Case study – Synthesis of a VQE Quantum Circuit Using Stellora.AI Quantum Flow

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

Abstract

This case study investigates the synthesis and execution of a **Variational Quantum Eigensolver (VQE) workflow using Stellora.AI Quantum Flow**, a research-grounded system designed to transform scientific literature, mathematical descriptions, validated quantum knowledge, and target-platform SDK documentation into executable quantum code and circuits [1]. For this experiment, the supplied corpus served as the primary and authoritative knowledge source for quantum-code generation, refinement, and troubleshooting.

Quantum Flow is built **on Stellora.AI Core**, described by Stellora.AI as a **proprietary** multilayer agentic retrieval-and-reasoning architecture that combines vectorized retrieval, specialized agents, cross-checking, refinement, and validation to reduce unsupported outputs [2]. The reported experiment used a six-qubit VQE configuration and a hybrid quantum-classical workflow comprising definition of a demonstrative six-qubit Hamiltonian, preparation of a parameterized ansatz, hardware-aware transpilation, execution of IBM Quantum Runtime Estimator jobs on a physical quantum processing unit (QPU), and iterative classical optimization using COBYLA.

The experimental evidence includes the generated **Qiskit**-oriented implementation, transpiled circuit output, optimizer traces, returned expectation values, and IBM Quantum Platform records showing completed jobs on the **ibm_marrakesh** backend. The six-qubit logical circuit was transpiled into a 156-qubit hardware representation corresponding to the physical width of the selected backend, while the underlying variational circuit remained six-qubit. Post-run validation subsequently identified an observable-layout issue in the generated implementation: the six-qubit Pauli observable was expanded through identity padding rather than transformed using the transpiler-recorded physical layout. Consequently, the returned objective values are treated in this study as execution diagnostics rather than as validated VQE energies for the intended six-qubit Hamiltonian.

The classical optimization stage also terminated within a limited evaluation budget. The objective of this paper is therefore not to claim quantum advantage, convergence to a physically meaningful ground-state energy, or complete scientific validity of the generated VQE implementation. Instead, it evaluates whether a research- and SDK-grounded AI workflow can transform a controlled corpus of scientific and technical material into executable quantum code, generate and transpile a non-trivial quantum circuit, submit workloads to physical quantum hardware, and expose implementation issues for subsequent troubleshooting and refinement. The study focuses on research-to-execution traceability, functional hardware execution, transparent limitation reporting, and reproducibility.

Contents

Introduction

A persistent bottleneck in practical quantum-computing work is the translation of research concepts, mathematical formulations, and platform documentation into executable, backend-compatible code. For this study, the research-to-execution process is represented as: **paper or model → quantum algorithm → SDK-specific code → transpilation and debugging → simulator or QPU test**. Vendor-specific SDKs, backend constraints, repeated porting, and the need for combined physics, mathematics, software-engineering, and SDK expertise contribute to the implementation overhead that Quantum Flow is intended to address.

Stellora.AI Quantum Flow is presented as a system intended **to reduce that translation** burden by grounding code generation and troubleshooting in the user-provided research material and the documentation of the target quantum platform [1]. The public Quantum Flow documentation states that its inputs can include current quantum research, validated quantum theories, and up-to-date SDKs or guides for the selected machine, together with a prompt specifying the computational goal and target framework [1].

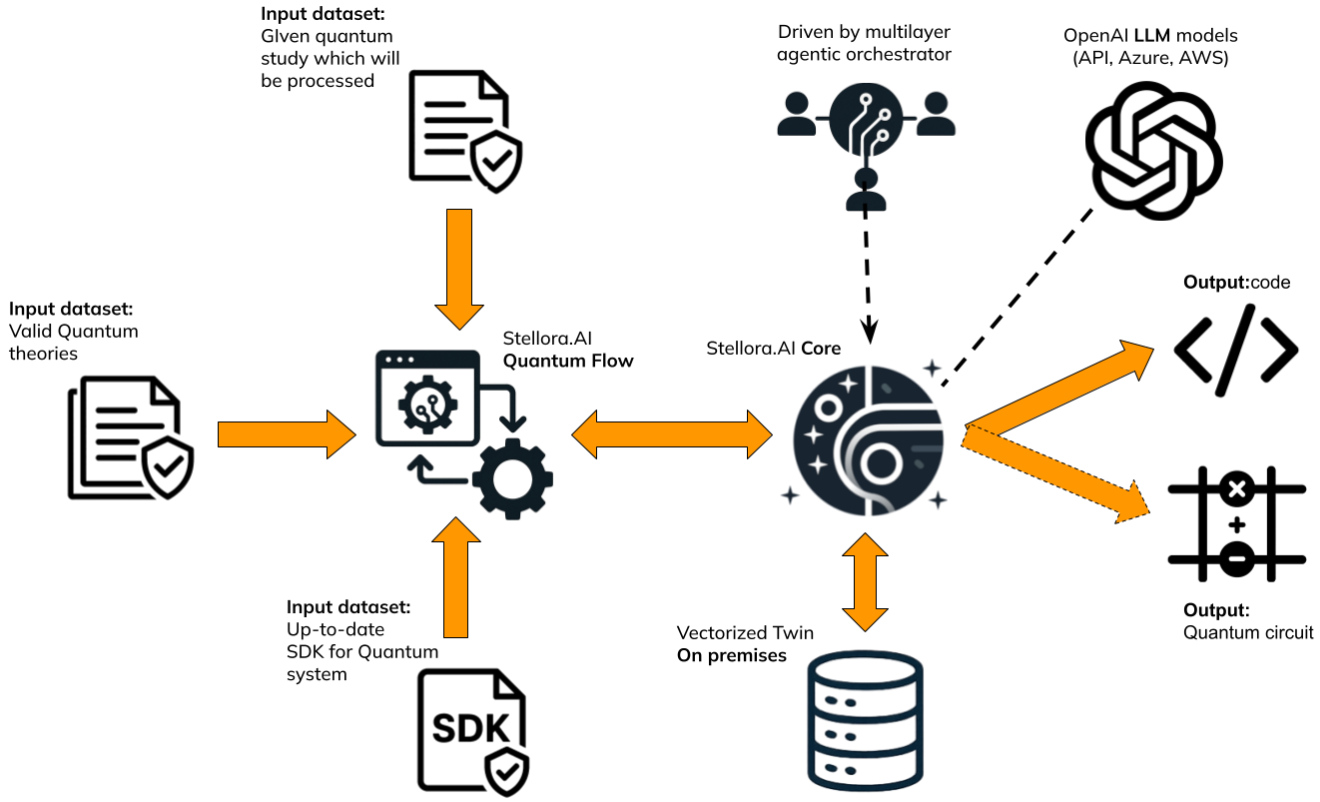


Figure 1: Stellora.AI Quantum Flow high-level overview

The present case study narrows that broader proposition to a concrete experimental question: can Quantum Flow synthesize and refine a VQE-oriented implementation that proceeds from a high-level research/task context to actual execution on an IBM QPU? The experimental prompt permitted either SPSA or COBYLA as the classical optimizer; the implementation generated by Quantum Flow selected COBYLA. This paper evaluates the resulting workflow from corpus-grounded code synthesis through transpilation and physical QPU execution, followed by post-run validation of the generated implementation. The study therefore distinguishes successful code generation and hardware execution from the separate requirement of scientific correctness and validation.

Security / Sensitive Information statement

Certain information has been intentionally omitted or anonymized for security and privacy reasons. This includes infrastructure-specific details that could expose or facilitate unauthorized access to the systems used in this study, such as the IP addresses of hosting machines, network identifiers, access-related configuration details, and similar sensitive technical information. These omissions are intended to reduce security risks and do not affect the interpretation of the reported results or the main conclusions of the study.

Environment

The **runtime environment** for Stellora.AI Core, Stellora.AI Quantum Flow and Stellora.AI Interactive Flow was as follows:

- Raspberry Pi 5 Model B Rev 1.1
- Operating system: Debian GNU/Linux 13.6 (trixie) aarch64
- RAM: 16GB LPDDR4X-4267
- CPU: BCM2712 quad-core Arm Cortex A76 processor @ 2.4GHz
- Storage:
  - SD card: Raspberry Pi 64GB microSDXC Class 10 UHS-I U1 A1
  - NVMe: Suptronics - X1001 2280 M.2 NVMe Shield w/ PATRIOT P300 SSD 1TB M.2 2280
- Add-ons: Raspberry Pi 5 Active Cooler
- Kernel: 6.18.46-v8-16k+
- Native installed Python version: 3.13.5

Stellora.AI component versions

- Stellora.AI (Core) version: Build 7bd06154fd14ac167f8d875d0b8a4868d43c234f
- Stellora.AI Flows (collection): Build d6f2c1e41fff91a669561e760aee7d24b9a9d292
- Stellora.AI Scripts (collection): HEAD 6bcdeee46d87c2b35919f536fc4111ca479ace36

Key Stellora.AI configuration

- LLM model used: OpenAI “gpt-5.4-mini” via API
- Embedding model used: OpenAI “text-embedding-3-large” via API
- Maximum tokens: 127999 (per inner request for cloud-based LLM)
- Dataset mode: unstructured
- Mode: FastMCP (used by Interactive Flow)
- Agentic mode is enabled
- AI memory is enabled
- AI memory limit: 250 (maximum number of messages before memory is purged)

Quantum backend

- IBM Quantum Platform access / account
- Access to the IBM Quantum open-instance tier
- API key generated

Input dataset

The dataset was stored under "~/stellora.ai.flows.kb/quantum.ibm.cloud.qpi-VQE" on the system. This directory contains the following input documents:

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*. Literature converted to OCR-enabled PDF before embedding.
- Quantum Algorithms.pdf [6] — *Quantum Algorithm Zoo, Stephen P. Jordan.* Web resource converted to PDF before embedding.

Quantum studies processed:

- Variational Quantum Eigensolver (VQE) Example · Joshua Goings.pdf [7] — *Variational Quantum Eigensolver (VQE) Example, Joshua Goings.* Original web resource converted to an OCR-enabled PDF prior to the embedding process
- 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 [17] — *Quantum Hamilton mechanics: Hamilton equations of quantum motion, origin of quantum operators, and proof of quantization axiom*

Up-to-date SDK documentation 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

Launching of Stellora.AI Quantum Flow

Embedding – creation of vectorized twin

Logged in as a user with unrestricted access to the Stellora.AI components listed above.

