {"directory": str(directory), "manifest": manifest, "analysis": None}

# ---------------------- Counts-based real-QPU execution ------------------
options = {
    "dynamical_decoupling": {"enable": False},
    "twirling": {"enable_gates": False, "enable_measure": False},
}
sampler = SamplerV2(mode=backend, options=options)
manifest["sampler_options"] = options
manifest["authorized_utc"] = utc()
manifest["status"] = "authorized"
save_json("manifest.json", manifest)
lookup = {item["item_id"]: item for item in plan}
observed = {}

try:
    for job_record in jobs:
        require(job_record["status"] == "planned" and job_record["job_id"] is None,
                "Unexpected job state; duplicate submission blocked.")
        job_record["status"] = "submitting"
        save_json("manifest.json", manifest)
        pubs = []
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        for key in job_record["items"]:
            item = lookup[key]
            cid = item["circuit_id"]
            require(
                sha((directory / registry[cid]["file"]).read_bytes()) == cid,
                "Pre-submission circuit artifact hash mismatch.",
            )
            # The actual submitted object is the validated object.
            pubs.append(SamplerPub(circuit=circuits[cid], shots=item["shots"]))
        job = sampler.run(pubs)
        job_record["job_id"] = job.job_id()
        job_record["status"] = "submitted"
        job_record["submitted_utc"] = utc()
        save_json("manifest.json", manifest)
        print("Submitted:", job_record["batch"], job.job_id())
        result = job.result()
        require(len(result) == len(job_record["items"]), "Returned PUB count mismatch.")
        rows = []
        for key, pub in zip(job_record["items"], result):
            item = lookup[key]
            counts = returned_counts(pub)
            rows.append({
                "item_id": key, "circuit_id": item["circuit_id"],
                "job_id": job.job_id(), "shots": item["shots"],
                "counts": counts, "metadata": getattr(pub, "metadata", {}),
            })
        relative = f"jobs/batch_{job_record['batch']:03d}.json"
        save_json(relative, {
            "job_id": job.job_id(), "batch": job_record["batch"],
            "rows": rows, "result_metadata": getattr(result, "metadata", None),
        })
        # Raw counts are saved before decoding/analysis.
        for row in rows:
            observed[row["item_id"]] = decode(row["counts"], row["shots"])
        job_record["raw_counts_sha256"] = sha((directory / relative).read_bytes())
        job_record["status"] = "complete"
        job_record["completed_utc"] = utc()
        save_json("manifest.json", manifest)
    require(len(observed) == len(plan), "Incomplete screening experiment.")
except BaseException as error:
    manifest["status"] = "interrupted_or_failed"
    manifest["failure"] = {"type": type(error).__name__, "utc": utc()}
    save_json("manifest.json", manifest)
    print("Stopped. Original counts and submitted job IDs are preserved:", directory)
    print("No automatic retry. A 'submitting' job without an ID is ambiguous.")
    raise

# ---------------------------- Readout validation -------------------------
assignment_counts = np.zeros((2, DIM, DIM), np.int64)
heldout_counts = np.zeros((B, DIM, DIM), np.int64)
diagnostics = []
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
science_rows = []
for item in plan:
    counts = observed[item["item_id"]]
    if item["kind"] == "assignment":
        assignment_counts[item["epoch"], :, item["prepared"]] += counts
    elif item["kind"] == "validation":
        heldout_counts[item["block"], :, item["prepared"]] += counts
    elif item["kind"] == "diagnostic":
        diagnostics.append({
            **item, "counts": counts,
            "expected_fraction": float(counts[item["prepared"]] / counts.sum()),
        })
    else:
        science_rows.append((item, counts))

pooled = assignment_counts.sum(axis=0)
A = pooled / pooled.sum(axis=0, keepdims=True)
heldout = heldout_counts.sum(axis=0)
heldout = heldout / heldout.sum(axis=0, keepdims=True)
heldout_TV = 0.5 * np.abs(A - heldout).sum(axis=0)
block_TV = [
    0.5 * np.abs(A - x / x.sum(axis=0, keepdims=True)).sum(axis=0)
    for x in heldout_counts
]
epoch_A = assignment_counts / assignment_counts.sum(axis=1, keepdims=True)
readout_report = {
    "training_counts_by_epoch": assignment_counts,
    "pooled_assignment": A, "heldout_counts": heldout_counts,
    "heldout_assignment": heldout, "heldout_TV": heldout_TV,
    "heldout_block_TV": block_TV,
    "training_epoch_TV": 0.5 * np.abs(epoch_A[0] - epoch_A[1]).sum(axis=0),
    "thresholds": {
        key: CFG[key] for key in (
            "readout_condition_limit", "readout_minimum_singular_value",
            "heldout_TV_limit", "heldout_block_TV_limit", "corrected_heldout_TV_limit"
        )
    },
    "qualification": (
        "Independent empirical screening validation, not proof of transferability. "
        "Calibration includes basis-state preparation error. No clipping or "
        "reference-dependent correction. Low held-out shots reduce sensitivity."
    ),
}
readout_valid = False
inverse = None
try:
    inverse, condition, minimum, inverse_error = safe_inverse(A)
    recovery_TV = 0.5 * np.abs(inverse @ heldout - np.eye(DIM)).sum(axis=0)
    readout_valid = bool(
        heldout_TV.max() <= CFG["heldout_TV_limit"]
        and max(x.max() for x in block_TV) <= CFG["heldout_block_TV_limit"]
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        and recovery_TV.max() <= CFG["corrected_heldout_TV_limit"]
    )
    readout_report.update({
        "condition": condition, "minimum_singular_value": minimum,
        "inverse_identity_error": inverse_error,
        "corrected_heldout_TV": recovery_TV,
    })
except RuntimeError as error:
    readout_report["reason"] = str(error)
readout_report["accepted"] = readout_valid
save_json("readout_validation.json", readout_report)
save_json("physical_diagnostics.json", diagnostics)

# Flatten actual publications. Every randomized publication has weight 1/4;
# baseline publications have weight 1. Complete group outcomes retain covariance.
counts_array, weight_array, shot_array, averaging, cells, row_meta = [], [], [], [], [], []
for item, counts in science_rows:
    permutation = np.array(item["logical_to_canonical_index"])
    canonical_counts = np.zeros(DIM, np.int64)
    canonical_values = np.zeros(DIM)
    canonical_counts[permutation] = counts
    canonical_values[permutation] = groups[item["group"]]["values"]
    counts_array.append(canonical_counts)
    weight_array.append(canonical_values)
    shot_array.append(item["shots"])
    averaging.append(1. if item["mode"] == 0 else 1 / R)
    cells.append((item["block"] * 2 + item["state"]) * 4 + item["mode"])
    row_meta.append(item)
counts_array = np.stack(counts_array)
W = np.stack(weight_array)
shots = np.array(shot_array)
averaging = np.array(averaging)
cell_indices = np.array(cells)
Pobs = counts_array / shots[:, None]
L = len(row_meta)

def evaluate(p, inv=None, group_output=False):
    effective = W if inv is None else W @ inv
    values = np.einsum("ij,ij->i", p, effective)
    total = np.bincount(
        cell_indices, weights=averaging * values, minlength=B * 2 * 4
    ).reshape(B, 2, 4)
    if not group_output:
        return total
    group_total = np.zeros((B, 2, 4, G))
    for i, item in enumerate(row_meta):
        group_total[item["block"], item["state"], item["mode"], item["group"]] += (
            averaging[i] * values[i]
        )
    return total, group_total
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
raw_blocks, raw_groups = evaluate(Pobs, group_output=True)
corrected_blocks = corrected_groups = None
if readout_valid:
    corrected_blocks, corrected_groups = evaluate(Pobs, inverse, True)

# -------------------- Randomization random-effects component -------------
# Estimate excess variation over multinomial science noise separately for
# each independently randomized factor. Retain paired raw/corrected covariance.
# Four randomizations make this component imprecise; it is screening evidence.
latent = {}
latent_records = []
for block in range(B):
    for s in range(2):
        for mode in (1, 2, 3):
            for g in range(G):
                ix = [
                    i for i, item in enumerate(row_meta)
                    if item["block"] == block and item["state"] == s
                    and item["mode"] == mode and item["group"] == g
                ]
                require(len(ix) == R, "Missing randomized publication.")
                ix.sort(key=lambda i: row_meta[i]["randomization"])
                p, w = Pobs[ix], W[ix]
                raw_values = np.einsum("ij,ij->i", p, w)
                dimensions = 2 if readout_valid else 1
                values = raw_values[:, None]
                if readout_valid:
                    z = w @ inverse
                    values = np.column_stack([
                        raw_values, np.einsum("ij,ij->i", p, z)
                    ])
                shot_cov = np.zeros((dimensions, dimensions))
                for j, i in enumerate(ix):
                    obs = w[j][None, :] if not readout_valid else np.stack([w[j], z[j]])
                    means = obs @ p[j]
                    shot_cov += (
                        (obs * p[j]) @ obs.T - np.outer(means, means)
                    ) / (shots[i] - 1) / R
                empirical = np.atleast_2d(np.cov(values, rowvar=False, ddof=1))
                excess = (empirical - shot_cov)
                eig, vec = np.linalg.eigh((excess + excess.T) / 2)
                psd = (vec * np.maximum(eig, 0)) @ vec.T / R
                latent[block, s, mode, g] = psd
                latent_records.append({
                    "block": block, "state": s, "mode": mode, "group": g,
                    "preprojection_eigenvalues": eig,
                    "covariance_of_randomization_mean": psd,
                })
save_json("randomization_uncertainty.json", {
    "estimates": latent_records,
    "qualification": (
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        "Four-draw random-effects screening estimate; subtract estimated "
        "science-shot covariance, project covariance only to PSD. No "
        "probability or energy is clipped. Factors use independent streams."
    ),
})

# ------------------- Shared-calibration / shot / RC bootstrap --------------
K = CFG["bootstrap_replicates"]
shape = (K, B, 2, 4)
raw_joint = np.zeros(shape)
raw_shot = np.zeros(shape)
corr_joint = np.full(shape, np.nan)
corr_shot = np.full(shape, np.nan)
corr_cal = np.full(shape, np.nan)
corr_random = np.full(shape, np.nan)
raw_random = np.zeros(shape)
invalid_calibration_draws = 0

for iteration in range(K):
    shot_rng, cal_rng, random_rng = rng(100, iteration), rng(101, iteration), rng(102, iteration)
    sampled = np.stack([
        shot_rng.multinomial(int(shots[i]), Pobs[i]) / shots[i] for i in range(L)
    ])
    raw_shot[iteration] = evaluate(sampled)
    raw_joint[iteration] = raw_shot[iteration]
    raw_random[iteration] = raw_blocks
    if readout_valid:
        corr_shot[iteration] = evaluate(sampled, inverse)
        draw_counts = np.zeros((DIM, DIM), np.int64)
        for epoch in range(2):
            for column in range(DIM):
                counts = assignment_counts[epoch, :, column]
                draw_counts[:, column] += cal_rng.multinomial(
                    int(counts.sum()), counts / counts.sum()
                )
        draw_A = draw_counts / draw_counts.sum(axis=0, keepdims=True)
        try:
            draw_inverse, _, _, _ = safe_inverse(draw_A)
            corr_cal[iteration] = evaluate(Pobs, draw_inverse)
            corr_joint[iteration] = evaluate(sampled, draw_inverse)
        except RuntimeError:
            invalid_calibration_draws += 1
        corr_random[iteration] = corrected_blocks

    for (block, s, mode, g), covariance in latent.items():
        delta = random_rng.multivariate_normal(np.zeros(len(covariance)), covariance)
        raw_joint[iteration, block, s, mode] += delta[0]
        raw_random[iteration, block, s, mode] += delta[0]
        if readout_valid:
            corr_joint[iteration, block, s, mode] += delta[1]
            corr_random[iteration, block, s, mode] += delta[1]
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
# Fail closed on calibration-draw failures, rather than dropping draws.
if invalid_calibration_draws:
    readout_valid = False

def interval(samples):
    return np.quantile(samples, [0.025, 0.975], axis=0)

def summary(point, samples):
    return {
        "estimate": np.asarray(point),
        "standard_error": samples.std(axis=0, ddof=1),
        "statistical_95_percentile_interval": interval(samples),
    }

raw_mean = raw_blocks.mean(axis=0)
raw_boot_mean = raw_joint.mean(axis=1)
report = {
    "scope": "TWO-BLOCK, FOUR-RANDOMIZATION SCREENING EXPERIMENT",
    "backend": CFG["backend"],
    "references_Ha": dict(zip(NAMES, references)),
    "continuum_reference_Ha": continuum,
    "finite_reference_Ha": fd_reference,
    "finite_representation_difference_Ha": fd_reference - continuum,
    "classical_variational_residual_Ha": references - fd_reference,
    "quantum_implementation_difference_Ha": ideal_group.sum(axis=1) - references,
    "local_sampling": local_rows,
    "baseline_raw": summary(raw_mean[:, 0], raw_boot_mean[:, :, 0]),
    "candidate_raw_factor1": summary(raw_mean[:, 1], raw_boot_mean[:, :, 1]),
    "candidate_raw_factors_1_3_5": summary(raw_mean[:, 1:4], raw_boot_mean[:, :, 1:4]),
    "paired_raw_candidate_minus_baseline": summary(
        raw_mean[:, 1] - raw_mean[:, 0],
        raw_boot_mean[:, :, 1] - raw_boot_mean[:, :, 0],
    ),
    "raw_block_energies": raw_blocks,
    "raw_between_block_sd": raw_blocks.std(axis=0, ddof=1),
    "raw_between_block_se_of_mean": raw_blocks.std(axis=0, ddof=1) / math.sqrt(B),
    "readout_valid_for_inference": readout_valid,
    "invalid_calibration_bootstrap_draws": invalid_calibration_draws,
    "uncertainty_assumptions": [
        "Conditional multinomial science counts for each measured circuit.",
        "The same bootstrap assignment matrix is shared across all science data.",
        "Four-randomization excess variance is an imprecise random-effects estimate.",
        "Science, calibration, and RC components are exported separately.",
        "Between-block variation is reported separately and is not added twice.",
        "Intervals condition on frozen allocation/selection and omit unquantified systematics.",
    ],
}
group_report = {
    "group_numbering": "zero-based",
    "ideal": ideal_group,
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    "raw_block_group_energies": raw_groups,
    "raw_mean_group_residuals": raw_groups.mean(axis=0) - ideal_group[:, None, :],
}
achieved = np.zeros(2, bool)

if readout_valid:
    corrected_mean = corrected_blocks.mean(axis=0)
    corrected_boot_mean = corr_joint.mean(axis=1)
    factors = corrected_mean[:, 1:4]
    factor_boot = corrected_boot_mean[:, :, 1:4]
    final = factors @ LINEAR
    final_boot = factor_boot @ LINEAR
    quadratic = factors @ QUADRATIC
    quadratic_boot = factor_boot @ QUADRATIC
    sensitivity = quadratic - final
    sensitivity_boot = quadratic_boot - final_boot
    curvature = factors @ np.array([1., -2., 1.])
    curvature_boot = factor_boot @ np.array([1., -2., 1.])
    curvature_se = curvature_boot.std(axis=0, ddof=1)
    lack_p = chi2.sf(curvature ** 2 / np.maximum(curvature_se ** 2, 1e-30), 1)
    amplification = (
        final_boot.std(axis=0, ddof=1)
        / np.maximum(factor_boot[:, :, 0].std(axis=0, ddof=1), 1e-15)
    )
    sensitivity_ci = interval(sensitivity_boot)
    sensitivity_halfwidth = (sensitivity_ci[1] - sensitivity_ci[0]) / 2
    diagnostic_inconclusive = sensitivity_halfwidth > CFG["diagnostic_precision_Ha"]
    model_limit = np.maximum(
        CFG["model_sensitivity_floor_Ha"], 2 * sensitivity_boot.std(axis=0, ddof=1)
    )
    zne_per_state = (
        (lack_p >= CFG["ZNE_lack_of_fit_p_min"])
        & (amplification <= CFG["ZNE_amplification_limit"])
        & (np.abs(sensitivity) <= model_limit)
        & ~diagnostic_inconclusive
    )
    both_zne_valid = bool(np.all(zne_per_state))
    block_final = corrected_blocks[:, :, 1:4] @ LINEAR
    block_final_boot = corr_joint[:, :, :, 1:4] @ LINEAR
    difference = block_final[1] - block_final[0]
    difference_boot = block_final_boot[:, 1] - block_final_boot[:, 0]
    difference_se = difference_boot.std(axis=0, ddof=1)
    heterogeneity_p = chi2.sf(
        difference ** 2 / np.maximum(difference_se ** 2, 1e-30), 1
    )
    ci = interval(final_boot)
    for s in range(2):
        lower, upper = references[s] - 0.05, references[s] + 0.05
        achieved[s] = bool(
            both_zne_valid
            and ci[0, s] >= lower and ci[1, s] <= upper
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        and np.all(block_final[:, s] >= lower)
        and np.all(block_final[:, s] <= upper)
        and heterogeneity_p[s] >= CFG["block_heterogeneity_p_min"]
    )

design = np.column_stack([np.ones(3), [1., 3., 5.]])
coefficients_fit = np.linalg.lstsq(design, factors.T, rcond=None)[0]
fit_residuals = factors - (design @ coefficients_fit).T
report.update({
    "baseline_readout_corrected": summary(
        corrected_mean[:, 0], corrected_boot_mean[:, :, 0]
    ),
    "candidate_readout_corrected_factor1": summary(
        corrected_mean[:, 1], corrected_boot_mean[:, :, 1]
    ),
    "measured_corrected_factors_1_3_5": summary(factors, factor_boot),
    "prespecified_linear_ZNE": summary(final, final_boot),
    "ZNE_error_against_fixed_state_Ha": final - references,
    "linear_fit_residuals_Ha": fit_residuals,
    "linear_lack_of_fit_p": lack_p,
    "uncertainty_amplification": amplification,
    "quadratic_sensitivity_only": summary(quadratic, quadratic_boot),
    "quadratic_minus_linear": summary(sensitivity, sensitivity_boot),
    "diagnostic_inconclusive": diagnostic_inconclusive,
    "ZNE_accepted_per_state": zne_per_state,
    "ZNE_accepted_for_both_states": both_zne_valid,
    "paired_corrected_minus_raw_baseline": summary(
        corrected_mean[:, 0] - raw_mean[:, 0],
        corrected_boot_mean[:, :, 0] - raw_boot_mean[:, :, 0],
    ),
    "paired_corrected_minus_raw_candidate_factor1": summary(
        corrected_mean[:, 1] - raw_mean[:, 1],
        corrected_boot_mean[:, :, 1] - raw_boot_mean[:, :, 1],
    ),
    "final_uncertainty_components": {
        "science_shot_se": (
            corr_shot.mean(axis=1)[:, :, 1:4] @ LINEAR
        ).std(axis=0, ddof=1),
        "shared_calibration_se": (
            corr_cal.mean(axis=1)[:, :, 1:4] @ LINEAR
        ).std(axis=0, ddof=1),
        "randomization_se": (
            corr_random.mean(axis=1)[:, :, 1:4] @ LINEAR
        ).std(axis=0, ddof=1),
        "joint_statistical_se": final_boot.std(axis=0, ddof=1),
    },
    "corrected_factor_covariance": [
        np.cov(factor_boot[:, s, :], rowvar=False) for s in range(2)
    ],
    "final_cross_state_covariance": np.cov(final_boot, rowvar=False),
    "block_final_estimates": block_final,
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        "block_final_intervals": interval(block_final_boot),
        "between_block_final_sd": block_final.std(axis=0, ddof=1),
        "between_block_final_se_of_mean": block_final.std(axis=0, ddof=1) / math.sqrt(B),
        "block_difference": summary(difference, difference_boot),
        "block_heterogeneity_p": heterogeneity_p,
        "absolute_error_reduction_over_raw_baseline": summary(
          np.abs(raw_mean[:, 0] - references) - np.abs(final - references),
          np.abs(raw_boot_mean[:, :, 0] - references)
          - np.abs(final_boot - references),
        ),
        "current_block2_corrected_minus_raw": corrected_blocks[1] - raw_blocks[1],
        "current_block2_paired_correction_intervals": interval(
          corr_joint[:, 1] - raw_joint[:, 1]
        ),
        "screening_interpretation": (
          "A non-rejected three-point fit is not proof of a valid noise "
          "model. Broad model-sensitivity intervals are explicitly "
          "inconclusive. No workload expansion or alternative fitted "
          "estimator is selected after viewing results."
        ),
      })
      corrected_group_mean = corrected_groups.mean(axis=0)
      zne_group = np.einsum(
        "smg,m->sg", corrected_group_mean[:, 1:4, :], LINEAR
      )
      residual = zne_group - ideal_group
      require(np.max(np.abs(residual.sum(axis=1) - (final - references))) <= ETOL,
              "Extrapolated group residual sum failed.")
      absolute = np.abs(residual).sum(axis=1)
      group_report.update({
        "corrected_block_group_energies": corrected_groups,
        "ZNE_group_energies": zne_group,
        "ZNE_group_residuals": residual,
        "signed_residual_sum": residual.sum(axis=1),
        "absolute_residual_sum": absolute,
        "cancellation_fraction": np.where(
          absolute > 0, 1 - np.abs(residual.sum(axis=1)) / np.maximum(absolute, 1e-30), 0.
        ),
        "prespecified_groups_2_8_14_16": {
          str(g): {
            "basis": groups[g]["basis"],
            "raw_mean": raw_groups.mean(axis=0)[:, :, g],
            "corrected_mean": corrected_group_mean[:, :, g],
            "ZNE_residual": residual[:, g],
            "block2_correction": corrected_groups[1, :, :, g] - raw_groups[1, :, :, g],
          }
          for g in (2, 8, 14, 16)
        },
      })
    else:
      report["corrected_and_ZNE_status"] = (
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        "UNAVAILABLE/REJECTED by readout or bootstrap-conditioning checks. "
        "No calibration draw is silently dropped. Raw results retained."
    )

report.update({
    "engineering_target_Ha": 0.05,
    "target_achieved": dict(zip(NAMES, map(bool, achieved))),
    "success_rule": (
        "Both-state readout/ZNE validity; final prespecified estimator's "
        "95% interval entirely inside each fixed-state reference +/-0.05 Ha; "
        "both block point estimates inside; block heterogeneity p>=0.01. "
        "Two-block evidence is screening, not robust long-term confirmation."
    ),
    "systematic_limitations": [
        "Readout calibration preparation errors, transferability, and calibration drift.",
        "Unmodelled leakage, coherent errors, correlations, and crosstalk.",
        "Entangler folding does not scale all physical errors equally.",
        "The extrapolated intercept is not proven noiseless execution.",
        "Only two blocks and four randomizations; uncertainty components are imprecise.",
        "Statistical intervals do not include every hardware systematic.",
    ],
    "job_ids": [job["job_id"] for job in jobs],
    "budget_by_kind": dict(budget), "total_shots": total_shots,
})
save_json("analysis.json", report)
save_json("group_diagnostics.json", group_report)
np.savez_compressed(
    directory / "bootstrap_samples.npz",
    raw_shot=raw_shot, raw_joint=raw_joint, raw_randomization=raw_random,
    corrected_shot=corr_shot, corrected_calibration=corr_cal,
    corrected_randomization=corr_random, corrected_joint=corr_joint,
)
os.chmod(directory / "bootstrap_samples.npz", 0o600)
try:
    after = backend.properties(refresh=True)
    save_json("calibration_after.json", {
        "utc": utc(), "properties": None if after is None else after.to_dict()
    })
except Exception as error:
    save_json("calibration_after_unavailable.json", {
        "utc": utc(), "exception_type": type(error).__name__
    })

manifest["status"] = "complete"
manifest["completed_utc"] = utc()
manifest["target_achieved"] = report["target_achieved"]
save_json("manifest.json", manifest)

print("\n=== MEASURED SCREENING RESULTS — HARTREE ===")
for s, name in enumerate(NAMES):
    print("\n", name, "fixed-state reference:", references[s])
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        for key in (
            "baseline_raw", "candidate_raw_factor1",
            "baseline_readout_corrected", "candidate_readout_corrected_factor1",
            "measured_corrected_factors_1_3_5", "prespecified_linear_ZNE",
        ):
            if key in report:
                print(
                    key,
                    report[key]["estimate"][s],
                    "95% interval:",
                    report[key]["statistical_95_percentile_interval"][:, s],
                )
        if readout_valid:
            print("ZNE diagnostics inconclusive:", bool(report["diagnostic_inconclusive"][s]))
            print("ZNE accepted for both states:", report["ZNE_accepted_for_both_states"])
            print("Between-block final SD:", report["between_block_final_sd"][s])
        else:
            print(report["corrected_and_ZNE_status"])
        print("±0.05 Ha target:", "PASS" if achieved[s] else "NOT ACHIEVED")
    print("\nNo chemical-accuracy or guaranteed device-improvement claim.")
    print("No automatic extra jobs, shots, randomizations, or follow-up run.")
    print("Job IDs:", report["job_ids"])
    print("Evidence:", directory)
    return {"directory": str(directory), "manifest": manifest, "analysis": report}


# Strict external-dependency sweep:
# Only these ten existing Part I definitions are read. Every other working
# symbol is defined/imported in this fresh cell. No evidence path or result
# from any earlier or cancelled experiment is accessed.
_compact_required = (
    "H", "r", "h", "n_interior", "R_max", "E_exact", "E0_fd",
    "alpha_cobyla", "alpha_spsa", "normalized_trial_state",
)
_compact_missing = [name for name in _compact_required if name not in globals()]
if _compact_missing:
    raise NameError(
        "Run authoritative Part I first. Missing: " + ", ".join(_compact_missing)
    )

part_ii_compact_screen_results = _fresh_compact_hydrogen_screen(
    **{name: globals()[name] for name in _compact_required}
)
```

Sixth-run results (verbatim)

```
SCREENING EXPERIMENT — FRESH, NOT A RESUMPTION
LOCAL ACCEPTANCE: PASS
Versions: {'numpy': '2.1.3', 'scipy': '1.16.3', 'qiskit': '2.5.2', 'qiskit-ibm-runtime': '0.50.0'}
Fixed-state references: {'COBYLA': np.float64(-0.46097392795976344), 'SPSA': np.float64(-
0.46077921338321903)}
Pauli terms / groups: 63 17
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
Hamiltonian reconstruction error: 1.7763568394002505e-15
LOCAL COBYLA -0.4668599737258183 +/- 0.019769806516494216 Ha
LOCAL SPSA -0.4456820696876161 +/- 0.019630157827082956 Ha
No cancelled-experiment artifacts have been opened.
qiskit_runtime_service._discover_account:WARNING:2026-10-15 18:35:05,103: Loading account with the given token. A saved account will not be used.
qiskit_runtime_service.__init__:WARNING:2026-10-15 18:35:08,036: Instance was not set at service instantiation. Free and trial plan instances will be prioritized. Based on the following filters: (tags: None, region: us-east, eu-de), and available plans: (open), the available account instances are: open-instance. If you need a specific instance set it explicitly either by using a saved account with a saved default instance or passing it in directly to QiskitRuntimeService(). Alternatively, pass instance='auto' or save it to your account for auto-selection without warning.
qiskit_runtime_service.backends:WARNING:2026-10-15 18:35:08,038: Using instance: open-instance, plan: open
Fresh local placement / orientation / schedule / DD screening...

