VISTA: Verifier-in-the-Loop Agentic Reinforcement Learning for Quantum Program Synthesis

Cong Yu
cong.yu@aalto.fi
Aalto University

Tuo Shi
tuo.shi@aalto.fi
Aalto University

Valter Uotila
valter.uotila@aalto.fi
Aalto University

Shilong Deng
shilong.deng@liverpool.ac.uk
University of Liverpool

Lei You
leiyo@dtu.dk
Technical University of Denmark

Bo Zhao
bo.zhao@aalto.fi
Aalto University

Abstract

Quantum program synthesis increasingly depends on external evaluators such as parsers, simulators, and optimizers. In OpenQASM 3.0 circuit generation, artifact quality is determined not by text plausibility but by staged execution against tool-defined quantum semantics. This makes verifier-in-the-loop training a systems problem: verifier stages differ sharply in cost, latency, and informativeness, so executing the full verifier on every candidate is inefficient, while collapsing all verifier outcomes into a single reward can destabilize learning.

We present VISTA, a verifier-in-the-loop agentic reinforcement learning (RL) system for quantum program synthesis, instantiated for OpenQASM 3.0 quantum circuit generation. VISTA introduces two mechanisms: (i) hierarchical verified reward optimization, which converts staged verifier outcomes into stable learning signals spanning feasibility, behavior, objective quality, and utility; and (ii) budget-aware gated evaluation, which schedules expensive verifier stages using partial evidence from earlier stages. VISTA outperforms four classes of baselines—frontier LLM agents, quantum-specific agents, RL post-training agents, and RL agentic tool-use agents. Across quantum optimization tasks, it achieves 1.13× higher executability at Pass@10, improves semantic solution quality by 1.10×, and cuts verifier cost by 1.77× under matched-budget evaluation.

ACM Reference Format:
Cong Yu, Tuo Shi, Valter Uotila, Shilong Deng, Lei You, and Bo Zhao. 2026. VISTA: Verifier-in-the-Loop Agentic Reinforcement Learning for Quantum Program Synthesis. In *ACM Conference on AI and Agentic Systems (CAIS '26), May 26–29, 2026, San Jose, CA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3786335.3813148

1 Introduction

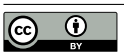
Quantum program synthesis increasingly depends on toolchains such as parsers, simulators, and optimizers. In OpenQASM 3.0 circuit generation, the key question is not whether a candidate circuit looks plausible, but whether it satisfies semantic requirements that only staged execution can validate. A generated circuit must parse, satisfy qubit and structural constraints, induce task-aligned behavior under simulation, and often remain useful for downstream refinement. Closing this generation–verification gap is therefore a core challenge for LLM-based quantum program synthesis.

For this workload, verifier-in-the-loop learning is also a systems problem. The verifier is an external, staged pipeline whose components differ widely in latency, cost, failure modes, and resource requirements, and may span local CPUs/GPUs as well as remote services. A practical system must therefore manage intermediate state, coordinate stage transitions, and decide when deeper verification is worth the added cost. Existing execution-feedback methods improve code generation when checks are cheap and decisive, e.g., through reinforcement learning (RL) from test outcomes or iterative self-reflection over tool interaction [25, 42]. However, they do not directly address settings such as quantum program synthesis, where verification dominates cost, later stages are substantially more expensive than earlier ones, and correctness depends on numerically sensitive task semantics rather than compile-and-test signals alone.

This paper asks: *How can we build a verifier-in-the-loop agentic RL framework for quantum program synthesis that is simultaneously stable, efficient, and semantically faithful under expensive verification?* We organize the design space through three questions: **what** semantic signals to optimize, **how** to shape them into stable learning objectives, and **when** to spend verification budget across a multi-stage verifier. In VISTA, the *what/how* questions are addressed through hierarchical verified rewards, while the *when* question is addressed through budget-aware verifier scheduling via cost-aware gated evaluation.