Checked that the components exist in the $HOME Linux user directory:

```
$ 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
```

For embedding runs, it is recommended to process each dataset separately:

```
$ 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-VQE false true
```

The following output was then written to 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-VQE
Dataset directory for the flow: /home/stellora/stellora.ai.flows.kb/quantum.ibm.cloud.qpi-VQE
Dataset cache directory for the flow: /home/stellora/stellora.ai.flows.kb.cache/quantum.ibm.cloud.qpi-VQE
Bind arguments for the flow: -a [redacted]
No folders to remove from dataset directory.
Running in embedding only mod
2026-08-26 18:10:13 [WARN]: Running as a foreground process (--no-daemon flag is set)
2026-08-26 18:10:13 [INFO]: PID file created at /tmp/Stellora.AI-core.pid
2026-08-26 18:10:13 [INFO]: Stellora.AI: Core is starting
2026-08-26 18:10:13 [INFO]: Author: Michal Charvát (mcharvat90@gmail.com)
2026-08-26 18:10:13 [INFO]: License: proprietary (usage and/or distribution without author's permission is prohibited
2026-08-26 18:10:13 [INFO]: Running in embedding-only mode, the program will exit after embedding the dataset. Dataset cache must be enabled for this mode.
2026-08-26 18:10:13 [INFO]: Embedding model: text-embedding-3-large
2026-08-26 18:10:13 [INFO]: Preparing the dataset for use with the AI model...
2026-08-26 18:10:13 [INFO]: Dataset directory: /home/stellora/stellora.ai.flows.kb/quantum.ibm.cloud.qpis-VQE
2026-08-26 18:10:13 [INFO]: Dataset mode: unstructured
2026-08-26 18:10:13 [INFO]: Dataset cache enable status: True
2026-08-26 18:10:13 [INFO]: Dataset cache directory: /home/stellora/stellora.ai.flows.kb.cache/quantum.ibm.cloud.qpi-VQE
2026-08-26 18:10:13 [INFO]: Creating the dataset cache directory (not existing)...
2026-08-26 18:10:13 [INFO]: Indexing the dataset...
2026-08-26 21:16:14 [INFO]: Creating vector index via DirectoryLoader...
2026-08-26 21:16:14 [WARN]: Purging old persistence data in '/home/stellora/stellora.ai.flows.kb.cache/quantum.ibm.cloud.qpi-VQE'…
2026-08-26 21:16:14 [INFO]: Creating vectorized index and saving to '/home/stellora/stellora.ai.flows.kb.cache/quantum.ibm.cloud.qpi-VQE'…
```

Launching full Quantum Flow with Interactive Flow (WebUI) over MCP

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

The following output was then written to 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-VQE
Dataset directory for the flow: /mnt/secure_nvme/stellora/stellora.ai.flows.kb/quantum.ibm.cloud.qpi-VQE
Dataset cache directory for the flow: /tmp/ramdisk/quantum.ibm.cloud.qpi-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-VQE
[INFO]: Using the dataset cache directory /tmp/ramdisk/quantum.ibm.cloud.qpi-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...
2026-08-26 22:20:16.319 Uvicorn server started on [redacted]

  You can now view your Streamlit app in your browser.

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

Accessing Stellora.AI Interactive Flow (WebUI Flow)

Opening the HTTPS URL provided in the standard output displays the login screen, where the configured username and password were used:

This opens the next stage, which accepts input through a chat-style interface:

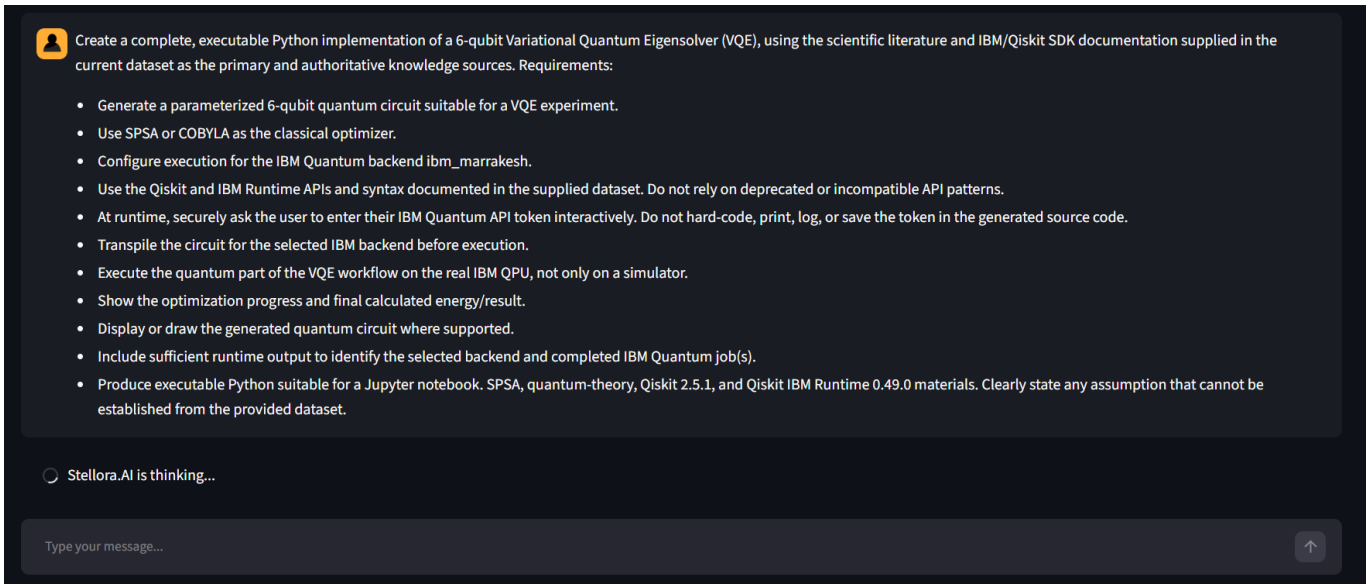
Submitting the prompt to Stellora.AI Quantum Flow through Interactive Flow:

Create a complete, executable Python implementation of a 6-qubit Variational Quantum Eigensolver (VQE), using the scientific literature and IBM/Qiskit SDK documentation supplied in the current dataset as the primary and authoritative knowledge sources.
Requirements:
* Generate a parameterized 6-qubit quantum circuit suitable for a VQE experiment.
* Use SPSA or COBYLA as the classical optimizer.
* Configure execution for the IBM Quantum backend ibm_marrakesh.
* Use the Qiskit and IBM Runtime APIs and syntax documented in the supplied dataset. Do not rely on deprecated or incompatible API patterns.
* At runtime, securely ask the user to enter their IBM Quantum API token interactively. Do not hard-code, print, log, or save the token in the generated source code.
* Transpile the circuit for the selected IBM backend before execution.
* Execute the quantum part of the VQE workflow on the real IBM QPU, not only on a simulator.
* Show the optimization progress and final calculated energy/result.

* Display or draw the generated quantum circuit where supported.
* Include sufficient runtime output to identify the selected backend and completed IBM Quantum job(s).
* Produce executable Python suitable for a Jupyter notebook.
SPSA, quantum-theory, Qiskit 2.5.1, and Qiskit IBM Runtime 0.49.0 materials.
Clearly state any assumption that cannot be established from the provided
dataset.

Reproducibility note: The experimental prompt above is reproduced verbatim from the request submitted to Quantum Flow, including its original wording and standalone phrase concerning the SPSA, quantum-theory, Qiskit 2.5.1, and Qiskit IBM Runtime 0.49.0 materials. No post-run corrections have been applied to the recorded experimental input.

Submitting the request



The WebUI returned the following response:

The complete response, including the generated code, is reproduced below:

%% [markdown]
6-qubit VQE on IBM Quantum hardware
Best-effort implementation based on the supplied context.
#
Assumptions:

```
# - The exact 6-qubit Hamiltonian is not fully specified in the supplied context.
#   A placeholder 6-qubit Hamiltonian is used below.
# - The notebook uses IBM Quantum Runtime on channel "ibm_quantum_platform".
# - The backend requested is "ibm_marrakesh".
# - No new session is created.
# - The circuit is transpiled for the selected backend before execution.
# - The observable is expanded/remapped to match the transpiled circuit width if needed.
# - The quantum execution is performed on a real IBM QPU.

# %%
import os
import numpy as np
from getpass import getpass
from scipy.optimize import minimize

from qiskit import transpile
from qiskit.circuit.library import EfficientSU2
from qiskit.quantum_info import SparsePauliOp

from qiskit_ibm_runtime import QiskitRuntimeService
from qiskit_ibm_runtime import EstimatorV2 as Estimator
from qiskit_ibm_runtime import EstimatorOptions

# Optional plotting support
try:
    from IPython.display import display
except Exception:
    display = None

# %%
# ------------------------------------------------
# Securely ask for IBM Quantum token at runtime
# ------------------------------------------------
token = os.environ.get("IBM_QUANTUM_TOKEN", "").strip()
if not token:
    token = getpass("Enter your IBM Quantum API token (input hidden): ").strip()

# %%
# ------------------------------------------------
# Connect to IBM Quantum Runtime platform
# ------------------------------------------------
service = QiskitRuntimeService(
    channel="ibm_quantum_platform",
    token=token
)

print("Initialized QiskitRuntimeService on channel: ibm_quantum_platform")

# %%
# ------------------------------------------------
# Select target backend
```

```
# ------------------------------------------------
backend_name = "ibm_marrakesh"
backend = service.backend(backend_name)

print(f"Selected backend: {backend.name}")
try:
    print(f"Backend num_qubits: {backend.num_qubits}")
except Exception:
    print("Backend qubit count unavailable from this runtime object.")