=== EXACT FROZEN SCREENING WORKLOAD ===
Backend: ibm_marrakesh physical width: 156 logical qubits: 5
Selected configuration: {'path': (2, 3, 4, 5, 6), 'direction': -1, 'schedule': 'ASAP', 'dd': False}
Baseline preparation metrics: [{'state': 'COBYLA', 'preparation_depth': 41, 'preparation_native_entanglers': 10}, {'state': 'SPSA', 'preparation_depth': 41, 'preparation_native_entanglers': 10}]
Two blocks; four randomizations; complete 17 groups; factors 1,3,5.
{'path': (4, 5, 6, 7, 8), 'direction': 1, 'schedule': 'ASAP', 'dd': False, 'valid': True, 'score': 4.038093225534932, 'mean_depth_per_measurement_circuit': 73.26470588235294, 'summed_depth_across_measurement_circuits': 2491, 'reason': None}
{'path': (4, 5, 6, 7, 8), 'direction': 1, 'schedule': 'ASAP', 'dd': True, 'valid': False, 'score': None, 'mean_depth_per_measurement_circuit': None, 'summed_depth_across_measurement_circuits': None, 'reason': 'RuntimeError: Nonterminal active operation.'}
{'path': (4, 5, 6, 7, 8), 'direction': 1, 'schedule': 'ALAP', 'dd': False, 'valid': True, 'score': 4.680486037646909, 'mean_depth_per_measurement_circuit': 73.26470588235294, 'summed_depth_across_measurement_circuits': 2491, 'reason': None}
{'path': (4, 5, 6, 7, 8), 'direction': 1, 'schedule': 'ALAP', 'dd': True, 'valid': True, 'score': 4.721469736803569, 'mean_depth_per_measurement_circuit': 73.26470588235294, 'summed_depth_across_measurement_circuits': 2491, 'reason': None}
{'path': (4, 5, 6, 7, 8), 'direction': -1, 'schedule': 'ASAP', 'dd': False, 'valid': True, 'score': 3.841386986821317, 'mean_depth_per_measurement_circuit': 70.94117647058823, 'summed_depth_across_measurement_circuits': 2412, 'reason': None}
{'path': (4, 5, 6, 7, 8), 'direction': -1, 'schedule': 'ASAP', 'dd': True, 'valid': False, 'score': None, 'mean_depth_per_measurement_circuit': None, 'summed_depth_across_measurement_circuits': None, 'reason': 'RuntimeError: Nonterminal active operation.'}
{'path': (4, 5, 6, 7, 8), 'direction': -1, 'schedule': 'ALAP', 'dd': False, 'valid': True, 'score': 4.669554097485253, 'mean_depth_per_measurement_circuit': 70.94117647058823, 'summed_depth_across_measurement_circuits': 2412, 'reason': None}
{'path': (4, 5, 6, 7, 8), 'direction': -1, 'schedule': 'ALAP', 'dd': True, 'valid': True, 'score': 4.704062839811565, 'mean_depth_per_measurement_circuit': 70.94117647058823, 'summed_depth_across_measurement_circuits': 2412, 'reason': None}
{'path': (8, 7, 6, 5, 4), 'direction': 1, 'schedule': 'ASAP', 'dd': False, 'valid': True, 'score': 3.955537108031374, 'mean_depth_per_measurement_circuit': 74.47058823529412, 'summed_depth_across_measurement_circuits': 2532, 'reason': None}
{'path': (8, 7, 6, 5, 4), 'direction': 1, 'schedule': 'ASAP', 'dd': True, 'valid': False, 'score': None, 'mean_depth_per_measurement_circuit': None, 'summed_depth_across_measurement_circuits': None, 'reason':
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

'RuntimeError: Nonterminal active operation.'}
{'path': (8, 7, 6, 5, 4), 'direction': 1, 'schedule': 'ALAP', 'dd': False, 'valid': True, 'score': 4.831420895132705, 'mean_depth_per_measurement_circuit': 74.47058823529412, 'summed_depth_across_measurement_circuits': 2532, 'reason': None}
{'path': (8, 7, 6, 5, 4), 'direction': 1, 'schedule': 'ALAP', 'dd': True, 'valid': True, 'score': 4.865929637459016, 'mean_depth_per_measurement_circuit': 74.47058823529412, 'summed_depth_across_measurement_circuits': 2532, 'reason': None}
{'path': (8, 7, 6, 5, 4), 'direction': -1, 'schedule': 'ASAP', 'dd': False, 'valid': True, 'score': 3.932992386084811, 'mean_depth_per_measurement_circuit': 71.20588235294117, 'summed_depth_across_measurement_circuits': 2421, 'reason': None}
{'path': (8, 7, 6, 5, 4), 'direction': -1, 'schedule': 'ASAP', 'dd': True, 'valid': False, 'score': None, 'mean_depth_per_measurement_circuit': None, 'summed_depth_across_measurement_circuits': None, 'reason': 'RuntimeError: Nonterminal active operation.'}
{'path': (8, 7, 6, 5, 4), 'direction': -1, 'schedule': 'ALAP', 'dd': False, 'valid': True, 'score': 4.568051042468449, 'mean_depth_per_measurement_circuit': 71.20588235294117, 'summed_depth_across_measurement_circuits': 2421, 'reason': None}
{'path': (8, 7, 6, 5, 4), 'direction': -1, 'schedule': 'ALAP', 'dd': True, 'valid': True, 'score': 4.609034741625109, 'mean_depth_per_measurement_circuit': 71.20588235294117, 'summed_depth_across_measurement_circuits': 2421, 'reason': None}
{'path': (2, 3, 4, 5, 6), 'direction': 1, 'schedule': 'ASAP', 'dd': False, 'valid': True, 'score': 3.833676432811097, 'mean_depth_per_measurement_circuit': 73.08823529411765, 'summed_depth_across_measurement_circuits': 2485, 'reason': None}
{'path': (2, 3, 4, 5, 6), 'direction': 1, 'schedule': 'ASAP', 'dd': True, 'valid': False, 'score': None, 'mean_depth_per_measurement_circuit': None, 'summed_depth_across_measurement_circuits': None, 'reason': 'RuntimeError: Nonterminal active operation.'}
{'path': (2, 3, 4, 5, 6), 'direction': 1, 'schedule': 'ALAP', 'dd': False, 'valid': True, 'score': 4.310017004787259, 'mean_depth_per_measurement_circuit': 73.08823529411765, 'summed_depth_across_measurement_circuits': 2485, 'reason': None}
{'path': (2, 3, 4, 5, 6), 'direction': 1, 'schedule': 'ALAP', 'dd': True, 'valid': True, 'score': 4.352001964725077, 'mean_depth_per_measurement_circuit': 73.08823529411765, 'summed_depth_across_measurement_circuits': 2485, 'reason': None}
{'path': (2, 3, 4, 5, 6), 'direction': -1, 'schedule': 'ASAP', 'dd': False, 'valid': True, 'score': 3.684843081876501, 'mean_depth_per_measurement_circuit': 70.52941176470588, 'summed_depth_across_measurement_circuits': 2398, 'reason': None}
{'path': (2, 3, 4, 5, 6), 'direction': -1, 'schedule': 'ASAP', 'dd': True, 'valid': False, 'score': None, 'mean_depth_per_measurement_circuit': None, 'summed_depth_across_measurement_circuits': None, 'reason': 'RuntimeError: Nonterminal active operation.'}
{'path': (2, 3, 4, 5, 6), 'direction': -1, 'schedule': 'ALAP', 'dd': False, 'valid': True, 'score': 4.282466443012068, 'mean_depth_per_measurement_circuit': 70.52941176470588, 'summed_depth_across_measurement_circuits': 2398, 'reason': None}
{'path': (2, 3, 4, 5, 6), 'direction': -1, 'schedule': 'ALAP', 'dd': True, 'valid': True, 'score': 4.3234501421687295, 'mean_depth_per_measurement_circuit': 70.52941176470588, 'summed_depth_across_measurement_circuits': 2398, 'reason': None}
{'path': (6, 5, 4, 3, 2), 'direction': 1, 'schedule': 'ASAP', 'dd': False, 'valid': True, 'score': 3.7524730120404115, 'mean_depth_per_measurement_circuit': 73.6470588235294, 'summed_depth_across_measurement_circuits': 2504, 'reason': None}
{'path': (6, 5, 4, 3, 2), 'direction': 1, 'schedule': 'ASAP', 'dd': True, 'valid': False, 'score': None, 'mean_depth_per_measurement_circuit': None, 'summed_depth_across_measurement_circuits': None, 'reason': 'RuntimeError: Nonterminal active operation.'}
{'path': (6, 5, 4, 3, 2), 'direction': 1, 'schedule': 'ALAP', 'dd': False, 'valid': True, 'score': 4.384013891519942, 'mean_depth_per_measurement_circuit': 73.6470588235294, 'summed_depth_across_measurement_circuits':

2504, 'reason': None}
{'path': (6, 5, 4, 3, 2), 'direction': 1, 'schedule': 'ALAP', 'dd': True, 'valid': True, 'score': 4.424997590676603, 'mean_depth_per_measurement_circuit': 73.6470588235294, 'summed_depth_across_measurement_circuits': 2504, 'reason': None}
{'path': (6, 5, 4, 3, 2), 'direction': -1, 'schedule': 'ASAP', 'dd': False, 'valid': True, 'score': 3.809133151183505, 'mean_depth_per_measurement_circuit': 69.91176470588235, 'summed_depth_across_measurement_circuits': 2377, 'reason': None}
{'path': (6, 5, 4, 3, 2), 'direction': -1, 'schedule': 'ASAP', 'dd': True, 'valid': False, 'score': None, 'mean_depth_per_measurement_circuit': None, 'summed_depth_across_measurement_circuits': None, 'reason': 'RuntimeError: Nonterminal active operation.'}
{'path': (6, 5, 4, 3, 2), 'direction': -1, 'schedule': 'ALAP', 'dd': False, 'valid': True, 'score': 4.272071306691911, 'mean_depth_per_measurement_circuit': 69.91176470588235, 'summed_depth_across_measurement_circuits': 2377, 'reason': None}
{'path': (6, 5, 4, 3, 2), 'direction': -1, 'schedule': 'ALAP', 'dd': True, 'valid': True, 'score': 4.314056266629731, 'mean_depth_per_measurement_circuit': 69.91176470588235, 'summed_depth_across_measurement_circuits': 2377, 'reason': None}
Mean depth and summed depth are distinct; summed depth is not job latency.
{'batch': 0, 'items': ['item_00000', 'item_00001', 'item_00002', 'item_00003', 'item_00004', 'item_00005', 'item_00006', 'item_00007', 'item_00008', 'item_00009', 'item_00010', 'item_00011', 'item_00012', 'item_00013', 'item_00014', 'item_00015', 'item_00016', 'item_00017', 'item_00018', 'item_00019', 'item_00020', 'item_00021', 'item_00022', 'item_00023', 'item_00024', 'item_00025', 'item_00026', 'item_00027', 'item_00028', 'item_00029', 'item_00030', 'item_00031'], 'stage': 'assignment_start', 'status': 'planned', 'job_id': None, 'publications': 32, 'shots': 16384, 'QPY_bytes_sum': 351908, 'instructions_sum': 5147}
{'batch': 1, 'items': ['item_00032', 'item_00033'], 'stage': 'diagnostics', 'status': 'planned', 'job_id': None, 'publications': 2, 'shots': 256, 'QPY_bytes_sum': 21781, 'instructions_sum': 317}
{'batch': 2, 'items': ['item_00034', 'item_00035', 'item_00036', 'item_00037', 'item_00038', 'item_00039', 'item_00040', 'item_00041', 'item_00042', 'item_00043', 'item_00044', 'item_00045', 'item_00046', 'item_00047', 'item_00048', 'item_00049', 'item_00050', 'item_00051', 'item_00052', 'item_00053', 'item_00054', 'item_00055', 'item_00056', 'item_00057', 'item_00058', 'item_00059', 'item_00060', 'item_00061', 'item_00062', 'item_00063', 'item_00064', 'item_00065', 'item_00066', 'item_00067', 'item_00068', 'item_00069', 'item_00070', 'item_00071', 'item_00072', 'item_00073', 'item_00074', 'item_00075', 'item_00076', 'item_00077', 'item_00078', 'item_00079', 'item_00080', 'item_00081', 'item_00082', 'item_00083', 'item_00084', 'item_00085', 'item_00086', 'item_00087', 'item_00088', 'item_00089', 'item_00090', 'item_00091', 'item_00092', 'item_00093', 'item_00094', 'item_00095', 'item_00096', 'item_00097', 'item_00098', 'item_00099', 'item_00100', 'item_00101', 'item_00102', 'item_00103', 'item_00104', 'item_00105', 'item_00106', 'item_00107', 'item_00108', 'item_00109', 'item_00110', 'item_00111', 'item_00112', 'item_00113', 'item_00114', 'item_00115', 'item_00116', 'item_00117', 'item_00118', 'item_00119', 'item_00120', 'item_00121', 'item_00122', 'item_00123', 'item_00124', 'item_00125', 'item_00126', 'item_00127', 'item_00128', 'item_00129', 'item_00130', 'item_00131', 'item_00132', 'item_00133', 'item_00134', 'item_00135', 'item_00136', 'item_00137', 'item_00138', 'item_00139', 'item_00140', 'item_00141', 'item_00142', 'item_00143', 'item_00144', 'item_00145', 'item_00146', 'item_00147', 'item_00148', 'item_00149', 'item_00150', 'item_00151', 'item_00152', 'item_00153', 'item_00154', 'item_00155', 'item_00156', 'item_00157', 'item_00158', 'item_00159', 'item_00160', 'item_00161', 'item_00162', 'item_00163', 'item_00164', 'item_00165', 'item_00166', 'item_00167', 'item_00168', 'item_00169', 'item_00170', 'item_00171', 'item_00172', 'item_00173', 'item_00174', 'item_00175', 'item_00176', 'item_00177', 'item_00178', 'item_00179', 'item_00180', 'item_00181', 'item_00182', 'item_00183', 'item_00184', 'item_00185', 'item_00186', 'item_00187', 'item_00188', 'item_00189', 'item_00190', 'item_00191', 'item_00192', 'item_00193', 'item_00194', 'item_00195', 'item_00196', 'item_00197', 'item_00198', 'item_00199', 'item_00200', 'item_00201', 'item_00202', 'item_00203', 'item_00204', 'item_00205', 'item_00206', 'item_00207', 'item_00208', 'item_00209', 'item_00210', 'item_00211', 'item_00212', 'item_00213', 'item_00214', 'item_00215', 'item_00216', 'item_00217', 'item_00218', 'item_00219', 'item_00220', 'item_00221', 'item_00222', 'item_00223', 'item_00224', 'item_00225', 'item_00226', 'item_00227', 'item_00228', 'item_00229', 'item_00230', 'item_00231', 'item_00232', 'item_00233', 'item_00234', 'item_00235', 'item_00236', 'item_00237', 'item_00238', 'item_00239',

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

'item_00240', 'item_00241', 'item_00242', 'item_00243', 'item_00244', 'item_00245', 'item_00246', 'item_00247', 'item_00248', 'item_00249', 'item_00250', 'item_00251', 'item_00252', 'item_00253', 'item_00254', 'item_00255', 'item_00256', 'item_00257', 'item_00258', 'item_00259', 'item_00260', 'item_00261', 'item_00262', 'item_00263', 'item_00264', 'item_00265', 'item_00266', 'item_00267', 'item_00268', 'item_00269', 'item_00270', 'item_00271', 'item_00272', 'item_00273', 'item_00274', 'item_00275', 'item_00276', 'item_00277', 'item_00278', 'item_00279', 'item_00280', 'item_00281', 'item_00282', 'item_00283', 'item_00284', 'item_00285', 'item_00286', 'item_00287', 'item_00288', 'item_00289'], 'stage': 'science_0', 'status': 'planned', 'job_id': None, 'publications': 256, 'shots': 63712, 'QPY_bytes_sum': 7406698, 'instructions_sum': 129484}
{'batch': 3, 'items': ['item_00290', 'item_00291', 'item_00292', 'item_00293', 'item_00294', 'item_00295', 'item_00296', 'item_00297', 'item_00298', 'item_00299', 'item_00300', 'item_00301', 'item_00302', 'item_00303', 'item_00304', 'item_00305', 'item_00306', 'item_00307', 'item_00308', 'item_00309', 'item_00310', 'item_00311', 'item_00312', 'item_00313', 'item_00314', 'item_00315', 'item_00316', 'item_00317', 'item_00318', 'item_00319', 'item_00320', 'item_00321', 'item_00322', 'item_00323', 'item_00324', 'item_00325', 'item_00326', 'item_00327', 'item_00328', 'item_00329', 'item_00330', 'item_00331', 'item_00332', 'item_00333', 'item_00334', 'item_00335', 'item_00336', 'item_00337', 'item_00338', 'item_00339', 'item_00340', 'item_00341', 'item_00342', 'item_00343', 'item_00344', 'item_00345', 'item_00346', 'item_00347', 'item_00348', 'item_00349', 'item_00350', 'item_00351', 'item_00352', 'item_00353', 'item_00354', 'item_00355', 'item_00356', 'item_00357', 'item_00358', 'item_00359', 'item_00360', 'item_00361', 'item_00362', 'item_00363', 'item_00364', 'item_00365', 'item_00366', 'item_00367', 'item_00368', 'item_00369', 'item_00370', 'item_00371', 'item_00372', 'item_00373', 'item_00374', 'item_00375', 'item_00376', 'item_00377', 'item_00378', 'item_00379', 'item_00380', 'item_00381', 'item_00382', 'item_00383', 'item_00384', 'item_00385', 'item_00386', 'item_00387', 'item_00388', 'item_00389', 'item_00390', 'item_00391', 'item_00392', 'item_00393', 'item_00394', 'item_00395', 'item_00396', 'item_00397', 'item_00398', 'item_00399', 'item_00400', 'item_00401', 'item_00402', 'item_00403', 'item_00404', 'item_00405', 'item_00406', 'item_00407', 'item_00408', 'item_00409', 'item_00410', 'item_00411', 'item_00412', 'item_00413', 'item_00414', 'item_00415', 'item_00416', 'item_00417', 'item_00418', 'item_00419', 'item_00420', 'item_00421', 'item_00422', 'item_00423', 'item_00424', 'item_00425', 'item_00426', 'item_00427', 'item_00428', 'item_00429', 'item_00430', 'item_00431', 'item_00432', 'item_00433', 'item_00434', 'item_00435', 'item_00436', 'item_00437', 'item_00438', 'item_00439', 'item_00440', 'item_00441', 'item_00442', 'item_00443', 'item_00444', 'item_00445', 'item_00446', 'item_00447', 'item_00448', 'item_00449', 'item_00450', 'item_00451', 'item_00452', 'item_00453', 'item_00454', 'item_00455', 'item_00456', 'item_00457', 'item_00458', 'item_00459', 'item_00460', 'item_00461', 'item_00462', 'item_00463', 'item_00464', 'item_00465', 'item_00466', 'item_00467', 'item_00468', 'item_00469', 'item_00470', 'item_00471', 'item_00472', 'item_00473', 'item_00474', 'item_00475'], 'stage': 'science_0', 'status': 'planned', 'job_id': None, 'publications': 186, 'shots': 42720, 'QPY_bytes_sum': 5361767, 'instructions_sum': 93725}
{'batch': 4, 'items': ['item_00476', 'item_00477', 'item_00478', 'item_00479', 'item_00480', 'item_00481', 'item_00482', 'item_00483', 'item_00484', 'item_00485', 'item_00486', 'item_00487', 'item_00488', 'item_00489', 'item_00490', 'item_00491', 'item_00492', 'item_00493', 'item_00494', 'item_00495', 'item_00496', 'item_00497', 'item_00498', 'item_00499', 'item_00500', 'item_00501', 'item_00502', 'item_00503', 'item_00504', 'item_00505', 'item_00506', 'item_00507'], 'stage': 'heldout_0', 'status': 'planned', 'job_id': None, 'publications': 32, 'shots': 2048, 'QPY_bytes_sum': 351908, 'instructions_sum': 5147}
{'batch': 5, 'items': ['item_00508', 'item_00509', 'item_00510', 'item_00511', 'item_00512', 'item_00513', 'item_00514', 'item_00515', 'item_00516', 'item_00517', 'item_00518', 'item_00519', 'item_00520', 'item_00521', 'item_00522', 'item_00523', 'item_00524', 'item_00525', 'item_00526', 'item_00527', 'item_00528', 'item_00529', 'item_00530', 'item_00531', 'item_00532', 'item_00533', 'item_00534', 'item_00535', 'item_00536', 'item_00537', 'item_00538', 'item_00539', 'item_00540', 'item_00541', 'item_00542', 'item_00543', 'item_00544', 'item_00545', 'item_00546', 'item_00547', 'item_00548', 'item_00549', 'item_00550', 'item_00551', 'item_00552', 'item_00553', 'item_00554', 'item_00555', 'item_00556', 'item_00557', 'item_00558', 'item_00559', 'item_00560', 'item_00561', 'item_00562', 'item_00563', 'item_00564', 'item_00565', 'item_00566', 'item_00567', 'item_00568', 'item_00569', 'item_00570', 'item_00571', 'item_00572', 'item_00573', 'item_00574', 'item_00575', 'item_00576', 'item_00577', 'item_00578', 'item_00579', 'item_00580', 'item_00581', 'item_00582', 'item_00583', 'item_00584', 'item_00585', 'item_00586', 'item_00587', 'item_00588', 'item_00589', 'item_00590', 'item_00591', 'item_00592', 'item_00593', 'item_00594', 'item_00595', 'item_00596', 'item_00597', 'item_00598', 'item_00599', 'item_00600', 'item_00601',

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

'item_00602', 'item_00603', 'item_00604', 'item_00605', 'item_00606', 'item_00607', 'item_00608', 'item_00609', 'item_00610', 'item_00611', 'item_00612', 'item_00613', 'item_00614', 'item_00615', 'item_00616', 'item_00617', 'item_00618', 'item_00619', 'item_00620', 'item_00621', 'item_00622', 'item_00623', 'item_00624', 'item_00625', 'item_00626', 'item_00627', 'item_00628', 'item_00629', 'item_00630', 'item_00631', 'item_00632', 'item_00633', 'item_00634', 'item_00635', 'item_00636', 'item_00637', 'item_00638', 'item_00639', 'item_00640', 'item_00641', 'item_00642', 'item_00643', 'item_00644', 'item_00645', 'item_00646', 'item_00647', 'item_00648', 'item_00649', 'item_00650', 'item_00651', 'item_00652', 'item_00653', 'item_00654', 'item_00655', 'item_00656', 'item_00657', 'item_00658', 'item_00659', 'item_00660', 'item_00661', 'item_00662', 'item_00663', 'item_00664', 'item_00665', 'item_00666', 'item_00667', 'item_00668', 'item_00669', 'item_00670', 'item_00671', 'item_00672', 'item_00673', 'item_00674', 'item_00675', 'item_00676', 'item_00677', 'item_00678', 'item_00679', 'item_00680', 'item_00681', 'item_00682', 'item_00683', 'item_00684', 'item_00685', 'item_00686', 'item_00687', 'item_00688', 'item_00689', 'item_00690', 'item_00691', 'item_00692', 'item_00693', 'item_00694', 'item_00695', 'item_00696', 'item_00697', 'item_00698', 'item_00699', 'item_00700', 'item_00701', 'item_00702', 'item_00703', 'item_00704', 'item_00705', 'item_00706', 'item_00707', 'item_00708', 'item_00709', 'item_00710', 'item_00711', 'item_00712', 'item_00713', 'item_00714', 'item_00715', 'item_00716', 'item_00717', 'item_00718', 'item_00719', 'item_00720', 'item_00721', 'item_00722', 'item_00723', 'item_00724', 'item_00725', 'item_00726', 'item_00727', 'item_00728', 'item_00729', 'item_00730', 'item_00731', 'item_00732', 'item_00733', 'item_00734', 'item_00735', 'item_00736', 'item_00737', 'item_00738', 'item_00739', 'item_00740', 'item_00741', 'item_00742', 'item_00743', 'item_00744', 'item_00745', 'item_00746', 'item_00747', 'item_00748', 'item_00749', 'item_00750', 'item_00751', 'item_00752', 'item_00753', 'item_00754', 'item_00755', 'item_00756', 'item_00757', 'item_00758', 'item_00759', 'item_00760', 'item_00761', 'item_00762', 'item_00763'], 'stage': 'science_1', 'status': 'planned', 'job_id': None, 'publications': 256, 'shots': 54304, 'QPY_bytes_sum': 7409342, 'instructions_sum': 129519}
{'batch': 6, 'items': ['item_00764', 'item_00765', 'item_00766', 'item_00767', 'item_00768', 'item_00769', 'item_00770', 'item_00771', 'item_00772', 'item_00773', 'item_00774', 'item_00775', 'item_00776', 'item_00777', 'item_00778', 'item_00779', 'item_00780', 'item_00781', 'item_00782', 'item_00783', 'item_00784', 'item_00785', 'item_00786', 'item_00787', 'item_00788', 'item_00789', 'item_00790', 'item_00791', 'item_00792', 'item_00793', 'item_00794', 'item_00795', 'item_00796', 'item_00797', 'item_00798', 'item_00799', 'item_00800', 'item_00801', 'item_00802', 'item_00803', 'item_00804', 'item_00805', 'item_00806', 'item_00807', 'item_00808', 'item_00809', 'item_00810', 'item_00811', 'item_00812', 'item_00813', 'item_00814', 'item_00815', 'item_00816', 'item_00817', 'item_00818', 'item_00819', 'item_00820', 'item_00821', 'item_00822', 'item_00823', 'item_00824', 'item_00825', 'item_00826', 'item_00827', 'item_00828', 'item_00829', 'item_00830', 'item_00831', 'item_00832', 'item_00833', 'item_00834', 'item_00835', 'item_00836', 'item_00837', 'item_00838', 'item_00839', 'item_00840', 'item_00841', 'item_00842', 'item_00843', 'item_00844', 'item_00845', 'item_00846', 'item_00847', 'item_00848', 'item_00849', 'item_00850', 'item_00851', 'item_00852', 'item_00853', 'item_00854', 'item_00855', 'item_00856', 'item_00857', 'item_00858', 'item_00859', 'item_00860', 'item_00861', 'item_00862', 'item_00863', 'item_00864', 'item_00865', 'item_00866', 'item_00867', 'item_00868', 'item_00869', 'item_00870', 'item_00871', 'item_00872', 'item_00873', 'item_00874', 'item_00875', 'item_00876', 'item_00877', 'item_00878', 'item_00879', 'item_00880', 'item_00881', 'item_00882', 'item_00883', 'item_00884', 'item_00885', 'item_00886', 'item_00887', 'item_00888', 'item_00889', 'item_00890', 'item_00891', 'item_00892', 'item_00893', 'item_00894', 'item_00895', 'item_00896', 'item_00897', 'item_00898', 'item_00899', 'item_00900', 'item_00901', 'item_00902', 'item_00903', 'item_00904', 'item_00905', 'item_00906', 'item_00907', 'item_00908', 'item_00909', 'item_00910', 'item_00911', 'item_00912', 'item_00913', 'item_00914', 'item_00915', 'item_00916', 'item_00917', 'item_00918', 'item_00919', 'item_00920', 'item_00921', 'item_00922', 'item_00923', 'item_00924', 'item_00925', 'item_00926', 'item_00927', 'item_00928', 'item_00929', 'item_00930', 'item_00931', 'item_00932', 'item_00933', 'item_00934', 'item_00935', 'item_00936', 'item_00937', 'item_00938', 'item_00939', 'item_00940', 'item_00941', 'item_00942', 'item_00943', 'item_00944', 'item_00945', 'item_00946', 'item_00947', 'item_00948', 'item_00949'], 'stage': 'science_1', 'status': 'planned', 'job_id': None, 'publications': 186, 'shots': 52128, 'QPY_bytes_sum': 5356693, 'instructions_sum': 93644}
{'batch': 7, 'items': ['item_00950', 'item_00951', 'item_00952', 'item_00953', 'item_00954', 'item_00955', 'item_00956', 'item_00957', 'item_00958', 'item_00959', 'item_00960', 'item_00961', 'item_00962', 'item_00963', 'item_00964', 'item_00965', 'item_00966', 'item_00967', 'item_00968', 'item_00969', 'item_00970', 'item_00971', 'item_00972', 'item_00973', 'item_00974', 'item_00975', 'item_00976', 'item_00977', 'item_00978', 'item_00979',

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

'item_00980', 'item_00981'], 'stage': 'heldout_1', 'status': 'planned', 'job_id': None, 'publications': 32, 'shots': 2048, 'QPY_bytes_sum': 351908, 'instructions_sum': 5147}
{'batch': 8, 'items': ['item_00982', 'item_00983', 'item_00984', 'item_00985', 'item_00986', 'item_00987', 'item_00988', 'item_00989', 'item_00990', 'item_00991', 'item_00992', 'item_00993', 'item_00994', 'item_00995', 'item_00996', 'item_00997', 'item_00998', 'item_00999', 'item_01000', 'item_01001', 'item_01002', 'item_01003', 'item_01004', 'item_01005', 'item_01006', 'item_01007', 'item_01008', 'item_01009', 'item_01010', 'item_01011', 'item_01012', 'item_01013'], 'stage': 'assignment_end', 'status': 'planned', 'job_id': None, 'publications': 32, 'shots': 16384, 'QPY_bytes_sum': 351908, 'instructions_sum': 5147}
{'jobs': 9, 'circuit_executions': 1014, 'unique_saved_circuits': 882, 'shots_by_kind': {'assignment': 32768, 'diagnostic': 256, 'science': 212864, 'validation': 4096}, 'total_shots': 249984, 'shot_cap': 250000, 'job_cap': 12, 'publications_per_job_limit': 256, 'evidence': '/content/hydrogen_compact_screen_20261015T183419_201635Z_a76143ce'}
All deterministic, negative-control, and actual-circuit checks: PASS.
No cancelled experiment was accessed. No QPU job has been submitted.
Mixed per-PUB shots use SamplerPub; runtime 0.50 may emit deprecation warnings.
Server-side twirling and DD disabled; all client transformations are saved.
No automatic workload expansion or retry.
/tmp/ipykernel_8303/942387080.py:1329: DeprecationWarning: The SamplerV2 class is deprecated as of qiskit-ibm-runtime 0.50.0 and will be removed no sooner than 3 months after the release date. `qiskit_ibm_runtime.executor_sampler.Sampler` provides a new Sampler implementation that runs all the pre- and post-processing in the client side. The top-level `qiskit_ibm_runtime.Sampler` import will be updated to point to the client-side sampler in an upcoming release, with the legacy Sampler being available until the deprecation is completed.
  sampler = SamplerV2(mode=backend, options=options)
Submitted: 0 db8hsdob3g5c73ega7r0
Submitted: 1 db8hsrob3g5c73ega8sg
/tmp/ipykernel_8303/942387080.py:1353: DeprecationWarning: Specifying different 'shots' across pubs is deprecated as of qiskit-ibm-runtime 0.48.0 and will be removed no sooner than 3 months after the release date. Submit one job for each desired shot count instead. To reduce overhead, Consider submitting these jobs inside a Batch execution mode.
  job = sampler.run(pubs)
Submitted: 2 db8ht2peukqs73aqkrf0
Submitted: 3 db8hu3lckpjs73b11ddg
Submitted: 4 db8i0he94fic73etee1g
Submitted: 5 db8i3m0b3g5c73egak10
Submitted: 6 db8ia1peukqs73aqlmhg
Submitted: 7 db8iea9eukqs73aqm10g
Submitted: 8 db8ifatckpjs73b12j7g

=== MEASURED SCREENING RESULTS — HARTREE ===

 COBYLA fixed-state reference: -0.46097392795976344
baseline_raw 0.0018641080379016989 95% interval: [-0.053190014897288 0.0562066504680292]
candidate_raw_factor1 0.12178396461105473 95% interval: [0.0571704701937871 0.1958049063023849]
UNAVAILABLE/REJECTED by readout or bootstrap-conditioning checks. No calibration draw is silently dropped. Raw results retained.
±0.05 Ha target: NOT ACHIEVED

 SPSA fixed-state reference: -0.46077921338321903
baseline_raw 0.026773392983728245 95% interval: [-0.0272799019414917 0.0812361495220332]
candidate_raw_factor1 0.1963914484150516 95% interval: [0.1277962843183974 0.2718736885139076]

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

UNAVAILABLE/REJECTED by readout or bootstrap-conditioning checks. No calibration draw is silently dropped. Raw results retained.
±0.05 Ha target: NOT ACHIEVED

No chemical-accuracy or guaranteed device-improvement claim.
No automatic extra jobs, shots, randomizations, or follow-up run.
Job IDs: ['db8hsdob3g5c73ega7r0', 'db8hsrob3g5c73ega8sg', 'db8ht2peukqs73aqkrf0', 'db8hu3lckpjs73b11ddg', 'db8i0he94fic73etee1g', 'db8i3m0b3g5c73egak10', 'db8ia1peukqs73aqlmhg', 'db8iea9eukqs73aqm10g', 'db8ifatckpjs73b12j7g']
Evidence: /content/hydroge_compact_screen_20261015T183419_201635Z_a76143ce

Sixth-run conclusion

Run 6 completed the resource-bounded screening on the ibm_marrakesh backend using nine jobs, 1,014 circuit instances, and 249,984 requested shots. Reported deterministic, negative-control, and actual-circuit checks passed under ideal execution. The raw candidate factor-1 means were +0.121784 Ha for the COBYLA-derived state and +0.196391 Ha for the SPSA-derived state, farther from their targets than the same-run baseline means of +0.001864 Ha and +0.026773 Ha. The ±0.05 Ha target was not achieved. The console combined readout and bootstrap failures into one message; the subsequent offline analysis specifically reproduced failure of pooled held-out readout validation for prepared indices 17 and 25. Corrected and zero-noise-extrapolated estimates remained unavailable. This run established a completed screening outcome and operational budget reduction, not improved energy accuracy or proof that device noise alone caused the discrepancy.

Screening-method qualification

The selected Run 6 configuration uses physical path (2, 3, 4, 5, 6), reverse-direction preparation, ASAP scheduling, and no dynamical decoupling. Its candidate circuits include explicit Pauli randomization and folds at factors 1, 3, and 5. Some DD candidates failed a local “Nonterminal active operation” check and were not submitted. Server-side DD and twirling were disabled because the client transformations were explicit. The log therefore does not support a claim that DD improved the final hardware run.

The candidate changes more than one physical implementation feature relative to the baseline. Its worse factor-1 mean does not isolate randomization, direction, or a particular error channel as the cause. Randomized compiling preserves the ideal circuit while tailoring the noise under stated assumptions [27]; finite ensembles and added physical pulses still require empirical evaluation. The selected calibration/exposure score is a heuristic, not a predicted energy or an optimality certificate [30, 32].

Prompt sent to Quantum Flow for final offline verification

We are closing Case Study 2. Perform final offline verification and reporting for the completed compact Part II screening. Do not propose or initiate another hardware experiment. Return one complete, self-contained Colab/Jupyter finalization cell. When executed, it must work only with the exact compact-screening source and locally available artifacts from: /content/hydrogen_compact_screen_20261015T183419_201635Z_a76143ce No IBM connection, job submission, additional measurements, recompilation, parameter optimization, or recovery of the cancelled 115-job experiment. Do not assume access to my notebook from this conversation; generate the cell that performs the checks in the notebook environment. Verify the completed record. Inspect the available manifest, saved source, circuit hashes, job associations, science counts, calibration counts, and held-out validation data. Verify the reported nine-job, 1,014-circuit-instance workload and reconcile requested versus returned shots. Independently reconstruct the baseline and candidate raw energies for both fixed states, by block and measurement group, preserving the actual shot allocations, randomization weights, identity contribution, and measurement mappings. Export the acquired raw results at folding factors 1, 3, and 5, including those omitted from the console summary. Explain the mitigation rejection precisely. Print the exact failed condition or exception and its numerical evidence: assignment-matrix condition numbers and minimum singular values, held-out validation statistics and thresholds, and failed bootstrap-draw counts and reasons. Distinguish calibration

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

incompatibility, numerical conditioning, insufficient statistical evidence, and implementation defects; do not assert a cause that the evidence cannot establish. Preserve the original rejection decision. Do not relax thresholds, discard unfavorable draws, select a different estimator, or fit corrections to the reference energies. Any demonstrable arithmetic or reporting correction must appear separately as a documented reanalysis, without overwriting the original results. Finalize the statistical interpretation. Verify the reported intervals and state exactly which uncertainty sources they include. Report baseline-versus-candidate paired differences where the saved data support them, keeping shot uncertainty, randomization variability, shared-calibration uncertainty, and between-block variation distinct. Explicitly acknowledge the limited repeatability evidence from two blocks and four randomizations. Keep corrected/ZNE estimates unavailable wherever the prescribed checks fail. Preserve the conclusion that the ±0.05 Ha target was not achieved; do not replace it with a weaker retrospective success criterion. Preserve the study's scientific scope. Part I remains authoritative and unchanged: the same Hamiltonian, five-qubit encoding, two fixed optimized states, 63 Pauli terms, and 17 measurement groups. Distinguish ideal simulation validation from physical-QPU accuracy, and fixed-state evaluation from a QPU-driven VQE optimization. Keep earlier completed experiments and their documented improvements separate from this final screening; do not pool results across different backends or configurations. Describe the cancelled large run only as an incomplete, resource-limited experiment, without inferring its unobserved final outcome. Produce closure artifacts: a concise final report, numerical comparison tables, an artifact inventory with hashes, and an archive of available evidence excluding credentials. Include a case-study-ready conclusion and limitations section. Preserve all original code, comments, counts, and outputs unchanged. If any required artifact is unavailable, list exactly which verification cannot be completed and generate a log-based report using the attached output. Do not invent missing measurements, claim unperformed checks, request a new QPU run, or make missing artifacts a reason to continue experimentation. The task ends with documentation and closure.