To make the execution structure explicit, we describe such a setting as synthesis under verifier-defined semantics: a policy proposes a candidate artifact, a staged verifier executes the corresponding evaluation pipeline and returns a structured report containing semantic signals, cost, and diagnostics, and learning optimizes rewards derived from that report rather than from text likelihood alone. In this paper, we instantiate this structure for quantum OpenQASM synthesis, where the verifier naturally exposes four domain-native semantic stages: *Feasibility*, *Behavior*, *Objective*, and *Utility*.

Quantum assembly code generation is a demanding setting because it combines hard semantic correctness with expensive verification. OpenQASM 3.0 is a vendor-neutral, hardware-facing representation used across modern quantum software stacks, and correctness is unforgiving: a generated circuit must parse, satisfy qubit constraints, and induce task-aligned behavior after simulation and often local optimization. The problem couples discrete structure with continuous parameters, so numerical errors can destroy objective quality even when the program compiles. Verification



CAIS '26, San Jose, CA, USA

ACM ISBN 979-8-4007-2415-2/2026/05
https://doi.org/10.1145/3786335.3813148

is also expensive in practice: IBM Quantum usage starts at $96 per minute [18], while AWS Braket charges substantially higher reservation-mode rates on real QPUs (e.g., IonQ Aria at $7,000 per hour) [2]. These costs make "verify everything" unrealistic and turn verifier scheduling into a first-class systems concern.

We address these questions through the following contributions.

(1) Verifier-in-the-loop RL for quantum program synthesis We present Vista, a verifier-in-the-loop agentic RL framework for OpenQASM-based quantum program synthesis. Vista makes staged quantum verification explicit in the training loop through structured verifier reports, semantic signals, and explicit verification budgets (§2 and §3).

(2) Hierarchical verified reward optimization We design a staged reward that converts multi-stage verifier outcomes into stable learning signals spanning *Feasibility*, *Behavior*, *Objective*, and *Utility*. This preserves the semantic structure of the verifier while making its outputs usable for RL post-training (§4).

(3) Budget-aware verifier scheduling We formulate verifier execution as a budgeted stage-scheduling problem and implement a cost-aware gated policy that selectively invokes expensive simulation and optimization stages based on partial verifier outcomes. This turns staged verification into a practical systems mechanism for reallocating budget toward more informative candidates (§5).

(4) Quantum OpenQASM instantiation and evaluation We instantiate Vista on OpenQASM-based quantum optimization tasks and evaluate it against four baseline classes: frontier LLM agents, quantum-specific agents, RL post-training agents, and RL tool-use agents. The evaluation covers end-to-end effectiveness and efficiency, ablations, scaling, and real QPU hardware deployment (§6).

Across quantum optimization tasks, Vista achieves 1.13× higher executability at Pass@10, improves semantic quality by 1.10×, and cuts verifier cost by 1.77× under matched-budget evaluation.

2 Problem Setting and Application Context

We now formalize the setting introduced in §1 and instantiate it for quantum circuit synthesis in OpenQASM 3.0.

2.1 Problem Setting and Execution Model

We study synthesis settings in which artifact quality is defined by an external verifier. Here, a synthesis instance is represented as $x_{\text{inst}} \triangleq (\texttt{spec}, \texttt{constraints}, \texttt{semantics}, \texttt{budget})$, where `spec` is the user-level task description, `constraints` are machine-checkable requirements, `semantics` specifies how task behavior is evaluated, and `budget` bounds verification cost.

Given an instance x_{inst}, a policy π_θ produces a candidate artifact $a_{\text{cand}} \sim \pi_\theta(\cdot \mid x_{\text{inst}})$. An external verifier V then evaluates $(x_{\text{inst}}, a_{\text{cand}})$ and returns a structured verifier report $\rho_{\text{ver}} \triangleq (\texttt{stage}, \texttt{signals}, \texttt{cost}, \texttt{diagnostics})$, where `stage` indicates how far evaluation progressed, `signals` are semantic measurements produced by the verifier, `cost` records verification effort, and `diagnostics` explain failure modes such as parser or constraint violations. Learning then optimizes the policy against a terminal reward derived from the verifier report: $R_{\text{ver}}(x_{\text{inst}}, a_{\text{cand}}) \triangleq R(\rho_{\text{ver}})$.