# %%
# ------------------------------------------------
# Build a parameterized 6-qubit ansatz
# ------------------------------------------------
num_qubits = 6
ansatz = EfficientSU2(num_qubits=num_qubits, reps=2, entanglement="linear")

print("Parameterized ansatz circuit:")
print(ansatz)

try:
    ansatz.draw("mpl")
except Exception:
    pass

# %%
# ------------------------------------------------
# Define the observable / Hamiltonian
# ------------------------------------------------
# Best-effort placeholder 6-qubit Hamiltonian.
# Replace with the exact Hamiltonian from your supplied dataset if available.
observable = SparsePauliOp.from_list([
    ("ZIIIII", 0.50),
    ("IZIIII", -0.30),
    ("IIZIII", 0.25),
    ("IIIZII", -0.20),
    ("IIIIZI", 0.15),
    ("IIIIIZ", -0.10),
    ("ZZIIII", 0.20),
    ("XXIIII", 0.10),
    ("IIZZII", 0.12),
    ("IIXXII", 0.08),
])

print("Original observable qubit count:", observable.num_qubits)

# %%
# ------------------------------------------------
# Transpile circuit for selected backend
# ------------------------------------------------
transpiled_ansatz = transpile(ansatz, backend=backend, optimization_level=1)
```

```
print("Transpiled circuit:")
print(transpiled_ansatz)

try:
    transpiled_ansatz.draw("mpl")
except Exception:
    pass

print("Transpiled circuit qubit count:", transpiled_ansatz.num_qubits)

# %%
# --------------------------------------------------
# Align / expand observable to match transpiled circuit width
# --------------------------------------------------
def align_observable_to_width(obs: SparsePauliOp, target_qubits: int) -> SparsePauliOp:
    """
    Expand a SparsePauliOp with identities so it acts on target_qubits.
    If obs already matches target_qubits, return it unchanged.
    If obs is wider than target_qubits, raise an error.
    """
    if obs.num_qubits == target_qubits:
        return obs

    if obs.num_qubits > target_qubits:
        raise ValueError(
            f"Observable has {obs.num_qubits} qubits but target circuit has {target_qubits}."
        )

    pad = target_qubits - obs.num_qubits
    padded_terms = []
    for pauli, coeff in zip(obs.paulis, obs.coeffs):
        label = pauli.to_label()
        padded_label = label + ("I" * pad)
        padded_terms.append((padded_label, coeff))

    return SparsePauliOp.from_list(padded_terms)

aligned_observable = align_observable_to_width(observable, transpiled_ansatz.num_qubits)

print("Aligned observable qubit count:", aligned_observable.num_qubits)

# %%
# --------------------------------------------------
# Configure runtime options
# --------------------------------------------------
estimator_options = EstimatorOptions(
    resilience_level=1,
    default_shots=4000
)
```

```
# %%
# ------------------------------------------------
# VQE cost function
# ------------------------------------------------
def extract_expectation_value(result):
    """
    Best-effort extraction of an expectation value from EstimatorV2 results.
    """
    candidate_paths = [
        lambda r: r.pub_results[0].data.evs,
        lambda r: r.pub_results[0].data.ev,
        lambda r: r.pub_results[0].data.value,
        lambda r: r[0].data.evs,
        lambda r: r[0].data.ev,
        lambda r: r[0].data.value,
        lambda r: r.values[0],
    ]

    for path in candidate_paths:
        try:
            val = path(result)
            arr = np.asarray(val).ravel()
            if arr.size > 0:
                return float(arr[0])
        except Exception:
            continue

    raise RuntimeError(f"Could not extract expectation value from result object of type {type(result)}")

def vqe_cost(theta, circuit, hamiltonian, estimator):
    """
    Best-effort EstimatorV2 VQE cost function.
    """
    theta = np.asarray(theta, dtype=float)

    pub_candidates = [
        (circuit, hamiltonian, theta),
        (circuit, hamiltonian, [theta]),
        (circuit, [hamiltonian], [theta]),
    ]

    last_exc = None
    for pub in pub_candidates:
        try:
            job = estimator.run([pub])
            print("Submitted IBM Quantum job.")
            result = job.result()
            energy = extract_expectation_value(result)
            print(f"theta = {np.array2string(theta, precision=3, suppress_small=True)}")
            print(f"energy = {energy:.12f}")
            return energy
```

```
        except Exception as exc:
            last_exc = exc

    raise RuntimeError(f"Estimator evaluation failed: {last_exc}")

# %%
# --------------------------------------------------
# Run VQE optimization on the real IBM QPU
# --------------------------------------------------
# We do not create a new session; use the runtime primitive directly.
estimator = Estimator(mode=backend, options=estimator_options)

initial_point = np.zeros(transpiled_ansatz.num_parameters)

iteration_counter = {"n": 0}
energy_history = []

def objective(theta):
    iteration_counter["n"] += 1
    print(f"\nOptimization iteration {iteration_counter['n']}")
    energy = vqe_cost(theta, transpiled_ansatz, aligned_observable, estimator)
    energy_history.append(energy)
    return energy

result = minimize(
    objective,
    initial_point,
    method="COBYLA",
    options={"maxiter": 15, "disp": True},
)

# %%
# --------------------------------------------------
# Report final results
# --------------------------------------------------
print("\n===== VQE Optimization Complete =====")
print("Selected backend:", backend.name)
print("Optimization success:", result.success)
print("Optimizer message:", result.message)
print("Final energy:", result.fun)
print("Final parameters:", result.x)
print("Total objective evaluations:", result.nfev)

# %%
# --------------------------------------------------
# Optional final evaluation
# --------------------------------------------------
final_energy = vqe_cost(result.x, transpiled_ansatz, aligned_observable, estimator)
print("Final recalculated energy:", final_energy)

# %%
```

```
# ------------------------------------------------
# Plot optimization progress if available
# ------------------------------------------------
try:
    import matplotlib.pyplot as plt

    plt.figure(figsize=(7, 4))
    plt.plot(energy_history, marker="o")
    plt.xlabel("Iteration")
    plt.ylabel("Energy")
    plt.title("VQE Optimization Progress")
    plt.grid(True)
    plt.show()
except Exception as exc:
    print("Could not plot optimization history:", exc)
```

Testing the quantum code / circuit

Stellora.AI Quantum Flow generates, by default, formatted code blocks suitable for JupyterLab or Google Colab notebooks.

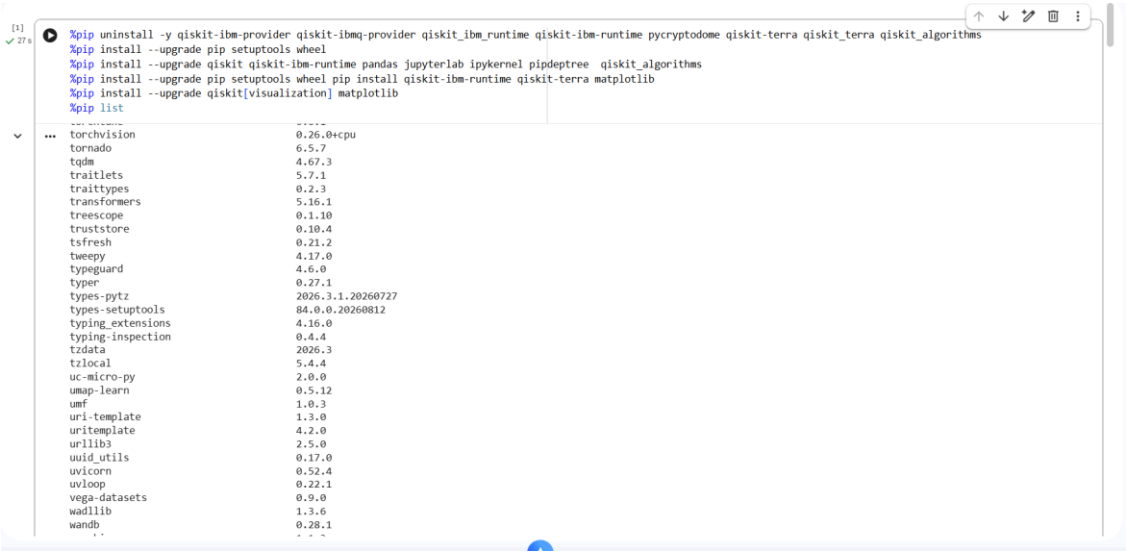
For our use case, we used Google Colab because it provides a Python environment where we can install the required Qiskit components and submit IBM Quantum Runtime jobs to the selected backend through the IBM Quantum Platform.

Setting up the Google Colab runtime

The following package-management commands were executed in the first cell of the Colab notebook to prepare the runtime environment:

```
%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
```

Package installation completed successfully



Pasting the generated code into the second code cell

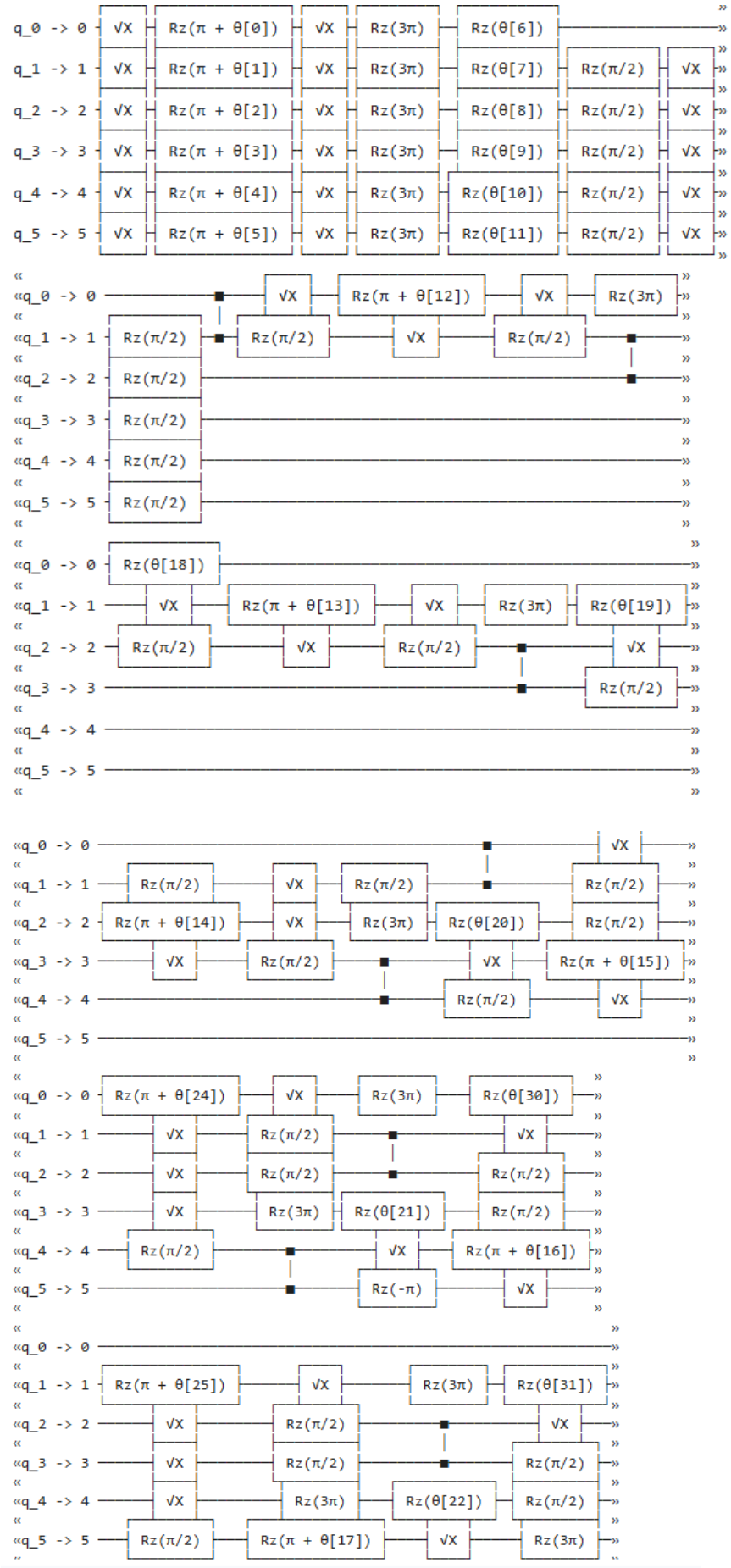
```
# %% [markdown]
# # 6-qubit VQE on IBM Quantum hardware
# Best-effort implementation based on the supplied context.
#
# Assumptions:
# - The exact 6-qubit Hamiltonian is not fully specified in the supplied context.
#   A placeholder 6-qubit Hamiltonian is used below.
# - The notebook uses IBM Quantum Runtime on channel "ibm_quantum_platform".
# - The backend requested is "ibm_marrakesh".
# - No new session is created.
# - The circuit is transpiled for the selected backend before execution.
# - The observable is expanded/remapped to match the transpiled circuit width if needed.
# - The quantum execution is performed on a real IBM QPU.