Code was
```
<code ommited>
```

Results were
```
<omitted>
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

Offline verification and closure code (verbatim)

```
# Case Study 2 — FINAL OFFLINE VERIFICATION AND CLOSURE
#
# This cell:
#   * reads ONLY the specified completed compact-screening directory;
#   * never connects to IBM, retrieves/submits jobs, recompiles circuits,
#     optimizes parameters, or executes saved experiment Python;
#   * independently reconstructs the exact compact-screen schema and math;
#   * preserves all original files;
#   * writes reports/tables/diagnostics and a credential-screened archive
#     to a NEW SIBLING directory.
#
# No Part I notebook variables are required or modified.
# Missing artifacts produce explicit verification gaps and a log-based closure.
# No pickle, marshal, object-array, or saved-code execution is performed.
#
# Local dependencies: numpy. Qiskit is optional for QPY inspection.
# No dependency is installed or upgraded.

def _finalize_case_study_2_offline():
    from pathlib import Path
    from collections import Counter, defaultdict
    from datetime import datetime, timezone
    import ast
    import csv
    import hashlib
    import io
    import json
    import math
    import os
    import re
    import tarfile
    import traceback
    import uuid

    ROOT = Path(
        "/content/hydrogen_compact_screen_20261015T183419_201635Z_a76143ce"
    )
    OUT = ROOT.parent / (
        ROOT.name + "_FINAL_CLOSURE_"
        + datetime.now(timezone.utc).strftime("%Y%m%dT%H%M%S_%fZ")
        + "_" + uuid.uuid4().hex[:8]
    )
    OUT.mkdir(parents=True, mode=0o700, exist_ok=False)

    try:
        import numpy as np
    except ImportError:
        np = None
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
STATES = ["COBYLA", "SPSA"]
LOG_REFERENCES = [-0.46097392795976344, -0.46077921338321903]
LOG_JOBS = [
    "db8hsdob3g5c73ega7r0", "db8hsrob3g5c73ega8sg",
    "db8ht2peukqs73aqkrf0", "db8hu3lckpjs73b11ddg",
    "db8i0he94fic73etee1g", "db8i3m0b3g5c73egak10",
    "db8ia1peukqs73aqlmhg", "db8iea9eukqs73aqm10g",
    "db8ifatckpjs73b12j7g",
]
LOG_RESULTS = [
    {
        "state": "COBYLA", "processing": "baseline raw", "factor": 1,
        "energy": 0.0018641080379016989,
        "low": -0.053190014897288, "high": 0.0562066504680292,
    },
    {
        "state": "COBYLA", "processing": "candidate raw", "factor": 1,
        "energy": 0.12178396461105473,
        "low": 0.0571704701937871, "high": 0.1958049063023849,
    },
    {
        "state": "SPSA", "processing": "baseline raw", "factor": 1,
        "energy": 0.026773392983728245,
        "low": -0.0272799019414917, "high": 0.0812361495220332,
    },
    {
        "state": "SPSA", "processing": "candidate raw", "factor": 1,
        "energy": 0.1963914484150516,
        "low": 0.1277962843183974, "high": 0.2718736885139076,
    },
]
EXPECTED = {
    "jobs": 9, "instances": 1014, "unique_circuits": 882,
    "shots": 249984,
    "shots_by_kind": {
        "assignment": 32768, "diagnostic": 256,
        "science": 212864, "validation": 4096,
    },
}

gaps, checks, exclusions, inventory = [], [], [], []
tables = {}
artifacts = {}
manifest = local = analysis = saved_readout = source_tree = None
source_text = None
cfg = None
counts_by_item = {}
independent_raw_blocks = independent_raw_groups = None
replay = {}
references = None
authority_verified = False
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
full_counts_verified = False
mapping_verified = False
qpy_verified = False

def normalize(value):
    if np is not None:
        if isinstance(value, np.ndarray):
            return normalize(value.tolist())
        if isinstance(value, np.generic):
            return normalize(value.item())
    if isinstance(value, complex):
        return {"real": value.real, "imag": value.imag}
    if isinstance(value, dict):
        return {str(k): normalize(v) for k, v in value.items()}
    if isinstance(value, (list, tuple)):
        return [normalize(v) for v in value]
    if isinstance(value, (Path, datetime)):
        return str(value)
    if isinstance(value, float) and not math.isfinite(value):
        return {"nonfinite": str(value)}
    if value is None or isinstance(value, (str, int, float, bool)):
        return value
    return repr(value)

def dump(name, value):
    path = OUT / name
    path.write_text(
        json.dumps(normalize(value), indent=2, ensure_ascii=False, allow_nan=False),
        encoding="utf-8",
    )
    os.chmod(path, 0o600)

def csv_table(name, rows):
    tables[name] = rows
    keys = list(dict.fromkeys(k for row in rows for k in row))
    with (OUT / name).open("w", newline="", encoding="utf-8") as handle:
        if not keys:
            handle.write("# No independently supported rows available.\n")
            return
        writer = csv.DictWriter(handle, fieldnames=keys)
        writer.writeheader()
        for row in rows:
            encoded = {}
            for key in keys:
                value = normalize(row.get(key))
                encoded[key] = (
                    json.dumps(value, ensure_ascii=False)
                    if isinstance(value, (dict, list)) else value
                )
            writer.writerow(encoded)
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
def gap(stage, detail):
    record = {"stage": stage, "detail": str(detail)}
    if record not in gaps:
        gaps.append(record)

def require(condition, message):
    if not bool(condition):
        raise RuntimeError(message)

def sha_bytes(data):
    return hashlib.sha256(data).hexdigest()

def sha_file(path):
    hasher = hashlib.sha256()
    with path.open("rb") as handle:
        for chunk in iter(lambda: handle.read(1024 * 1024), b""):
            hasher.update(chunk)
    return hasher.hexdigest()

def array_hash(value):
    a = np.ascontiguousarray(value)
    return sha_bytes(str(a.dtype).encode() + str(a.shape).encode() + a.tobytes())

def phase(name, function):
    try:
        result = function()
        checks.append({"stage": name, "completed": True})
        return result
    except Exception as error:
        gap(name, f"{type(error).__name__}: {error}")
        checks.append({
            "stage": name, "completed": False,
            "exception": f"{type(error).__name__}: {error}",
        })
        return None

# Credential detection is heuristic and documented. Nothing is redacted in
# place. Excluded original files remain untouched.
secret_patterns = [
    re.compile(rb"-----BEGIN (?:RSA |EC |OPENSSH |DSA )?PRIVATE KEY-----"),
    re.compile(
        rb"""(?im)["']?(?:api[_-]?key|access[_-]?token|refresh[_-]?token|"""
        rb"""password|client[_-]?secret|ibm_quantum_token)["']?\s*[:=]\s*"""
        rb"""["']([^"'\r\n]{12,})["']"""
    ),
    re.compile(
        rb"\beyJ[A-Za-z0-9_-]{15,}\.[A-Za-z0-9_-]{10,}\."
        rb"[A-Za-z0-9_-]{10,}\b"
    ),
]
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    def credential_reason(path):
        name = path.name.lower()
        if (
            name in {
                ".env", ".netrc", "credentials", "credentials.json",
                "qiskit-ibm.json", "qiskitrc",
            }
            or name.endswith((".pem", ".key", ".p12", ".pfx"))
            or any(word in name for word in ("credential", "access_token", "refresh_token"))
        ):
            return "credential-like filename"
        tail = b""
        with path.open("rb") as handle:
            for chunk in iter(lambda: handle.read(1024 * 1024), b""):
                data = tail + chunk
                for pattern in secret_patterns:
                    match = pattern.search(data)
                    if match:
                        candidate = match.group(1) if match.lastindex else match.group(0)
                        if not any(word in candidate.lower() for word in (
                            b"placeholder", b"your_", b"replace", b"os.environ",
                            b"getenv", b"<token", b"insert_",
                        )):
                            return "credential-like content"
                tail = data[-8192:]
        return None

    def inventory_originals():
        require(ROOT.is_dir() and not ROOT.is_symlink(),
                f"Specified experiment directory unavailable: {ROOT}")
        for path in sorted(ROOT.rglob("*")):
            relative = str(path.relative_to(ROOT))
            if path.is_symlink() or any(
                ancestor.is_symlink()
                for ancestor in path.parents
                if ancestor != ROOT.parent
            ):
                exclusions.append({"path": relative, "reason": "symlink not followed"})
                continue
            if not path.is_file():
                continue
            try:
                reason = credential_reason(path)
                record = {
                    "path": relative, "bytes": path.stat().st_size,
                    "sha256": sha_file(path),
                    "credential_screen": reason or "no heuristic match",
                }
                inventory.append(record)
                if reason:
                    exclusions.append({"path": relative, "reason": reason})
```

```
            else:
                artifacts[relative] = path
        except Exception as error:
            exclusions.append({
                "path": relative, "reason": f"unreadable: {type(error).__name__}"
            })
    return True

phase("artifact inventory and credential screening", inventory_originals)

def read_json(relative, required=True):
    path = artifacts.get(relative)
    if path is None:
        if required:
            raise FileNotFoundError(
                f"{ROOT / relative}: missing, unreadable, or excluded by credential screening"
            )
        return None

    def pairs(items):
        output = {}
        for key, value in items:
            require(key not in output, f"Duplicate JSON key {key!r}: {relative}")
            output[key] = value
        return output

    def bad_constant(value):
        raise ValueError(f"Invalid JSON nonfinite constant {value}: {relative}")

    return json.loads(
        path.read_text(encoding="utf-8"),
        object_pairs_hook=pairs,
        parse_constant=bad_constant,
    )

manifest = phase("load exact compact manifest", lambda: read_json("manifest.json"))
local = phase("load local validation", lambda: read_json("local_validation.json"))
analysis = phase("load original numerical report", lambda: read_json("analysis.json"))
saved_readout = phase(
    "load original readout decision", lambda: read_json("readout_validation.json")
)

# ---------- Static inspection of the actual compact source ----------
source_functions = {}
static_excerpts = {}
source_verified = False

def inspect_source():
    nonlocal source_text, source_tree, cfg, source_verified
    require(manifest is not None, "Manifest unavailable.")
    require(manifest.get("schema") == "compact-screen-1", "Unexpected schema.")
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
source_path = artifacts.get("fresh_function_source.py")
require(source_path is not None,
      "Exact saved fresh_function_source.py unavailable; source replay cannot be certified.")
source_text = source_path.read_text(encoding="utf-8")
recorded = manifest.get("source", {}).get("function_source_sha256")
require(recorded is not None, "No recorded function source-byte hash.")
actual = sha_bytes(source_text.encode("utf-8"))
require(actual == recorded, "Recorded source-byte SHA256 mismatch.")
source_tree = ast.parse(source_text)
functions = [
   node for node in source_tree.body
   if isinstance(node, ast.FunctionDef)
   and node.name == "_fresh_compact_hydrogen_screen"
]
require(len(functions) == 1, "Exact compact entry function not identified.")
function = functions[0]
for node in function.body:
   if isinstance(node, ast.FunctionDef):
      source_functions[node.name] = node
   if isinstance(node, ast.Assign):
      if any(isinstance(t, ast.Name) and t.id == "CFG" for t in node.targets):
         cfg = ast.literal_eval(node.value)
require(cfg == manifest["configuration"], "Source/manifest CFG mismatch.")
require(
   cfg["blocks"] == 2 and cfg["randomizations"] == 4
   and cfg["engineering_target_Ha"] == 0.05,
   "Compact scope/acceptance rule mismatch.",
)
for name in (
   "decode", "evaluate", "safe_inverse", "rng", "summary", "interval"
):
   require(name in source_functions, f"Saved pure source definition absent: {name}")
   static_excerpts[name] = ast.get_source_segment(source_text, source_functions[name])
lines = source_text.splitlines()
relevant = [
   "invalid_calibration_draws", "if invalid_calibration_draws:",
   "readout_valid = bool(", "np.random.SeedSequence",
   "raw_joint[iteration", "corr_joint[iteration",
   "np.quantile", "heldout_TV.max()", "recovery_TV.max()",
]
selected_lines = set()
for i, line in enumerate(lines):
   if any(token in line for token in relevant):
      selected_lines.update(range(max(0, i - 3), min(len(lines), i + 7)))
static_excerpts["decision_and_resampling_lines"] = [
   {"line": i + 1, "text": lines[i]} for i in sorted(selected_lines)
]
source_verified = True
dump("exact_source_inspection.json", {
   "byte_hash": actual,
   "source_executed": False,
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        "code_object_hash_qualification": (
            "The recorded code-object digest is inventoried but not regenerated "
            "by executing or compiling the saved experiment."
        ),
        "pure_function_and_decision_excerpts": static_excerpts,
    })
    return {"source_sha256": actual, "configuration_matches": True}

phase("static executed-source verification", inspect_source)
if manifest is not None and cfg is None:
    # Manifest values can support arithmetic, but do not certify source replay.
    cfg = manifest.get("configuration")
    gap("source-specific replay",
        "Using recorded manifest configuration for independent arithmetic only; "
        "exact source verification is unavailable.")

# ---------------------- Explicit compact-schema associations ----------------
plan = registry = jobs = None
association_rows, job_rows, circuit_rows = [], [], []
raw_groups_table, block_table, result_table, paired_table = [], [], [], []
pub_table, outcome_table = [], []
data = None
assignment_counts = heldout_counts = None
group_values = None
Hsaved = vectors = ideal_group = None

def decode(counts, shots):
    require(np is not None, "NumPy unavailable.")
    output = np.zeros(32, np.int64)
    require(isinstance(counts, dict) and bool(counts), "Missing counts.")
    for bits, number in counts.items():
        require(
            isinstance(bits, str) and len(bits) == 5
            and set(bits) <= {"0", "1"},
            "Unsupported count string; expected the source's single five-bit register.",
        )
        require(
            isinstance(number, int) and not isinstance(number, bool) and number >= 0,
            "Invalid count multiplicity.",
        )
        output[int(bits, 2)] += number
    require(int(output.sum()) == int(shots), "Requested/returned shot mismatch.")
    return output

def verify_workload():
    nonlocal plan, registry, jobs, full_counts_verified
    require(np is not None and manifest is not None, "NumPy/manifest unavailable.")
    plan, registry, jobs = manifest["plan"], manifest["registry"], manifest["jobs"]
    require(len(jobs) == 9, "Saved manifest does not have nine jobs.")
    require([job["batch"] for job in jobs] == list(range(9)), "Batch order mismatch.")
    require(len(plan) == 1014, "Saved plan does not have 1,014 executions.")
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    require(len(registry) == 882, "Saved registry does not have 882 unique byte artifacts.")
    lookup = {item["item_id"]: item for item in plan}
    require(len(lookup) == len(plan), "Duplicate planned item ID.")
    require(
        [key for job in jobs for key in job["items"]]
        == [item["item_id"] for item in plan],
        "Batch-to-publication ordering differs from the frozen plan.",
    )
    require([job["job_id"] for job in jobs] == LOG_JOBS, "Manifest/log job IDs differ.")
    expected_kind_totals = Counter()
    returned_kind_totals = Counter()
    for item in plan:
        require(item["circuit_id"] in registry, "Planned circuit absent from registry.")
        expected_kind_totals[item["kind"]] += int(item["shots"])
    for job in jobs:
        relative = f"jobs/batch_{job['batch']:03d}.json"
        path = artifacts.get(relative)
        require(path is not None, f"Required actual counts unavailable: {ROOT / relative}")
        recorded_hash = job.get("raw_counts_sha256")
        require(recorded_hash is not None, f"Raw-count hash missing for batch {job['batch']}.")
        require(sha_file(path) == recorded_hash, f"Raw-count hash mismatch: {relative}")
        content = read_json(relative)
        require(content["job_id"] == job["job_id"], "Saved job-ID association mismatch.")
        require(content["batch"] == job["batch"], "Saved batch association mismatch.")
        require(
            [row["item_id"] for row in content["rows"]] == job["items"],
            "Saved publication order mismatch.",
        )
        actual_total = 0
        for pub_index, row in enumerate(content["rows"]):
            item = lookup[row["item_id"]]
            require(
                row["item_id"] not in counts_by_item,
                "Duplicate returned publication identity.",
            )
            require(
                row["circuit_id"] == item["circuit_id"]
                and row["job_id"] == job["job_id"]
                and row["shots"] == item["shots"],
                "Saved publication circuit/job/requested-shot mismatch.",
            )
            frequencies = decode(row["counts"], item["shots"])
            counts_by_item[item["item_id"]] = frequencies
            actual_shots = int(frequencies.sum())
            actual_total += actual_shots
            returned_kind_totals[item["kind"]] += actual_shots
            association_rows.append({
                "batch": job["batch"], "job_id": job["job_id"],
                "publication_index": pub_index, "item_id": item["item_id"],
                "circuit_id": item["circuit_id"], "kind": item["kind"],
                "state": item.get("state"), "block": item.get("block"),
```

```
                "group": item.get("group"), "mode": item.get("mode"),
                "factor": item.get("factor"),
                "randomization": item.get("randomization"),
                "requested_shots": item["shots"],
                "actual_returned_shots": actual_shots,
                "saved_association_verified": True,
                "remote_execution_independently_verified": False,
            })
        require(
            actual_total == job["shots"]
            == sum(lookup[key]["shots"] for key in job["items"]),
            "Job requested/returned total mismatch.",
        )
        job_rows.append({
            "batch": job["batch"], "job_id": job["job_id"],
            "publications": len(job["items"]),
            "requested_shots": job["shots"], "returned_shots": actual_total,
            "raw_counts_sha256": recorded_hash,
            "saved_associations_verified": True,
            "remote_inputs_checked": False,
        })
    require(len(counts_by_item) == 1014, "Not all actual publications are present.")
    require(dict(expected_kind_totals) == EXPECTED["shots_by_kind"],
            "Requested shots by kind differ from the completed record.")
    require(returned_kind_totals == expected_kind_totals,
            "Requested and actual shot totals differ.")
    require(sum(returned_kind_totals.values()) == 249984, "Actual total differs.")
    full_counts_verified = True
    return {
        "nine_jobs": True, "executions": len(plan), "unique_registry_entries": len(registry),
        "requested_shots_by_kind": dict(expected_kind_totals),
        "actual_shots_by_kind": dict(returned_kind_totals),
        "actual_total_shots": sum(returned_kind_totals.values()),
        "qualification": (
            "Local manifest/result associations and byte hashes verified. "
            "No remote service inputs or execution were independently checked."
        ),
    }

reconciliation = phase("nine-job workload and actual counts reconciliation", verify_workload)
csv_table("job_reconciliation.csv", job_rows)
csv_table("publication_associations.csv", association_rows)
dump("workload_reconciliation.json", reconciliation)

# ---------------------- Authoritative arrays and Pauli terms ----------------
def complex_coefficient(value):
    if isinstance(value, dict):
        return complex(value["real"], value["imag"])
    return complex(value)

def verify_scientific_inputs():
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    nonlocal references, authority_verified, group_values, Hsaved, vectors, ideal_group
    require(np is not None and manifest is not None and local is not None,
            "NumPy/manifest/local-validation artifacts unavailable.")
    path = artifacts.get("inputs.npz")
    require(path is not None, f"Required authoritative array file missing: {ROOT / 'inputs.npz'}")
    with np.load(path, allow_pickle=False) as loaded:
        Hsaved = np.array(loaded["H"], dtype=complex, copy=True)
        rsaved = np.array(loaded["r"], copy=True)
        vectors = [
            np.array(loaded["COBYLA"], dtype=complex, copy=True),
            np.array(loaded["SPSA"], dtype=complex, copy=True),
        ]
    a = manifest["authority"]
    require(Hsaved.shape == (32, 32) and rsaved.shape == (32,), "Invalid saved dimensions.")
    require(array_hash(Hsaved) == a["H_sha256"], "Saved H authority hash mismatch.")
    require(array_hash(rsaved) == a["r_sha256"], "Saved grid authority hash mismatch.")
    require([array_hash(v) for v in vectors] == a["state_sha256"],
            "Saved optimized-state authority hash mismatch.")
    require(np.max(np.abs(Hsaved - Hsaved.conj().T)) <= 1e-12, "Saved H is not Hermitian.")
    references = np.array([
        float(np.vdot(v, Hsaved @ v).real) for v in vectors
    ])
    for v in vectors:
        require(v.shape == (32,) and np.all(np.isfinite(v))
                and abs(np.vdot(v, v) - 1) <= 1e-12, "Saved state invalid.")
    require(
        np.max(np.abs(references - np.asarray(a["references_Ha"]))) <= 1e-10,
        "Saved references do not reproduce real(vdot(psi,H@psi)).",
    )
    groups = local["groups"]
    require(len(groups) == 17, "Saved QWC groups incomplete.")
    group_values = np.zeros((17, 32))
    ideal_group = np.zeros((2, 17))
    matrices = {
        "I": np.eye(2, dtype=complex),
        "X": np.array([[0, 1], [1, 0]], complex),
        "Y": np.array([[0, -1j], [1j, 0]], complex),
        "Z": np.diag([1, -1]).astype(complex),
    }
    reconstructed = np.zeros((32, 32), complex)
    labels, identity_count = [], 0
    for g, group in enumerate(groups):
        require(group["index"] == g, "Group index mismatch.")
        axes = ["I"] * 5
        group_matrix = np.zeros((32, 32), complex)
        for label, saved_c in group["terms"]:
            c = complex_coefficient(saved_c)
            require(len(label) == 5 and abs(c.imag) <= 1e-12, "Invalid Pauli term.")
            labels.append(label)
            identity_count += int(label == "IIIII")
            mask = 0
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
            term_matrix = np.array([[1]], complex)
            for axis in label:
                term_matrix = np.kron(term_matrix, matrices[axis])
            group_matrix += c * term_matrix
            for q in range(5):
                axis = label[4 - q]
                if axis != "I":
                    require(axes[q] in ("I", axis), "Saved group is not QWC.")
                    axes[q] = axis
                    mask |= 1 << q
            group_values[g] += c.real * np.array([
                -1 if (k & mask).bit_count() % 2 else 1 for k in range(32)
            ])
        axes = ["Z" if x == "I" else x for x in axes]
        require(axes == group["axes"] and "".join(reversed(axes)) == group["basis"],
                "Saved basis/Pauli ordering mismatch.")
        require(
            np.max(np.abs(group_values[g] - np.asarray(group["values"]))) <= 1e-12,
            "Saved weighted outcomes do not match Pauli parity semantics.",
        )
        reconstructed += group_matrix
        for s in range(2):
            ideal_group[s, g] = np.vdot(vectors[s], group_matrix @ vectors[s]).real
    require(len(labels) == 63 and len(set(labels)) == 63 and identity_count == 1,
            "Term coverage/identity accounting failed.")
    require(np.max(np.abs(reconstructed - Hsaved)) <= 1e-12,
            "Full grouped Hamiltonian does not reconstruct authoritative H.")
    require(np.max(np.abs(ideal_group.sum(axis=1) - references)) <= 1e-10,
            "Exact grouped expectations fail reference reconstruction.")
    authority_verified = True
    return {
        "references_from_unmodified_saved_vectors_Ha": references,
        "authority_hashes_verified": True,
        "Hermiticity_error": float(np.max(np.abs(Hsaved - Hsaved.conj().T))),
        "Hamiltonian_reconstruction_error": float(np.max(np.abs(reconstructed - Hsaved))),
        "unique_Pauli_terms": 63, "groups": 17, "identity_occurrences": identity_count,
        "reference_minus_continuum_Ha": (
            references - a["continuum_reference_Ha"]
        ),
        "finite_representation_difference_Ha": (
            a["finite_reference_Ha"] - a["continuum_reference_Ha"]
        ),
        "classical_variational_residual_Ha": (
            references - a["finite_reference_Ha"]
        ),
        "alpha_full_precision_hex": a["alpha_full_precision_hex"],
        "qualification": (
            "Agreement with locally recorded Part I authority verified. "
            "No live Part I variables were read or changed; external historical "
            "attestation of that authority is outside offline verification."
        ),
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    }

reference_report = phase("saved Part I authority and full Hamiltonian", verify_scientific_inputs)
dump("reference_verification.json", reference_report)

# ------------------------ Circuit file and measurement maps ----------------
def verify_circuit_artifacts():
    nonlocal qpy_verified, mapping_verified
    require(manifest is not None, "Manifest unavailable.")
    try:
        from qiskit import qpy
    except ImportError:
        qpy = None
        gap("QPY semantics", "Local qiskit.qpy unavailable; byte hashes still checked.")

    for cid, record in manifest["registry"].items():
        relative = record["file"]
        require(relative in artifacts, f"Missing circuit artifact: {ROOT / relative}")
        path = artifacts[relative]
        require(
            sha_file(path) == cid == record["sha256"],
            f"Recorded QPY byte hash mismatch: {relative}",
        )
        row = {
            "circuit_id": cid, "file": relative, "byte_sha256_verified": True,
            "QPY_measurement_map_verified": False,
        }
        if qpy is not None:
            with path.open("rb") as handle:
                loaded = qpy.load(handle)
            require(len(loaded) == 1, "Expected one circuit per saved QPY.")
            circuit = loaded[0]
            require(
                circuit.num_clbits == 5 and len(circuit.cregs) == 1
                and circuit.cregs[0].name == manifest["configuration"]["register"],
                "QPY register convention mismatch.",
            )
            mapping = {}
            measured = set()
            after_measurement = []
            twoq = Counter()
            for item in circuit.data:
                qs = [circuit.find_bit(q).index for q in item.qubits]
                if item.operation.name == "measure":
                    require(len(qs) == 1 and len(item.clbits) == 1, "Invalid QPY measure.")
                    c = circuit.find_bit(item.clbits[0]).index
                    require(c not in mapping, "Duplicate QPY measurement destination.")
                    mapping[c] = qs[0]
                    measured.update(qs)
                elif item.operation.name not in ("barrier", "delay"):
                    if measured.intersection(qs):
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
                    after_measurement.append(item.operation.name)
                if len(qs) == 2:
                    twoq[item.operation.name] += 1
        expected_map = tuple(record["measurement_map"])
        require(
            set(mapping) == set(range(5))
            and tuple(mapping[c] for c in range(5)) == expected_map
            and len(set(expected_map)) == 5,
            "QPY/registry logical-classical-to-physical map mismatch.",
        )
        require(not after_measurement, "Saved active instruction follows measurement.")
        row.update({
            "QPY_measurement_map_verified": True,
            "physical_width": circuit.num_qubits,
            "measurement_map": expected_map,
            "native_two_qubit_count": sum(twoq.values()),
            "two_qubit_operations": dict(twoq),
            "depth": circuit.depth(),
            "instructions": len(circuit.data),
        })
    circuit_rows.append(row)
qpy_verified = qpy is not None
require(plan is not None, "Frozen plan unavailable.")
canonical_set = tuple(manifest["physical_readout_set"])
require(len(canonical_set) == 5 and len(set(canonical_set)) == 5,
        "Invalid canonical physical readout set.")
positions = {q: i for i, q in enumerate(canonical_set)}
for item in plan:
    record = manifest["registry"][item["circuit_id"]]
    mapping = record["measurement_map"]
    require(set(mapping) == set(canonical_set), "Science/calibration readout set differs.")
    if item["kind"] == "science":
        explicit = [
            sum(((k >> c) & 1) << positions[mapping[c]] for c in range(5))
            for k in range(32)
        ]
        require(explicit == item["logical_to_canonical_index"],
                "Saved canonical permutation differs from explicit physical mapping.")
        require(len(set(explicit)) == 32, "Nonbijective canonical permutation.")
    else:
        require(tuple(mapping) == canonical_set,
                "Calibration/diagnostic canonical measurement order differs.")
mapping_verified = True
return {
    "registry_byte_hashes_verified": len(circuit_rows),
    "QPY_maps_independently_checked": qpy_verified,
    "plan_permutations_checked_against_registry": True,
    "scope": "Local QPY inspection only; no compilation or simulation.",
}

circuit_report = phase("circuit-byte hashes and actual measurement maps", verify_circuit_artifacts)
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
csv_table("circuit_artifact_verification.csv", circuit_rows)
dump("circuit_verification.json", circuit_report)

# ------------------- Independent source-specific raw reconstruction --------
def reconstruct_raw():
    nonlocal data, assignment_counts, heldout_counts
    nonlocal independent_raw_blocks, independent_raw_groups
    require(
        full_counts_verified and authority_verified and mapping_verified,
        "Complete verified counts, authority, and mappings are required.",
    )
    data = defaultdict(list)
    assignment_counts = np.zeros((2, 32, 32), np.int64)
    heldout_counts = np.zeros((2, 32, 32), np.int64)
    term_covariances = []
    for item in plan:
        counts = counts_by_item[item["item_id"]]
        if item["kind"] == "assignment":
            assignment_counts[item["epoch"], :, item["prepared"]] += counts
            continue
        if item["kind"] == "validation":
            heldout_counts[item["block"], :, item["prepared"]] += counts
            continue
        if item["kind"] != "science":
            continue

        b, s, mode, g = item["block"], item["state"], item["mode"], item["group"]
        require(b in (0, 1) and s in (0, 1) and mode in (0, 1, 2, 3)
                and 0 <= g < 17, "Invalid science cell.")
        require(item["factor"] == (1, 1, 3, 5)[mode], "Mode/factor mismatch.")
        require(item.get("classical_xor", 0) == 0, "Unexpected unmodeled frame correction.")
        permutation = np.asarray(item["logical_to_canonical_index"], int)
        physical_counts = np.zeros(32, np.int64)
        physical_values = np.zeros(32)
        physical_counts[permutation] = counts
        physical_values[permutation] = group_values[g]
        p_logical = counts / counts.sum()
        p_physical = physical_counts / physical_counts.sum()
        mu = float(p_logical @ group_values[g])
        require(abs(mu - p_physical @ physical_values) <= 1e-12,
                "Logical/canonical weighted outcomes disagree.")
        shot_var = float(
            p_logical @ ((group_values[g] - mu) ** 2) / (counts.sum() - 1)
        )
        weight = 1.0 if mode == 0 else 0.25
        data[b, s, mode, g].append({
            "item": item, "counts": physical_counts, "p": p_physical,
            "w": physical_values, "mean": mu, "shot_var": shot_var,
        })
        pub_table.append({
            "item_id": item["item_id"], "state": STATES[s], "block": b,
```

```
            "mode": mode, "factor": item["factor"], "group": g,
            "randomization": item["randomization"], "shots": int(counts.sum()),
            "randomization_weight": weight, "group_mean": mu,
            "conditional_shot_variance": shot_var,
        })
        for k in range(32):
            outcome_table.append({
                "item_id": item["item_id"], "logical_count_index": k,
                "returned_bitstring": format(k, "05b"),
                "canonical_physical_index": int(permutation[k]),
                "actual_count": int(counts[k]),
                "complete_weighted_group_outcome": float(group_values[g, k]),
            })