This interface makes the execution structure explicit. The setting is naturally verifier-in-the-loop and agentic: policy improvement depends on how the system coordinates generation, verifier execution, and reward construction. Vista instantiates this loop by coupling an LLM policy to a staged quantum verification pipeline and optimizing against rewards derived from verifier reports.

2.2 Systems Challenges

The challenge is that the verifier-defined synthesis requires a heterogeneous execution pipeline. First, verification is typically *multi-stage* with heterogeneous costs. A candidate may fail immediately on cheap checks (e.g., parsing or resource constraints), or only later after expensive simulation, objective evaluation, or downstream optimization. Verification therefore forms a staged pipeline whose stages differ sharply in latency, cost, and informativeness.

Second, verifier feedback is usually *black-box* and delayed. Semantic quality often becomes clear only after full artifact execution, and the resulting signals may be noisy or expensive to obtain. This creates a systems problem: evaluating every candidate with the full verifier pipeline wastes budget, while collapsing all verifier outcomes into a single reward value can hurt training stability.

Addressing these challenges requires the design space (see §1): *what* semantic signals to optimize, *how* to shape them into stable learning objectives, and *when* to spend verification budget across stages. Vista makes this design space explicit by exposing stage, semantic signals, and cost through the verifier report ρ_{ver}.

2.3 Quantum Circuit Synthesis and Optimization

Vista instantiates the quantum program synthesis setting with the OpenQASM 3.0 circuits. OpenQASM is a vendor-neutral, hardware-facing representation that serves as a common interchange format across modern quantum software stacks [8, 10, 20, 34, 43]. In our setting, the synthesis input specifies a graph-optimization problem in natural-language form together with resource constraints and a task-specific semantic descriptor (e.g., generated circuits may be used for portfolio optimization [46] or quantum chemistry problems [33]), and the output is a complete OpenQASM circuit with associated parameters.

The semantic target is not merely syntactic validity. A generated circuit must parse, satisfy qubit and structural constraints, and ensure task-aligned behavior for the underlying optimization objective. Concretely, many of these tasks are Hamiltonian-based optimization problems [12] in which a circuit is assessed through the distribution over measured bitstrings, the resulting expectation value, and its usefulness as a warm start for downstream optimization. OpenQASM synthesis is therefore an instance in which artifact quality is inherently verifier-defined.

The quantum setting is also a natural stage structure for semantic verification. The dominant semantic stages include:

- **Feasibility:** whether the generated artifact parses and satisfies resource or structural constraints;
- **Behavior:** whether the induced output distribution aligns with the intended task behavior;
- **Objective:** whether the circuit achieves strong task-level semantic quality under the target objective;
- **Utility:** whether the generated circuit is useful for downstream optimization, e.g., as a warm start.