# %%
import os
import numpy as np
from getpass import getpass
from scipy.optimize import minimize

from qiskit import transpile
from qiskit.circuit.library import EfficientSU2
from qiskit.quantum_info import SparsePauliOp

from qiskit_ibm_runtime import QiskitRuntimeService
from qiskit_ibm_runtime import EstimatorV2 as Estimator
from qiskit_ibm_runtime import EstimatorOptions
```

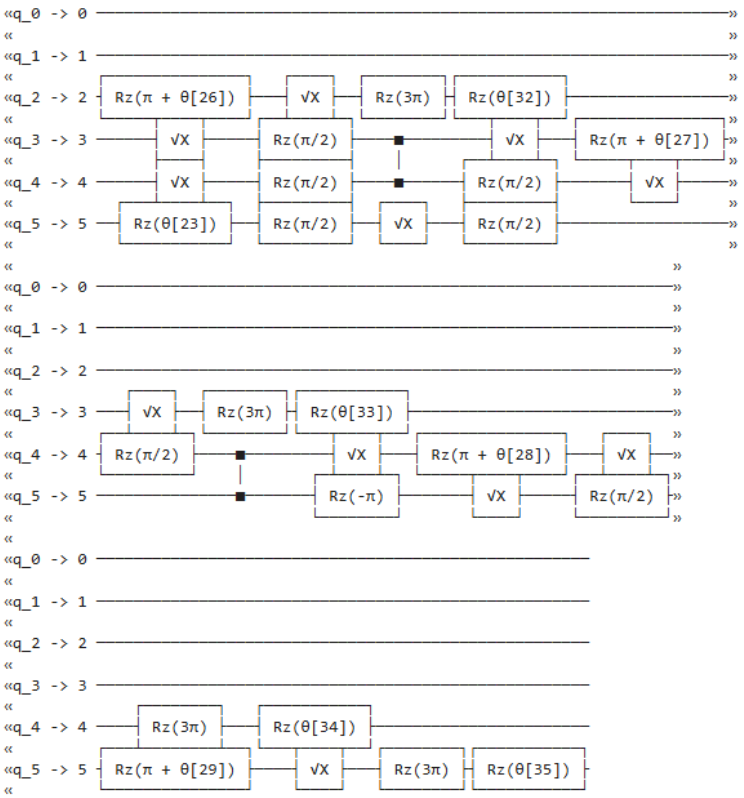
Results after execution: parameterized ansatz circuit

The run took some time to finish due to target-backend availability and queueing on the IBM Quantum Platform.