        # Full joint covariance of term means, including identity in its
        # original group. Identity is NOT added again outside the groups.
        coefficients, signs, labels = [], [], []
        for label, c in local["groups"][g]["terms"]:
            mask = sum((1 << q) for q in range(5) if label[4 - q] != "I")
            signs.append([
                -1.0 if (k & mask).bit_count() % 2 else 1.0 for k in range(32)
            ])
            coefficients.append(complex_coefficient(c).real)
            labels.append(label)
        signs = np.asarray(signs).T
        coefficients = np.asarray(coefficients)
        means = p_logical @ signs
        covariance = (
            (signs.T * p_logical) @ signs - np.outer(means, means)
        ) / (counts.sum() - 1)
        require(
            abs(float(coefficients @ covariance @ coefficients) - shot_var) <= 1e-10,
            "Full same-shot Pauli covariance disagrees with weighted-outcome variance.",
        )
        term_covariances.append({
            "item_id": item["item_id"], "labels": labels,
            "coefficients": coefficients, "term_means": means,
            "same_shot_term_mean_covariance": covariance,
        })

    independent_raw_blocks = np.zeros((2, 2, 4))
    independent_raw_groups = np.zeros((2, 2, 4, 17))
    block_shot_var = np.zeros((2, 2, 4))
    for b in range(2):
        for s in range(2):
            for mode in range(4):
                for g in range(17):
                    rows = data[b, s, mode, g]
                    rows.sort(key=lambda x: x["item"]["randomization"])
                    expected_n = 1 if mode == 0 else 4
                    require(len(rows) == expected_n, "Missing/duplicate randomizations.")
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
            expected_ids = [-1] if mode == 0 else [0, 1, 2, 3]
            require([x["item"]["randomization"] for x in rows] == expected_ids,
                "Randomization identities incomplete.")
            require(len({int(x["counts"].sum()) for x in rows}) == 1,
                "Randomization shot allocations differ within a group.")
            mu = float(np.mean([x["mean"] for x in rows]))
            var = math.fsum(x["shot_var"] for x in rows) / expected_n**2
            empirical = (
              float(np.var([x["mean"] for x in rows], ddof=1) / 4)
              if mode else None
            )
            independent_raw_groups[b, s, mode, g] = mu
            independent_raw_blocks[b, s, mode] += mu
            block_shot_var[b, s, mode] += var
            raw_groups_table.append({
              "state": STATES[s], "block": b, "mode": mode,
              "factor": (1, 1, 3, 5)[mode], "group": g,
              "basis": local["groups"][g]["basis"],
              "energy_contribution": mu,
              "conditional_science_shot_variance": var,
              "actual_shots_per_compilation": [
                int(x["counts"].sum()) for x in rows
              ],
              "compilation_weights": [1 / expected_n] * expected_n,
              "compilation_means": [x["mean"] for x in rows],
              "empirical_RC_mean_variance_including_shots": empirical,
              "signed_RC_excess_variance": (
                empirical - var if empirical is not None else None
              ),
            })
          block_table.append({
            "state": STATES[s], "block": b, "mode": mode,
            "processing": "baseline raw" if mode == 0 else "candidate raw",
            "factor": (1, 1, 3, 5)[mode],
            "energy_Ha": independent_raw_blocks[b, s, mode],
            "conditional_science_shot_se": math.sqrt(block_shot_var[b, s, mode]),
            "groups": 17, "Pauli_terms": 63,
            "identity_accounting": "Included exactly once in its saved group.",
          })
    for s in range(2):
      for mode in range(4):
        vals = independent_raw_blocks[:, s, mode]
        result_table.append({
          "state": STATES[s],
          "processing": "baseline raw" if mode == 0 else "candidate raw",
          "factor": (1, 1, 3, 5)[mode],
          "energy_Ha": float(vals.mean()),
          "block_0_Ha": float(vals[0]), "block_1_Ha": float(vals[1]),
          "conditional_science_shot_se_of_mean": math.sqrt(
            block_shot_var[:, s, mode].sum() / 4
          ),
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
                "between_block_sample_sd": float(vals.std(ddof=1)),
                "between_block_se_of_mean": float(vals.std(ddof=1) / math.sqrt(2)),
                "reference_Ha": float(references[s]),
                "signed_fixed_state_error_Ha": float(vals.mean() - references[s]),
                "absolute_fixed_state_error_Ha": float(abs(vals.mean() - references[s])),
            })
        for mode in (1, 2, 3):
            delta = independent_raw_blocks[:, s, mode] - independent_raw_blocks[:, s, 0]
            for b in range(2):
                paired_table.append({
                    "state": STATES[s], "block": b, "factor": (1, 1, 3, 5)[mode],
                    "candidate_minus_baseline_Ha": float(delta[b]),
                    "conditional_shot_se": math.sqrt(
                        block_shot_var[b, s, mode] + block_shot_var[b, s, 0]
                    ),
                })
            paired_table.append({
                "state": STATES[s], "block": "equal-block mean",
                "factor": (1, 1, 3, 5)[mode],
                "candidate_minus_baseline_Ha": float(delta.mean()),
                "conditional_shot_se": math.sqrt(
                    (block_shot_var[:, s, mode] + block_shot_var[:, s, 0]).sum() / 4
                ),
                "between_block_se_of_paired_difference": float(
                    delta.std(ddof=1) / math.sqrt(2)
                ),
            })
    dump("same_shot_term_covariances.json", term_covariances)
    return {"complete_cells": 16, "science_publications": len(pub_table)}

phase("independent raw energies and complete group covariance", reconstruct_raw)
csv_table("raw_publication_statistics.csv", pub_table)
csv_table("weighted_group_outcomes.csv", outcome_table)
csv_table("raw_group_energies.csv", raw_groups_table)
csv_table("raw_block_energies.csv", block_table)
csv_table("raw_factors_1_3_5.csv", result_table)
csv_table("paired_raw_differences.csv", paired_table)

# ---------------------- Exact readout rejection arithmetic ----------------
readout_diagnostics = None
point_readout_valid = False
inverse = None

def diagnosed_inverse(A, source_cfg):
    diagnostics = {
        "finite": bool(np.all(np.isfinite(A))),
        "minimum_entry": float(np.min(A)),
        "column_normalization_error": float(np.max(np.abs(A.sum(axis=0) - 1))),
    }
    if not (
        A.shape == (32, 32) and diagnostics["finite"]
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        and A.min() >= 0
        and diagnostics["column_normalization_error"] <= source_cfg["matrix_tolerance"]
    ):
        return None, diagnostics, "Invalid assignment matrix."
    singular = np.linalg.svd(A, compute_uv=False)
    condition = float(singular[0] / singular[-1]) if singular[-1] > 0 else float("inf")
    diagnostics.update({
        "condition_number": condition,
        "minimum_singular_value": float(singular[-1]),
        "condition_threshold": source_cfg["readout_condition_limit"],
        "minimum_singular_value_threshold": source_cfg["readout_minimum_singular_value"],
        "condition_pass": condition <= source_cfg["readout_condition_limit"],
        "minimum_singular_value_pass": (
            singular[-1] >= source_cfg["readout_minimum_singular_value"]
        ),
    })
    if not diagnostics["condition_pass"] or not diagnostics["minimum_singular_value_pass"]:
        return None, diagnostics, "Assignment conditioning rejected."
    try:
        inv = np.linalg.solve(A, np.eye(32))
    except Exception as error:
        return None, diagnostics, f"{type(error).__name__}: {error}"
    error = float(np.max(np.abs(inv @ A - np.eye(32))))
    diagnostics["inverse_identity_error"] = error
    diagnostics["inverse_error_threshold"] = source_cfg["energy_tolerance"]
    if error > source_cfg["energy_tolerance"]:
        return None, diagnostics, "Assignment numerical inverse failed."
    return inv, diagnostics, None

def verify_readout():
    nonlocal readout_diagnostics, point_readout_valid, inverse
    require(assignment_counts is not None and cfg is not None,
            "Verified actual calibration counts/configuration unavailable.")
    require(np.all(assignment_counts.sum(axis=1) == cfg["assignment_shots"]),
            "Assignment shots/prepared state/epoch differ.")
    require(np.all(heldout_counts.sum(axis=1) == cfg["heldout_shots"]),
            "Held-out shots/prepared state/block differ.")
    pooled = assignment_counts.sum(axis=0)
    A = pooled / pooled.sum(axis=0, keepdims=True)
    held = heldout_counts.sum(axis=0)
    held = held / held.sum(axis=0, keepdims=True)
    pooled_TV = 0.5 * np.abs(A - held).sum(axis=0)
    block_TV = [
        0.5 * np.abs(A - x / x.sum(axis=0, keepdims=True)).sum(axis=0)
        for x in heldout_counts
    ]
    epoch_A = assignment_counts / assignment_counts.sum(axis=1, keepdims=True)
    inverse, numerical, exception = diagnosed_inverse(A, cfg)
    pooled_fail = np.flatnonzero(pooled_TV > cfg["heldout_TV_limit"]).tolist()
    block_fail = [
        np.flatnonzero(x > cfg["heldout_block_TV_limit"]).tolist() for x in block_TV
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    ]
    recovery_TV = None
    recovery_fail = None
    if inverse is not None:
        recovery_TV = 0.5 * np.abs(inverse @ held - np.eye(32)).sum(axis=0)
        recovery_fail = np.flatnonzero(
            recovery_TV > cfg["corrected_heldout_TV_limit"]
        ).tolist()
        point_readout_valid = not (
            pooled_fail or any(block_fail) or recovery_fail
        )
    readout_diagnostics = {
        "orientation": "A[observed canonical physical index, prepared canonical index]",
        "training_counts_by_epoch": assignment_counts,
        "pooled_assignment": A,
        "assignment_epoch_matrices": epoch_A,
        "heldout_counts": heldout_counts,
        "heldout_assignment": held,
        "pooled_numerical_checks": numerical,
        "exact_inverse_exception_or_failed_condition": exception,
        "pooled_heldout_TV": pooled_TV,
        "pooled_heldout_threshold": cfg["heldout_TV_limit"],
        "pooled_heldout_failed_prepared_indices": pooled_fail,
        "per_block_heldout_TV": block_TV,
        "per_block_threshold": cfg["heldout_block_TV_limit"],
        "per_block_failed_prepared_indices": block_fail,
        "corrected_heldout_TV": recovery_TV,
        "corrected_heldout_threshold": cfg["corrected_heldout_TV_limit"],
        "corrected_heldout_failed_prepared_indices": recovery_fail,
        "epoch_change_TV": 0.5 * np.abs(epoch_A[0] - epoch_A[1]).sum(axis=0),
        "point_calibration_accepted_by_original_rules": point_readout_valid,
        "per_epoch_numerics": [
            diagnosed_inverse(epoch_A[e], cfg)[1:] for e in range(2)
        ],
        "interpretation": (
            "A TV threshold failure is empirical incompatibility under the "
            "prespecified screening test, not proof of drift or implementation "
            "defect. The test does not explicitly adjust for finite calibration/"
            "held-out uncertainty. A numerical inverse failure is a different "
            "condition from held-out incompatibility."
        ),
    }
    comparisons = {}
    if saved_readout is not None:
        for key, reconstructed in (
            ("pooled_assignment", A), ("heldout_assignment", held),
            ("heldout_TV", pooled_TV), ("heldout_block_TV", np.asarray(block_TV)),
        ):
            if key in saved_readout:
                error = float(np.max(np.abs(
                    np.asarray(saved_readout[key]) - reconstructed
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
            )))
            comparisons[key] = error
            require(error <= 1e-10, f"Saved readout {key} arithmetic discrepancy.")
        require(
            bool(saved_readout["accepted"]) == point_readout_valid,
            "Saved point-calibration acceptance disagrees with original arithmetic.",
        )
    readout_diagnostics["saved_arithmetic_max_errors"] = comparisons
    np.savez_compressed(
        OUT / "assignment_and_heldout_matrices.npz",
        assignment=A, epochs=epoch_A, heldout=held,
        heldout_counts=heldout_counts, training_counts=assignment_counts,
    )
    dump("precise_readout_diagnostics.json", readout_diagnostics)
    print("\nEXACT READOUT CHECKS")
    print(json.dumps(normalize(readout_diagnostics["pooled_numerical_checks"]), indent=2))
    print("Inverse exception/condition:", exception)
    print("Pooled held-out TV failures:", pooled_fail)
    print("Block held-out TV failures:", block_fail)
    print("Corrected held-out TV failures:", recovery_fail)
    print("Point-calibration accepted:", point_readout_valid)
    return readout_diagnostics

phase("independent original readout checks", verify_readout)

# ---------- Independent exact-bootstrap arithmetic ----------
# No saved source is executed. This implementation reproduces the explicitly
# supplied compact algorithm only when its required local source is verified.
# Random number call order, row order, dimensions, and fail-closed policy
# match that algorithm. Saved bootstrap arrays are checked afterwards.
replay_failures = []
interval_rows = []
bootstrap_comparisons = []
original_draws = {}

def replay_original_bootstrap():
    nonlocal replay
    require(
        source_verified and full_counts_verified and mapping_verified
        and authority_verified and data is not None and cfg is not None,
        "Exact-source verification and complete validated inputs are required "
        "for original-bootstrap reproduction.",
    )
    required_source_fragments = [
        "rng(100, iteration), rng(101, iteration), rng(102, iteration)",
        "np.maximum(eig, 0)",
        "invalid_calibration_draws += 1",
        "if invalid_calibration_draws:",
        "np.quantile(samples, [0.025, 0.975], axis=0)",
    ]
    require(
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    all(fragment in source_text for fragment in required_source_fragments),
    "Saved algorithm differs from the explicitly supported compact bootstrap.",
)
require(np.__version__ == manifest["versions"]["numpy"],
        "NumPy version differs from the recorded RNG/quantile environment; "
        "exact-draw replay not claimed.")

rows = []
for item in plan:
    if item["kind"] != "science":
        continue
    g = item["group"]
    permutation = np.asarray(item["logical_to_canonical_index"], int)
    counts = np.zeros(32, np.int64)
    values = np.zeros(32)
    counts[permutation] = counts_by_item[item["item_id"]]
    values[permutation] = group_values[g]
    rows.append((item, counts, values))

W = np.stack([x[2] for x in rows])
shots = np.array([x[0]["shots"] for x in rows])
Pobs = np.stack([x[1] / x[1].sum() for x in rows])
averaging = np.array([1.0 if x[0]["mode"] == 0 else 0.25 for x in rows])
cell = np.array([
    (x[0]["block"] * 2 + x[0]["state"]) * 4 + x[0]["mode"] for x in rows
])

def evaluate(p, inv=None):
    effective = W if inv is None else W @ inv
    values = np.einsum("ij,ij->i", p, effective)
    return np.bincount(
        cell, weights=averaging * values, minlength=16
    ).reshape(2, 2, 4)

raw_blocks = evaluate(Pobs)
require(np.max(np.abs(raw_blocks - independent_raw_blocks)) <= 1e-10,
        "Independent raw reconstruction and source-order aggregation disagree.")
corrected_blocks = evaluate(Pobs, inverse) if point_readout_valid else None

latent = {}
latent_evidence = []
for block in range(2):
    for s in range(2):
        for mode in (1, 2, 3):
            for g in range(17):
                ix = [
                    i for i, row in enumerate(rows)
                    if row[0]["block"] == block and row[0]["state"] == s
                    and row[0]["mode"] == mode and row[0]["group"] == g
                ]
                require(len(ix) == 4, "Incomplete four-compilation ensemble.")
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
                ix.sort(key=lambda i: rows[i][0]["randomization"])
                p, w = Pobs[ix], W[ix]
                raw_values = np.einsum("ij,ij->i", p, w)
                dimensions = 2 if point_readout_valid else 1
                values = raw_values[:, None]
                if point_readout_valid:
                    z = w @ inverse
                    values = np.column_stack([
                        raw_values, np.einsum("ij,ij->i", p, z)
                    ])
                shot_cov = np.zeros((dimensions, dimensions))
                for j, i in enumerate(ix):
                    obs = (
                        w[j][None, :] if not point_readout_valid
                        else np.stack([w[j], z[j]])
                    )
                    means = obs @ p[j]
                    shot_cov += (
                        (obs * p[j]) @ obs.T - np.outer(means, means)
                    ) / (shots[i] - 1) / 4
                empirical = np.atleast_2d(np.cov(values, rowvar=False, ddof=1))
                excess = empirical - shot_cov
                eig, vec = np.linalg.eigh((excess + excess.T) / 2)
                covariance = (vec * np.maximum(eig, 0)) @ vec.T / 4
                latent[block, s, mode, g] = covariance
                latent_evidence.append({
                    "block": block, "state": STATES[s], "mode": mode, "group": g,
                    "preprojection_eigenvalues": eig,
                    "randomization_mean_covariance": covariance,
                    "interpretation": (
                        "PSD projection is the original covariance-model rule, "
                        "not clipping of measured outcomes or energies."
                    ),
                })

    def seeded(*coordinates):
        return np.random.default_rng(
            np.random.SeedSequence([cfg["seed"], *map(int, coordinates)])
        )

    K = cfg["bootstrap_replicates"]
    shape = (K, 2, 2, 4)
    raw_joint = np.zeros(shape)
    raw_shot = np.zeros(shape)
    raw_random = np.zeros(shape)
    corr_joint = np.full(shape, np.nan)
    corr_shot = np.full(shape, np.nan)
    corr_cal = np.full(shape, np.nan)
    corr_random = np.full(shape, np.nan)

    pooled = assignment_counts.sum(axis=0)
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    for iteration in range(K):
        shot_rng = seeded(100, iteration)
        cal_rng = seeded(101, iteration)
        random_rng = seeded(102, iteration)
        sampled = np.stack([
            shot_rng.multinomial(int(shots[i]), Pobs[i]) / shots[i]
            for i in range(len(rows))
        ])
        raw_shot[iteration] = evaluate(sampled)
        raw_joint[iteration] = raw_shot[iteration]
        raw_random[iteration] = raw_blocks

        if point_readout_valid:
            corr_shot[iteration] = evaluate(sampled, inverse)
            draw_counts = np.zeros((32, 32), np.int64)
            for epoch in range(2):
                for column in range(32):
                    counts = assignment_counts[epoch, :, column]
                    draw_counts[:, column] += cal_rng.multinomial(
                        int(counts.sum()), counts / counts.sum()
                    )
            draw_A = draw_counts / draw_counts.sum(axis=0, keepdims=True)
            draw_inverse, diagnostics, error = diagnosed_inverse(draw_A, cfg)
            if error is None:
                corr_cal[iteration] = evaluate(Pobs, draw_inverse)
                corr_joint[iteration] = evaluate(sampled, draw_inverse)
            else:
                replay_failures.append({
                    "iteration_zero_based": iteration,
                    "exact_failed_condition_or_exception": error,
                    **diagnostics,
                })
            corr_random[iteration] = corrected_blocks

        for (block, s, mode, g), covariance in latent.items():
            delta = random_rng.multivariate_normal(
                np.zeros(len(covariance)), covariance
            )
            raw_joint[iteration, block, s, mode] += delta[0]
            raw_random[iteration, block, s, mode] += delta[0]
            if point_readout_valid:
                corr_joint[iteration, block, s, mode] += delta[1]
                corr_random[iteration, block, s, mode] += delta[1]

    generated = {
        "raw_shot": raw_shot, "raw_joint": raw_joint,
        "raw_randomization": raw_random,
        "corrected_shot": corr_shot,
        "corrected_calibration": corr_cal,
        "corrected_randomization": corr_random,
        "corrected_joint": corr_joint,
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
}
path = artifacts.get("bootstrap_samples.npz")
if path is None:
    gap("saved bootstrap comparison",
        f"Missing {ROOT / 'bootstrap_samples.npz'}; independent replay "
        "can be exported but exact agreement with original saved draws is unverified.")
else:
    with np.load(path, allow_pickle=False) as loaded:
        for key, reconstructed in generated.items():
            require(key in loaded.files, f"Saved bootstrap member absent: {key}")
            saved = np.array(loaded[key], copy=True)
            original_draws[key] = saved
            require(saved.shape == reconstructed.shape, "Bootstrap shape mismatch.")
            same_nonfinite = np.array_equal(np.isnan(saved), np.isnan(reconstructed))
            finite = np.isfinite(saved) & np.isfinite(reconstructed)
            maximum_error = (
                float(np.max(np.abs(saved[finite] - reconstructed[finite])))
                if finite.any() else 0.0
            )
            exact = np.array_equal(saved, reconstructed, equal_nan=True)
            bootstrap_comparisons.append({
                "member": key, "shape": list(saved.shape),
                "bitwise_equal_including_NaN_positions": exact,
                "same_NaN_positions": same_nonfinite,
                "maximum_finite_absolute_difference": maximum_error,
            })
            require(
                same_nonfinite and maximum_error <= 1e-10,
                f"Original bootstrap {key} differs from independent exact-algorithm replay.",
            )

effective_valid = point_readout_valid and len(replay_failures) == 0
if analysis is not None:
    require(
        analysis["invalid_calibration_bootstrap_draws"] == len(replay_failures),
        "Original failed-draw count differs from independent replay.",
    )
    require(
        bool(analysis["readout_valid_for_inference"]) == effective_valid,
        "Original final readout decision differs from original-rule replay.",
    )

# Use saved original draws if available and verified, otherwise clearly
# labeled independent original-algorithm replay; never a replacement estimator.
interval_draws = original_draws.get("raw_joint", raw_joint)
mean_boot = interval_draws.mean(axis=1)
raw_mean = raw_blocks.mean(axis=0)
all_mean_ci = np.quantile(mean_boot, [0.025, 0.975], axis=0)
paired_boot = mean_boot[:, :, 1] - mean_boot[:, :, 0]
paired_ci = np.quantile(paired_boot, [0.025, 0.975], axis=0)
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    comparisons = []
    if analysis is not None:
        for key, estimate, samples in (
            ("baseline_raw", raw_mean[:, 0], mean_boot[:, :, 0]),
            ("candidate_raw_factor1", raw_mean[:, 1], mean_boot[:, :, 1]),
            ("candidate_raw_factors_1_3_5", raw_mean[:, 1:4], mean_boot[:, :, 1:4]),
            ("paired_raw_candidate_minus_baseline",
             raw_mean[:, 1] - raw_mean[:, 0], paired_boot),
        ):
            require(key in analysis, f"Original report field absent: {key}")
            point_error = float(np.max(np.abs(
                np.asarray(analysis[key]["estimate"]) - estimate
            )))
            reproduced_ci = np.quantile(samples, [0.025, 0.975], axis=0)
            interval_error = float(np.max(np.abs(
                np.asarray(analysis[key]["statistical_95_percentile_interval"])
                - reproduced_ci
            )))
            comparisons.append({
                "field": key, "point_error": point_error,
                "interval_endpoint_error": interval_error,
                "percentile_rule": "numpy.quantile, [0.025,0.975], default linear",
            })
            require(point_error <= 1e-10 and interval_error <= 1e-10,
                    f"Original report {key} arithmetic/interval discrepancy.")

    for s in range(2):
        for mode in range(4):
            interval_rows.append({
                "state": STATES[s],
                "processing": "baseline raw" if mode == 0 else "candidate raw",
                "factor": (1, 1, 3, 5)[mode],
                "energy_Ha": raw_mean[s, mode],
                "low_95_Ha": all_mean_ci[0, s, mode],
                "high_95_Ha": all_mean_ci[1, s, mode],
                "joint_statistical_se": mean_boot[:, s, mode].std(ddof=1),
                "conditional_science_shot_se": (
                    raw_shot.mean(axis=1)[:, s, mode].std(ddof=1)
                ),
                "modeled_randomization_se": (
                    raw_random.mean(axis=1)[:, s, mode].std(ddof=1)
                ),
                "shared_calibration_included": False,
                "between_block_variation_resampled": False,
                "between_block_sd": raw_blocks[:, s, mode].std(ddof=1),
                "interval_semantics": (
                    "Frozen equal-block estimator; conditional multinomial shot "
                    "draws plus original PSD-projected excess-RC random effects. "
                    "Baseline has no RC term. No block resampling; no raw readout correction."
                ),
            })
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    replay = {
        "point_readout_valid": point_readout_valid,
        "bootstrap_attempted": point_readout_valid,
        "bootstrap_replicates_configured": K,
        "failed_calibration_draw_count": len(replay_failures),
        "failed_calibration_draw_reasons": dict(Counter(
            x["exact_failed_condition_or_exception"] for x in replay_failures
        )),
        "final_readout_valid_under_original_rules": effective_valid,
        "bootstrap_array_comparisons": bootstrap_comparisons,
        "report_arithmetic_comparisons": comparisons,
        "raw_point_energies": raw_mean,
        "raw_intervals": all_mean_ci,
        "paired_factor1_interval": paired_ci,
        "qualification": (
            "Independent offline implementation of the saved source's original "
            "math and RNG call order. No experiment source executed; no draw "
            "dropped/redrawn; no threshold or estimator changed."
        ),
    }
    dump("independent_exact_algorithm_replay.json", replay)
    dump("failed_bootstrap_draws.json", replay_failures)
    dump("randomization_covariance_reanalysis.json", latent_evidence)
    np.savez_compressed(
        OUT / "independent_raw_bootstrap_replay.npz",
        raw_shot=raw_shot, raw_joint=raw_joint, raw_randomization=raw_random,
    )
    print("\nEXACT MITIGATION REJECTION EVIDENCE")
    print("Point readout accepted:", point_readout_valid)
    print("Failed calibration bootstrap draws:", len(replay_failures))
    print("Failure reasons:", replay["failed_calibration_draw_reasons"])
    for failure in replay_failures:
        print(json.dumps(normalize(failure)))
    if not point_readout_valid:
        print("Calibration bootstrap branch was NOT entered; failed-draw count is zero.")
        print("Do not misdescribe this as bootstrap-conditioning failure.")
    print("Original final readout acceptance reproduced:", effective_valid)
    return replay

phase("independent original bootstrap, rejection, and interval replay",
      replay_original_bootstrap)
csv_table("raw_energies_and_verified_intervals.csv", interval_rows)
csv_table("bootstrap_array_comparison.csv", bootstrap_comparisons)
csv_table("failed_bootstrap_draws.csv", replay_failures)

# ---------------------------- Saved interval checks ----------------------
# Even when exact replay is unavailable, directly saved original draws can
# verify their associated raw percentile calculation under the saved source.
if not interval_rows and np is not None and source_verified:
    def verify_saved_raw_intervals_only():
        path = artifacts.get("bootstrap_samples.npz")
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        require(path is not None and analysis is not None,
                "Saved bootstrap samples/original analysis unavailable.")
        with np.load(path, allow_pickle=False) as loaded:
            draws = np.array(loaded["raw_joint"], copy=True)
        require(draws.shape == (cfg["bootstrap_replicates"], 2, 2, 4)
                and np.all(np.isfinite(draws)), "Invalid saved raw bootstrap.")
        means = draws.mean(axis=1)
        for s in range(2):
            for mode, key in ((0, "baseline_raw"), (1, "candidate_raw_factor1")):
                ci = np.quantile(means[:, s, mode], [0.025, 0.975])
                saved_ci = np.asarray(
                    analysis[key]["statistical_95_percentile_interval"]
                )[:, s]
                error = float(np.max(np.abs(ci - saved_ci)))
                require(error <= 1e-10, "Saved raw interval arithmetic mismatch.")
                interval_rows.append({
                    "state": STATES[s],
                    "processing": "baseline raw" if mode == 0 else "candidate raw",
                    "factor": 1,
                    "energy_Ha": analysis[key]["estimate"][s],
                    "low_95_Ha": ci[0], "high_95_Ha": ci[1],
                    "verification": (
                        "Original saved draws -> original percentile calculation "
                        "verified; underlying draw-generation replay not verified."
                    ),
                })
        return True
    phase("saved-draw raw percentile arithmetic", verify_saved_raw_intervals_only)
    csv_table("raw_energies_and_verified_intervals.csv", interval_rows)

# ---------------------- Reporting corrections kept separate ----------------
corrections = []
if analysis is not None and readout_diagnostics is not None:
    generic_status = analysis.get("corrected_and_ZNE_status")
    if generic_status and not point_readout_valid:
        corrections.append({
            "type": "Reporting specificity, not estimator correction",
            "original_text": generic_status,
            "documented_reanalysis": (
                "Point assignment/held-out validation failed before the "
                "calibration bootstrap branch. Consequently a reported "
                "bootstrap failure count of zero is expected and does not "
                "mean mitigation was validated. Exact failed checks are in "
                "precise_readout_diagnostics.json."
            ),
            "original_decision_preserved": True,
        })
dump("documented_reporting_or_arithmetic_reanalysis.json", corrections)

# ------------------------- Log fallback / fixed closure -------------------
log_rows = []
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
for row in LOG_RESULTS:
    s = STATES.index(row["state"])
    ref = LOG_REFERENCES[s]
    log_rows.append({
        **row,
        "provenance": "User-supplied completed compact-run console",
        "reference_Ha": ref,
        "absolute_fixed_state_error_Ha": abs(row["energy"] - ref),
        "target_outcome": "NOT ACHIEVED",
        "interval_status": (
            "See independent verification tables; log values alone do not "
            "verify interval generation."
        ),
    })
csv_table("provided_console_results.csv", log_rows)

# Preserve the original rejection even if an offline discrepancy is found.
original_rejected = bool(
    analysis is not None and analysis.get("readout_valid_for_inference") is False
)
original_target = (
    analysis.get("target_achieved") if analysis is not None else
    {"COBYLA": False, "SPSA": False}
)
if analysis is not None:
    if original_target != {"COBYLA": False, "SPSA": False}:
        gap("original acceptance conflict",
            "Saved target flags differ from supplied completed output. "
            "No retrospective success declaration is made.")
    if not original_rejected:
        gap("original mitigation decision conflict",
            "Saved readout flag differs from the supplied rejection output. "
            "Closure does not promote corrected/ZNE estimates.")
if source_text is None:
    gap("exact source evidence",
        f"{ROOT / 'fresh_function_source.py'} could not be verified. "
        "No earlier source is substituted.")
if not full_counts_verified:
    gap("complete raw measurement reconstruction",
        "All 1,014 exact count associations were not verified. Supported "
        "partial tables are evidence only, not complete energy estimates.")
if not replay:
    gap("exact failed bootstrap draw attribution",
        "Original random-stream/bootstrap replay could not be completed. "
        "Only explicitly saved counts, reasons, and flags are reported; "
        "no bootstrap failure cause or favorable replacement interval is invented.")