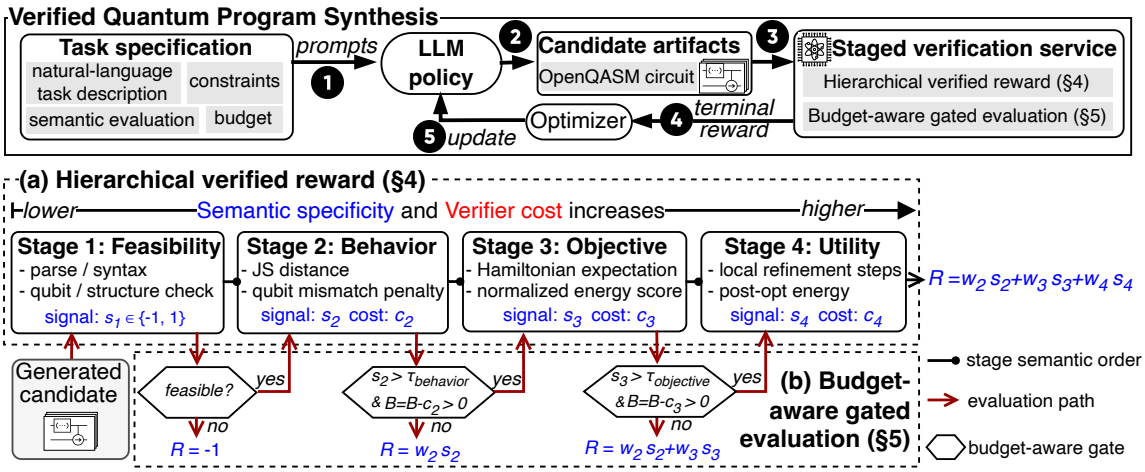


Fig. 1: VISTA design overview with two key mechanisms: (i) *Hierarchical verified reward*, which turns multi-stage verifier outcomes into progressive learning signals; and (ii) *Budget-aware gated evaluation*, which filters candidates to avoid unnecessary expensive verification.

These stages are not introduced as arbitrary reward terms; they arise directly from the failure modes and verifier stages of quantum optimization tasks [7, 32]. This stage structure provides the foundation for the hierarchical reward design in §4 and the budget-aware verifier scheduling policy in §5.

In this paper, we use *verification* to refer to the staged external evaluation pipeline used to assess generated quantum circuits. *Feasibility* is a strict executability and structural-validity check. If a quantum circuit fails the feasibility checking, it is directly rejected. By contrast, *Behavior*, *Objective*, and *Utility* are soft semantic criteria that assess distributional alignment, task-level quality, and downstream usefulness through numerical measurements rather than binary formal-correctness predicates. Thus, we use "verification" in a broad systems sense, closer to acceptability-oriented computing [39] and conditionally correct evaluation [41] than to classical full functional verification.

3 VISTA Design

VISTA is a verifier-in-the-loop agentic RL system for expensive-semantics synthesis in quantum circuit generation. It operationalizes a staged synthesis loop from task instance to candidate artifact, verifier report, and terminal reward. At a high level, VISTA combines two mechanisms: (i) *Hierarchical verified reward*, which determines what semantic signals are exposed to learning and how they are converted into stable reward signals; and (ii) *Budget-aware gated evaluation*, which determines when verification budget should be spent on deeper stages of the verifier pipeline.

Figure 1 shows the overall architecture and verifier-in-the-loop workflow. VISTA begins by constructing the synthesis setting from the input prompt (❶), which initializes an instance $x_{\text{inst}} \triangleq (\texttt{spec}, \texttt{constraints}, \texttt{semantics}, \texttt{budget})$. This instance specifies the task description (`spec`), machine-checkable constraints (`constraints`), semantic evaluation criteria (`semantics`), and the verification budget (`budget`). Conditioned on this instance, the policy model generates a set of OpenQASM 3.0 circuit candidates (❷).

The generated candidates are then dispatched to the staged verification service (❸), backed by quantum simulation or a physical quantum processing unit (QPU). Verification proceeds through VISTA's two core mechanisms. First, *Hierarchical verified reward* (§4) determines which semantic signals are exposed to learning and how multi-stage verifier outcomes are converted into terminal rewards (❹). In our quantum instantiation, these signals are organized into four semantic stages: feasibility, behavior, objective, and utility. Second, *Budget-aware gated evaluation* (§5) determines when candidates should advance to more expensive verifier stages, using early-stage evidence to stop weak candidates and reserve deeper evaluation for promising ones.

Finally, VISTA updates the policy model (❺) using the resulting verified rewards. In this way, VISTA closes the loop between candidate generation, staged semantic verification, and policy optimization under an explicit verification budget.

4 Hierarchical Verified Reward Optimization

This section investigates the *what* and *how* parts of the design space for quantum synthesis. We first specify the semantic signals of the verifier, then define the four stage-aligned verifier questions in our setting, and finally show how these staged outcomes are combined into a scalar reward for the agentic RL loop. The key design principle is to preserve the semantic structure exposed by the verifier, rather than collapsing all verifier outcomes into a single undifferentiated score.