```
Enter your IBM Quantum API token (input hidden): ·········
qiskit_runtime_service._discover_account:WARNING:2026-08-26 22:32:28,061: Loading account with the given token. A saved account will not be used.
qiskit_runtime_service.__init__:WARNING:2026-08-26 22:32:32,083: 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 av
qiskit_runtime_service.backends:WARNING:2026-08-26 22:32:32,084: Using instance: open-instance, plan: open
Initialized QiskitRuntimeService on channel: ibm_quantum_platform
Selected backend: ibm_marrakesh
Backend num_qubits: 156
Parameterized ansatz circuit:
      »
q_0: »
      »
q_1: »
      »
q_2: »
      »
q_3: »
      »
q_4: »
      »
q_5: »
      »
«
«q_0: ┤0
«
«q_1: ┤1
«
«q_2: ┤2
«     │  EfficientSU2(θ[0],θ[1],θ[2],θ[3],θ[4],θ[5],θ[6],θ[7],θ[8],θ[9],θ[10],θ[11],θ[12],θ[13],θ[14],θ[15],θ[16],θ[17],θ[18],θ[19],θ[20],θ[21],θ[22],θ[23],θ[24],θ[25],θ[26],θ[27],θ[28],θ[29],θ[30],θ[31],θ[32],θ[33],θ[34],θ[35])
«q_3: ┤3
«
«q_4: ┤4
«
«q_5: ┤5
«
Original observable qubit count: 6
Transpiled circuit:
global phase: π/2
```

Results after execution: hardware-transpiled circuit





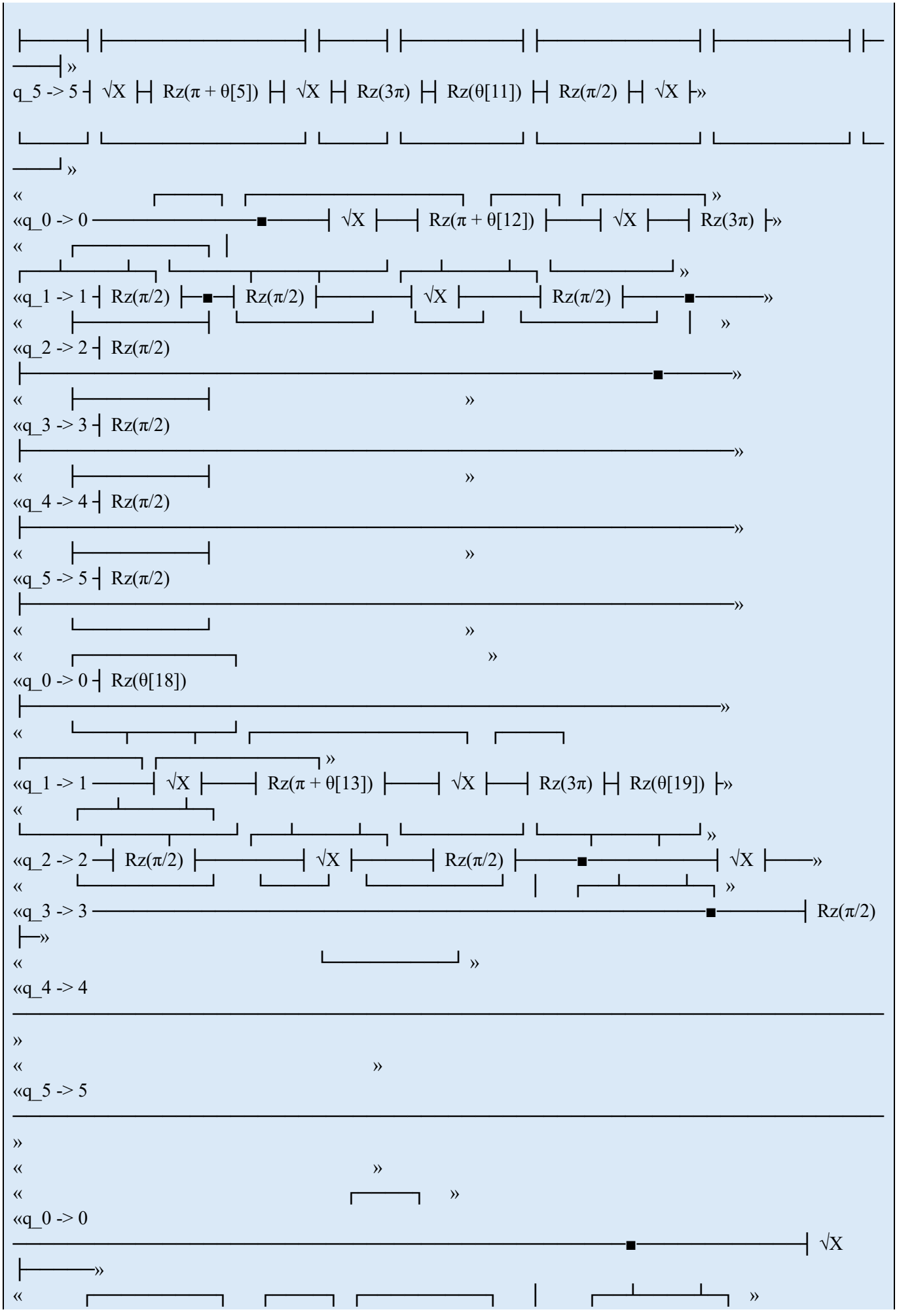
Full runtime output (verbatim)

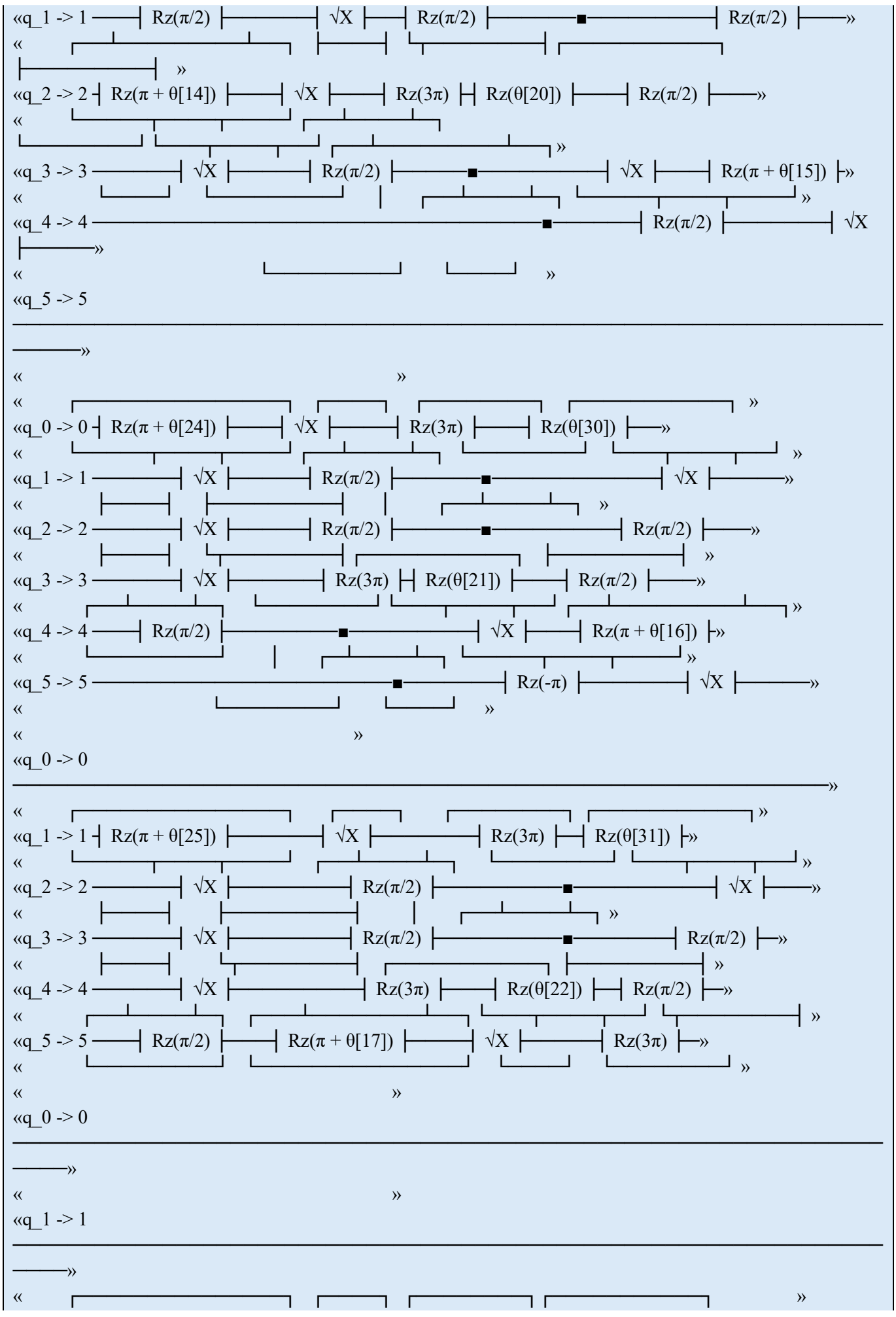
```
Enter your IBM Quantum API token (input hidden): ··········
qiskit_runtime_service._discover_account:WARNING:2026-08-26 22:32:28,061: Loading account with the given token. A saved account will not be used.
qiskit_runtime_service.__init__:WARNING:2026-08-26 22:32:32,083: Instance was not set at service 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:2026-08-26 22:32:32,084: Using instance: open-instance, plan: open
Initialized QiskitRuntimeService on channel: ibm_quantum_platform
Selected backend: ibm_marrakesh
Backend num_qubits: 156
Parameterized ansatz circuit:
     »
q_0: »
     »
q_1: »
     »
q_2: »
     »
q_3: »
     »
q_4: »
     »
q_5: »
     »
```

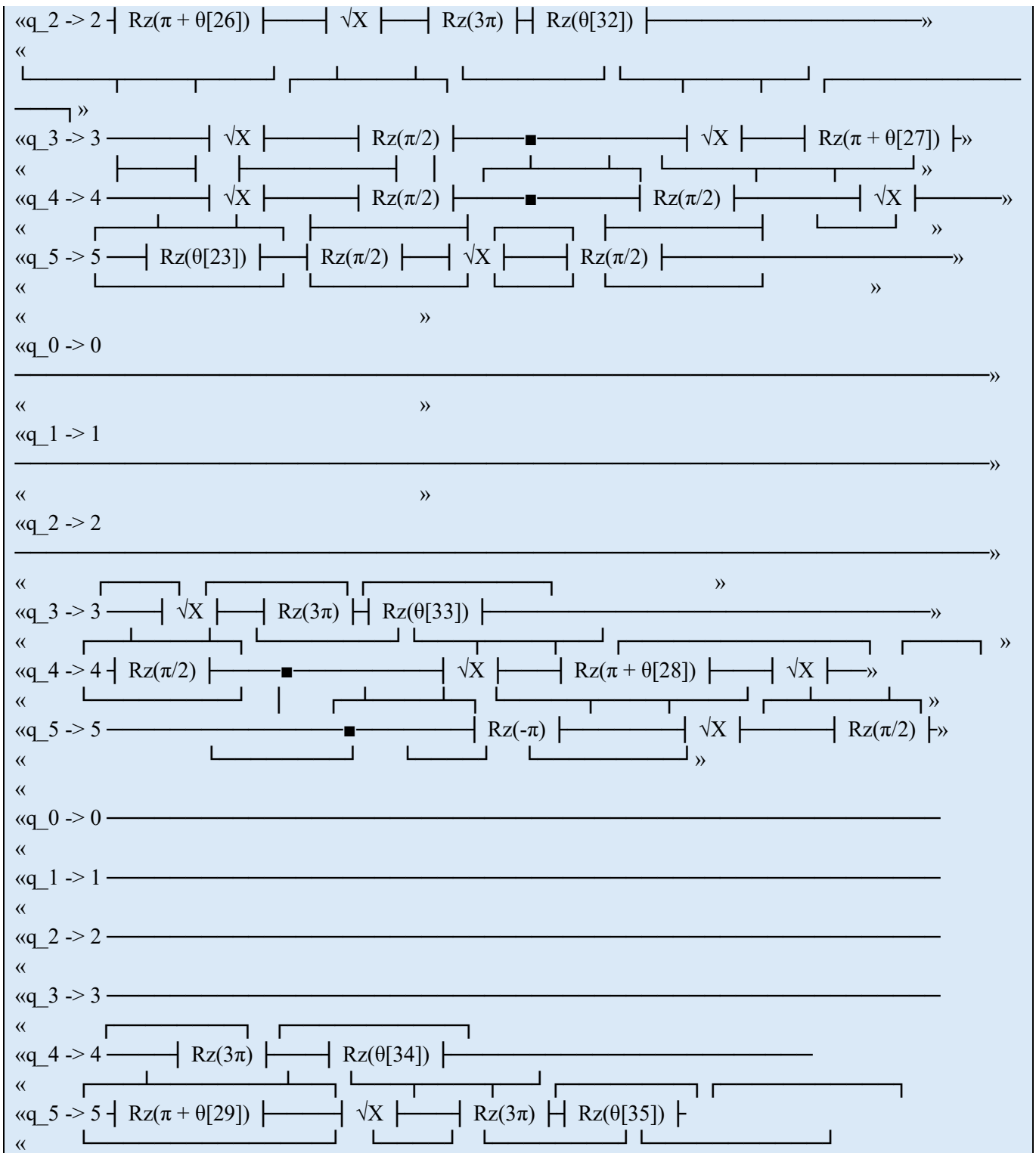
```
«
┌──────────────────────────────────────────────────────────────────────
───────────────────────────────────────────────────────────────────────
───────────────────────────────────────────────────────────────────────
──────────────────────────────────┐
«q_0: ┤0
├
«     │
│
«q_1: ┤1
├
«     │
│
«q_2: ┤2
├
«     │
EfficientSU2(θ[0],θ[1],θ[2],θ[3],θ[4],θ[5],θ[6],θ[7],θ[8],θ[9],θ[10],θ[11],θ[12],θ[13],θ[14],θ[15],θ[16],θ[17],θ[1
8],θ[19],θ[20],θ[21],θ[22],θ[23],θ[24],θ[25],θ[26],θ[27],θ[28],θ[29],θ[30],θ[31],θ[32],θ[33],θ[34],θ[35]) │
«q_3: ┤3
├
«     │
│
«q_4: ┤4
├
«     │
│
«q_5: ┤5
├
«
└──────────────────────────────────────────────────────────────────────
───────────────────────────────────────────────────────────────────────
───────────────────────────────────────────────────────────────────────
──────────────────────────────────┘
```

Original observable qubit count: 6
Transpiled circuit:
global phase: π/2

```
       ┌────┐ ┌───────────────┐ ┌────┐ ┌───────┐ ┌──────────┐ ┌────────────┐      »
q_0 -> 0 ┤ √X ├┤ Rz(π + θ[0]) ├┤ √X ├┤ Rz(3π) ├──┤ Rz(θ[6]) ├──────────────────»
       ├────┤ ├───────────────┤ ├────┤ ├──────────┤
├──────────┤ ┌────────────┐ ┌─────┐»
q_1 -> 1 ┤ √X ├┤ Rz(π + θ[1]) ├┤ √X ├┤ Rz(3π) ├──┤ Rz(θ[7]) ├┤ Rz(π/2) ├┤ √X ├»
       ├────┤ ├───────────────┤ ├────┤ ├──────────┤
├──────────┤ ├────────────┤ ├─────┤»
q_2 -> 2 ┤ √X ├┤ Rz(π + θ[2]) ├┤ √X ├┤ Rz(3π) ├──┤ Rz(θ[8]) ├┤ Rz(π/2) ├┤ √X ├»
       ├────┤ ├───────────────┤ ├────┤ ├──────────┤
├──────────┤ ├────────────┤ ├─────┤»
q_3 -> 3 ┤ √X ├┤ Rz(π + θ[3]) ├┤ √X ├┤ Rz(3π) ├──┤ Rz(θ[9]) ├┤ Rz(π/2) ├┤ √X ├»