# ------------------------ Concise case-study final report ------------------
if readout_diagnostics is not None:
    failure_description = (
        "Inverse check: "
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        + str(readout_diagnostics["exact_inverse_exception_or_failed_condition"])
        + "; pooled held-out failed prepared indices: "
        + str(readout_diagnostics["pooled_heldout_failed_prepared_indices"])
        + "; per-block failed prepared indices: "
        + str(readout_diagnostics["per_block_failed_prepared_indices"])
        + "; corrected held-out failed prepared indices: "
        + str(readout_diagnostics["corrected_heldout_failed_prepared_indices"])
        + "."
    )
else:
    failure_description = (
        "Exact local mitigation failure diagnostics unavailable; see the "
        "precisely listed missing/failed verification stages."
    )
bootstrap_description = (
    f"Original-rule bootstrap failures independently reproduced: "
    f"{replay['failed_calibration_draw_count']}; reasons "
    f"{replay['failed_calibration_draw_reasons']}."
    if replay else
    "Failed bootstrap draw counts/reasons were not independently reproduced."
)

numerical_rows = interval_rows if interval_rows else log_rows
markdown_rows = []
for row in numerical_rows:
    energy = row.get("energy_Ha", row.get("energy"))
    lower = row.get("low_95_Ha", row.get("low"))
    upper = row.get("high_95_Ha", row.get("high"))
    markdown_rows.append(
        f"| {row['state']} | {row['processing']} | {row.get('factor', 1)} | "
        f"{energy:.12f} | "
        + (f"[{lower:.12f}, {upper:.12f}]" if lower is not None else "unavailable")
        + " |"
    )

scientific_conclusion = (
    "Case Study 2 evaluated two already optimized fixed five-qubit states "
    "on a physical QPU; it was not a QPU-driven VQE optimizer. The completed "
    "compact screening did not achieve the prespecified fixed-state "
    "reference ±0.05 Ha target for either state. The supplied raw baseline "
    "point estimates were closer to the fixed-state references than the "
    "candidate factor-1 point estimates, but neither met the target. "
    "Readout-corrected and ZNE results remain unavailable/rejected; no "
    "thresholds, draws, reference energies, or estimators are altered."
)
limitations = [
    "Only two blocks and four randomizations: limited repeatability evidence.",
    "Raw intervals include conditional science shot uncertainty plus the original "
    "excess-randomization random-effects approximation for randomized circuits.",
    "Raw baseline intervals have no randomization term. No raw interval includes "
    "readout-calibration uncertainty because raw estimates do not use calibration.",
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        "Between-block SD/SE is descriptive and separate; blocks were not resampled "
        "and that variation is not added a second time.",
        "Shared calibration uncertainty would use one common assignment draw across "
        "states/groups/factors/blocks. Rejected calibrated estimates are not restored.",
        "Calibration numerical conditioning, held-out incompatibility, finite "
        "statistical sensitivity, and implementation defects are distinct.",
        "A failed empirical threshold does not identify drift or decoder error. "
        "A small inverse residual does not establish empirical calibration validity.",
        "Local hashes, QPY maps, and count consistency do not independently prove "
        "historical remote submission/execution; no remote check was performed.",
        "Ideal validation is separate from physical-QPU accuracy. Continuum and "
        "finite-model/variational differences are separate from hardware error.",
        "Earlier completed experiments are separate evidence and are not pooled. "
        "The cancelled large run is only an incomplete, resource-limited experiment; "
        "no unobserved final outcome is inferred.",
        "This closure does not initiate or propose further hardware work.",
    ]
    gaps_markdown = "\n".join(
        f"- **{x['stage']}**: {x['detail']}" for x in gaps
    ) or "- No additional verification gap was recorded."

    report = f"""# Case Study 2 — final offline closure

## Conclusion

{scientific_conclusion}

**Final disposition: COBYLA NOT ACHIEVED; SPSA NOT ACHIEVED.
Readout-corrected/ZNE UNAVAILABLE/REJECTED. Study closed.**

## Numerical comparison

{"Independent original-algorithm/saved-draw verification is documented in the tables."
 if interval_rows else
 "LOG-BASED FALLBACK: these values come from the supplied console; missing local checks are not
claimed."}

| State | Processing | Factor | Energy (Ha) | 95% interval (Ha) |
|---|---|---:|---:|---|
{chr(10).join(markdown_rows)}

All acquired candidate raw factors 1/3/5 are exported in
`raw_factors_1_3_5.csv` where complete measured reconstruction succeeds.
Block/group/publication details, actual allocations, equal four-compilation
weights, complete weighted outcomes, and full same-shot term covariance are
exported separately. Identity remains in its original group and is counted
exactly once, not added again globally.

Fixed-state console references: COBYLA {LOG_REFERENCES[0]:.17g} Ha;
SPSA {LOG_REFERENCES[1]:.17g} Ha. These are not −0.5 Ha.
`reference_verification.json` independently recalculates references from
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

unmodified hash-linked saved arrays when available.

Workload and artifact verification

Specified source directory: `{ROOT}`

Reported: 9 jobs; 1,014 executions; 882 unique saved QPY byte artifacts;
249,984 shots = 32,768 assignment + 256 diagnostic + 212,864 science +
4,096 held-out.

Complete requested/returned count reconciliation: **{full_counts_verified}**.
Part I saved authority/Hamiltonian reconstruction: **{authority_verified}**.
Manifest canonical measurement permutations verified: **{mapping_verified}**.
Local QPY measurement maps independently checked: **{qpy_verified}**.
Exact saved compact source-byte hash verified: **{source_verified}**.

Circuit hashes use the original byte definition: SHA256 of each saved QPY.
They are not confused with regenerated semantic-circuit hashes. Job associations
are local manifest/result associations, not remote-input attestations.

Precise mitigation rejection

{failure_description}

{bootstrap_description}

Full condition numbers, minimum singular values, inverse residuals, held-out
per-column/per-block statistics, original thresholds, and failed-draw diagnostics
are exported to `precise_readout_diagnostics.json`,
`independent_exact_algorithm_replay.json`, and `failed_bootstrap_draws.json`
where the exact required inputs are available. Every failed draw is retained
in diagnostics; no resampling until success or filtering occurs.

Original saved decisions remain unchanged. Any demonstrable discrepancy or
reporting clarification is separate in
`documented_reporting_or_arithmetic_reanalysis.json`.

Statistical interpretation

The original algorithm uses an equal mean over four randomized compilations
inside each candidate group, sums all 17 groups, then averages two blocks.
Its conditional shot covariance uses complete weighted outcomes, retaining
jointly measured Pauli-term covariance. The original randomization component
subtracts estimated shot covariance from four-draw empirical covariance and
projects only that covariance to PSD; it never clips energies.

The original raw percentile intervals use `numpy.quantile` at 0.025/0.975
on equal-block bootstrap means. Baseline/candidate paired differences reuse
the same simulated draws. Shot, randomization, shared-calibration, and
between-block evidence remain distinct. An independently replayed interval
is not represented as covering systematic device errors.

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
Two blocks and four randomizations are screening evidence, not robust
long-term repeatability or universal backend-performance evidence.

## Scope and limitations

{chr(10).join("- " + text for text in limitations)}

## Verification gaps

{gaps_markdown}

## Preservation

Original files, code, comments, counts, and outputs were opened read-only.
All closure artifacts are in the new sibling directory `{OUT}`.
The archive excludes credential-like files/content, symlinks, and opaque
nested/pickle containers. Credential detection is heuristic, not a guarantee
of recognizing arbitrary secrets. Pickles/object NumPy data were not loaded.
Saved Python was parsed only; it was never executed.

No IBM connection, network access, retrieval, submission, measurements,
recompilation, parameter optimization, or cancelled-run recovery occurred.
"""
    (OUT / "FINAL_REPORT.md").write_text(report, encoding="utf-8")

    dump("case_study_summary.json", {
        "conclusion": scientific_conclusion,
        "fixed_state_target": "±0.05 Ha",
        "target_outcome": {"COBYLA": "NOT ACHIEVED", "SPSA": "NOT ACHIEVED"},
        "corrected_ZNE_status": "UNAVAILABLE/REJECTED; original decision preserved",
        "reconciliation": reconciliation,
        "references": reference_report,
        "raw_reconstructed_results": result_table,
        "raw_verified_intervals": interval_rows,
        "log_fallback": log_rows,
        "mitigation_failure_summary": failure_description,
        "bootstrap_failure_summary": bootstrap_description,
        "limitations": limitations,
        "verification_gaps": gaps,
        "offline_only": True,
    })
    dump("verification_checks.json", checks)
    dump("verification_gaps.json", gaps)
    dump("original_artifact_inventory.json", inventory)
    dump("credential_and_archive_exclusions.json", exclusions)

    # ---------- Credential-screened preservation archive ----------
    # Do not copy opaque containers that could conceal unscreened credentials.
    opaque = (
        ".pkl", ".pickle", ".joblib", ".dill", ".zip", ".tar",
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    ".tar.gz", ".tgz", ".gz", ".7z", ".rar", ".xz", ".bz2",
)
initial_hashes = {row["path"]: row["sha256"] for row in inventory}
archive_members = []
for relative, path in artifacts.items():
    if relative.lower().endswith(opaque):
        exclusions.append({
            "path": relative,
            "reason": "opaque container/unsafe serialization not archived",
        })
        continue
    try:
        if sha_file(path) != initial_hashes[relative]:
            exclusions.append({"path": relative, "reason": "changed externally during closure"})
            gap("concurrent modification", relative)
        else:
            archive_members.append((relative, path))
    except Exception as error:
        exclusions.append({
            "path": relative, "reason": f"archive recheck failed: {type(error).__name__}"
        })

dump("credential_and_archive_exclusions.json", exclusions)
dump("verification_gaps.json", gaps)
closure_manifest = [
    {"file": path.name, "bytes": path.stat().st_size, "sha256": sha_file(path)}
    for path in sorted(OUT.iterdir()) if path.is_file()
]
dump("closure_artifact_hashes.json", closure_manifest)

archive = OUT / "case_study_2_evidence.tar.gz"
late_exclusions = []
with tarfile.open(archive, "w:gz") as handle:
    for relative, path in archive_members:
        try:
            payload = path.read_bytes()
            if sha_bytes(payload) != initial_hashes[relative]:
                late_exclusions.append({
                    "path": relative, "reason": "changed before archive read",
                })
                continue
            member = tarfile.TarInfo("original_evidence/" + relative)
            member.size, member.mode, member.mtime = len(payload), 0o600, 0
            handle.addfile(member, io.BytesIO(payload))
        except Exception as error:
            late_exclusions.append({
                "path": relative, "reason": f"{type(error).__name__}: {error}"
            })
    for path in sorted(OUT.iterdir()):
        if path.is_file() and path != archive:
            payload = path.read_bytes()
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
            member = tarfile.TarInfo("closure/" + path.name)
            member.size, member.mode, member.mtime = len(payload), 0o600, 0
            handle.addfile(member, io.BytesIO(payload))
        payload = json.dumps(late_exclusions, indent=2).encode()
        member = tarfile.TarInfo("closure/archive_late_exclusions.json")
        member.size, member.mode, member.mtime = len(payload), 0o600, 0
        handle.addfile(member, io.BytesIO(payload))
    dump("archive_late_exclusions.json", late_exclusions)
    (OUT / "case_study_2_evidence.sha256").write_text(
        sha_file(archive) + "  " + archive.name + "\n", encoding="utf-8"
    )

    print("\n=== FINAL OFFLINE CLOSURE ===")
    print(scientific_conclusion)
    print("\nCOBYLA: NOT ACHIEVED")
    print("SPSA:   NOT ACHIEVED")
    print("Readout-corrected/ZNE: UNAVAILABLE/REJECTED; original decision preserved.")
    print("\nWorkload verified:", full_counts_verified)
    print("Reference/Hamiltonian verified:", authority_verified)
    print("Exact source verified:", source_verified)
    print("QPY measurement-map checks:", qpy_verified)
    print("Mitigation evidence:", failure_description)
    print("Bootstrap evidence:", bootstrap_description)
    print("\nAcquired raw factors, including 3 and 5:")
    for row in result_table:
        print(row)
    if not result_table:
        print("Unavailable from complete local reconstruction; see log fallback and gaps.")
    print("\nVerification gaps:")
    for item in gaps:
        print(item["stage"] + ": " + item["detail"])
    print("\nReport:", OUT / "FINAL_REPORT.md")
    print("Numerical tables:", OUT)
    print("Archive:", archive)
    print("Archive SHA256:", sha_file(archive))
    print("Original directory unchanged by this cell:", ROOT)
    print("No IBM/network call or additional experiment was performed.")
    return {
        "closure_directory": str(OUT),
        "report": str(OUT / "FINAL_REPORT.md"),
        "archive": str(archive),
        "workload_verified": full_counts_verified,
        "target_outcome": {"COBYLA": "NOT ACHIEVED", "SPSA": "NOT ACHIEVED"},
        "verification_gaps": gaps,
    }


case_study_2_final_closure = _finalize_case_study_2_offline()
```

Final verification output (verbatim)

```
EXACT READOUT CHECKS
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

{
 "finite": true,
 "minimum_entry": 0.0,
 "column_normalization_error": 0.0,
 "condition_number": 1.2956909439733226,
 "minimum_singular_value": 0.7765500515736593,
 "condition_threshold": 10.0,
 "minimum_singular_value_threshold": 0.1,
 "condition_pass": true,
 "minimum_singular_value_pass": true,
 "inverse_identity_error": 4.440892098500626e-16,
 "inverse_error_threshold": 1e-10
}
Inverse exception/condition: None
Pooled held-out TV failures: [17, 25]
Block held-out TV failures: [[], []]
Corrected held-out TV failures: []
Point-calibration accepted: False

=== FINAL OFFLINE CLOSURE ===
Case Study 2 evaluated two already optimized fixed five-qubit states on a physical QPU; it was not a QPU-driven VQE optimizer. The completed compact screening did not achieve the prespecified fixed-state reference ±0.05 Ha target for either state. The supplied raw baseline point estimates were closer to the fixed-state references than the candidate factor-1 point estimates, but neither met the target. Readout-corrected and ZNE results remain unavailable/rejected; no thresholds, draws, reference energies, or estimators are altered.

COBYLA: NOT ACHIEVED
SPSA: NOT ACHIEVED
Readout-corrected/ZNE: UNAVAILABLE/REJECTED; original decision preserved.

Workload verified: True
Reference/Hamiltonian verified: True
Exact source verified: False
QPY measurement-map checks: True
Mitigation evidence: Inverse check: None; pooled held-out failed prepared indices: [17, 25]; per-block failed prepared indices: [[], []]; corrected held-out failed prepared indices: [].
Bootstrap evidence: Failed bootstrap draw counts/reasons were not independently reproduced.

Acquired raw factors, including 3 and 5:
{'state': 'COBYLA', 'processing': 'baseline raw', 'factor': 1, 'energy_Ha': 0.0018641080379017405, 'block_0_Ha': -0.21519213081716287, 'block_1_Ha': 0.21892034689296636, 'conditional_science_shot_se_of_mean': 0.02814284055587061, 'between_block_sample_sd': 0.30696387678652637, 'between_block_se_of_mean': 0.21705623885506461, 'reference_Ha': -0.46097392795976344, 'signed_fixed_state_error_Ha': 0.46283803599766515, 'absolute_fixed_state_error_Ha': 0.46283803599766515}
{'state': 'COBYLA', 'processing': 'candidate raw', 'factor': 1, 'energy_Ha': 0.12178396461105453, 'block_0_Ha': -0.10037855613179264, 'block_1_Ha': 0.3439464853539017, 'conditional_science_shot_se_of_mean': 0.028896382180757286, 'between_block_sample_sd': 0.31418524988552854, 'between_block_se_of_mean': 0.22216252074284717, 'reference_Ha': -0.46097392795976344, 'signed_fixed_state_error_Ha': 0.582757892570818, 'absolute_fixed_state_error_Ha': 0.582757892570818}
{'state': 'COBYLA', 'processing': 'candidate raw', 'factor': 3, 'energy_Ha': 0.4002022400971804, 'block_0_Ha': 0.166614995905767, 'block_1_Ha': 0.6337894842885938, 'conditional_science_shot_se_of_mean':

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

0.055209641671300144, 'between_block_sample_sd': 0.33034224873285284, 'between_block_se_of_mean': 0.2335872441914134, 'reference_Ha': -0.46097392795976344, 'signed_fixed_state_error_Ha': 0.8611761680569439, 'absolute_fixed_state_error_Ha': 0.8611761680569439}
{'state': 'COBYLA', 'processing': 'candidate raw', 'factor': 5, 'energy_Ha': 0.5313030569556607, 'block_0_Ha': 0.18876973187581358, 'block_1_Ha': 0.8738363820355078, 'conditional_science_shot_se_of_mean': 0.05114834179019423, 'between_block_sample_sd': 0.484415273892672, 'between_block_se_of_mean': 0.34253332507984713, 'reference_Ha': -0.46097392795976344, 'signed_fixed_state_error_Ha': 0.9922769849154242, 'absolute_fixed_state_error_Ha': 0.9922769849154242}
{'state': 'SPSA', 'processing': 'baseline raw', 'factor': 1, 'energy_Ha': 0.026773392983728522, 'block_0_Ha': -0.23024900135273946, 'block_1_Ha': 0.2837957873201965, 'conditional_science_shot_se_of_mean': 0.028174014510025578, 'between_block_sample_sd': 0.36348455590423884, 'between_block_se_of_mean': 0.257022394336468, 'reference_Ha': -0.46077921338321903, 'signed_fixed_state_error_Ha': 0.4875526063669475, 'absolute_fixed_state_error_Ha': 0.4875526063669475}
{'state': 'SPSA', 'processing': 'candidate raw', 'factor': 1, 'energy_Ha': 0.19639144841505213, 'block_0_Ha': 0.03307989451253068, 'block_1_Ha': 0.3597030023175736, 'conditional_science_shot_se_of_mean': 0.02898086843887317, 'between_block_sample_sd': 0.2309574144211706, 'between_block_se_of_mean': 0.16331155390252144, 'reference_Ha': -0.46077921338321903, 'signed_fixed_state_error_Ha': 0.6571706617982711, 'absolute_fixed_state_error_Ha': 0.6571706617982711}
{'state': 'SPSA', 'processing': 'candidate raw', 'factor': 3, 'energy_Ha': 0.380728973331558, 'block_0_Ha': 0.2981361363822391, 'block_1_Ha': 0.4633218102808769, 'conditional_science_shot_se_of_mean': 0.05427453703655842, 'between_block_sample_sd': 0.11680391016859644, 'between_block_se_of_mean': 0.08259283694931888, 'reference_Ha': -0.46077921338321903, 'signed_fixed_state_error_Ha': 0.841508186714777, 'absolute_fixed_state_error_Ha': 0.841508186714777}
{'state': 'SPSA', 'processing': 'candidate raw', 'factor': 5, 'energy_Ha': 0.6570118873337103, 'block_0_Ha': 0.38111132107824963, 'block_1_Ha': 0.932912453589171, 'conditional_science_shot_se_of_mean': 0.04995317038105473, 'between_block_sample_sd': 0.39018232266488917, 'between_block_se_of_mean': 0.27590056625546067, 'reference_Ha': -0.46077921338321903, 'signed_fixed_state_error_Ha': 1.1177911007169294, 'absolute_fixed_state_error_Ha': 1.1177911007169294}

Verification gaps:
static executed-source verification: ValueError: malformed node or string on line 76: <ast.BinOp object at 0x7c48dccd41d0>
source-specific replay: Using recorded manifest configuration for independent arithmetic only; exact source verification is unavailable.
independent original bootstrap, rejection, and interval replay: RuntimeError: Exact-source verification and complete validated inputs are required for original-bootstrap reproduction.
exact failed bootstrap draw attribution: Original random-stream/bootstrap replay could not be completed. Only explicitly saved counts, reasons, and flags are reported; no bootstrap failure cause or favorable replacement interval is invented.

Report:
/content/hydrogen_compact_screen_20261015T183419_201635Z_a76143ce_FINAL_CLOSURE_20261015T200853_724102Z_a3f02847/FINAL_REPORT.md
Numerical tables:
/content/hydrogen_compact_screen_20261015T183419_201635Z_a76143ce_FINAL_CLOSURE_20261015T200853_724102Z_a3f02847
Archive:
/content/hydrogen_compact_screen_20261015T183419_201635Z_a76143ce_FINAL_CLOSURE_20261015T200853_724102Z_a3f02847/case_study_2_evidence.tar.gz
Archive SHA256: 4f7cffc5b4d6644af9741f019c69204f6580391e0c9b2c020e8d5dd6f4495f0e
Original directory unchanged by this cell:

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

/content/hydrogen_compact_screen_20261015T183419_201635Z_a76143ce
No IBM/network call or additional experiment was performed.

Follow-up prompt requesting correction of the offline verifier

Finalize the existing closure report only. No new experiment, IBM connection, job retrieval, additional shots, recompilation, or changes to Part I.
Fix the finalization cell's static-source parser failure on ast.BinOp. Inspect and report the exact offending expression. Separate source-byte/hash verification against the original recorded hash from configuration parsing. For parsing, support only the necessary bounded, explicitly whitelisted constant arithmetic; do not execute the original notebook cell or use unrestricted eval/exec. If source identity cannot be established, retain that verification gap explicitly.
Export the actual pooled held-out total-variation values and original threshold for prepared indices 17 and 25. State that this reproduced point-calibration failure—not a demonstrated inversion or bootstrap-conditioning failure—is the established reason for rejecting readout correction. Keep corrected/ZNE inference unavailable and preserve the original target decisions.
Add the two-block energy table and clearly separate conditional science-shot uncertainty from observed between-block variation. Retain the original intervals as reported, with their independent-replay status; do not replace them with favorable intervals or claim a complete bootstrap replay.
Preserve all original source, counts, outputs, thresholds, and estimators. Write a versioned report addendum and updated evidence archive without overwriting the original closure package. End with "Experimental phase closed; accuracy target not achieved; remaining verification limitations documented." Do not propose further hardware work.References