4.1 Verifier Signals for OpenQASM Synthesis

VISTA's verifier returns a structured report ρ containing the current semantic stage, the corresponding semantic signals, the cost of obtaining those signals, and diagnostics for failure cases. As illustrated in Fig. 1(a), VISTA instantiates ρ with four stages: (i) *Feasibility*, which captures executability and constraint satisfaction; (ii) *Behavior*, which captures distributional alignment; (iii) *Objective*, which captures task-level semantic quality; and (iv) *Utility*, which captures usefulness for downstream optimization.

These signals differ in semantics, cost, and failure mode. We therefore organize reward construction hierarchically: earlier stages provide cheap, necessary feedback, while later stages provide richer

but more expensive semantic information. This ordering follows the verifier pipeline itself—*feasibility → coarse behavior → task-level objective quality → deployment utility*—and yields a minimal domain-native decomposition. Merging stages conflates distinct semantic roles, while introducing more stages would over-fragment the hierarchy.

4.2 Four-Stage Verifier Semantics

We now define the four semantic stages used in Vista. Each stage answers a distinct verifier question, and each later stage becomes meaningful only after earlier preconditions have been satisfied.

Stage 1: Feasibility. *Is the generated quantum circuit executable?* A circuit is feasible if it passes the OpenQASM 3.0 parser. If parsing fails, reward computation terminates immediately and returns a negative value. This stage is computationally cheap and filters invalid artifacts before deeper semantic evaluation. Formally, let $s_1(a) \in \{-1, 1\}$ denote the *Feasibility* signal, where $s_1(a) = 1$ if parsing succeeds and $s_1(a) = -1$ otherwise.

Stage 2: Behavior. *If feasible, does the program behave broadly like a good solution?* We capture coarse semantic alignment using the measurement distribution induced by the generated circuit. Prior work [21] used KL-based measures; here we use the Jensen–Shannon (JS) distance [28] because it is bounded and empirically more stable for RL. Let p and q denote the measurement distributions of the generated circuit a and reference circuit a', respectively. We define

$$d_{\mathrm{JS}}(p, q) = \sqrt{\frac{\mathrm{JS}(p \,\|\, q)}{\log 2}}, \tag{1}$$

where $\mathrm{JS}(p \,\|\, q) = \frac{1}{2} D_{\mathrm{KL}}(p \,\|\, m) + \frac{1}{2} D_{\mathrm{KL}}(q \,\|\, m)$ and $m = \frac{1}{2}(p + q)$. The resulting behavior score is

$$s_2(a) = 1 - d_{\mathrm{JS}}(p, q) \in [0, 1].$$

This stage provides a smooth semantic surrogate before exact task quality is known.

Stage 3: Objective. *Is the circuit good for the task itself?* Behavioral similarity alone is insufficient: two circuits may induce different distributions yet still have similar or different task quality. We therefore include an Objective-stage signal that measures task-level semantic quality directly through the cost Hamiltonian [12]:

$$J(a) := \langle \psi_a | H | \psi_a \rangle = \sum\nolimits_z p_a(z)\, E(z),$$

where a is the generated circuit, H is the task Hamiltonian, and p_a is the induced measurement distribution. For a feasible circuit, we compute E_{gen} and normalize it using the Hamiltonian eigenvalue bounds $E_{\min}$ and $E_{\max}$:

$$s_3(a) := 1 - \frac{E_{\mathrm{gen}} - E_{\min}}{E_{\max} - E_{\min}}. \tag{2}$$

This term measures task-level semantic quality more directly than distributional alignment.

