AI Report

TEAM QAT

February 1, 2026

Hybrid Generative Quantum-Enhanced Memetic Tabu Search for LABS

Team Name: QAT
GitHub Repository: `https://github.com/rnsln/iquhack2026-nvidia`
Github Profiles:
Hatice Boyar: https://github.com/dhaticeboyar
Eren Aslan: https://github.com/rnsln
Hüseyin Umut Işık: https://github.com/HUmutI
Chang Jen Yu: https://github.com/leo07010
Ilayda Dilek : https://github.com/ilayd-a

1 The Workflow

Our development process utilized a Dual-Layer AI Strategy, distinguishing between Development Agents (Tooling) that accelerated our coding and learning the problem stage velocity, and Runtime Agents (Embedded AI) that executed the core physics logic.

- **Development Agent(Tooling):** We organized our coding agents into a strict Human-in-the-Loop Sandwich Workflow (AI $\rightarrow$ Unit test $\rightarrow$ Hand $\rightarrow$ AI) to balance coding and learning speed with architectural correctness.With this method, we can easily generate, control and discard the code for reaching the optimal code that we use in the challenge. Besides this advantage, we also utilize AI as a learning tool. We use the AI to learn challenge and complete gaps in our knowledge. These two combination makes the AI an accelerator in this project. Also AI help the documentation of this project by creating templates, giving advice on grammatical issues, etc.

We use Google Gemini, Antigravity, NotebookLLM, ChatGPT, Cursor, and CUDO Compute as an AI agent in this project. Google Gemini, NotebookLLM, ChatGPT used as a learning tool, and the CUDO Compute, Google Gemini, Antigravity, Cursor, and ChatGPT used as a coding tool. Also, we used Mermaid AI to create our project's flowchart.

- **Runtime Agent(Embedded AI):** Beyond development, our architecture integrates Generative AI directly into the execution loop as a Generative Quantum Eigensolver (GQE) augmented by Generative AI for circuit synthesis. This embedded intelligence replaces traditional variational parameter optimization, which often fails in high-dimensional landscapes due to the "Barren Plateau" problem.

  - **Physics Aware Problem Pool:**Unlike standard quantum chemistry solvers, our runtime agent treats LABS as a distribution problem. We construct the sampling space using specific two-body and four-body operators derived from quantum annealing principles and the problem's interaction graph.
  - **Basin Attraction:** The ultimate goal of this embedded AI is not immediate perfection but locating the Basin of Attraction. By concentrating the wavefunction on low-energy states, the GQE provides high-quality seeds that allow the classical MTS phase to find the Global Minimum extremely fast.
  - **Hierarchical Parameter Strategy:** The embedded agent manages circuit execution using a decaying parameter strategy, moving from π to $\pi/8$. This allows for coarse tuning via large geometric rotations first, followed by fine-tuning through quantum interference adjustments

2 Verification Strategy

To ensure the integrity of the Hybrid GQE-MTS solver, we implemented a multi-layered verification strategy. This strategy guards against AI hallucinations during code generation and ensures that the Quantum Engine, which consists of GQE and Transfer Learning, respects the fundamental physical laws of the LABS problem. All code is given in test.py in the documentation of the project.

- **Physical Correctness Invariants:** We utilized the LABSPhysics class to enforce mathematical consistency throughout the optimization loop.

  - **Hamiltonian Validity:** We verify that the built Hamilton function correctly generates only Pauli-Z terms, as the LABS problem is mapped to an Ising-like spin configuration.
  - **Symmetry Checks:** The system is programmed to verify the reversal symmetry ($E(S) = E(S_{reversed})$) and the negative symmetry ($E(S) = E(-S)$).

Any implementation of the compute_energy function that fails these invariants is flagged as a logic error.

