NVIDIA Ecosystem Award

Team name: Team 17
Link to github repository that contains your challenge code:
https://github.com/andxeg/iquhack_2026_iqm/tree/main

Challenge you completed during iQuHACK: IQM Challenge — *Witness My Entanglement!*

Problem statement

We aimed to demonstrate, using experiments on IQM hardware, that entanglement must be present in prepared states of the QPU. We focused on (1) rigorous entanglement witnesses with clear separable bounds, (2) showing entanglement across qualitatively different state families, and (3) pushing the approach to as many qubits as feasible, while diagnosing what prevents scaling.

What was the "hard part"?

The hard part is that entanglement witnesses are unforgiving to noise: routing overhead (SWAPs), weak couplers, and readout errors quickly destroy witness margins as the number of qubits increases. We had to combine hardware-aware qubit selection + compilation with measurement designs that minimize circuit count while still producing a real "certificate" (crossing a known separable threshold).

Core technical approach (what we built)

We implemented two complementary witness pipelines:

1. Multipartite GHZ fidelity witness (few-circuit, scalable measurement)

- Prepare GHZ using a nearest-neighbor entangling chain mapped onto a connected hardware path.
- Measure in Z basis to estimate GHZ populations and in X basis (Hadamards before measurement) to estimate global coherence (parity).
- Combine these into a GHZ fidelity estimate and use the separable bound $F \leq 0.5$as the certificate: if measured $F > 0.5$, entanglement must be present.
- In our repo results, we demonstrate this GHZ witness on 25 qubits with fidelity $F = 0.68$learly above the 0.5 bound).

2. IQP / graph-state stabilizer-style witness (flexibility across state family)

- Prepare an IQP-style state (Hadamards + entangling CZ structure + measurements).

- Estimate stabilizer-like correlators per qubit and aggregate them into a witness score / fidelity-style metric with a 0.5 threshold for separability.
- In our repo results, we demonstrate this IQP witness on 15 qubits with measured fidelity $F = 0.7777$.

Hardware-aware optimization (how we improved results)

- Calibration-driven selection: We use device topology and per-qubit/per-edge quality to avoid bad qubits/couplers.
- Custom mapping vs baseline: We compare an "optimal path / guided layout" approach against a naive transpile baseline to show the effect of routing overhead.
- Mitigation where it matters: We include readout-error mitigation steps for measurements (where applicable in the notebooks) to reduce obvious measurement bias.

Scaling + diagnosis

We also ran larger-n GHZ experiments on Emerald to stress-test scaling and compare "optimal vs naive" compilation. These runs are used primarily to identify bottlenecks (coherence + depth growth), not necessarily to claim a witness proof at the largest n.

NVIDIA tools + hardware

How did you use NVIDIA software tools (CUDA, CUDA-X, CUDA-Q, cuQuantum, CUDA-Q QEC, CUDA-Q Solvers)?
We did not use NVIDIA software tools in the final solution implementation. Our execution and analysis pipeline is based on IQM Resonance + Qiskit/Qrisp + iqm-client, with simulation support from Qiskit Aer.

What (if any) NVIDIA hardware did you use?
We used an NVIDIA GPU to accelerate hybrid (classical) post-processing, including: (1) readout-error mitigation (M3), (2) circuit cutting and stitching/knitting post-processing, and (3) graph-based optimization used for mapping/qubit selection.

Why did you select these tools?
We selected Qiskit/Qrisp + IQM tooling because it provided the quickest path to: (1) submit jobs to IQM hardware, (2) control transpilation/layout decisions, and (3) implement witness-specific measurement circuits and post-processing.

How did the tool(s) benefit you during iQuHACK?
They enabled fast iteration on witness definitions, hardware-aware mapping, and repeatable measurement/post-processing for GHZ and IQP state families.

CUDA-Q alongside IQM tools — interaction/friction?
We did not integrate CUDA-Q into the final pipeline. (Our repo includes IQM “connecting” materials for multiple SDKs, but our solution notebooks run via Qiskit/Qrisp.)