Stage 4: Utility. *Is the circuit a good warm start for downstream refinement?* Even a semantically reasonable circuit may not be useful in practice [7]. In hybrid quantum–classical workflows [38], a common deployment pattern is to refine a generated circuit through local optimization. We therefore measure warm-start usefulness. Let n denote the number of optimization steps needed to reach the stopping criterion, and let E_{opt} denote the post-optimization expectation value. We define

$$s_4(a) := \frac{1}{1 + n} + \left(1 - \frac{E_{\mathrm{opt}} - E_{\min}}{E_{\max} - E_{\min}}\right). \tag{3}$$

The first term captures optimization efficiency, while the second captures post-optimization quality. Together, they measure downstream utility.

4.3 Reward Construction and Scalarization

Having defined the four semantic stages, we combine them into a single terminal reward for RL:

$$R(a) = \begin{cases} -1, & s_1(a) = -1, \\ w_2\, s_2(a) + w_3\, s_3(a) + w_4\, s_4(a), & s_1(a) = 1, \end{cases} \tag{4}$$

where $w_2, w_3, w_4 \geq 0$ are nonnegative weights controlling the relative contribution of the Behavior, Objective, and Utility stages.

This scalarization preserves the hierarchy. *Feasibility* acts as a hard gate on invalid artifacts, while the later three stages contribute progressively richer semantic feedback. The resulting reward is therefore a scalar training signal constructed from semantically ordered components that become meaningful only after earlier preconditions have been satisfied.

5 Budget-Aware Verifier Scheduling

This section instantiates the *when* part of the design space. After §4 defines the semantic stages, the remaining systems question is how to schedule them under a limited verification budget. Because later stages are more informative but also much more expensive, Vista introduces *budget-aware gated evaluation*, which uses partial verifier outcomes to decide whether a candidate merits deeper evaluation. By stopping weak candidates early, Vista reduces wasted simulator and optimizer invocations and reallocates the saved budget toward deeper semantic supervision for promising samples, improving RL training efficiency.

5.1 Verifier Scheduling as a Policy

Verification is staged. For each candidate artifact a, the verifier does not expose all semantic signals at once; instead, it produces progressively richer reports, denoted as $\rho = \{(s_i, c_i)\}_{i=1}^{m}$, where s_i is the semantic signal acquired at stage i and c_i is the associated computational cost. Early stages are typically cheap and structurally required, whereas later stages are more semantically informative but can dominate total verification cost. Verifier execution is therefore not a fixed evaluation routine, but a scheduling problem.

Verifier scheduling problem. Vista invokes a policy π to decide which stages to execute for each candidate artifact a, inducing a selected subset $\pi(a) \subseteq \{1, \ldots, m\}$ and a corresponding partial report $\rho' = \{(s_i, c_i)\}_{i \in \pi(a)}$. The goal is to maximize semantic utility across all candidates under a verification budget B:

$$\max_{\pi} \sum_{a} \sum_{i \in \pi(a)} u_i(s_i) \quad \text{s.t.} \quad \sum_{a} \sum_{i \in \pi(a)} c_i \leq B,$$

where u_i measures the utility of semantic feedback at stage i.

This problem is NP-hard, since 0–1 knapsack [23] reduces to it by treating each stage as an item with value u_i and weight c_i.

We therefore instantiate a lightweight scheduling policy for the quantum setting through progressive semantic screening.

5.2 Quantum Instantiation: Progressive Semantic Screening

We instantiate the scheduling policy for the four-stage quantum verifier (§4). The core idea is progressive semantic screening: every candidate first undergoes cheap structural checks, and only those with sufficiently promising early-stage semantics are escalated to more expensive stages.

As illustrated in Fig. 1, *Feasibility* is always executed because it is inexpensive and establishes the first semantic gate; invalid OpenQASM programs are rejected immediately. For feasible candidates, the verifier next computes the *Behavior* signal, which is cheaper than optimization-oriented evaluation and provides a bounded, lower-variance indicator of whether the candidate behaves like a useful solution. Only candidates whose partial reports indicate nontrivial learning value advance to *Objective* and *Utility* evaluation.