├─────┤ ├───────────────┤ ├─────┤ ├─────────┤ ┌┴───────────┤ ├──────────┤ ├─
───┤»
q_4 -> 4 ┤ √X ├┤ Rz(π + θ[4]) ├┤ √X ├┤ Rz(3π) ├┤ Rz(θ[10]) ├┤ Rz(π/2) ├┤ √X ├»
```


q_5 -> 5 √X Rz(π + θ[5]) √X Rz(3π) Rz(θ[11]) Rz(π/2) √X
q_0 -> 0 √X Rz(π + θ[12]) √X Rz(3π)
q_1 -> 1 Rz(π/2) Rz(π/2) √X Rz(π/2)
q_2 -> 2 Rz(π/2)
q_3 -> 3 Rz(π/2)
q_4 -> 4 Rz(π/2)
q_5 -> 5 Rz(π/2)
q_0 -> 0 Rz(θ[18])
q_1 -> 1 √X Rz(π + θ[13]) √X Rz(3π) Rz(θ[19])
q_2 -> 2 Rz(π/2) √X Rz(π/2) √X
q_3 -> 3 Rz(π/2)
q_4 -> 4
q_5 -> 5
q_0 -> 0 √X





/tmp/ipykernel_2290/6541229.py:72: DeprecationWarning: The class ``qiskit.circuit.library.n_local.efficient_su2.EfficientSU2`` is deprecated as of Qiskit 2.1. It will be removed in Qiskit 3.0. Use the function qiskit.circuit.library.efficient_su2 instead.
  ansatz = EfficientSU2(num_qubits=num_qubits, reps=2, entanglement="linear")
Transpiled circuit qubit count: 156
Aligned observable qubit count: 156

Optimization iteration 1
Submitted IBM Quantum job.
theta = [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
energy = 0.625123533079

```
Optimization iteration 2
/usr/local/lib/python3.13/dist-packages/scipy/_lib/pyprima/common/preproc.py:68: UserWarning: COBYLA:
Invalid MAXFUN; it should be at least num_vars + 2; it is set to 38
  warn(f'{solver}: Invalid MAXFUN; it should be at least {min_maxfun_str}; it is set to {maxfun}')
Submitted IBM Quantum job.
theta = [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
energy = 0.628045359400

Optimization iteration 3
Submitted IBM Quantum job.
theta = [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
energy = 0.626358419042

Optimization iteration 4
Submitted IBM Quantum job.
theta = [0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
energy = 0.626892397547

Optimization iteration 5
Submitted IBM Quantum job.
theta = [0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
energy = 0.624108435336

Optimization iteration 6
Submitted IBM Quantum job.
theta = [0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
energy = 0.624606699556

Optimization iteration 7
Submitted IBM Quantum job.
theta = [0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
energy = 0.626689548758

Optimization iteration 8
Submitted IBM Quantum job.
theta = [0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
energy = 0.620098872109

Optimization iteration 9
Submitted IBM Quantum job.
theta = [0. 0. 0. 1. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
energy = 0.623385644347
```

Optimization iteration 10
Submitted IBM Quantum job.
theta = [0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
energy = 0.618966453839

Optimization iteration 11
Submitted IBM Quantum job.
theta = [0. 0. 0. 1. 0. 0. 1. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
energy = 0.628871516886

Optimization iteration 12
Submitted IBM Quantum job.
theta = [0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
energy = 0.623903974228

Optimization iteration 13
Submitted IBM Quantum job.
theta = [0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
energy = 0.620411468748

Optimization iteration 14
Submitted IBM Quantum job.
theta = [0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
energy = 0.628490509469

Optimization iteration 15
Submitted IBM Quantum job.
theta = [0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
energy = 0.621278253823

Optimization iteration 16
Submitted IBM Quantum job.
theta = [0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
energy = 0.621654006142

Optimization iteration 17
Submitted IBM Quantum job.
theta = [0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
energy = 0.621551126177

Optimization iteration 18
Submitted IBM Quantum job.

```
theta = [0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
energy = 0.622607996820

Optimization iteration 19
Submitted IBM Quantum job.
theta = [0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
energy = 0.624108517704

Optimization iteration 20
Submitted IBM Quantum job.
theta = [0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
energy = 0.621790472743

Optimization iteration 21
Submitted IBM Quantum job.
theta = [0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
energy = 0.623254802863

Optimization iteration 22
Submitted IBM Quantum job.
theta = [0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
energy = 0.623620532166

Optimization iteration 23
Submitted IBM Quantum job.
theta = [0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
energy = 0.625729617168

Optimization iteration 24
Submitted IBM Quantum job.
theta = [0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
energy = 0.621534012681

Optimization iteration 25
Submitted IBM Quantum job.
theta = [0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
energy = 0.626888066720

Optimization iteration 26
Submitted IBM Quantum job.
theta = [0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
energy = 0.628083372580
```

```
Optimization iteration 27
Submitted IBM Quantum job.
theta = [0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
energy = 0.625729782343

Optimization iteration 28
Submitted IBM Quantum job.
theta = [0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
energy = 0.624437644092

Optimization iteration 29
Submitted IBM Quantum job.
theta = [0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
energy = 0.618390850001

Optimization iteration 30
Submitted IBM Quantum job.
theta = [0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0.]
energy = 0.623500829062

Optimization iteration 31
Submitted IBM Quantum job.
theta = [0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0.]
energy = 0.624650009815

Optimization iteration 32
Submitted IBM Quantum job.
theta = [0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0.]
energy = 0.624856443301

Optimization iteration 33
Submitted IBM Quantum job.
theta = [0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0.]
energy = 0.625870838157

Optimization iteration 34
Submitted IBM Quantum job.
theta = [0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0.]
energy = 0.629601058658

Optimization iteration 35
Submitted IBM Quantum job.
```

```
theta = [0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0.]
energy = 0.622859017100

Optimization iteration 36
Submitted IBM Quantum job.
theta = [0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0.]
energy = 0.621482500834

Optimization iteration 37
Submitted IBM Quantum job.
theta = [0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1.]
energy = 0.624439482919

Optimization iteration 38
Submitted IBM Quantum job.
theta = [-0.092 -0.039 -0.056  1.032 -0.016 -0.082  1.127 -0.104  1.036 -0.313
 -0.156 -0.046 -0.301 -0.073 -0.085 -0.082 -0.115 -0.163 -0.089 -0.136
 -0.147 -0.214 -0.081 -0.25  -0.288 -0.214 -0.173  1.018 -0.162 -0.198
 -0.204 -0.236 -0.354 -0.141 -0.098 -0.191]
energy = 0.631433943083
Return from COBYLA because the objective function has been evaluated MAXFUN times.
Number of function values = 38   Least value of F = 0.6183908500012651
The corresponding X is:
[0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]