Returned follow-up verification code (verbatim)

```
# Case Study 2 — FINAL OFFLINE VERIFICATION AND CLOSURE
#
# This cell:
#   * reads ONLY the specified completed compact-screening directory;
#   * never connects to IBM, retrieves/submits jobs, recompiles circuits,
#     optimizes parameters, or executes saved experiment Python;
#   * independently reconstructs the exact compact-screen schema and math;
#   * preserves all original files;
#   * writes reports/tables/diagnostics and a credential-screened archive
#     to a NEW SIBLING directory.
#
# No Part I notebook variables are required or modified.
# Missing artifacts produce explicit verification gaps and a log-based closure.
# No pickle, marshal, object-array, or saved-code execution is performed.
#
# Local dependencies: numpy. Qiskit is optional for QPY inspection.
# No dependency is installed or upgraded.

def _finalize_case_study_2_offline():
    from pathlib import Path
    from collections import Counter, defaultdict
    from datetime import datetime, timezone
    import ast
    import csv
    import hashlib
    import io
    import json
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
import math
import os
import re
import tarfile
import traceback
import uuid

ROOT = Path(
    "/content/hydrogen_compact_screen_20261015T183419_201635Z_a76143ce"
)
OUT = ROOT.parent / (
    ROOT.name + "_FINAL_CLOSURE_"
    + datetime.now(timezone.utc).strftime("%Y%m%dT%H%M%S_%fZ")
    + "_" + uuid.uuid4().hex[:8]
)
OUT.mkdir(parents=True, mode=0o700, exist_ok=False)

try:
    import numpy as np
except ImportError:
    np = None

STATES = ["COBYLA", "SPSA"]
LOG_REFERENCES = [-0.46097392795976344, -0.46077921338321903]
LOG_JOBS = [
    "db8hsdob3g5c73ega7r0", "db8hsrob3g5c73ega8sg",
    "db8ht2peukqs73aqkrf0", "db8hu3lckpjs73b11ddg",
    "db8i0he94fic73etee1g", "db8i3m0b3g5c73egak10",
    "db8ia1peukqs73aqlmhg", "db8iea9eukqs73aqm10g",
    "db8ifatckpjs73b12j7g",
]
LOG_RESULTS = [
    {
        "state": "COBYLA", "processing": "baseline raw", "factor": 1,
        "energy": 0.0018641080379016989,
        "low": -0.053190014897288, "high": 0.0562066504680292,
    },
    {
        "state": "COBYLA", "processing": "candidate raw", "factor": 1,
        "energy": 0.12178396461105473,
        "low": 0.0571704701937871, "high": 0.1958049063023849,
    },
    {
        "state": "SPSA", "processing": "baseline raw", "factor": 1,
        "energy": 0.026773392983728245,
        "low": -0.0272799019414917, "high": 0.0812361495220332,
    },
    {
        "state": "SPSA", "processing": "candidate raw", "factor": 1,
        "energy": 0.1963914484150516,
        "low": 0.1277962843183974, "high": 0.2718736885139076,
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    },
]
EXPECTED = {
    "jobs": 9, "instances": 1014, "unique_circuits": 882,
    "shots": 249984,
    "shots_by_kind": {
        "assignment": 32768, "diagnostic": 256,
        "science": 212864, "validation": 4096,
    },
}

gaps, checks, exclusions, inventory = [], [], [], []
tables = {}
artifacts = {}
manifest = local = analysis = saved_readout = source_tree = None
source_text = None
cfg = None
counts_by_item = {}
independent_raw_blocks = independent_raw_groups = None
replay = {}
references = None
authority_verified = False
full_counts_verified = False
mapping_verified = False
qpy_verified = False

def normalize(value):
    if np is not None:
        if isinstance(value, np.ndarray):
            return normalize(value.tolist())
        if isinstance(value, np.generic):
            return normalize(value.item())
    if isinstance(value, complex):
        return {"real": value.real, "imag": value.imag}
    if isinstance(value, dict):
        return {str(k): normalize(v) for k, v in value.items()}
    if isinstance(value, (list, tuple)):
        return [normalize(v) for v in value]
    if isinstance(value, (Path, datetime)):
        return str(value)
    if isinstance(value, float) and not math.isfinite(value):
        return {"nonfinite": str(value)}
    if value is None or isinstance(value, (str, int, float, bool)):
        return value
    return repr(value)

def dump(name, value):
    path = OUT / name
    path.write_text(
        json.dumps(normalize(value), indent=2, ensure_ascii=False, allow_nan=False),
        encoding="utf-8",
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    )
    os.chmod(path, 0o600)

def csv_table(name, rows):
    tables[name] = rows
    keys = list(dict.fromkeys(k for row in rows for k in row))
    with (OUT / name).open("w", newline="", encoding="utf-8") as handle:
        if not keys:
            handle.write("# No independently supported rows available.\n")
            return
        writer = csv.DictWriter(handle, fieldnames=keys)
        writer.writeheader()
        for row in rows:
            encoded = {}
            for key in keys:
                value = normalize(row.get(key))
                encoded[key] = (
                    json.dumps(value, ensure_ascii=False)
                    if isinstance(value, (dict, list)) else value
                )
            writer.writerow(encoded)

def gap(stage, detail):
    record = {"stage": stage, "detail": str(detail)}
    if record not in gaps:
        gaps.append(record)

def require(condition, message):
    if not bool(condition):
        raise RuntimeError(message)

def sha_bytes(data):
    return hashlib.sha256(data).hexdigest()

def sha_file(path):
    hasher = hashlib.sha256()
    with path.open("rb") as handle:
        for chunk in iter(lambda: handle.read(1024 * 1024), b""):
            hasher.update(chunk)
    return hasher.hexdigest()

def array_hash(value):
    a = np.ascontiguousarray(value)
    return sha_bytes(str(a.dtype).encode() + str(a.shape).encode() + a.tobytes())

def phase(name, function):
    try:
        result = function()
        checks.append({"stage": name, "completed": True})
        return result
    except Exception as error:
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        gap(name, f"{type(error).__name__}: {error}")
        checks.append({
            "stage": name, "completed": False,
            "exception": f"{type(error).__name__}: {error}",
        })
        return None

# Credential detection is heuristic and documented. Nothing is redacted in
# place. Excluded original files remain untouched.
secret_patterns = [
    re.compile(rb"-----BEGIN (?:RSA |EC |OPENSSH |DSA )?PRIVATE KEY-----"),
    re.compile(
        rb"""(?im)["']?(?:api[_-]?key|access[_-]?token|refresh[_-]?token|"""
        rb"""password|client[_-]?secret|ibm_quantum_token)["']?\s*[:=]\s*"""
        rb"""["']([^"'\r\n]{12,})["']"""
    ),
    re.compile(
        rb"\beyJ[A-Za-z0-9_-]{15,}\.[A-Za-z0-9_-]{10,}\."
        rb"[A-Za-z0-9_-]{10,}\b"
    ),
]

def credential_reason(path):
    name = path.name.lower()
    if (
        name in {
            ".env", ".netrc", "credentials", "credentials.json",
            "qiskit-ibm.json", "qiskitrc",
        }
        or name.endswith((".pem", ".key", ".p12", ".pfx"))
        or any(word in name for word in ("credential", "access_token", "refresh_token"))
    ):
        return "credential-like filename"
    tail = b""
    with path.open("rb") as handle:
        for chunk in iter(lambda: handle.read(1024 * 1024), b""):
            data = tail + chunk
            for pattern in secret_patterns:
                match = pattern.search(data)
                if match:
                    candidate = match.group(1) if match.lastindex else match.group(0)
                    if not any(word in candidate.lower() for word in (
                        b"placeholder", b"your_", b"replace", b"os.environ",
                        b"getenv", b"<token", b"insert_",
                    )):
                        return "credential-like content"
            tail = data[-8192:]
    return None

def inventory_originals():
    require(ROOT.is_dir() and not ROOT.is_symlink(),
```

```
            f"Specified experiment directory unavailable: {ROOT}")
    for path in sorted(ROOT.rglob("*")):
        relative = str(path.relative_to(ROOT))
        if path.is_symlink() or any(
            ancestor.is_symlink()
            for ancestor in path.parents
            if ancestor != ROOT.parent
        ):
            exclusions.append({"path": relative, "reason": "symlink not followed"})
            continue
        if not path.is_file():
            continue
        try:
            reason = credential_reason(path)
            record = {
                "path": relative, "bytes": path.stat().st_size,
                "sha256": sha_file(path),
                "credential_screen": reason or "no heuristic match",
            }
            inventory.append(record)
            if reason:
                exclusions.append({"path": relative, "reason": reason})
            else:
                artifacts[relative] = path
        except Exception as error:
            exclusions.append({
                "path": relative, "reason": f"unreadable: {type(error).__name__}"
            })
    return True

phase("artifact inventory and credential screening", inventory_originals)

def read_json(relative, required=True):
    path = artifacts.get(relative)
    if path is None:
        if required:
            raise FileNotFoundError(
                f"{ROOT / relative}: missing, unreadable, or excluded by credential screening"
            )
        return None

    def pairs(items):
        output = {}
        for key, value in items:
            require(key not in output, f"Duplicate JSON key {key!r}: {relative}")
            output[key] = value
        return output

    def bad_constant(value):
        raise ValueError(f"Invalid JSON nonfinite constant {value}: {relative}")
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    return json.loads(
        path.read_text(encoding="utf-8"),
        object_pairs_hook=pairs,
        parse_constant=bad_constant,
    )

manifest = phase("load exact compact manifest", lambda: read_json("manifest.json"))
local = phase("load local validation", lambda: read_json("local_validation.json"))
analysis = phase("load original numerical report", lambda: read_json("analysis.json"))
saved_readout = phase(
    "load original readout decision", lambda: read_json("readout_validation.json")
)

# ---------- Static inspection of the actual compact source ----------
source_functions = {}
static_excerpts = {}
source_verified = False

def inspect_source():
    nonlocal source_text, source_tree, cfg, source_verified
    require(manifest is not None, "Manifest unavailable.")
    require(manifest.get("schema") == "compact-screen-1", "Unexpected schema.")
    source_path = artifacts.get("fresh_function_source.py")
    require(source_path is not None,
            "Exact saved fresh_function_source.py unavailable; source replay cannot be certified.")
    source_text = source_path.read_text(encoding="utf-8")
    recorded = manifest.get("source", {}).get("function_source_sha256")
    require(recorded is not None, "No recorded function source-byte hash.")
    actual = sha_bytes(source_text.encode("utf-8"))
    require(actual == recorded, "Recorded source-byte SHA256 mismatch.")
    source_tree = ast.parse(source_text)
    functions = [
        node for node in source_tree.body
        if isinstance(node, ast.FunctionDef)
        and node.name == "_fresh_compact_hydrogen_screen"
    ]
    require(len(functions) == 1, "Exact compact entry function not identified.")
    function = functions[0]
    for node in function.body:
        if isinstance(node, ast.FunctionDef):
            source_functions[node.name] = node
        if isinstance(node, ast.Assign):
            if any(isinstance(t, ast.Name) and t.id == "CFG" for t in node.targets):
                cfg = ast.literal_eval(node.value)
    require(cfg == manifest["configuration"], "Source/manifest CFG mismatch.")
    require(
        cfg["blocks"] == 2 and cfg["randomizations"] == 4
        and cfg["engineering_target_Ha"] == 0.05,
        "Compact scope/acceptance rule mismatch.",
    )
    for name in (
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        "decode", "evaluate", "safe_inverse", "rng", "summary", "interval"
    ):
        require(name in source_functions, f"Saved pure source definition absent: {name}")
        static_excerpts[name] = ast.get_source_segment(source_text, source_functions[name])
    lines = source_text.splitlines()
    relevant = [
        "invalid_calibration_draws", "if invalid_calibration_draws:",
        "readout_valid = bool(", "np.random.SeedSequence",
        "raw_joint[iteration", "corr_joint[iteration",
        "np.quantile", "heldout_TV.max()", "recovery_TV.max()",
    ]
    selected_lines = set()
    for i, line in enumerate(lines):
        if any(token in line for token in relevant):
            selected_lines.update(range(max(0, i - 3), min(len(lines), i + 7)))
    static_excerpts["decision_and_resampling_lines"] = [
        {"line": i + 1, "text": lines[i]} for i in sorted(selected_lines)
    ]
    source_verified = True
    dump("exact_source_inspection.json", {
        "byte_hash": actual,
        "source_executed": False,
        "code_object_hash_qualification": (
            "The recorded code-object digest is inventoried but not regenerated "
            "by executing or compiling the saved experiment."
        ),
        "pure_function_and_decision_excerpts": static_excerpts,
    })
    return {"source_sha256": actual, "configuration_matches": True}

phase("static executed-source verification", inspect_source)
if manifest is not None and cfg is None:
    # Manifest values can support arithmetic, but do not certify source replay.
    cfg = manifest.get("configuration")
    gap("source-specific replay",
        "Using recorded manifest configuration for independent arithmetic only; "
        "exact source verification is unavailable.")

# ---------- Explicit compact-schema associations ----------
plan = registry = jobs = None
association_rows, job_rows, circuit_rows = [], [], []
raw_groups_table, block_table, result_table, paired_table = [], [], [], []
pub_table, outcome_table = [], []
data = None
assignment_counts = heldout_counts = None
group_values = None
Hsaved = vectors = ideal_group = None

def decode(counts, shots):
    require(np is not None, "NumPy unavailable.")
    output = np.zeros(32, np.int64)
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    require(isinstance(counts, dict) and bool(counts), "Missing counts.")
    for bits, number in counts.items():
        require(
            isinstance(bits, str) and len(bits) == 5
            and set(bits) <= {"0", "1"},
            "Unsupported count string; expected the source's single five-bit register.",
        )
        require(
            isinstance(number, int) and not isinstance(number, bool) and number >= 0,
            "Invalid count multiplicity.",
        )
        output[int(bits, 2)] += number
    require(int(output.sum()) == int(shots), "Requested/returned shot mismatch.")
    return output

def verify_workload():
    nonlocal plan, registry, jobs, full_counts_verified
    require(np is not None and manifest is not None, "NumPy/manifest unavailable.")
    plan, registry, jobs = manifest["plan"], manifest["registry"], manifest["jobs"]
    require(len(jobs) == 9, "Saved manifest does not have nine jobs.")
    require([job["batch"] for job in jobs] == list(range(9)), "Batch order mismatch.")
    require(len(plan) == 1014, "Saved plan does not have 1,014 executions.")
    require(len(registry) == 882, "Saved registry does not have 882 unique byte artifacts.")
    lookup = {item["item_id"]: item for item in plan}
    require(len(lookup) == len(plan), "Duplicate planned item ID.")
    require(
        [key for job in jobs for key in job["items"]]
        == [item["item_id"] for item in plan],
        "Batch-to-publication ordering differs from the frozen plan.",
    )
    require([job["job_id"] for job in jobs] == LOG_JOBS, "Manifest/log job IDs differ.")
    expected_kind_totals = Counter()
    returned_kind_totals = Counter()
    for item in plan:
        require(item["circuit_id"] in registry, "Planned circuit absent from registry.")
        expected_kind_totals[item["kind"]] += int(item["shots"])
    for job in jobs:
        relative = f"jobs/batch_{job['batch']:03d}.json"
        path = artifacts.get(relative)
        require(path is not None, f"Required actual counts unavailable: {ROOT / relative}")
        recorded_hash = job.get("raw_counts_sha256")
        require(recorded_hash is not None, f"Raw-count hash missing for batch {job['batch']}.")
        require(sha_file(path) == recorded_hash, f"Raw-count hash mismatch: {relative}")
        content = read_json(relative)
        require(content["job_id"] == job["job_id"], "Saved job-ID association mismatch.")
        require(content["batch"] == job["batch"], "Saved batch association mismatch.")
        require(
            [row["item_id"] for row in content["rows"]] == job["items"],
            "Saved publication order mismatch.",
        )
        actual_total = 0
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        for pub_index, row in enumerate(content["rows"]):
            item = lookup[row["item_id"]]
            require(
                row["item_id"] not in counts_by_item,
                "Duplicate returned publication identity.",
            )
            require(
                row["circuit_id"] == item["circuit_id"]
                and row["job_id"] == job["job_id"]
                and row["shots"] == item["shots"],
                "Saved publication circuit/job/requested-shot mismatch.",
            )
            frequencies = decode(row["counts"], item["shots"])
            counts_by_item[item["item_id"]] = frequencies
            actual_shots = int(frequencies.sum())
            actual_total += actual_shots
            returned_kind_totals[item["kind"]] += actual_shots
            association_rows.append({
                "batch": job["batch"], "job_id": job["job_id"],
                "publication_index": pub_index, "item_id": item["item_id"],
                "circuit_id": item["circuit_id"], "kind": item["kind"],
                "state": item.get("state"), "block": item.get("block"),
                "group": item.get("group"), "mode": item.get("mode"),
                "factor": item.get("factor"),
                "randomization": item.get("randomization"),
                "requested_shots": item["shots"],
                "actual_returned_shots": actual_shots,
                "saved_association_verified": True,
                "remote_execution_independently_verified": False,
            })
        require(
            actual_total == job["shots"]
            == sum(lookup[key]["shots"] for key in job["items"]),
            "Job requested/returned total mismatch.",
        )
        job_rows.append({
            "batch": job["batch"], "job_id": job["job_id"],
            "publications": len(job["items"]),
            "requested_shots": job["shots"], "returned_shots": actual_total,
            "raw_counts_sha256": recorded_hash,
            "saved_associations_verified": True,
            "remote_inputs_checked": False,
        })
    require(len(counts_by_item) == 1014, "Not all actual publications are present.")
    require(dict(expected_kind_totals) == EXPECTED["shots_by_kind"],
            "Requested shots by kind differ from the completed record.")
    require(returned_kind_totals == expected_kind_totals,
            "Requested and actual shot totals differ.")
    require(sum(returned_kind_totals.values()) == 249984, "Actual total differs.")
    full_counts_verified = True
    return {
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        "nine_jobs": True, "executions": len(plan), "unique_registry_entries": len(registry),
        "requested_shots_by_kind": dict(expected_kind_totals),
        "actual_shots_by_kind": dict(returned_kind_totals),
        "actual_total_shots": sum(returned_kind_totals.values()),
        "qualification": (
            "Local manifest/result associations and byte hashes verified. "
            "No remote service inputs or execution were independently checked."
        ),
    }

reconciliation = phase("nine-job workload and actual counts reconciliation", verify_workload)
csv_table("job_reconciliation.csv", job_rows)
csv_table("publication_associations.csv", association_rows)
dump("workload_reconciliation.json", reconciliation)

# ---------------------- Authoritative arrays and Pauli terms ----------------
def complex_coefficient(value):
    if isinstance(value, dict):
        return complex(value["real"], value["imag"])
    return complex(value)

def verify_scientific_inputs():
    nonlocal references, authority_verified, group_values, Hsaved, vectors, ideal_group
    require(np is not None and manifest is not None and local is not None,
            "NumPy/manifest/local-validation artifacts unavailable.")
    path = artifacts.get("inputs.npz")
    require(path is not None, f"Required authoritative array file missing: {ROOT / 'inputs.npz'}")
    with np.load(path, allow_pickle=False) as loaded:
        Hsaved = np.array(loaded["H"], dtype=complex, copy=True)
        rsaved = np.array(loaded["r"], copy=True)
        vectors = [
            np.array(loaded["COBYLA"], dtype=complex, copy=True),
            np.array(loaded["SPSA"], dtype=complex, copy=True),
        ]
    a = manifest["authority"]
    require(Hsaved.shape == (32, 32) and rsaved.shape == (32,), "Invalid saved dimensions.")
    require(array_hash(Hsaved) == a["H_sha256"], "Saved H authority hash mismatch.")
    require(array_hash(rsaved) == a["r_sha256"], "Saved grid authority hash mismatch.")
    require([array_hash(v) for v in vectors] == a["state_sha256"],
            "Saved optimized-state authority hash mismatch.")
    require(np.max(np.abs(Hsaved - Hsaved.conj().T)) <= 1e-12, "Saved H is not Hermitian.")
    references = np.array([
        float(np.vdot(v, Hsaved @ v).real) for v in vectors
    ])
    for v in vectors:
        require(v.shape == (32,) and np.all(np.isfinite(v))
                and abs(np.vdot(v, v) - 1) <= 1e-12, "Saved state invalid.")
    require(
        np.max(np.abs(references - np.asarray(a["references_Ha"]))) <= 1e-10,
        "Saved references do not reproduce real(vdot(psi,H@psi)).",
    )
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    groups = local["groups"]
    require(len(groups) == 17, "Saved QWC groups incomplete.")
    group_values = np.zeros((17, 32))
    ideal_group = np.zeros((2, 17))
    matrices = {
        "I": np.eye(2, dtype=complex),
        "X": np.array([[0, 1], [1, 0]], complex),
        "Y": np.array([[0, -1j], [1j, 0]], complex),
        "Z": np.diag([1, -1]).astype(complex),
    }
    reconstructed = np.zeros((32, 32), complex)
    labels, identity_count = [], 0
    for g, group in enumerate(groups):
        require(group["index"] == g, "Group index mismatch.")
        axes = ["I"] * 5
        group_matrix = np.zeros((32, 32), complex)
        for label, saved_c in group["terms"]:
            c = complex_coefficient(saved_c)
            require(len(label) == 5 and abs(c.imag) <= 1e-12, "Invalid Pauli term.")
            labels.append(label)
            identity_count += int(label == "IIIII")
            mask = 0
            term_matrix = np.array([[1]], complex)
            for axis in label:
                term_matrix = np.kron(term_matrix, matrices[axis])
            group_matrix += c * term_matrix
            for q in range(5):
                axis = label[4 - q]
                if axis != "I":
                    require(axes[q] in ("I", axis), "Saved group is not QWC.")
                    axes[q] = axis
                    mask |= 1 << q
            group_values[g] += c.real * np.array([
                -1 if (k & mask).bit_count() % 2 else 1 for k in range(32)
            ])
        axes = ["Z" if x == "I" else x for x in axes]
        require(axes == group["axes"] and "".join(reversed(axes)) == group["basis"],
                "Saved basis/Pauli ordering mismatch.")
        require(
            np.max(np.abs(group_values[g] - np.asarray(group["values"]))) <= 1e-12,
            "Saved weighted outcomes do not match Pauli parity semantics.",
        )
        reconstructed += group_matrix
        for s in range(2):
            ideal_group[s, g] = np.vdot(vectors[s], group_matrix @ vectors[s]).real
    require(len(labels) == 63 and len(set(labels)) == 63 and identity_count == 1,
            "Term coverage/identity accounting failed.")
    require(np.max(np.abs(reconstructed - Hsaved)) <= 1e-12,
            "Full grouped Hamiltonian does not reconstruct authoritative H.")
    require(np.max(np.abs(ideal_group.sum(axis=1) - references)) <= 1e-10,
            "Exact grouped expectations fail reference reconstruction.")
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    authority_verified = True
    return {
        "references_from_unmodified_saved_vectors_Ha": references,
        "authority_hashes_verified": True,
        "Hermiticity_error": float(np.max(np.abs(Hsaved - Hsaved.conj().T))),
        "Hamiltonian_reconstruction_error": float(np.max(np.abs(reconstructed - Hsaved))),
        "unique_Pauli_terms": 63, "groups": 17, "identity_occurrences": identity_count,
        "reference_minus_continuum_Ha": (
            references - a["continuum_reference_Ha"]
        ),
        "finite_representation_difference_Ha": (
            a["finite_reference_Ha"] - a["continuum_reference_Ha"]
        ),
        "classical_variational_residual_Ha": (
            references - a["finite_reference_Ha"]
        ),
        "alpha_full_precision_hex": a["alpha_full_precision_hex"],
        "qualification": (
            "Agreement with locally recorded Part I authority verified. "
            "No live Part I variables were read or changed; external historical "
            "attestation of that authority is outside offline verification."
        ),
    }

reference_report = phase("saved Part I authority and full Hamiltonian", verify_scientific_inputs)
dump("reference_verification.json", reference_report)

# ------------------------- Circuit file and measurement maps ----------------
def verify_circuit_artifacts():
    nonlocal qpy_verified, mapping_verified
    require(manifest is not None, "Manifest unavailable.")
    try:
        from qiskit import qpy
    except ImportError:
        qpy = None
        gap("QPY semantics", "Local qiskit.qpy unavailable; byte hashes still checked.")

    for cid, record in manifest["registry"].items():
        relative = record["file"]
        require(relative in artifacts, f"Missing circuit artifact: {ROOT / relative}")
        path = artifacts[relative]
        require(
            sha_file(path) == cid == record["sha256"],
            f"Recorded QPY byte hash mismatch: {relative}",
        )
        row = {
            "circuit_id": cid, "file": relative, "byte_sha256_verified": True,
            "QPY_measurement_map_verified": False,
        }
        if qpy is not None:
            with path.open("rb") as handle:
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
            loaded = qpy.load(handle)
        require(len(loaded) == 1, "Expected one circuit per saved QPY.")
        circuit = loaded[0]
        require(
            circuit.num_clbits == 5 and len(circuit.cregs) == 1
            and circuit.cregs[0].name == manifest["configuration"]["register"],
            "QPY register convention mismatch.",
        )
        mapping = {}
        measured = set()
        after_measurement = []
        twoq = Counter()
        for item in circuit.data:
            qs = [circuit.find_bit(q).index for q in item.qubits]
            if item.operation.name == "measure":
                require(len(qs) == 1 and len(item.clbits) == 1, "Invalid QPY measure.")
                c = circuit.find_bit(item.clbits[0]).index
                require(c not in mapping, "Duplicate QPY measurement destination.")
                mapping[c] = qs[0]
                measured.update(qs)
            elif item.operation.name not in ("barrier", "delay"):
                if measured.intersection(qs):
                    after_measurement.append(item.operation.name)
                if len(qs) == 2:
                    twoq[item.operation.name] += 1
        expected_map = tuple(record["measurement_map"])
        require(
            set(mapping) == set(range(5))
            and tuple(mapping[c] for c in range(5)) == expected_map
            and len(set(expected_map)) == 5,
            "QPY/registry logical-classical-to-physical map mismatch.",
        )
        require(not after_measurement, "Saved active instruction follows measurement.")
        row.update({
            "QPY_measurement_map_verified": True,
            "physical_width": circuit.num_qubits,
            "measurement_map": expected_map,
            "native_two_qubit_count": sum(twoq.values()),
            "two_qubit_operations": dict(twoq),
            "depth": circuit.depth(),
            "instructions": len(circuit.data),
        })
    circuit_rows.append(row)
qpy_verified = qpy is not None
require(plan is not None, "Frozen plan unavailable.")
canonical_set = tuple(manifest["physical_readout_set"])
require(len(canonical_set) == 5 and len(set(canonical_set)) == 5,
        "Invalid canonical physical readout set.")
positions = {q: i for i, q in enumerate(canonical_set)}
for item in plan:
    record = manifest["registry"][item["circuit_id"]]
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        mapping = record["measurement_map"]
        require(set(mapping) == set(canonical_set), "Science/calibration readout set differs.")
        if item["kind"] == "science":
            explicit = [
                sum(((k >> c) & 1) << positions[mapping[c]] for c in range(5))
                for k in range(32)
            ]
            require(explicit == item["logical_to_canonical_index"],
                    "Saved canonical permutation differs from explicit physical mapping.")
            require(len(set(explicit)) == 32, "Nonbijective canonical permutation.")
        else:
            require(tuple(mapping) == canonical_set,
                    "Calibration/diagnostic canonical measurement order differs.")
    mapping_verified = True
    return {
        "registry_byte_hashes_verified": len(circuit_rows),
        "QPY_maps_independently_checked": qpy_verified,
        "plan_permutations_checked_against_registry": True,
        "scope": "Local QPY inspection only; no compilation or simulation.",
    }

circuit_report = phase("circuit-byte hashes and actual measurement maps", verify_circuit_artifacts)
csv_table("circuit_artifact_verification.csv", circuit_rows)
dump("circuit_verification.json", circuit_report)

# ------------------- Independent source-specific raw reconstruction --------
def reconstruct_raw():
    nonlocal data, assignment_counts, heldout_counts
    nonlocal independent_raw_blocks, independent_raw_groups
    require(
        full_counts_verified and authority_verified and mapping_verified,
        "Complete verified counts, authority, and mappings are required.",
    )
    data = defaultdict(list)
    assignment_counts = np.zeros((2, 32, 32), np.int64)
    heldout_counts = np.zeros((2, 32, 32), np.int64)
    term_covariances = []
    for item in plan:
        counts = counts_by_item[item["item_id"]]
        if item["kind"] == "assignment":
            assignment_counts[item["epoch"], :, item["prepared"]] += counts
            continue
        if item["kind"] == "validation":
            heldout_counts[item["block"], :, item["prepared"]] += counts
            continue
        if item["kind"] != "science":
            continue

        b, s, mode, g = item["block"], item["state"], item["mode"], item["group"]
        require(b in (0, 1) and s in (0, 1) and mode in (0, 1, 2, 3)
                and 0 <= g < 17, "Invalid science cell.")
```

```
        require(item["factor"] == (1, 1, 3, 5)[mode], "Mode/factor mismatch.")
        require(item.get("classical_xor", 0) == 0, "Unexpected unmodeled frame correction.")
        permutation = np.asarray(item["logical_to_canonical_index"], int)
        physical_counts = np.zeros(32, np.int64)
        physical_values = np.zeros(32)
        physical_counts[permutation] = counts
        physical_values[permutation] = group_values[g]
        p_logical = counts / counts.sum()
        p_physical = physical_counts / physical_counts.sum()
        mu = float(p_logical @ group_values[g])
        require(abs(mu - p_physical @ physical_values) <= 1e-12,
                "Logical/canonical weighted outcomes disagree.")
        shot_var = float(
            p_logical @ ((group_values[g] - mu) ** 2) / (counts.sum() - 1)
        )
        weight = 1.0 if mode == 0 else 0.25
        data[b, s, mode, g].append({
            "item": item, "counts": physical_counts, "p": p_physical,
            "w": physical_values, "mean": mu, "shot_var": shot_var,
        })
        pub_table.append({
            "item_id": item["item_id"], "state": STATES[s], "block": b,
            "mode": mode, "factor": item["factor"], "group": g,
            "randomization": item["randomization"], "shots": int(counts.sum()),
            "randomization_weight": weight, "group_mean": mu,
            "conditional_shot_variance": shot_var,
        })
        for k in range(32):
            outcome_table.append({
                "item_id": item["item_id"], "logical_count_index": k,
                "returned_bitstring": format(k, "05b"),
                "canonical_physical_index": int(permutation[k]),
                "actual_count": int(counts[k]),
                "complete_weighted_group_outcome": float(group_values[g, k]),
            })

        # Full joint covariance of term means, including identity in its
        # original group. Identity is NOT added again outside the groups.
        coefficients, signs, labels = [], [], []
        for label, c in local["groups"][g]["terms"]:
            mask = sum((1 << q) for q in range(5) if label[4 - q] != "I")
            signs.append([
                -1.0 if (k & mask).bit_count() % 2 else 1.0 for k in range(32)
            ])
            coefficients.append(complex_coefficient(c).real)
            labels.append(label)
        signs = np.asarray(signs).T
        coefficients = np.asarray(coefficients)
        means = p_logical @ signs
        covariance = (
            (signs.T * p_logical) @ signs - np.outer(means, means)
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        ) / (counts.sum() - 1)
        require(
            abs(float(coefficients @ covariance @ coefficients) - shot_var) <= 1e-10,
            "Full same-shot Pauli covariance disagrees with weighted-outcome variance.",
        )
        term_covariances.append({
            "item_id": item["item_id"], "labels": labels,
            "coefficients": coefficients, "term_means": means,
            "same_shot_term_mean_covariance": covariance,
        })

    independent_raw_blocks = np.zeros((2, 2, 4))
    independent_raw_groups = np.zeros((2, 2, 4, 17))
    block_shot_var = np.zeros((2, 2, 4))
    for b in range(2):
        for s in range(2):
            for mode in range(4):
                for g in range(17):
                    rows = data[b, s, mode, g]
                    rows.sort(key=lambda x: x["item"]["randomization"])
                    expected_n = 1 if mode == 0 else 4
                    require(len(rows) == expected_n, "Missing/duplicate randomizations.")
                    expected_ids = [-1] if mode == 0 else [0, 1, 2, 3]
                    require([x["item"]["randomization"] for x in rows] == expected_ids,
                            "Randomization identities incomplete.")
                    require(len({int(x["counts"].sum()) for x in rows}) == 1,
                            "Randomization shot allocations differ within a group.")
                    mu = float(np.mean([x["mean"] for x in rows]))
                    var = math.fsum(x["shot_var"] for x in rows) / expected_n**2
                    empirical = (
                        float(np.var([x["mean"] for x in rows], ddof=1) / 4)
                        if mode else None
                    )
                    independent_raw_groups[b, s, mode, g] = mu
                    independent_raw_blocks[b, s, mode] += mu
                    block_shot_var[b, s, mode] += var
                    raw_groups_table.append({
                        "state": STATES[s], "block": b, "mode": mode,
                        "factor": (1, 1, 3, 5)[mode], "group": g,
                        "basis": local["groups"][g]["basis"],
                        "energy_contribution": mu,
                        "conditional_science_shot_variance": var,
                        "actual_shots_per_compilation": [
                            int(x["counts"].sum()) for x in rows
                        ],
                        "compilation_weights": [1 / expected_n] * expected_n,
                        "compilation_means": [x["mean"] for x in rows],
                        "empirical_RC_mean_variance_including_shots": empirical,
                        "signed_RC_excess_variance": (
                            empirical - var if empirical is not None else None
                        ),
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
                })
            block_table.append({
                "state": STATES[s], "block": b, "mode": mode,
                "processing": "baseline raw" if mode == 0 else "candidate raw",
                "factor": (1, 1, 3, 5)[mode],
                "energy_Ha": independent_raw_blocks[b, s, mode],
                "conditional_science_shot_se": math.sqrt(block_shot_var[b, s, mode]),
                "groups": 17, "Pauli_terms": 63,
                "identity_accounting": "Included exactly once in its saved group.",
            })
    for s in range(2):
        for mode in range(4):
            vals = independent_raw_blocks[:, s, mode]
            result_table.append({
                "state": STATES[s],
                "processing": "baseline raw" if mode == 0 else "candidate raw",
                "factor": (1, 1, 3, 5)[mode],
                "energy_Ha": float(vals.mean()),
                "block_0_Ha": float(vals[0]), "block_1_Ha": float(vals[1]),
                "conditional_science_shot_se_of_mean": math.sqrt(
                    block_shot_var[:, s, mode].sum() / 4
                ),
                "between_block_sample_sd": float(vals.std(ddof=1)),
                "between_block_se_of_mean": float(vals.std(ddof=1) / math.sqrt(2)),
                "reference_Ha": float(references[s]),
                "signed_fixed_state_error_Ha": float(vals.mean() - references[s]),
                "absolute_fixed_state_error_Ha": float(abs(vals.mean() - references[s])),
            })
        for mode in (1, 2, 3):
            delta = independent_raw_blocks[:, s, mode] - independent_raw_blocks[:, s, 0]
            for b in range(2):
                paired_table.append({
                    "state": STATES[s], "block": b, "factor": (1, 1, 3, 5)[mode],
                    "candidate_minus_baseline_Ha": float(delta[b]),
                    "conditional_shot_se": math.sqrt(
                        block_shot_var[b, s, mode] + block_shot_var[b, s, 0]
                    ),
                })
            paired_table.append({
                "state": STATES[s], "block": "equal-block mean",
                "factor": (1, 1, 3, 5)[mode],
                "candidate_minus_baseline_Ha": float(delta.mean()),
                "conditional_shot_se": math.sqrt(
                    (block_shot_var[:, s, mode] + block_shot_var[:, s, 0]).sum() / 4
                ),
                "between_block_se_of_paired_difference": float(
                    delta.std(ddof=1) / math.sqrt(2)
                ),
            })
    dump("same_shot_term_covariances.json", term_covariances)
    return {"complete_cells": 16, "science_publications": len(pub_table)}
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
phase("independent raw energies and complete group covariance", reconstruct_raw)
csv_table("raw_publication_statistics.csv", pub_table)
csv_table("weighted_group_outcomes.csv", outcome_table)
csv_table("raw_group_energies.csv", raw_groups_table)
csv_table("raw_block_energies.csv", block_table)
csv_table("raw_factors_1_3_5.csv", result_table)
csv_table("paired_raw_differences.csv", paired_table)

# ------------------------ Exact readout rejection arithmetic ----------------
readout_diagnostics = None
point_readout_valid = False
inverse = None

def diagnosed_inverse(A, source_cfg):
    diagnostics = {
        "finite": bool(np.all(np.isfinite(A))),
        "minimum_entry": float(np.min(A)),
        "column_normalization_error": float(np.max(np.abs(A.sum(axis=0) - 1))),
    }
    if not (
        A.shape == (32, 32) and diagnostics["finite"]
        and A.min() >= 0
        and diagnostics["column_normalization_error"] <= source_cfg["matrix_tolerance"]
    ):
        return None, diagnostics, "Invalid assignment matrix."
    singular = np.linalg.svd(A, compute_uv=False)
    condition = float(singular[0] / singular[-1]) if singular[-1] > 0 else float("inf")
    diagnostics.update({
        "condition_number": condition,
        "minimum_singular_value": float(singular[-1]),
        "condition_threshold": source_cfg["readout_condition_limit"],
        "minimum_singular_value_threshold": source_cfg["readout_minimum_singular_value"],
        "condition_pass": condition <= source_cfg["readout_condition_limit"],
        "minimum_singular_value_pass": (
            singular[-1] >= source_cfg["readout_minimum_singular_value"]
        ),
    })
    if not diagnostics["condition_pass"] or not diagnostics["minimum_singular_value_pass"]:
        return None, diagnostics, "Assignment conditioning rejected."
    try:
        inv = np.linalg.solve(A, np.eye(32))
    except Exception as error:
        return None, diagnostics, f"{type(error).__name__}: {error}"
    error = float(np.max(np.abs(inv @ A - np.eye(32))))
    diagnostics["inverse_identity_error"] = error
    diagnostics["inverse_error_threshold"] = source_cfg["energy_tolerance"]
    if error > source_cfg["energy_tolerance"]:
        return None, diagnostics, "Assignment numerical inverse failed."
    return inv, diagnostics, None
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    def verify_readout():
        nonlocal readout_diagnostics, point_readout_valid, inverse
        require(assignment_counts is not None and cfg is not None,
                "Verified actual calibration counts/configuration unavailable.")
        require(np.all(assignment_counts.sum(axis=1) == cfg["assignment_shots"]),
                "Assignment shots/prepared state/epoch differ.")
        require(np.all(heldout_counts.sum(axis=1) == cfg["heldout_shots"]),
                "Held-out shots/prepared state/block differ.")
        pooled = assignment_counts.sum(axis=0)
        A = pooled / pooled.sum(axis=0, keepdims=True)
        held = heldout_counts.sum(axis=0)
        held = held / held.sum(axis=0, keepdims=True)
        pooled_TV = 0.5 * np.abs(A - held).sum(axis=0)
        block_TV = [
            0.5 * np.abs(A - x / x.sum(axis=0, keepdims=True)).sum(axis=0)
            for x in heldout_counts
        ]
        epoch_A = assignment_counts / assignment_counts.sum(axis=1, keepdims=True)
        inverse, numerical, exception = diagnosed_inverse(A, cfg)
        pooled_fail = np.flatnonzero(pooled_TV > cfg["heldout_TV_limit"]).tolist()
        block_fail = [
            np.flatnonzero(x > cfg["heldout_block_TV_limit"]).tolist() for x in block_TV
        ]
        recovery_TV = None
        recovery_fail = None
        if inverse is not None:
            recovery_TV = 0.5 * np.abs(inverse @ held - np.eye(32)).sum(axis=0)
            recovery_fail = np.flatnonzero(
                recovery_TV > cfg["corrected_heldout_TV_limit"]
            ).tolist()
            point_readout_valid = not (
                pooled_fail or any(block_fail) or recovery_fail
            )
        readout_diagnostics = {
            "orientation": "A[observed canonical physical index, prepared canonical index]",
            "training_counts_by_epoch": assignment_counts,
            "pooled_assignment": A,
            "assignment_epoch_matrices": epoch_A,
            "heldout_counts": heldout_counts,
            "heldout_assignment": held,
            "pooled_numerical_checks": numerical,
            "exact_inverse_exception_or_failed_condition": exception,
            "pooled_heldout_TV": pooled_TV,
            "pooled_heldout_threshold": cfg["heldout_TV_limit"],
            "pooled_heldout_failed_prepared_indices": pooled_fail,
            "per_block_heldout_TV": block_TV,
            "per_block_threshold": cfg["heldout_block_TV_limit"],
            "per_block_failed_prepared_indices": block_fail,
            "corrected_heldout_TV": recovery_TV,
            "corrected_heldout_threshold": cfg["corrected_heldout_TV_limit"],
            "corrected_heldout_failed_prepared_indices": recovery_fail,
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
        "epoch_change_TV": 0.5 * np.abs(epoch_A[0] - epoch_A[1]).sum(axis=0),
        "point_calibration_accepted_by_original_rules": point_readout_valid,
        "per_epoch_numerics": [
            diagnosed_inverse(epoch_A[e], cfg)[1:] for e in range(2)
        ],
        "interpretation": (
            "A TV threshold failure is empirical incompatibility under the "
            "prespecified screening test, not proof of drift or implementation "
            "defect. The test does not explicitly adjust for finite calibration/"
            "held-out uncertainty. A numerical inverse failure is a different "
            "condition from held-out incompatibility."
        ),
    }
    comparisons = {}
    if saved_readout is not None:
        for key, reconstructed in (
            ("pooled_assignment", A), ("heldout_assignment", held),
            ("heldout_TV", pooled_TV), ("heldout_block_TV", np.asarray(block_TV)),
        ):
            if key in saved_readout:
                error = float(np.max(np.abs(
                    np.asarray(saved_readout[key]) - reconstructed
                )))
                comparisons[key] = error
                require(error <= 1e-10, f"Saved readout {key} arithmetic discrepancy.")
        require(
            bool(saved_readout["accepted"]) == point_readout_valid,
            "Saved point-calibration acceptance disagrees with original arithmetic.",
        )
    readout_diagnostics["saved_arithmetic_max_errors"] = comparisons
    np.savez_compressed(
        OUT / "assignment_and_heldout_matrices.npz",
        assignment=A, epochs=epoch_A, heldout=held,
        heldout_counts=heldout_counts, training_counts=assignment_counts,
    )
    dump("precise_readout_diagnostics.json", readout_diagnostics)
    print("\nEXACT READOUT CHECKS")
    print(json.dumps(normalize(readout_diagnostics["pooled_numerical_checks"]), indent=2))
    print("Inverse exception/condition:", exception)
    print("Pooled held-out TV failures:", pooled_fail)
    print("Block held-out TV failures:", block_fail)
    print("Corrected held-out TV failures:", recovery_fail)
    print("Point-calibration accepted:", point_readout_valid)
    return readout_diagnostics

phase("independent original readout checks", verify_readout)

# ---------- Independent exact-bootstrap arithmetic ----------
# No saved source is executed. This implementation reproduces the explicitly
# supplied compact algorithm only when its required local source is verified.
# Random number call order, row order, dimensions, and fail-closed policy
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
# match that algorithm. Saved bootstrap arrays are checked afterwards.
replay_failures = []
interval_rows = []
bootstrap_comparisons = []
original_draws = {}

def replay_original_bootstrap():
    nonlocal replay
    require(
        source_verified and full_counts_verified and mapping_verified
        and authority_verified and data is not None and cfg is not None,
        "Exact-source verification and complete validated inputs are required "
        "for original-bootstrap reproduction.",
    )
    required_source_fragments = [
        "rng(100, iteration), rng(101, iteration), rng(102, iteration)",
        "np.maximum(eig, 0)",
        "invalid_calibration_draws += 1",
        "if invalid_calibration_draws:",
        "np.quantile(samples, [0.025, 0.975], axis=0)",
    ]
    require(
        all(fragment in source_text for fragment in required_source_fragments),
        "Saved algorithm differs from the explicitly supported compact bootstrap.",
    )
    require(np.__version__ == manifest["versions"]["numpy"],
            "NumPy version differs from the recorded RNG/quantile environment; "
            "exact-draw replay not claimed.")

    rows = []
    for item in plan:
        if item["kind"] != "science":
            continue
        g = item["group"]
        permutation = np.asarray(item["logical_to_canonical_index"], int)
        counts = np.zeros(32, np.int64)
        values = np.zeros(32)
        counts[permutation] = counts_by_item[item["item_id"]]
        values[permutation] = group_values[g]
        rows.append((item, counts, values))

    W = np.stack([x[2] for x in rows])
    shots = np.array([x[0]["shots"] for x in rows])
    Pobs = np.stack([x[1] / x[1].sum() for x in rows])
    averaging = np.array([1.0 if x[0]["mode"] == 0 else 0.25 for x in rows])
    cell = np.array([
        (x[0]["block"] * 2 + x[0]["state"]) * 4 + x[0]["mode"] for x in rows
    ])

    def evaluate(p, inv=None):
        effective = W if inv is None else W @ inv
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    values = np.einsum("ij,ij->i", p, effective)
    return np.bincount(
        cell, weights=averaging * values, minlength=16
    ).reshape(2, 2, 4)

raw_blocks = evaluate(Pobs)
require(np.max(np.abs(raw_blocks - independent_raw_blocks)) <= 1e-10,
        "Independent raw reconstruction and source-order aggregation disagree.")
corrected_blocks = evaluate(Pobs, inverse) if point_readout_valid else None

latent = {}
latent_evidence = []
for block in range(2):
    for s in range(2):
        for mode in (1, 2, 3):
            for g in range(17):
                ix = [
                    i for i, row in enumerate(rows)
                    if row[0]["block"] == block and row[0]["state"] == s
                    and row[0]["mode"] == mode and row[0]["group"] == g
                ]
                require(len(ix) == 4, "Incomplete four-compilation ensemble.")
                ix.sort(key=lambda i: rows[i][0]["randomization"])
                p, w = Pobs[ix], W[ix]
                raw_values = np.einsum("ij,ij->i", p, w)
                dimensions = 2 if point_readout_valid else 1
                values = raw_values[:, None]
                if point_readout_valid:
                    z = w @ inverse
                    values = np.column_stack([
                        raw_values, np.einsum("ij,ij->i", p, z)
                    ])
                shot_cov = np.zeros((dimensions, dimensions))
                for j, i in enumerate(ix):
                    obs = (
                        w[j][None, :] if not point_readout_valid
                        else np.stack([w[j], z[j]])
                    )
                    means = obs @ p[j]
                    shot_cov += (
                        (obs * p[j]) @ obs.T - np.outer(means, means)
                    ) / (shots[i] - 1) / 4
                empirical = np.atleast_2d(np.cov(values, rowvar=False, ddof=1))
                excess = empirical - shot_cov
                eig, vec = np.linalg.eigh((excess + excess.T) / 2)
                covariance = (vec * np.maximum(eig, 0)) @ vec.T / 4
                latent[block, s, mode, g] = covariance
                latent_evidence.append({
                    "block": block, "state": STATES[s], "mode": mode, "group": g,
                    "preprojection_eigenvalues": eig,
                    "randomization_mean_covariance": covariance,
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
                "interpretation": (
                    "PSD projection is the original covariance-model rule, "
                    "not clipping of measured outcomes or energies."
                ),
            })

def seeded(*coordinates):
    return np.random.default_rng(
        np.random.SeedSequence([cfg["seed"], *map(int, coordinates)])
    )

K = cfg["bootstrap_replicates"]
shape = (K, 2, 2, 4)
raw_joint = np.zeros(shape)
raw_shot = np.zeros(shape)
raw_random = np.zeros(shape)
corr_joint = np.full(shape, np.nan)
corr_shot = np.full(shape, np.nan)
corr_cal = np.full(shape, np.nan)
corr_random = np.full(shape, np.nan)

pooled = assignment_counts.sum(axis=0)
for iteration in range(K):
    shot_rng = seeded(100, iteration)
    cal_rng = seeded(101, iteration)
    random_rng = seeded(102, iteration)
    sampled = np.stack([
        shot_rng.multinomial(int(shots[i]), Pobs[i]) / shots[i]
        for i in range(len(rows))
    ])
    raw_shot[iteration] = evaluate(sampled)
    raw_joint[iteration] = raw_shot[iteration]
    raw_random[iteration] = raw_blocks

    if point_readout_valid:
        corr_shot[iteration] = evaluate(sampled, inverse)
        draw_counts = np.zeros((32, 32), np.int64)
        for epoch in range(2):
            for column in range(32):
                counts = assignment_counts[epoch, :, column]
                draw_counts[:, column] += cal_rng.multinomial(
                    int(counts.sum()), counts / counts.sum()
                )
        draw_A = draw_counts / draw_counts.sum(axis=0, keepdims=True)
        draw_inverse, diagnostics, error = diagnosed_inverse(draw_A, cfg)
        if error is None:
            corr_cal[iteration] = evaluate(Pobs, draw_inverse)
            corr_joint[iteration] = evaluate(sampled, draw_inverse)
        else:
            replay_failures.append({
                "iteration_zero_based": iteration,
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
                "exact_failed_condition_or_exception": error,
                **diagnostics,
            })
        corr_random[iteration] = corrected_blocks

    for (block, s, mode, g), covariance in latent.items():
        delta = random_rng.multivariate_normal(
            np.zeros(len(covariance)), covariance
        )
        raw_joint[iteration, block, s, mode] += delta[0]
        raw_random[iteration, block, s, mode] += delta[0]
        if point_readout_valid:
            corr_joint[iteration, block, s, mode] += delta[1]
            corr_random[iteration, block, s, mode] += delta[1]

generated = {
    "raw_shot": raw_shot, "raw_joint": raw_joint,
    "raw_randomization": raw_random,
    "corrected_shot": corr_shot,
    "corrected_calibration": corr_cal,
    "corrected_randomization": corr_random,
    "corrected_joint": corr_joint,
}
path = artifacts.get("bootstrap_samples.npz")
if path is None:
    gap("saved bootstrap comparison",
        f"Missing {ROOT / 'bootstrap_samples.npz'}; independent replay "
        "can be exported but exact agreement with original saved draws is unverified.")
else:
    with np.load(path, allow_pickle=False) as loaded:
        for key, reconstructed in generated.items():
            require(key in loaded.files, f"Saved bootstrap member absent: {key}")
            saved = np.array(loaded[key], copy=True)
            original_draws[key] = saved
            require(saved.shape == reconstructed.shape, "Bootstrap shape mismatch.")
            same_nonfinite = np.array_equal(np.isnan(saved), np.isnan(reconstructed))
            finite = np.isfinite(saved) & np.isfinite(reconstructed)
            maximum_error = (
                float(np.max(np.abs(saved[finite] - reconstructed[finite])))
                if finite.any() else 0.0
            )
            exact = np.array_equal(saved, reconstructed, equal_nan=True)
            bootstrap_comparisons.append({
                "member": key, "shape": list(saved.shape),
                "bitwise_equal_including_NaN_positions": exact,
                "same_NaN_positions": same_nonfinite,
                "maximum_finite_absolute_difference": maximum_error,
            })
            require(
                same_nonfinite and maximum_error <= 1e-10,
                f"Original bootstrap {key} differs from independent exact-algorithm replay.",
```

```
            )

    effective_valid = point_readout_valid and len(replay_failures) == 0
    if analysis is not None:
        require(
            analysis["invalid_calibration_bootstrap_draws"] == len(replay_failures),
            "Original failed-draw count differs from independent replay.",
        )
        require(
            bool(analysis["readout_valid_for_inference"]) == effective_valid,
            "Original final readout decision differs from original-rule replay.",
        )

    # Use saved original draws if available and verified, otherwise clearly
    # labeled independent original-algorithm replay; never a replacement estimator.
    interval_draws = original_draws.get("raw_joint", raw_joint)
    mean_boot = interval_draws.mean(axis=1)
    raw_mean = raw_blocks.mean(axis=0)
    all_mean_ci = np.quantile(mean_boot, [0.025, 0.975], axis=0)
    paired_boot = mean_boot[:, :, 1] - mean_boot[:, :, 0]
    paired_ci = np.quantile(paired_boot, [0.025, 0.975], axis=0)

    comparisons = []
    if analysis is not None:
        for key, estimate, samples in (
            ("baseline_raw", raw_mean[:, 0], mean_boot[:, :, 0]),
            ("candidate_raw_factor1", raw_mean[:, 1], mean_boot[:, :, 1]),
            ("candidate_raw_factors_1_3_5", raw_mean[:, 1:4], mean_boot[:, :, 1:4]),
            ("paired_raw_candidate_minus_baseline",
             raw_mean[:, 1] - raw_mean[:, 0], paired_boot),
        ):
            require(key in analysis, f"Original report field absent: {key}")
            point_error = float(np.max(np.abs(
                np.asarray(analysis[key]["estimate"]) - estimate
            )))
            reproduced_ci = np.quantile(samples, [0.025, 0.975], axis=0)
            interval_error = float(np.max(np.abs(
                np.asarray(analysis[key]["statistical_95_percentile_interval"])
                - reproduced_ci
            )))
            comparisons.append({
                "field": key, "point_error": point_error,
                "interval_endpoint_error": interval_error,
                "percentile_rule": "numpy.quantile, [0.025,0.975], default linear",
            })
            require(point_error <= 1e-10 and interval_error <= 1e-10,
                    f"Original report {key} arithmetic/interval discrepancy.")

    for s in range(2):
        for mode in range(4):
            interval_rows.append({
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
            "state": STATES[s],
            "processing": "baseline raw" if mode == 0 else "candidate raw",
            "factor": (1, 1, 3, 5)[mode],
            "energy_Ha": raw_mean[s, mode],
            "low_95_Ha": all_mean_ci[0, s, mode],
            "high_95_Ha": all_mean_ci[1, s, mode],
            "joint_statistical_se": mean_boot[:, s, mode].std(ddof=1),
            "conditional_science_shot_se": (
                raw_shot.mean(axis=1)[:, s, mode].std(ddof=1)
            ),
            "modeled_randomization_se": (
                raw_random.mean(axis=1)[:, s, mode].std(ddof=1)
            ),
            "shared_calibration_included": False,
            "between_block_variation_resampled": False,
            "between_block_sd": raw_blocks[:, s, mode].std(ddof=1),
            "interval_semantics": (
                "Frozen equal-block estimator; conditional multinomial shot "
                "draws plus original PSD-projected excess-RC random effects. "
                "Baseline has no RC term. No block resampling; no raw readout correction."
            ),
        })
replay = {
    "point_readout_valid": point_readout_valid,
    "bootstrap_attempted": point_readout_valid,
    "bootstrap_replicates_configured": K,
    "failed_calibration_draw_count": len(replay_failures),
    "failed_calibration_draw_reasons": dict(Counter(
        x["exact_failed_condition_or_exception"] for x in replay_failures
    )),
    "final_readout_valid_under_original_rules": effective_valid,
    "bootstrap_array_comparisons": bootstrap_comparisons,
    "report_arithmetic_comparisons": comparisons,
    "raw_point_energies": raw_mean,
    "raw_intervals": all_mean_ci,
    "paired_factor1_interval": paired_ci,
    "qualification": (
        "Independent offline implementation of the saved source's original "
        "math and RNG call order. No experiment source executed; no draw "
        "dropped/redrawn; no threshold or estimator changed."
    ),
}
dump("independent_exact_algorithm_replay.json", replay)
dump("failed_bootstrap_draws.json", replay_failures)
dump("randomization_covariance_reanalysis.json", latent_evidence)
np.savez_compressed(
    OUT / "independent_raw_bootstrap_replay.npz",
    raw_shot=raw_shot, raw_joint=raw_joint, raw_randomization=raw_random,
)
print("\nEXACT MITIGATION REJECTION EVIDENCE")
print("Point readout accepted:", point_readout_valid)
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    print("Failed calibration bootstrap draws:", len(replay_failures))
    print("Failure reasons:", replay["failed_calibration_draw_reasons"])
    for failure in replay_failures:
        print(json.dumps(normalize(failure)))
    if not point_readout_valid:
        print("Calibration bootstrap branch was NOT entered; failed-draw count is zero.")
        print("Do not misdescribe this as bootstrap-conditioning failure.")
    print("Original final readout acceptance reproduced:", effective_valid)
    return replay

phase("independent original bootstrap, rejection, and interval replay",
      replay_original_bootstrap)
csv_table("raw_energies_and_verified_intervals.csv", interval_rows)
csv_table("bootstrap_array_comparison.csv", bootstrap_comparisons)
csv_table("failed_bootstrap_draws.csv", replay_failures)

# ----------------------------- Saved interval checks ----------------------
# Even when exact replay is unavailable, directly saved original draws can
# verify their associated raw percentile calculation under the saved source.
if not interval_rows and np is not None and source_verified:
    def verify_saved_raw_intervals_only():
        path = artifacts.get("bootstrap_samples.npz")
        require(path is not None and analysis is not None,
                "Saved bootstrap samples/original analysis unavailable.")
        with np.load(path, allow_pickle=False) as loaded:
            draws = np.array(loaded["raw_joint"], copy=True)
        require(draws.shape == (cfg["bootstrap_replicates"], 2, 2, 4)
                and np.all(np.isfinite(draws)), "Invalid saved raw bootstrap.")
        means = draws.mean(axis=1)
        for s in range(2):
            for mode, key in ((0, "baseline_raw"), (1, "candidate_raw_factor1")):
                ci = np.quantile(means[:, s, mode], [0.025, 0.975])
                saved_ci = np.asarray(
                    analysis[key]["statistical_95_percentile_interval"]
                )[:, s]
                error = float(np.max(np.abs(ci - saved_ci)))
                require(error <= 1e-10, "Saved raw interval arithmetic mismatch.")
                interval_rows.append({
                    "state": STATES[s],
                    "processing": "baseline raw" if mode == 0 else "candidate raw",
                    "factor": 1,
                    "energy_Ha": analysis[key]["estimate"][s],
                    "low_95_Ha": ci[0], "high_95_Ha": ci[1],
                    "verification": (
                        "Original saved draws -> original percentile calculation "
                        "verified; underlying draw-generation replay not verified."
                    ),
                })
        return True
    phase("saved-draw raw percentile arithmetic", verify_saved_raw_intervals_only)
    csv_table("raw_energies_and_verified_intervals.csv", interval_rows)
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
# ---------------------- Reporting corrections kept separate ----------------
corrections = []
if analysis is not None and readout_diagnostics is not None:
    generic_status = analysis.get("corrected_and_ZNE_status")
    if generic_status and not point_readout_valid:
        corrections.append({
            "type": "Reporting specificity, not estimator correction",
            "original_text": generic_status,
            "documented_reanalysis": (
                "Point assignment/held-out validation failed before the "
                "calibration bootstrap branch. Consequently a reported "
                "bootstrap failure count of zero is expected and does not "
                "mean mitigation was validated. Exact failed checks are in "
                "precise_readout_diagnostics.json."
            ),
            "original_decision_preserved": True,
        })
dump("documented_reporting_or_arithmetic_reanalysis.json", corrections)

# -------------------------- Log fallback / fixed closure -------------------
log_rows = []
for row in LOG_RESULTS:
    s = STATES.index(row["state"])
    ref = LOG_REFERENCES[s]
    log_rows.append({
        **row,
        "provenance": "User-supplied completed compact-run console",
        "reference_Ha": ref,
        "absolute_fixed_state_error_Ha": abs(row["energy"] - ref),
        "target_outcome": "NOT ACHIEVED",
        "interval_status": (
            "See independent verification tables; log values alone do not "
            "verify interval generation."
        ),
    })
csv_table("provided_console_results.csv", log_rows)

# Preserve the original rejection even if an offline discrepancy is found.
original_rejected = bool(
    analysis is not None and analysis.get("readout_valid_for_inference") is False
)
original_target = (
    analysis.get("target_achieved") if analysis is not None else
    {"COBYLA": False, "SPSA": False}
)
if analysis is not None:
    if original_target != {"COBYLA": False, "SPSA": False}:
        gap("original acceptance conflict",
            "Saved target flags differ from supplied completed output. "
            "No retrospective success declaration is made.")
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    if not original_rejected:
        gap("original mitigation decision conflict",
            "Saved readout flag differs from the supplied rejection output. "
            "Closure does not promote corrected/ZNE estimates.")
if source_text is None:
    gap("exact source evidence",
        f"{ROOT / 'fresh_function_source.py'} could not be verified. "
        "No earlier source is substituted.")
if not full_counts_verified:
    gap("complete raw measurement reconstruction",
        "All 1,014 exact count associations were not verified. Supported "
        "partial tables are evidence only, not complete energy estimates.")
if not replay:
    gap("exact failed bootstrap draw attribution",
        "Original random-stream/bootstrap replay could not be completed. "
        "Only explicitly saved counts, reasons, and flags are reported; "
        "no bootstrap failure cause or favorable replacement interval is invented.")

# ---------- Concise case-study final report ----------
if readout_diagnostics is not None:
    failure_description = (
        "Inverse check: "
        + str(readout_diagnostics["exact_inverse_exception_or_failed_condition"])
        + "; pooled held-out failed prepared indices: "
        + str(readout_diagnostics["pooled_heldout_failed_prepared_indices"])
        + "; per-block failed prepared indices: "
        + str(readout_diagnostics["per_block_failed_prepared_indices"])
        + "; corrected held-out failed prepared indices: "
        + str(readout_diagnostics["corrected_heldout_failed_prepared_indices"])
        + "."
    )
else:
    failure_description = (
        "Exact local mitigation failure diagnostics unavailable; see the "
        "precisely listed missing/failed verification stages."
    )
bootstrap_description = (
    f"Original-rule bootstrap failures independently reproduced: "
    f"{replay['failed_calibration_draw_count']}; reasons "
    f"{replay['failed_calibration_draw_reasons']}."
    if replay else
    "Failed bootstrap draw counts/reasons were not independently reproduced."
)

numerical_rows = interval_rows if interval_rows else log_rows
markdown_rows = []
for row in numerical_rows:
    energy = row.get("energy_Ha", row.get("energy"))
    lower = row.get("low_95_Ha", row.get("low"))
    upper = row.get("high_95_Ha", row.get("high"))
    markdown_rows.append(
```

```
            f"| {row['state']} | {row['processing']} | {row.get('factor', 1)} | "
            f"{energy:.12f} | "
            + (f"[{lower:.12f}, {upper:.12f}]" if lower is not None else "unavailable")
            + " |"
        )

    scientific_conclusion = (
        "Case Study 2 evaluated two already optimized fixed five-qubit states "
        "on a physical QPU; it was not a QPU-driven VQE optimizer. The completed "
        "compact screening did not achieve the prespecified fixed-state "
        "reference ±0.05 Ha target for either state. The supplied raw baseline "
        "point estimates were closer to the fixed-state references than the "
        "candidate factor-1 point estimates, but neither met the target. "
        "Readout-corrected and ZNE results remain unavailable/rejected; no "
        "thresholds, draws, reference energies, or estimators are altered."
    )
    limitations = [
        "Only two blocks and four randomizations: limited repeatability evidence.",
        "Raw intervals include conditional science shot uncertainty plus the original "
        "excess-randomization random-effects approximation for randomized circuits.",
        "Raw baseline intervals have no randomization term. No raw interval includes "
        "readout-calibration uncertainty because raw estimates do not use calibration.",
        "Between-block SD/SE is descriptive and separate; blocks were not resampled "
        "and that variation is not added a second time.",
        "Shared calibration uncertainty would use one common assignment draw across "
        "states/groups/factors/blocks. Rejected calibrated estimates are not restored.",
        "Calibration numerical conditioning, held-out incompatibility, finite "
        "statistical sensitivity, and implementation defects are distinct.",
        "A failed empirical threshold does not identify drift or decoder error. "
        "A small inverse residual does not establish empirical calibration validity.",
        "Local hashes, QPY maps, and count consistency do not independently prove "
        "historical remote submission/execution; no remote check was performed.",
        "Ideal validation is separate from physical-QPU accuracy. Continuum and "
        "finite-model/variational differences are separate from hardware error.",
        "Earlier completed experiments are separate evidence and are not pooled. "
        "The cancelled large run is only an incomplete, resource-limited experiment; "
        "no unobserved final outcome is inferred.",
        "This closure does not initiate or propose further hardware work.",
    ]
    gaps_markdown = "\n".join(
        f"- **{x['stage']}**: {x['detail']}" for x in gaps
    ) or "- No additional verification gap was recorded."

    report = f"""# Case Study 2 — final offline closure

## Conclusion

{scientific_conclusion}

**Final disposition: COBYLA NOT ACHIEVED; SPSA NOT ACHIEVED.
Readout-corrected/ZNE UNAVAILABLE/REJECTED. Study closed.**
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

Numerical comparison

{"Independent original-algorithm/saved-draw verification is documented in the tables."
 if interval_rows else
 "LOG-BASED FALLBACK: these values come from the supplied console; missing local checks are not claimed."}

State	Processing	Factor	Energy (Ha)	95% interval (Ha)
{chr(10).join(markdown_rows)}

All acquired candidate raw factors 1/3/5 are exported in
`raw_factors_1_3_5.csv` where complete measured reconstruction succeeds.
Block/group/publication details, actual allocations, equal four-compilation
weights, complete weighted outcomes, and full same-shot term covariance are
exported separately. Identity remains in its original group and is counted
exactly once, not added again globally.

Fixed-state console references: COBYLA {LOG_REFERENCES[0]:.17g} Ha;
SPSA {LOG_REFERENCES[1]:.17g} Ha. These are not −0.5 Ha.
`reference_verification.json` independently recalculates references from
unmodified hash-linked saved arrays when available.

Workload and artifact verification

Specified source directory: `{ROOT}`

Reported: 9 jobs; 1,014 executions; 882 unique saved QPY byte artifacts;
249,984 shots = 32,768 assignment + 256 diagnostic + 212,864 science +
4,096 held-out.

Complete requested/returned count reconciliation: **{full_counts_verified}**.
Part I saved authority/Hamiltonian reconstruction: **{authority_verified}**.
Manifest canonical measurement permutations verified: **{mapping_verified}**.
Local QPY measurement maps independently checked: **{qpy_verified}**.
Exact saved compact source-byte hash verified: **{source_verified}**.

Circuit hashes use the original byte definition: SHA256 of each saved QPY.
They are not confused with regenerated semantic-circuit hashes. Job associations
are local manifest/result associations, not remote-input attestations.

Precise mitigation rejection

{failure_description}

{bootstrap_description}

Full condition numbers, minimum singular values, inverse residuals, held-out
per-column/per-block statistics, original thresholds, and failed-draw diagnostics
are exported to `precise_readout_diagnostics.json`,

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

`independent_exact_algorithm_replay.json`, and `failed_bootstrap_draws.json`
where the exact required inputs are available. Every failed draw is retained
in diagnostics; no resampling until success or filtering occurs.

Original saved decisions remain unchanged. Any demonstrable discrepancy or
reporting clarification is separate in
`documented_reporting_or_arithmetic_reanalysis.json`.

Statistical interpretation

The original algorithm uses an equal mean over four randomized compilations
inside each candidate group, sums all 17 groups, then averages two blocks.
Its conditional shot covariance uses complete weighted outcomes, retaining
jointly measured Pauli-term covariance. The original randomization component
subtracts estimated shot covariance from four-draw empirical covariance and
projects only that covariance to PSD; it never clips energies.

The original raw percentile intervals use `numpy.quantile` at 0.025/0.975
on equal-block bootstrap means. Baseline/candidate paired differences reuse
the same simulated draws. Shot, randomization, shared-calibration, and
between-block evidence remain distinct. An independently replayed interval
is not represented as covering systematic device errors.

Two blocks and four randomizations are screening evidence, not robust
long-term repeatability or universal backend-performance evidence.

Scope and limitations

{chr(10).join("- " + text for text in limitations)}

Verification gaps

{gaps_markdown}

Preservation

Original files, code, comments, counts, and outputs were opened read-only.
All closure artifacts are in the new sibling directory `{OUT}`.
The archive excludes credential-like files/content, symlinks, and opaque
nested/pickle containers. Credential detection is heuristic, not a guarantee
of recognizing arbitrary secrets. Pickles/object NumPy data were not loaded.
Saved Python was parsed only; it was never executed.

No IBM connection, network access, retrieval, submission, measurements,
recompilation, parameter optimization, or cancelled-run recovery occurred.
"""
    (OUT / "FINAL_REPORT.md").write_text(report, encoding="utf-8")

    dump("case_study_summary.json", {
        "conclusion": scientific_conclusion,
        "fixed_state_target": "±0.05 Ha",

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    "target_outcome": {"COBYLA": "NOT ACHIEVED", "SPSA": "NOT ACHIEVED"},
    "corrected_ZNE_status": "UNAVAILABLE/REJECTED; original decision preserved",
    "reconciliation": reconciliation,
    "references": reference_report,
    "raw_reconstructed_results": result_table,
    "raw_verified_intervals": interval_rows,
    "log_fallback": log_rows,
    "mitigation_failure_summary": failure_description,
    "bootstrap_failure_summary": bootstrap_description,
    "limitations": limitations,
    "verification_gaps": gaps,
    "offline_only": True,
})
dump("verification_checks.json", checks)
dump("verification_gaps.json", gaps)
dump("original_artifact_inventory.json", inventory)
dump("credential_and_archive_exclusions.json", exclusions)

# -------------------- Credential-screened preservation archive ------------
# Do not copy opaque containers that could conceal unscreened credentials.
opaque = (
    ".pkl", ".pickle", ".joblib", ".dill", ".zip", ".tar",
    ".tar.gz", ".tgz", ".gz", ".7z", ".rar", ".xz", ".bz2",
)
initial_hashes = {row["path"]: row["sha256"] for row in inventory}
archive_members = []
for relative, path in artifacts.items():
    if relative.lower().endswith(opaque):
        exclusions.append({
            "path": relative,
            "reason": "opaque container/unsafe serialization not archived",
        })
        continue
    try:
        if sha_file(path) != initial_hashes[relative]:
            exclusions.append({"path": relative, "reason": "changed externally during closure"})
            gap("concurrent modification", relative)
        else:
            archive_members.append((relative, path))
    except Exception as error:
        exclusions.append({
            "path": relative, "reason": f"archive recheck failed: {type(error).__name__}"
        })

dump("credential_and_archive_exclusions.json", exclusions)
dump("verification_gaps.json", gaps)
closure_manifest = [
    {"file": path.name, "bytes": path.stat().st_size, "sha256": sha_file(path)}
    for path in sorted(OUT.iterdir()) if path.is_file()
]
dump("closure_artifact_hashes.json", closure_manifest)
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
archive = OUT / "case_study_2_evidence.tar.gz"
late_exclusions = []
with tarfile.open(archive, "w:gz") as handle:
    for relative, path in archive_members:
        try:
            payload = path.read_bytes()
            if sha_bytes(payload) != initial_hashes[relative]:
                late_exclusions.append({
                    "path": relative, "reason": "changed before archive read",
                })
                continue
            member = tarfile.TarInfo("original_evidence/" + relative)
            member.size, member.mode, member.mtime = len(payload), 0o600, 0
            handle.addfile(member, io.BytesIO(payload))
        except Exception as error:
            late_exclusions.append({
                "path": relative, "reason": f"{type(error).__name__}: {error}"
            })
    for path in sorted(OUT.iterdir()):
        if path.is_file() and path != archive:
            payload = path.read_bytes()
            member = tarfile.TarInfo("closure/" + path.name)
            member.size, member.mode, member.mtime = len(payload), 0o600, 0
            handle.addfile(member, io.BytesIO(payload))
    payload = json.dumps(late_exclusions, indent=2).encode()
    member = tarfile.TarInfo("closure/archive_late_exclusions.json")
    member.size, member.mode, member.mtime = len(payload), 0o600, 0
    handle.addfile(member, io.BytesIO(payload))
dump("archive_late_exclusions.json", late_exclusions)
(OUT / "case_study_2_evidence.sha256").write_text(
    sha_file(archive) + "  " + archive.name + "\n", encoding="utf-8"
)

print("\n=== FINAL OFFLINE CLOSURE ===")
print(scientific_conclusion)
print("\nCOBYLA: NOT ACHIEVED")
print("SPSA:   NOT ACHIEVED")
print("Readout-corrected/ZNE: UNAVAILABLE/REJECTED; original decision preserved.")
print("\nWorkload verified:", full_counts_verified)
print("Reference/Hamiltonian verified:", authority_verified)
print("Exact source verified:", source_verified)
print("QPY measurement-map checks:", qpy_verified)
print("Mitigation evidence:", failure_description)
print("Bootstrap evidence:", bootstrap_description)
print("\nAcquired raw factors, including 3 and 5:")
for row in result_table:
    print(row)
if not result_table:
    print("Unavailable from complete local reconstruction; see log fallback and gaps.")
print("\nVerification gaps:")
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9