- **Ground Truth Calibration:** Before executing large-scale production runs to reach the global minimum, we utilize the test.py suite as calibration code to verify solver accuracy against known physical benchmarks. This verification phase includes the followings:
  - **Small N:** For N=7 the system must return a global minimum of $E = 3$.
  - **Medium N:** For $N = 20$, the MTS Optimizer must target the known minimum of $E = 16$
  - **Performance Test:** We compare the h_rand (Random Seed) vs. h_quant (Quantum Seed) convergence histories to verify that the GQE-provided efficient seeds actually provide a statistical advantage in convergence speed.
- **Automated Unit Testing & Hallucination Guardrails:** Our workflow utilizes Test-Driven Prompting. We provide the test.py structure to the AI agents before requesting logic implementation
  - **Logic Error Detection:** We use unit tests to catch instances where the AI might hallucinate $O(N^2)$ energy calculations instead of the optimized Delta-Evaluation ($O(N)$) required for the MTS phase.
  - **State-Vector Validation:**Because we target the NVIDIA backend, our tests include checks for state-vector consistency to ensure the AI has not hallucinated deprecated cudaq syntax or incompatible MPI decomposition calls
- **Execution Timing & Benchmarking:** The `test.py` script includes an Execution Time Report. This allows us to verify if we are meeting our Success Metric of achieving global minimums for different N values.
  - **GQE Training Latency:** Monitored to ensure the Transfer Learning overhead does not exceed the time saved during the MTS phase.
  - **Sampling Efficiency:** Validates that the `shots_count` (e.g., 2000) is sufficient to populate the MTS population with high-quality seeds without stalling the pipeline.

3 The "Vibe" Log

3.1 Win

The most significant win occurred during the optimization of the Classical Memetic Search (MTS) phase. Initially, the local search process was a computational bottleneck because the AI-generated code utilized a standard $O(N^2)$ approach to recalculate the full autocorrelation energy for every potential bit flip.

By utilizing the CUDO Compute agent for specialized refactoring, the AI identified and implemented a Delta-Evaluation Optimization. Instead of a total recalculation, the logic was shifted to compute only the energy change (ΔE) induced by a single bit-flip, successfully reducing the complexity from $O(N^2)$ to $O(N)$. This optimization transformed a process that previously took hours of execution and debugging into a streamlined routine, serving as the critical factor in enabling our system to achieve the global minimum in less time.

3.2 Learn

Initially, our interaction with the coding agents resulted in significant technical debt because the agents defaulted to deprecated quantum programming syntaxes or generic Python-based simulators. To resolve this, we altered our prompting strategy by implementing a Context Injection method. We created skills.md file that strictly defined the requirements for the CUDA-Q 0.13.0 environment and the NVIDIA target backend. By providing this specialized context, we shifted the AI's output from generic code to hardware-aware implementations. This allowed the agents to successfully generate complex circuit optimization and correctly manage MPI decomposition via nvidia-mgpu for state-vector simulations where $N > 30$. This transition from "blind prompting" to "skill-based context provisioning" ensured that the AI-generated code was stable enough to handle the high-dimensional landscapes of the LABS problem without constant manual refactoring.

3.3 Fail

- **The Circuit Generation Hallucination:** During the initial attempt to implement Phase 1: Quantum Generative Optimization (GQE Phase), the AI agent failed significantly by generating random quantum gates that bore no relation to the physical structure of the problem. Instead of utilizing the specific 2-body and 4-body operators derived from the interaction graph, the agent hallucinated a standard hardware-efficient ansatz. This resulted in an optimized quantum state that failed to concentrate the wavefunction on low-energy states, as the gates did not represent the valid physical evolution of the LABS Hamiltonian. Hand check corrected the circuit generation logic, ensuring that every quantum step was physics-aware and efficient for the state-vector simulation