===== VQE Optimization Complete =====
Selected backend: ibm_marrakesh
Optimization success: False
Optimizer message: Return from COBYLA because the objective function has been evaluated MAXFUN times.
Final energy: 0.6183908500012651
Final parameters: [0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
Total objective evaluations: 38
Submitted IBM Quantum job.
theta = [0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
energy = 0.625633164848
Final recalculated energy: 0.6256331648480276
```

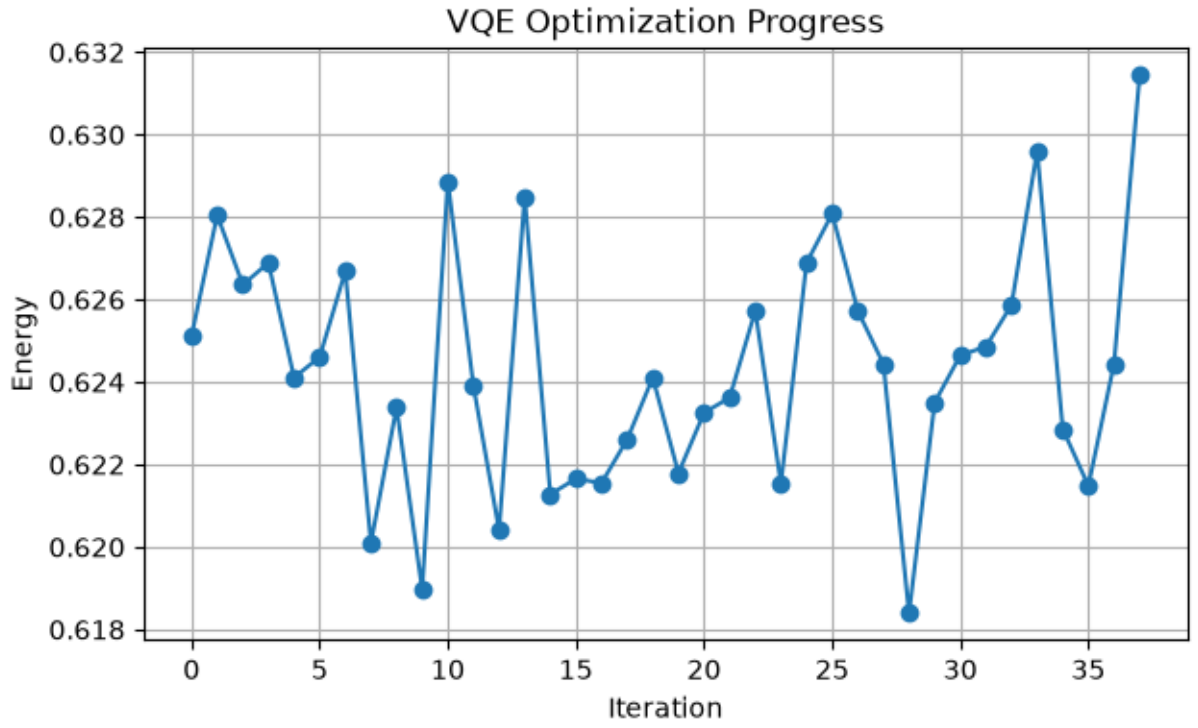


Figure 2: Objective values returned during the COBYLA evaluations

The generated program labels the vertical axis as “Energy.” Because the experiment used a demonstrative Hamiltonian and post-run validation identified an observable-layout mismatch, these measurements are reported here as diagnostic objective values rather than as validated physical VQE energies.

IBM Quantum Platform – Backend Execution Evidence

Platform-side record of jobs submitted and completed during the VQE optimization process

ID	Status	Instance	Mode	Created	Completed	QPU	Usage	User	Tags
da7mr3e0ukec7...	Completed	open-instance	Job	8/27/26, 12:49 AM	8/27/26, 12:50 AM	ibm_marrakesh	13s	Michal Charvát	
da7mqtm0ukec...	Completed	open-instance	Job	8/27/26, 12:49 AM	8/27/26, 12:49 AM	ibm_marrakesh	13s	Michal Charvát	
da7mqkc6l22c7...	Completed	open-instance	Job	8/27/26, 12:48 AM	8/27/26, 12:49 AM	ibm_marrakesh	13s	Michal Charvát	
da7mqb60ukec...	Completed	open-instance	Job	8/27/26, 12:48 AM	8/27/26, 12:48 AM	ibm_marrakesh	13s	Michal Charvát	
da7mq1k6l22c7...	Completed	open-instance	Job	8/27/26, 12:47 AM	8/27/26, 12:48 AM	ibm_marrakesh	13s	Michal Charvát	
da7mps3sq5js7...	Completed	open-instance	Job	8/27/26, 12:47 AM	8/27/26, 12:47 AM	ibm_marrakesh	13s	Michal Charvát	
da7mpmusidac...	Completed	open-instance	Job	8/27/26, 12:46 AM	8/27/26, 12:47 AM	ibm_marrakesh	13s	Michal Charvát	
da7mpdusidac7...	Completed	open-instance	Job	8/27/26, 12:46 AM	8/27/26, 12:46 AM	ibm_marrakesh	13s	Michal Charvát	
da7mp56sidac7...	Completed	open-instance	Job	8/27/26, 12:45 AM	8/27/26, 12:46 AM	ibm_marrakesh	13s	Michal Charvát	
da7mor3sq5js7...	Completed	open-instance	Job	8/27/26, 12:45 AM	8/27/26, 12:45 AM	ibm_marrakesh	13s	Michal Charvát	
da7molesidac7...	Completed	open-instance	Job	8/27/26, 12:44 AM	8/27/26, 12:44 AM	ibm_marrakesh	13s	Michal Charvát	
da7mofmsidac7...	Completed	open-instance	Job	8/27/26, 12:44 AM	8/27/26, 12:44 AM	ibm_marrakesh	13s	Michal Charvát	
da7mo6c6l22c7...	Completed	open-instance	Job	8/27/26, 12:43 AM	8/27/26, 12:44 AM	ibm_marrakesh	13s	Michal Charvát	

da7mnnk6l22...	Completed	open-instance	Job	8/27/26, 12:42 AM	8/27/26, 12:42 AM	ibm_marrakesh	13s	Michal Charvát	⋮
da7mnhrsq5js7...	Completed	open-instance	Job	8/27/26, 12:42 AM	8/27/26, 12:42 AM	ibm_marrakesh	13s	Michal Charvát	⋮
da7mn8u0ukec...	Completed	open-instance	Job	8/27/26, 12:41 AM	8/27/26, 12:42 AM	ibm_marrakesh	13s	Michal Charvát	⋮
da7mn33sq5js7...	Completed	open-instance	Job	8/27/26, 12:41 AM	8/27/26, 12:41 AM	ibm_marrakesh	13s	Michal Charvát	⋮
da7mmom0uke...	Completed	open-instance	Job	8/27/26, 12:40 AM	8/27/26, 12:41 AM	ibm_marrakesh	13s	Michal Charvát	⋮
da7mmjbsq5js7...	Completed	open-instance	Job	8/27/26, 12:40 AM	8/27/26, 12:40 AM	ibm_marrakesh	13s	Michal Charvát	⋮
da7mmdjsq5js7...	Completed	open-instance	Job	8/27/26, 12:39 AM	8/27/26, 12:40 AM	ibm_marrakesh	13s	Michal Charvát	⋮
da7mm7usidac...	Completed	open-instance	Job	8/27/26, 12:39 AM	8/27/26, 12:39 AM	ibm_marrakesh	13s	Michal Charvát	⋮
da7mm2c6l22c...	Completed	open-instance	Job	8/27/26, 12:39 AM	8/27/26, 12:39 AM	ibm_marrakesh	13s	Michal Charvát	⋮
da7mlsesidac73...	Completed	open-instance	Job	8/27/26, 12:38 AM	8/27/26, 12:39 AM	ibm_marrakesh	13s	Michal Charvát	⋮
da7mlmc6l22c7...	Completed	open-instance	Job	8/27/26, 12:38 AM	8/27/26, 12:38 AM	ibm_marrakesh	13s	Michal Charvát	⋮
da7mlh3sq5js7...	Completed	open-instance	Job	8/27/26, 12:37 AM	8/27/26, 12:38 AM	ibm_marrakesh	13s	Michal Charvát	⋮
da7mlbjsq5js73...	Completed	open-instance	Job	8/27/26, 12:37 AM	8/27/26, 12:37 AM	ibm_marrakesh	13s	Michal Charvát	⋮
da7ml66sidac7...	Completed	open-instance	Job	8/27/26, 12:37 AM	8/27/26, 12:37 AM	ibm_marrakesh	13s	Michal Charvát	⋮
da7ml0k6l22...	Completed	open-instance	Job	8/27/26, 12:36 AM	8/27/26, 12:37 AM	ibm_marrakesh	13s	Michal Charvát	⋮
da7mkqmsidac...	Completed	open-instance	Job	8/27/26, 12:36 AM	8/27/26, 12:36 AM	ibm_marrakesh	13s	Michal Charvát	⋮
da7mkl46l22c7...	Completed	open-instance	Job	8/27/26, 12:36 AM	8/27/26, 12:36 AM	ibm_marrakesh	13s	Michal Charvát	⋮
da7mkfc6l22c7...	Completed	open-instance	Job	8/27/26, 12:35 AM	8/27/26, 12:36 AM	ibm_marrakesh	13s	Michal Charvát	⋮
da7mk9jsq5js7...	Completed	open-instance	Job	8/27/26, 12:35 AM	8/27/26, 12:35 AM	ibm_marrakesh	13s	Michal Charvát	⋮
da7mk360ukec...	Completed	open-instance	Job	8/27/26, 12:34 AM	8/27/26, 12:35 AM	ibm_marrakesh	13s	Michal Charvát	⋮
da7mjtk6l22c7...	Completed	open-instance	Job	8/27/26, 12:34 AM	8/27/26, 12:34 AM	ibm_marrakesh	13s	Michal Charvát	⋮
da7mjo3sq5js7...	Completed	open-instance	Job	8/27/26, 12:34 AM	8/27/26, 12:34 AM	ibm_marrakesh	13s	Michal Charvát	⋮
da7mjhrsq5js73...	Completed	open-instance	Job	8/27/26, 12:33 AM	8/27/26, 12:34 AM	ibm_marrakesh	13s	Michal Charvát	⋮
da7mj7u0ukec7...	Completed	open-instance	Job	8/27/26, 12:33 AM	8/27/26, 12:33 AM	ibm_marrakesh	13s	Michal Charvát	⋮
da7mj0msidac7...	Completed	open-instance	Job	8/27/26, 12:32 AM	8/27/26, 12:33 AM	ibm_marrakesh	13s	Michal Charvát	⋮
da7m493sq5js7...	Completed	open-instance	Job	8/27/26, 12:01 AM	8/27/26, 12:21 AM	ibm_marrakesh	13s	Michal Charvát	⋮
da7m0uusidac7...	Completed	open-instance	Job	8/26/26, 11:54 PM	8/26/26, 11:54 PM	ibm_marrakesh	13s	Michal Charvát	⋮
da7lnte0ukec73...	Completed	open-instance	Job	8/26/26, 11:34 PM	8/26/26, 11:34 PM	ibm_marrakesh	3s	Michal Charvát	⋮

Conclusion

The primary objective of this case study was to evaluate whether **Stellora.AI Quantum Flow can transform a heterogeneous collection of raw scientific literature, quantum-theory material, algorithm-specific studies, and version-specific SDK documentation into executable quantum-computing code capable of reaching a real quantum processing unit**. The objective was therefore not primarily to demonstrate an optimized VQE ground-state result, quantum advantage, or a scientifically significant energy value. Instead, the experiment focused on the complete research-to-execution path**: source documents → embedding and retrieval → generated quantum implementation → hardware-aware transpilation → IBM Quantum execution → measurable QPU output**.

From this perspective, the experiment achieved its principal objective. Quantum Flow generated a complete Python/Jupyter implementation from the supplied corpus and comparatively concise user prompt. The generated program constructed a parameterized six-qubit VQE ansatz, defined a demonstrative six-qubit Hamiltonian, authenticated against IBM Quantum at runtime, selected the requested **ibm_marrakesh backend**, transpiled the circuit for that hardware, invoked the IBM Runtime Estimator, submitted multiple jobs to the physical QPU, obtained expectation values, and passed these values back to a classical optimizer. The IBM Quantum Platform job history corroborates the notebook output by showing the corresponding jobs reaching the selected backend and completing successfully.

Six logical qubits and the 156-qubit transpiled representation

An initially unexpected result was that the original six-qubit circuit was reported after transpilation as having 156 qubits. This does not mean that the experiment became a 156-qubit VQE computation or that all 156 physical qubits participated actively in the variational calculation.

The original ansatz remained a six-qubit logical circuit, as demonstrated by the six logical wires q_0 through q_5 and by the original Hamiltonian width of six qubits. The selected ibm_marrakesh backend, however, exposes 156 physical qubits. During hardware-aware transpilation, Qiskit maps the virtual qubits of the input circuit onto physical qubit locations of the target processor and can represent the resulting circuit over the complete physical width of the backend [12], [14].

Consequently, the following distinction must be maintained:

Logical VQE problem size: 6 qubits.

Physical capacity of the selected backend: 156 qubits.

Width of the transpiled hardware representation: 156 qubits.

Number of logical qubits carrying the intended VQE computation: 6.

The generated circuit output shows the logical qubits being mapped to physical locations beginning with q_0 -> 0 through q_5 -> 5. The remaining physical positions are part of the backend-level representation rather than an expansion of the underlying VQE problem itself.

The increase of the circuit representation from six to 156 qubits is itself consistent with hardware-aware transpilation for the selected 156-qubit target and does not imply that the VQE problem became a 156-qubit calculation. Post-run inspection, however, identified a separate issue in the handling of the observable. The generated “align_observable_to_width()” function expanded each six-qubit Pauli label by appending identity operators until a width of 156 qubits was reached. In Qiskit's Pauli-string convention, qubit 0 corresponds to the right-most character of the Pauli label [19]. Consequently, appending identities to the right changed the physical qubit positions on which the original non-identity Pauli operators acted.

In this experiment, the displayed transpiled circuit mapped logical qubits q_0 through q_5 to physical qubits 0 through 5, whereas the generic padding procedure did not explicitly apply this recorded layout to the observable. The appropriate hardware-aware procedure is to transform the original observable using the physical layout recorded by the transpiled circuit, for example by using “Qiskit's SparsePauliOp.apply_layout()” method [18]. This finding does

not invalidate the evidence that real QPU jobs were successfully submitted and completed, but it means that the returned expectation values cannot be treated as validated measurements of the intended six-qubit Hamiltonian.

Interpretation of the optimization result

The classical optimization stage did not converge, but post-run analysis indicates that the limited optimizer budget was not the only relevant factor. A more fundamental issue was identified in the transformation of the six-qubit observable to the 156-qubit hardware representation.

The generated implementation expanded the Pauli labels by appending identity operators rather than applying the physical layout recorded during transpilation. Because Qiskit associates qubit 0 with the right-most Pauli-string character, this procedure did not preserve the original observable on the physical qubits carrying the transpiled ansatz. The runtime output shows the six logical circuit qubits mapped to physical qubits 0 through 5; consequently, the Estimator evaluations cannot be regarded as validated measurements of the intended Hamiltonian acting on those six qubits.

The returned objective values are also consistent with this interpretation. For the demonstrative Hamiltonian generated in this experiment, the expectation value of its diagonal Z and ZZ terms on ideal idle $|0\rangle$ qubits is 0.62, while its XX terms contribute zero. The measured objective values cluster closely around this value, approximately between 0.618 and 0.631. This observation is consistent with an objective that was not correctly coupled to the parameterized six-qubit ansatz, although the experiment alone does not isolate every contribution from finite-shot statistics, hardware noise, and runtime processing. The reported values should therefore be treated as execution diagnostics rather than evidence of VQE convergence or physically meaningful ground-state energies.