```
    for item in gaps:
        print(item["stage"] + ": " + item["detail"])
    print("\nReport:", OUT / "FINAL_REPORT.md")
    print("Numerical tables:", OUT)
    print("Archive:", archive)
    print("Archive SHA256:", sha_file(archive))
    print("Original directory unchanged by this cell:", ROOT)
    print("No IBM/network call or additional experiment was performed.")
    return {
        "closure_directory": str(OUT),
        "report": str(OUT / "FINAL_REPORT.md"),
        "archive": str(archive),
        "workload_verified": full_counts_verified,
        "target_outcome": {"COBYLA": "NOT ACHIEVED", "SPSA": "NOT ACHIEVED"},
        "verification_gaps": gaps,
    }


case_study_2_final_closure = _finalize_case_study_2_offline()
```

Follow-up offline verification output (verbatim)

```
EXACT READOUT CHECKS
{
  "finite": true,
  "minimum_entry": 0.0,
  "column_normalization_error": 0.0,
  "condition_number": 1.2956909439733226,
  "minimum_singular_value": 0.7765500515736593,
  "condition_threshold": 10.0,
  "minimum_singular_value_threshold": 0.1,
  "condition_pass": true,
  "minimum_singular_value_pass": true,
  "inverse_identity_error": 4.440892098500626e-16,
  "inverse_error_threshold": 1e-10
}
Inverse exception/condition: None
Pooled held-out TV failures: [17, 25]
Block held-out TV failures: [[], []]
Corrected held-out TV failures: []
Point-calibration accepted: False

=== FINAL OFFLINE CLOSURE ===
Case Study 2 evaluated two already optimized fixed five-qubit states on a physical QPU; it was not a QPU-driven VQE optimizer. The completed compact screening did not achieve the prespecified fixed-state reference ±0.05 Ha target for either state. The supplied raw baseline point estimates were closer to the fixed-state references than the candidate factor-1 point estimates, but neither met the target. Readout-corrected and ZNE results remain unavailable/rejected; no thresholds, draws, reference energies, or estimators are altered.