- **The Non-Physical Operator Hallucination:** During the development of our Quantum Engine, the AI agent attempted to generate a custom operator pool that was physically impossible to implement within the constraints of quantum mechanics. Specifically, it hallucinated non-unitary operators and non-Hermitian Hamiltonian terms, suggesting "imaginary phase rotations" that violated the requirement for energy to be a real-valued observable. The agent also attempted to create many-body operators that spanned non-adjacent qubits without accounting for the actual connectivity of the NVIDIA backend, resulting in a circuit that could

not be physically synthesized.
We identified these non-physical hallucinations during the Verification Loop when the cudaq compiler rejected the non-unitary gate definitions. We fixed this by manually defining the LABSPhysics and QuantumEngine classes to strictly enforce the use of Pauli-Z-based spin operators. We then implemented a Symmetry invariant check in our unit tests to ensure that every generated operator preserved the physical properties of the Ising-like spin configuration. This forced the AI to stay within the bounds of physical reality, ensuring our GQE Phase could actually be executed on the target hardware.

3.4 Context Dump

This section provides the prompts and technical artifacts that governed our Dual-Layer AI Strategy. These logs demonstrate how we enforced physical laws and hardware constraints on the generative process.

- **The Skills Artifact (skills.md):** We injected **skills.md** specific context into the system prompt of every coding agent to ensure hardware optimization and project compliance from the first line of code.

- **The Architect Prompt:** To fix the initial failure where the AI generated non-physical circuits, we utilized a "Physics-Aware" prompt. For example, as a prompt, "Decompose the Generative Quantum Eigensolver (GQE) into a modular interaction-graph-based ansatz. Instead of a hardware-efficient approach, construct the operator pool using exclusively 2-body and 4-body Pauli-Z interactions (G_2 and G_4) derived from the LABS Hamiltonian. Apply a hierarchical parameter schedule where rotations decay from π to $\pi/8$ to avoid barren plateaus. Ensure every operator is unitary and Hermitian—reject any hallucinations of imaginary phases or non-physical gate types." was used to generate a circuit for obtaining less physical error in the generated output.

- **The Verifier Protocol:**This protocol was used to automate the Human-in-the-Loop "Sandwich" Workflow, forcing the AI to audit its own code against our unit tests. For example, as a prompt, "Analyze the following CUDA-Q kernel. Before finalizing, run a mental execution against our tests.py suite" shows that the prompt strategy to obtain results that pass the filter.

- **Resource Protection** To prevent the AI from hallucinating high-level cloud API calls, we provided this explicit implementation template for resource management. For example, as a prompt, "Write a raw shell-integrated Python script that queries nvidia-smi directly. If GPU utilization remains below 5% for a 600-second window, execute the termination command. This is our primary Zombie Prevention strategy to protect project credits," was used to get rid of zombie

References

[1] Nakaji, Kouhei, et al. *The generative quantum eigensolver (GQE) and its application for ground state search.* arXiv preprint arXiv:2401.09253 (2024).

[2] Gomez Cadavid, A., Chandarana, P., Romero, S. V., et al. *Scaling advantage with quantum-enhanced memetic tabu search for LABS.* arXiv preprint arXiv:2511.04553 (2025).

[3] Packebusch, T., and Mertens, S. *Low autocorrelation binary sequences.* Journal of Physics A: Mathematical and Theoretical, 49(16), 165001 (2016). DOI: 10.1088/1751-8113/49/16/165001.

[4] Tilly, J., Chen, H., Cao, S., et al. *The Variational Quantum Eigensolver: A review of methods and best practices.* Physics Reports, 986, 1–128 (2022). DOI: 10.1016/j.physrep.2022.08.003.

[5] Farhi, E., Goldstone, J., and Gutmann, S. *A Quantum Approximate Optimization Algorithm.* arXiv preprint arXiv:1411.4028 (2014). URL: https://arxiv.org/abs/1411.4028.

[6] Larocca, M., Thanasilp, S., Wang, S., et al. *Barren plateaus in variational quantum computing.* Nature Reviews Physics, 7(4), 174–189 (2025). DOI: 10.1038/s42254-025-00813-9.

[7] Zhuang, F., Qi, Z., Duan, K., et al. *A Comprehensive Survey on Transfer Learning.* Proceedings of the IEEE, 109(1), 43–76 (2020). arXiv preprint arXiv:1911.02685.