A second limitation was the COBYLA evaluation budget. The generated EfficientSU2 ansatz contained 36 optimization parameters, while the implementation requested maxiter=15. During execution, SciPy reported that the COBYLA function-evaluation budget must be at least num_vars + 2, resulting in 38 objective evaluations before termination with Optimization success: False. This budget was insufficient for a meaningful optimization study; however, increasing the evaluation budget alone would not resolve the observable-layout issue. Correct observable mapping is required before optimizer convergence can be meaningfully evaluated.

Experimental limitations

Several limitations of this first experiment are intentionally retained because they are directly relevant to evaluating AI-assisted quantum-code generation. Most importantly, the user prompt did not provide an exact scientific six-qubit Hamiltonian. Quantum Flow explicitly recognized this absence and generated a demonstrative placeholder Hamiltonian. Consequently, the reported objective values were never intended to represent the ground-state energy of a specified molecule, material, or other physical system.

Post-run validation additionally identified that the generated observable-expansion routine did not apply the transpiler's recorded logical-to-physical layout. Although the resulting code was accepted by the IBM Runtime Estimator and successfully produced real QPU jobs, successful execution did not guarantee that the intended observable was evaluated on the physical qubits carrying the parameterized ansatz. This demonstrates an important distinction between operationally executable quantum code and a scientifically validated quantum-computing implementation.

The generated ansatz also used the deprecated EfficientSU2 class, despite the experimental prompt explicitly requesting that deprecated or incompatible API patterns not be used. The class remained executable in the tested environment, but the runtime warning represents a documented deviation from that requirement and another implementation detail requiring validation.

These limitations do not negate the demonstrated research-to-execution capability of the workflow. They instead define the boundary of the result: the experiment demonstrates corpus-grounded code synthesis, quantum-circuit generation, backend-aware transpilation, authentication, and physical-QPU job execution, while complete scientific validity of the generated VQE calculation requires further correction and verification.

Next steps

The immediate next step is to correct the observable transformation by applying the transpiled circuit's recorded logical-to-physical layout to the original SparsePauliOp rather than expanding the operator through generic string padding. The transformed observable should then be inspected to confirm that its non-identity terms act on the same physical qubits selected for the six logical circuit qubits.

Before repeating a resource-intensive QPU optimization, the corrected circuit-observable combination should be validated using a simulator or another classical reference calculation to confirm that changes in the variational parameters produce the expected changes in the objective function. A subsequent scientific experiment should then replace the demonstrative Hamiltonian with a precisely specified physical Hamiltonian for which an independent classical reference value can be calculated.

Once these conditions are satisfied, the optimization experiment can be repeated with an appropriate evaluation budget and compared across COBYLA and SPSA. Future runs should additionally record the exact transpiler layout, physical qubit selection, circuit depth, two-qubit gate count, installed SDK versions, shot counts, optimizer configuration, IBM Quantum job identifiers, backend calibration information where available, random seeds where applicable, and repeated final measurements.

Overall conclusion

The central result of this case study is not the numerical minimum returned by COBYLA. The more significant result is that a corpus assembled from scientific publications, converted PDF material, quantum-theory references, VQE and optimization studies, IBM Quantum documentation, and version-specific Qiskit SDK sources was transformed by Stellora.AI Quantum Flow into executable Python code that constructed a non-trivial six-qubit parameterized quantum circuit, authenticated against the IBM Quantum Platform, transpiled the circuit for a specified physical backend, and submitted repeated workloads that completed successfully on a real IBM Quantum processor.

Post-run validation also exposed important limitations in the generated implementation, most notably an incorrect observable-layout transformation, use of a deprecated ansatz API, and an insufficient optimization budget. The observable-layout issue means that the returned objective values cannot be considered validated VQE energies for the intended six-qubit Hamiltonian. Accordingly, the experiment demonstrates successful **research-to-execution synthesis and physical QPU execution**, but not yet a scientifically validated end-to-end VQE calculation.

This distinction is itself relevant to the purpose of the study. AI-assisted quantum-code generation must be evaluated not only by whether generated code executes, but also by whether its scientific semantics remain correct after backend mapping, transpilation, observable transformation, and optimization. The issues identified during this experiment therefore provide concrete inputs for the troubleshooting and refinement capabilities of Stellora.AI Quantum Flow and define the next iteration of the experiment.

Accordingly, this first case study supports a narrower and directly verifiable conclusion: Stellora.AI Quantum Flow successfully transformed a controlled corpus of scientific and technical documents into a hardware-executable six-qubit variational quantum workflow and completed real IBM Quantum QPU executions. Further troubleshooting and validation are required before the generated workflow can be considered a scientifically validated VQE implementation.

References

[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. The bibliographic details are independently consistent with the Wiley/CERN record for the 1985 edition.

[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. Oxford sources identify Binney and Skinner as the authors and Oxford University Press as the publisher.

[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. Cambridge University Press lists the paperback edition as published in December 2011 with ISBN 9781107602762.

[6] S. P. Jordan, “Quantum Algorithm Zoo,” Quantum Algorithm Zoo, online resource. Available: https://quantumalgorithmzoo.org/. Accessed: Aug. 10, 2026. The site describes itself as a comprehensive catalog of quantum algorithms and provides the associated spj.jordan@gmail.com contact.

[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. The article explicitly evaluates stochastic optimization methods in applications including VQE.

[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. The final journal version was published in AVS Quantum Science in 2022; the ResearchGate record also contains the earlier 2021 preprint.

[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, 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: Aug. 31, 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: Aug. 31, 2026.

Disclaimer and Usage Limitations

Results and user experience with Stellora.AI Quantum Flow may vary depending on the quality, completeness, relevance, and scientific validity of the input dataset; the accuracy and specificity of the scientific problem definition; the clarity of the user request; the selected quantum algorithm; and the documentation available for the intended quantum-computing platform.

Quantum Flow is designed to use the supplied scientific material, validated quantum knowledge, target-platform documentation, and other provided sources as the primary basis for generating and refining quantum-computing implementations. Consequently, incomplete, outdated, contradictory, incorrectly interpreted, or scientifically invalid source material may directly affect the quality, correctness, compatibility, and scientific relevance of the generated output.

The quality of the generated implementation may also depend on whether the user provides sufficient scientific information to define the intended problem. Parameters such as the Hamiltonian, mathematical model, observables, ansatz requirements, optimization strategy, initial conditions, backend constraints, or other domain-specific inputs may need to be explicitly supplied when they cannot be reliably established from the available dataset.

Successful generation of executable code or successful execution on a simulator or quantum processing unit does not by itself establish scientific correctness, convergence, quantum advantage, or validity of the resulting scientific interpretation. Generated code, circuits, optimization results, expectation values, and other outputs should be reviewed and validated against the requirements of the particular experiment before they are used for scientific conclusions, publication, production workloads, or other consequential applications.

The experience may further vary according to the user's level of expertise in quantum computing, physics, mathematics, optimization, software development, and the target quantum platform. Quantum Flow is intended to assist and accelerate research-to-execution workflows, but appropriate domain expertise remains important for selecting scientifically meaningful inputs, evaluating generated implementations, interpreting experimental results,

and determining whether further validation or refinement is required.

Results can additionally be affected by factors outside the direct control of Quantum Flow, including SDK and API versions, changes in vendor interfaces, dependency versions, backend availability, QPU topology, calibration state, hardware noise, queue times, shot counts, transpilation configuration, optimizer parameters, random initialization, execution limits, network connectivity, account permissions, service-plan restrictions, model and embedding configurations, and other software, hardware, infrastructure, or operational variables.

Quantum-computing platforms and their SDKs evolve rapidly. Users should therefore provide documentation that corresponds as closely as possible to the exact SDK, runtime environment, backend, and hardware configuration intended for execution and should independently verify compatibility where appropriate.

If an issue is encountered during code generation, transpilation, dependency resolution, backend configuration, execution, optimization, or result processing, Stellora.AI Quantum Flow supports iterative troubleshooting and refinement. Users may provide the encountered error message, runtime output, unexpected result, platform response, or other relevant diagnostic information back to Quantum Flow so that the implementation can be analyzed, corrected, adapted, or further refined using the available scientific and technical context.

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.