COBYLA: NOT ACHIEVED
SPSA:   NOT ACHIEVED
Readout-corrected/ZNE: UNAVAILABLE/REJECTED; original decision preserved.
```

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

Workload verified: True
Reference/Hamiltonian verified: True
Exact source verified: False
QPY measurement-map checks: True
Mitigation evidence: Inverse check: None; pooled held-out failed prepared indices: [17, 25]; per-block failed prepared indices: [[], []]; corrected held-out failed prepared indices: [].
Bootstrap evidence: Failed bootstrap draw counts/reasons were not independently reproduced.

Acquired raw factors, including 3 and 5:
{'state': 'COBYLA', 'processing': 'baseline raw', 'factor': 1, 'energy_Ha': 0.0018641080379017405, 'block_0_Ha': -0.21519213081716287, 'block_1_Ha': 0.21892034689296636, 'conditional_science_shot_se_of_mean': 0.02814284055587061, 'between_block_sample_sd': 0.30696387678652637, 'between_block_se_of_mean': 0.21705623885506461, 'reference_Ha': -0.46097392795976344, 'signed_fixed_state_error_Ha': 0.46283803599766515, 'absolute_fixed_state_error_Ha': 0.46283803599766515}
{'state': 'COBYLA', 'processing': 'candidate raw', 'factor': 1, 'energy_Ha': 0.12178396461105453, 'block_0_Ha': -0.10037855613179264, 'block_1_Ha': 0.3439464853539017, 'conditional_science_shot_se_of_mean': 0.028896382180757286, 'between_block_sample_sd': 0.31418524988552854, 'between_block_se_of_mean': 0.22216252074284717, 'reference_Ha': -0.46097392795976344, 'signed_fixed_state_error_Ha': 0.582757892570818, 'absolute_fixed_state_error_Ha': 0.582757892570818}
{'state': 'COBYLA', 'processing': 'candidate raw', 'factor': 3, 'energy_Ha': 0.4002022400971804, 'block_0_Ha': 0.166614995905767, 'block_1_Ha': 0.6337894842885938, 'conditional_science_shot_se_of_mean': 0.055209641671300144, 'between_block_sample_sd': 0.33034224873285284, 'between_block_se_of_mean': 0.2335872441914134, 'reference_Ha': -0.46097392795976344, 'signed_fixed_state_error_Ha': 0.8611761680569439, 'absolute_fixed_state_error_Ha': 0.8611761680569439}
{'state': 'COBYLA', 'processing': 'candidate raw', 'factor': 5, 'energy_Ha': 0.5313030569556607, 'block_0_Ha': 0.18876973187581358, 'block_1_Ha': 0.8738363820355078, 'conditional_science_shot_se_of_mean': 0.05114834179019423, 'between_block_sample_sd': 0.484415273892672, 'between_block_se_of_mean': 0.34253332507984713, 'reference_Ha': -0.46097392795976344, 'signed_fixed_state_error_Ha': 0.9922769849154242, 'absolute_fixed_state_error_Ha': 0.9922769849154242}
{'state': 'SPSA', 'processing': 'baseline raw', 'factor': 1, 'energy_Ha': 0.026773392983728522, 'block_0_Ha': -0.23024900135273946, 'block_1_Ha': 0.2837957873201965, 'conditional_science_shot_se_of_mean': 0.028174014510025578, 'between_block_sample_sd': 0.36348455590423884, 'between_block_se_of_mean': 0.257022394336468, 'reference_Ha': -0.46077921338321903, 'signed_fixed_state_error_Ha': 0.4875526063669475, 'absolute_fixed_state_error_Ha': 0.4875526063669475}
{'state': 'SPSA', 'processing': 'candidate raw', 'factor': 1, 'energy_Ha': 0.19639144841505213, 'block_0_Ha': 0.03307989451253068, 'block_1_Ha': 0.3597030023175736, 'conditional_science_shot_se_of_mean': 0.02898086843887317, 'between_block_sample_sd': 0.2309574144211706, 'between_block_se_of_mean': 0.16331155390252144, 'reference_Ha': -0.46077921338321903, 'signed_fixed_state_error_Ha': 0.6571706617982711, 'absolute_fixed_state_error_Ha': 0.6571706617982711}
{'state': 'SPSA', 'processing': 'candidate raw', 'factor': 3, 'energy_Ha': 0.380728973331558, 'block_0_Ha': 0.2981361363822391, 'block_1_Ha': 0.4633218102808769, 'conditional_science_shot_se_of_mean': 0.05427453703655842, 'between_block_sample_sd': 0.11680391016859644, 'between_block_se_of_mean': 0.08259283694931888, 'reference_Ha': -0.46077921338321903, 'signed_fixed_state_error_Ha': 0.841508186714777, 'absolute_fixed_state_error_Ha': 0.841508186714777}
{'state': 'SPSA', 'processing': 'candidate raw', 'factor': 5, 'energy_Ha': 0.6570118873337103, 'block_0_Ha': 0.38111132107824963, 'block_1_Ha': 0.932912453589171, 'conditional_science_shot_se_of_mean': 0.04995317038105473, 'between_block_sample_sd': 0.39018232266488917, 'between_block_se_of_mean': 0.27590056625546067, 'reference_Ha': -0.46077921338321903, 'signed_fixed_state_error_Ha': 1.1177911007169294, 'absolute_fixed_state_error_Ha': 1.1177911007169294}

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

Verification gaps:
static executed-source verification: ValueError: malformed node or string on line 76: <ast.BinOp object at 0x7c48def49850>
source-specific replay: Using recorded manifest configuration for independent arithmetic only; exact source verification is unavailable.
independent original bootstrap, rejection, and interval replay: RuntimeError: Exact-source verification and complete validated inputs are required for original-bootstrap reproduction.
exact failed bootstrap draw attribution: Original random-stream/bootstrap replay could not be completed. Only explicitly saved counts, reasons, and flags are reported; no bootstrap failure cause or favorable replacement interval is invented.

Report:
/content/hydrogen_compact_screen_20261015T183419_201635Z_a76143ce_FINAL_CLOSURE_20261015T204555_641132Z_8df8409f/FINAL_REPORT.md
Numerical tables:
/content/hydrogen_compact_screen_20261015T183419_201635Z_a76143ce_FINAL_CLOSURE_20261015T204555_641132Z_8df8409f
Archive:
/content/hydrogen_compact_screen_20261015T183419_201635Z_a76143ce_FINAL_CLOSURE_20261015T204555_641132Z_8df8409f/case_study_2_evidence.tar.gz
Archive SHA256: 9cb7aea096d1c7c70c4505bac302cd359f3be0161c951cb1fc966cba22311572
Original directory unchanged by this cell:
/content/hydrogen_compact_screen_20261015T183419_201635Z_a76143ce
No IBM/network call or additional experiment was performed.

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

Overall findings, scientific interpretation, and final conclusion

The completed record supports a positive but bounded finding: in a human-guided workflow, Quantum Flow generated a consistent finite-model implementation, introduced strict local validation and evidence capture, and implemented a structure-aware synthesis that materially reduced compiled resources and improved hardware point estimates in Run 3. Those benefits remain part of the study even though the final, more elaborate screening did not improve accuracy. The strongest defensible performance statement is “best observed among the tested variants in the identified run and resource setting,” not “best achievable on all currently available IBM hardware.”

The final compact experiment ran on the ibm_marrakesh backend, as recorded in its nine job associations. It missed the prescribed accuracy target for both states. Pooled held-out calibration failures prevented accepted readout-corrected and ZNE inference, while exact-source verification and independent bootstrap replay remained incomplete. The study closes with both its measured engineering improvements and its unsuccessful accuracy endpoint intact. The following synthesis explains the theoretical background, the limits of attribution, and the evidence required for each claim.

1. Strongest demonstrated engineering result

The clearest positive result is the same-run comparison in Run 3, not the last numerical value in the chronology. The generated sequential preparation retained the authoritative Part I states and full Hamiltonian while reducing native two-qubit gates by 78.3%. For each fixed state, the calibrated sequential variant had the smallest observed absolute discrepancy between its mean measured energy and the corresponding reference energy among the variants submitted in Run 3. The table below preserves both the size of that improvement and the remaining error.

State	Run 3 baseline mean (Ha)	Best tested Run 3 mean (Ha)	Remaining absolute discrepancy (Ha)	Observed reduction
COBYLA-derived	+0.132891	−0.313877 ± 0.014890	0.147097	75.2%
SPSA-derived	+0.097022	−0.316532 ± 0.014855	0.144247	74.1%

Table 2. Run 3 within-run comparison on the ibm_fez backend. “±” denotes sampling standard error, not total error. Reductions are calculated as 1 − |Ecandidate − Eref|/|Ebaseline − Eref|. They are not percentages of state fidelity, chemical accuracy, or quantum speedup. Selection from tested variants and the two-repetition design limit generalization; no multiplicity-adjusted or multi-day confirmation is reported.

2. Separating model error, sampling, and physical execution

An additive accounting identity separates the reference levels: $E_{observed} - E_{continuum} = (E_{0,fd} - E_{continuum}) + (E_{ref} - E_{0,fd}) + (E_{ideal\ circuit} - E_{ref}) + (E_{observed} - E_{ideal\ circuit})$. The first term is finite representation, the second is the classical variational residual, the third is the tested ideal implementation difference, and the last is the observed physical-execution/estimation discrepancy. The last term includes sampling and any unisolated systematic or implementation effects; it is not automatically a pure hardware-noise estimate.

For any normalized state ψ in the finite model, the variational principle gives $\psi^\dagger H \psi \geq E_{0,fd}$. A finite-shot energy estimate may nevertheless fall below $E_{0,fd}$ because of sampling fluctuations, and a readout-corrected quasi-probability estimator need not correspond to a physical quantum state [25, 29]. Energy agreement alone is not full state tomography. Similarly, measurement-probability equivalence on the specified initial state validates the intended workload, not the action of an arbitrary generated circuit on every possible input.

For measurement group g, define the complete weighted outcome Fg as the sum of its Pauli coefficients multiplied by the corresponding measured parities. Under independent shots, the conditional variance of its sample mean is Var(Fg)/Ng. This retains covariance among jointly measured terms; treating every Pauli term as independently sampled would generally be wrong [25]. For R equally weighted independent repetitions with shot standard errors sr, the conditional shot standard error is $\sqrt{(\Sigma_r\, s_r^2)}/R$. This is distinct from variation among repetition means.

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

3. Final screening and limited repeatability

Run 6 did not meet the prespecified ±0.05 Ha engineering target. That threshold belongs to the finite fixed-state energy, not the −0.5 Ha continuum value, and is not presented as chemical accuracy. The absolute discrepancies between the final candidate's mean energies and their corresponding references were 0.582758 Ha and 0.657171 Ha, compared with baseline discrepancies of 0.462838 Ha and 0.487553 Ha. The observed candidate means were therefore worse, not a continuation of the earlier Run 3 improvement.

State / implementation	Block 1 (Ha)	Block 2 (Ha)	Mean (Ha)	Conditional shot SE / between-block SE (Ha)
COBYLA / baseline	−0.215192	+0.218920	+0.001864	0.028143 / 0.217056
COBYLA / candidate f=1	−0.100379	+0.343946	+0.121784	0.028896 / 0.222163
SPSA / baseline	−0.230249	+0.283796	+0.026773	0.028174 / 0.257022
SPSA / candidate f=1	+0.033080	+0.359703	+0.196391	0.028981 / 0.163312

Table 3. Final raw measurements reconstructed in the recorded offline output. The between-block standard error is the sample standard deviation divided by √2 and is descriptive with only two blocks; it is not an independently calibrated total-error model. The original raw intervals condition on the reported shot/randomization procedure and do not resample execution blocks. Their independent bootstrap replay was not completed. The large block changes must therefore remain visible rather than being hidden by the relatively small conditional shot errors.

4. Why a well-conditioned matrix did not justify mitigation

Readout calibration estimates an assignment matrix A with columns A[:,j] = P(measured outcome | prepared basis index j). Numerical invertibility and empirical applicability are different requirements. The final offline report gives condition number 1.295691, minimum singular value 0.776550, and inverse-identity residual 4.44×10^{-16}, all within the stated numerical limits. Nevertheless, pooled held-out validation failed for prepared indices 17 and 25, so the recorded point-calibration decision was false.

The preserved compact code sets the pooled total-variation limit to 0.15 and the per-block and corrected-held-out limits to 0.30. For each prepared index j, $TVj = \frac{1}{2}\Sigma i|Aij - Bij|$ compares training and held-out distributions. Passing the looser per-block checks while failing the stricter pooled check is not inherently contradictory. The final console identifies the failed indices but does not print their TV values; the numerical margin and physical cause of rejection remain unestablished. Sampling uncertainty, calibration drift, preparation error, and model mismatch cannot be distinguished from that flag alone. No threshold was relaxed retrospectively.

Readout correction would modify the estimator, not physically repair the state, and would require propagation of the shared calibration uncertainty across groups, factors, and states [29]. The reproduced point-calibration failure is sufficient to preserve rejection of the prespecified corrected estimator. The original bootstrap rejection and failed-draw attribution were not independently reproduced, so they are not asserted as a separately established cause.

5. Error suppression and zero-noise extrapolation in context

Pauli randomization and dynamical decoupling are established error-suppression tools, but their benefit depends on the noise and implementation. IBM documents both their compatibility constraints and additional processing cost [32]. Ideal Pauli-frame or DD-sequence equivalence does not prove a beneficial noisy channel. This study's accepted final configuration had DD disabled; the failed local DD candidates remain recorded rather than being described as successful physical suppression.

Zero-noise extrapolation estimates an expectation-value limit from measurements taken at amplified noise levels [28, 29]. The implemented nominal factors 1, 3, and 5 scale native entangling-gate repetitions through identity folds; they do not establish uniform scaling of readout error, idle dephasing, leakage, crosstalk, or calibration changes. The prespecified linear intercept is (13/12)E1 + (1/3)E3 − (5/12)E5, with its variance determined by the full covariance

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

of those estimates. This is a model-based estimator with potential uncertainty amplification and model bias, not error correction or a proof of noiseless execution.

The final offline record exposes raw factor-3 and factor-5 means as well as factor 1. Their increasing means show that the executed folded workloads differed in their measured response, but do not validate the extrapolation model. Because the required readout check failed, no accepted corrected-ZNE estimate was produced. Selecting another fit or a raw extrapolation after seeing the known hydrogen target would answer a different, exploratory question; it is not used to reverse the recorded outcome.

6. Reproducibility and verification boundaries

The closure output reports verification of workload accounting, the reference/Hamiltonian, and QPY measurement mappings, and reproduces the raw-energy arithmetic. It also reports “Exact source verified: False” and no independent replay of the original bootstrap. Inspection of the preserved listings shows that the verifier applies ast.literal_eval to a configuration dictionary containing constant arithmetic such as 16 * 1024 * 1024. Python’s literal-only parser does not evaluate ordinary arithmetic expressions [35]. This explains a plausible parser failure path, not a completed repair, proof of source tampering, or certification of the original execution.

The two offline closure outputs refer to the same compact-screening evidence and are not independent experiments. A new archive timestamp or checksum identifies a new byte package; it does not turn an incomplete verification into a pass. The study records local paths and archive digests, but access to the actual archived source, circuit bytes, counts, calibrations, and environment is required for a third party to reproduce the reported analysis. The Word code listings and screenshots are records, not substitutes for a machine-readable evidence release. No independent IBM-account or remote-job verification is claimed by the offline-only checks.

7. What this establishes about Quantum Flow and IBM execution

The supported contribution is a documented AI-assisted engineering process: implement the specified finite model, preserve it across circuit revisions, validate computational and measurement semantics, exploit exact state structure, produce native workloads, compare physical results with references, and retain negative outcomes. The measured resource reduction and Run 3 within-run energy improvement are substantive results. The cancelled large design and unsuccessful final screening also show that resource planning and hardware-model selection required further human direction; they are not omitted from the assessment.

IBM backends expose calibration-dependent properties rather than a single universal accuracy number [30]. The accessed devices and selected subsets were suitable platforms for this small benchmark, but the study neither compared every available processor nor exhausted compilations, layouts, pulse/control methods, or mitigation protocols. Consequently, it cannot certify that Quantum Flow reached IBM’s physical performance limit or the best result obtainable with another implementation. “Best observed” is restricted to the tested variants and stated metric. The results also cannot be used to conclude that IBM hardware in general is incapable of accurate small-system calculations; established hardware VQE experiments address different Hamiltonians and protocols [36].

The practical relevance is the traceable research-to-implementation workflow and its validation discipline, not a classically inaccessible hydrogen calculation. This 32-dimensional, single-electron instance is deliberately classically verifiable and its trial states have low tensor rank. No quantum advantage, industrial-scale chemistry result, independently quantified productivity gain, or global superiority over expert or general-model workflows is demonstrated. Those claims require different controlled comparisons rather than stronger wording.

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

Final conclusion

Stellora.AI Quantum Flow, operated with explicit scientific constraints and iterative human feedback, produced a simulator-validated implementation of the prescribed finite hydrogen problem and a structure-aware state synthesis with a 78.3% reduction in two-qubit gates. Its best completed within-run comparison on the ibm_fez backend reduced observed energy discrepancies by approximately 75% relative to the same-run baseline, while retaining measurable bias and limited repeatability evidence. These are defensible achievements in problem-specific quantum-software engineering under the recorded hardware-access conditions.

The final ibm_marrakesh backend screening completed the reduced nine-job budget but did not reproduce the fixed-state energies within the ±0.05 Ha target. Required readout validation failed, corrected/ZNE estimates were withheld, and source/interval replay limitations remained documented. The experimental phase is closed. The study's positive engineering results and negative accuracy endpoint are both retained; the dominant cause of the final discrepancy and the globally best attainable IBM implementation remain unestablished.

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

References

Reference provenance: Entries [1]–[24] identify the scientific and software-documentation materials reported in the study's input corpus. Access dates are those reported by the author. Entries [25]–[36] were added during scientific review as external explanatory sources and are not claimed to have been included in the original input corpus. The additional web documentation was consulted on October 15, 2026; it provides context rather than independent verification of historical account access or calibration values.

[1] Stellora.AI, "Stellora.AI Quantum Flow," 2026. Available: https://stellora.ai/quantum/. Accessed: Aug. 9, 2026.

[2] Stellora.AI, "Stellora.AI Core," 2026. Available: https://stellora.ai/core/. Accessed: Aug. 9, 2026.

[3] R. M. Eisberg and R. Resnick, Quantum Physics of Atoms, Molecules, Solids, Nuclei, and Particles, 2nd ed. New York, NY, USA: John Wiley & Sons, 1985. ISBN: 978-0-471-87373-0. Available: https://www.amazon.com/Quantum-Physics-Molecules-Solids-Particles/dp/047187373X. Accessed: Aug. 10, 2026.

[4] J. Binney and D. Skinner, The Physics of Quantum Mechanics. Oxford, UK: Oxford University Press, 2014. ISBN: 978-0-19-968857-9. Available: https://global.oup.com/academic/product/the-physics-of-quantum-mechanics-9780199688579. Accessed: Aug. 10, 2026.

[5] M. Le Bellac, Quantum Physics. Cambridge, UK: Cambridge University Press, 2011. ISBN: 978-1-107-60276-2. Available: https://www.cambridge.org/gb/universitypress/subjects/physics/quantum-physics-quantum-information-and-quantum-computation/quantum-physics. Accessed: Aug. 10, 2026.

[6] S. P. Jordan, "Quantum Algorithm Zoo," Quantum Algorithm Zoo, online resource. Available: https://quantumalgorithmzoo.org/. Accessed: Aug. 10, 2026.

[7] J. Goings, "Variational Quantum Eigensolver (VQE) Example," Joshua Goings, Aug. 20, 2020. Available: https://joshuagoings.com/2020/08/20/VQE/. Accessed: Aug. 10, 2026.

[8] X. Tan, "VQE Algorithm in Quantum Chemistry," Highlights in Science, Engineering and Technology, vol. 5, pp. 173–178, 2022, doi: 10.54097/hset.v5i.739. Available: https://www.researchgate.net/publication/362019513_VQE_Algorithm_in_Quantum_Chemistry. Accessed: Aug. 10, 2026.

[9] O. Granichin and N. Amelina, "Simultaneous Perturbation Stochastic Approximation for Tracking Under Unknown but Bounded Disturbances," IEEE Transactions on Automatic Control, vol. 60, no. 6, pp. 1653–1658, Jun. 2015, doi: 10.1109/TAC.2014.2359711. Available via ResearchGate: https://www.researchgate.net/publication/277028715_Simultaneous_Perturbation_Stochastic_Approximation_for_Tracking_Under_Unknown_but_Bounded_Disturbances. Accessed: Aug. 12, 2026.

[10] J. Gidi, B. Candia, A. D. Muñoz-Moller, A. Rojas, L. Pereira, M. Muñoz, L. Zambrano, and A. Delgado, "Stochastic optimization algorithms for quantum applications," Physical Review A, vol. 108, no. 3, Art. no. 032409, 2023, doi: 10.1103/PhysRevA.108.032409. Available via ResearchGate: https://www.researchgate.net/publication/373791728_Stochastic_optimization_algorithms_for_quantum_applications. Accessed: Aug. 12, 2026.

[11] R. J. P. T. de Keijzer, V. E. Colussi, B. Škorić, and S. J. J. M. F. Kokkelmans, "Optimization of the variational quantum eigensolver for quantum chemistry applications," AVS Quantum Science, vol. 4, no. 1, Art. no. 013803, 2022, doi: 10.1116/5.0076435. Available via ResearchGate: https://www.researchgate.net/publication/349025671_Optimization_of_the_Variational_Quantum_Eigensolver_for_Quantum_Chemistry_Applications. Accessed: Aug. 12, 2026.

[12] Qiskit Development Team, "Qiskit 2.5.1," Qiskit/qiskit, GitHub release, Jul. 23, 2026. Available: https://github.com/Qiskit/qiskit/releases/tag/2.5.1. Accessed: Aug. 12, 2026.

[13] Qiskit Development Team, "Qiskit IBM Runtime 0.49.0," Qiskit/qiskit-ibm-runtime, GitHub release, Aug. 10,

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

2026. Available: https://github.com/Qiskit/qiskit-ibm-runtime/releases/tag/0.49.0. Accessed: Aug. 12, 2026.

[14] Qiskit Development Team, “Qiskit Documentation, branch ajc/qiskit-251-take-two,” Qiskit/documentation, GitHub repository branch. Available: https://github.com/Qiskit/documentation/tree/ajc/qiskit-251-take-two. Accessed: Aug. 12, 2026.

[15] IBM Quantum, “Build a Qiskit Function template for Hamiltonian simulation,” IBM Quantum Documentation. Available: https://quantum.cloud.ibm.com/docs/en/guides/function-template-hamiltonian-simulation. Accessed: Aug. 25, 2026.

[16] H.-H. Tseng, H.-Y. Lin, S. Y.-C. Chen, Y. M. Chan, T.-C. Wei, and S. Yoo, “Observable Geometry for Effective Quantum Circuits,” arXiv preprint arXiv:2607.20198, 2026. Available via ResearchGate: https://www.researchgate.net/publication/410720988_Observable_Geometry_for_Effective_Quantum_Circuits. Also available: https://arxiv.org/abs/2607.20198. Accessed: Aug. 25, 2026.

[17] C.-D. Yang, “Quantum Hamilton mechanics: Hamilton equations of quantum motion, origin of quantum operators, and proof of quantization axiom,” Annals of Physics, vol. 321, no. 12, pp. 2876–2926, Dec. 2006, doi: 10.1016/j.aop.2006.07.008. Available via ResearchGate: https://www.researchgate.net/publication/243370682_Quantum_Hamilton_mechanics_Hamilton_equations_of_quantum_motion_origin_of_quantum_operators_and_proof_of_quantization_axiom. Accessed: Aug. 25, 2026.

[18] IBM Quantum, “SparsePauliOp,” Qiskit API Documentation. Available: https://quantum.cloud.ibm.com/docs/en/api/qiskit/qiskit.quantum_info.SparsePauliOp. Accessed: Oct. 5, 2026.

[19] IBM Quantum, “Pauli,” Qiskit API Documentation. Available: https://eu-de.quantum.cloud.ibm.com/docs/en/api/qiskit/qiskit.quantum_info.Pauli. Accessed: Oct. 5, 2026.

[20] IBM Quantum, “Variational Quantum Eigensolver,” IBM Quantum Learning. Available: https://quantum.cloud.ibm.com/learning/en/modules/computer-science/vqe. Accessed: Oct. 5, 2026.

[21] N. Barnea, “High-Order Matrix Numerov for Singular Potentials,” arXiv preprint, arXiv:2603.09350v1, Mar. 10, 2026, doi: 10.48550/arXiv.2603.09350. Available: https://arxiv.org/abs/2603.09350v1. Accessed: Oct. 5, 2026.

[22] M. Motta, W. Kirby, I. Liepuoniute, K. J. Sung, J. Cohn, A. Mezzacapo, K. Klymko, N. Nguyen, N. Yoshioka, and J. E. Rice, “Subspace methods for electronic structure simulations on quantum computers,” Electronic Structure, vol. 6, Art. no. 013001, 2024, doi: 10.1088/2516-1075/ad3592. Preprint version: arXiv:2312.00178v1, Nov. 30, 2023. Available: https://arxiv.org/abs/2312.00178v1. Accessed: Oct. 5, 2026.

[23] W. M. Kirby, A. Tranter, and P. J. Love, “Contextual Subspace Variational Quantum Eigensolver,” Quantum, vol. 5, Art. no. 456, May 14, 2021, doi: 10.22331/q-2021-05-14-456. arXiv manuscript version: arXiv:2011.10027v2, May 12, 2021. Available: https://arxiv.org/abs/2011.10027v2. Accessed: Oct. 5, 2026.

[24] C. C. Chang, K. S. McElvain, E. Rrapaj, and Y. Wu, “Improving Schrödinger Equation Implementations with Gray Code for Adiabatic Quantum Computers,” PRX Quantum, vol. 3, Art. no. 020356, Jun. 15, 2022, doi: 10.1103/PRXQuantum.3.020356. arXiv manuscript version: arXiv:2103.08056v4, Jun. 14, 2022. Available: https://arxiv.org/abs/2103.08056v4. Accessed: Oct. 5, 2026.

[25] J. R. McClean, J. Romero, R. Babbush, and A. Aspuru-Guzik, “The theory of variational hybrid quantum-classical algorithms,” New Journal of Physics, vol. 18, Art. no. 023023, 2016, doi: 10.1088/1367-2630/18/2/023023. https://arxiv.org/abs/1509.04279.

[26] C. Schön, E. Solano, F. Verstraete, J. I. Cirac, and M. M. Wolf, “Sequential Generation of Entangled Multiqubit States,” Physical Review Letters, vol. 95, Art. no. 110503, 2005, doi: 10.1103/PhysRevLett.95.110503. https://arxiv.org/abs/quant-ph/0501096.

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

[27] J. J. Wallman and J. Emerson, “Noise tailoring for scalable quantum computation via randomized compiling,” Physical Review A, vol. 94, Art. no. 052325, 2016, doi: 10.1103/PhysRevA.94.052325. https://arxiv.org/abs/1512.01098.

[28] K. Temme, S. Bravyi, and J. M. Gambetta, “Error Mitigation for Short-Depth Quantum Circuits,” Physical Review Letters, vol. 119, Art. no. 180509, 2017, doi: 10.1103/PhysRevLett.119.180509. https://arxiv.org/abs/1612.02058.

[29] Z. Cai, R. Babbush, S. C. Benjamin, S. Endo, W. J. Huggins, Y. Li, J. R. McClean, and T. E. O’Brien, “Quantum error mitigation,” Reviews of Modern Physics, vol. 95, Art. no. 045005, 2023, doi: 10.1103/RevModPhys.95.045005. https://arxiv.org/abs/2210.00921.

[30] IBM Quantum, “View backend details,” IBM Quantum Documentation. https://quantum.cloud.ibm.com/docs/en/guides/qpu-information. Consulted for editorial context: Oct. 15, 2026.

[31] IBM Quantum, “Processor types,” IBM Quantum Documentation. https://quantum.cloud.ibm.com/docs/en/guides/processor-types. Consulted for editorial context: Oct. 15, 2026.

[32] IBM Quantum, “Configure noise management with Sampler” and “Error mitigation and suppression techniques,” IBM Quantum Documentation. https://quantum.cloud.ibm.com/docs/en/guides/sampler-noise-management; https://quantum.cloud.ibm.com/docs/en/guides/error-mitigation-and-suppression-techniques. Consulted: Oct. 15, 2026. Interface availability is primitive- and version-dependent.

[33] IBM Quantum, “Workload usage,” IBM Quantum Documentation. https://quantum.cloud.ibm.com/docs/en/guides/estimate-job-run-time. Consulted: Oct. 15, 2026. Usage and wall-clock latency are not inferred from shot or job counts alone.

[34] IBM Quantum, “Initialize,” Qiskit API Documentation. https://quantum.cloud.ibm.com/docs/en/api/qiskit/qiskit.circuit.library.Initialize. Consulted: Oct. 15, 2026.

[35] Python Software Foundation, “ast — Abstract syntax trees,” Python Documentation, section ast.literal_eval. https://docs.python.org/3/library/ast.html#ast.literal_eval. Consulted: Oct. 15, 2026.

[36] A. Kandala, A. Mezzacapo, K. Temme, M. Takita, M. Brink, J. M. Chow, and J. M. Gambetta, “Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets,” Nature, vol. 549, pp. 242–246, 2017, doi: 10.1038/nature23879. https://arxiv.org/abs/1704.05018.

Docusign Envelope ID: E9FADC5D-280C-8D0D-82F8-9383023221F9

Disclaimer and Usage Limitations

This study is a developer-authored, human-guided case study, not an independent certification of Quantum Flow, IBM hardware, or general quantum-computing performance. Its findings apply to the documented finite hydrogen instance, prompts, software, selected circuits, access conditions, and recorded experiments. Results on other tasks or configurations may differ.

Stellora.AI describes Quantum Flow as using supplied scientific material and target-platform documentation to support code generation and refinement [1, 2]. Retrieval, agent agreement, and fluent explanations do not replace independent numerical or experimental checks. Incomplete, outdated, contradictory, or misinterpreted sources can affect correctness; source inclusion alone does not demonstrate traceable grounding of every generated decision.

The user supplied the physical target, discretization, interface constraints, reference workflow, and substantial revision guidance. These inputs are part of the demonstrated process. Where scientific information cannot be established from the available material, it must be identified as missing rather than inferred without validation.

Executable code, accepted jobs, and ideal simulation agreement do not by themselves establish accurate physical results, optimizer convergence, quantum advantage, or production readiness. The reported code and results are preserved as records, including warnings and unsuccessful validation outcomes. They are not a warranty or a recommendation to execute archived installation commands unchanged in a future environment.

Qualified review remains necessary to evaluate the physics, numerical representation, circuit transformations, measurements, uncertainty model, and conclusions. This study did not measure Quantum Flow's development-time or cost advantage over an expert implementation or a matched general-purpose model, and does not establish autonomous scientific discovery.

Observed outcomes can depend on generated implementation choices, source interpretation, SDK/API compatibility, compilation, state preparation, readout, qubit calibration, device conditions, shot allocation, scheduling, queueing, and account limits. This study did not isolate their individual contributions. A residual discrepancy must not be attributed exclusively to factors outside Quantum Flow's control merely because ideal checks passed.

Public IBM documentation distinguishes backend properties, processor families, calibration-dependent quantities, and account-specific availability [30, 31]. The preserved run-time environment and historical backend records remain authoritative for this experiment; current documentation cannot retrospectively certify a historical execution or prove that the hardware's best attainable accuracy was reached.

The recorded troubleshooting sequence demonstrates iterative revision and fault detection in this instance, while also retaining an unresolved offline-parser error and an unreproduced bootstrap. The experimental phase is closed. Any later methodological improvement or new device run would constitute additional work and would require its own versioned evidence and claims.

Quantum Flow should therefore be understood as an AI-assisted scientific and quantum-software engineering system, rather than as a replacement for scientific verification or qualified quantum expertise. The reliability of the overall workflow ultimately depends on the combination of appropriate source material, scientifically valid problem formulation, suitable target-platform documentation, correct experimental configuration, iterative validation, and informed interpretation of the resulting quantum-computing outputs.

Author name: Michal Charvát

Author signature: Signed by: 4F44C53BC0D1463...

Revision date: 16-10-2026