In this quantum instantiation, the scheduling problem therefore becomes a sequential gated evaluation policy. The policy π_{gate} decides whether to proceed to stage $k+1$ based on the current partial report $\rho_{\le k} = \{(s_i, c_i)\}_{i=1}^{k}$ and the fixed stage order *Feasibility* → *Behavior* → *Objective* → *Utility*. Concretely, $\pi_{\text{gate}}(\rho_{\le k}) \in \{\texttt{stop}, \texttt{continue}\}$, where `continue` is chosen only when the current signal s_k is sufficiently promising. This policy mirrors the semantic ordering used in §4, but uses that ordering for budget allocation rather than reward construction.

5.3 Policy Implementation

Fig. 1 illustrates the gated evaluation policy. VISTA implements a stage-aware, thresholded escalation policy over partial verifier outcomes: (i) infeasible candidates stop immediately; (ii) feasible candidates are evaluated at *Behavior*; (iii) only candidates with $s_2 \ge \tau_{\text{behavior}}$ and sufficient remaining budget proceed to *Objective*; and (iv) *Utility* is evaluated only for candidates with $s_3 \ge \tau_{\text{objective}}$ and remaining budget.

This policy is intentionally lightweight. Its purpose is not to solve the general verifier-scheduling problem optimally, but to expose the key systems mechanism in a practical form: expensive verifier stages should be invoked selectively from partial semantic evidence rather than executed uniformly for every candidate. By stopping weak candidates early, VISTA reduces wasted verifier work and reallocates budget toward candidates that are more likely to benefit from deeper semantic supervision. §6.4 and §6.5 evaluate this policy along three dimensions: verifier-cost reduction, quality under matched budget, and end-to-end latency in the hybrid classical–quantum setting.

6 Evaluation

We evaluate VISTA and answer the following four questions.

Q1: Effectiveness. Does VISTA outperform SOTA baselines on both executability and semantic quality? (§6.2)

Q2: Mechanism. Which design choices in VISTA contribute most to the gains? (§6.3)

Q3: Efficiency. Does cost-aware gating reduce verifier cost while retaining strong semantic performance? (§6.4)

Q4: Deployment. Can the verifier-in-the-loop RL pipeline run end-to-end on real QPUs and hybrid classical–quantum systems? (§6.5)

6.1 Experimental Setup and Metrics

Training data. We use the state-of-the-art quantum-specific LLM training dataset from Agent-Q [21]. It contains 13,914 circuits and covers both typical and hard-tail circuit regimes, with qubit counts ranging from 8 to 16, gate counts from 48 to 1,138, and circuit depth from 8 to 839. The dataset also provides offline-optimized parameters, Hamiltonian metadata, and *ground-truth* circuits with numerically optimized parameters, which we use as reward references during verification. Additional dataset statistics and construction details are provided in Appendix A.4.

Models and testbed. We post-train a 4B Qwen3 model [50] using GRPO [40] with the VISTA staged quantum verification service. Experiments are conducted on a hybrid classical–quantum testbed consisting of a national supercomputer connected to a 5-qubit superconducting quantum processor. We use up to 16 compute nodes, each equipped with 8× AMD MI250X GPUs and a 64-core AMD EPYC Trento CPU, for a total of 128 GPUs and 1,024 CPU cores. Detailed training hyperparameters and deployment settings are provided in Appendix A.4.

Evaluation metrics. We report five complementary metrics under both Pass@1 and Pass@10. Pass@1 uses one sampled completion per prompt, while Pass@10 samples up to 10 completions per prompt. We use the following stage-aligned metrics throughout the evaluation: *Feasibility* (SCR), *Behavior* (RE), *Objective* (SREV), and *Utility* (HQCR). We additionally report ΔE as a fine-grained objective-error metric. These metrics correspond to different levels of staged external evaluation: SCR captures a hard feasibility check, whereas RE, SREV, HQCR, and ΔE capture progressively stronger soft semantic criteria.

(1) *Feasibility (SCR, Syntax Compilation Rate)* measures the fraction of generations that produce executable OpenQASM programs, i.e., $\text{SCR} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}[\text{parse pass}]$. Higher is better.

(2) *Behavior (RE, Relative Entropy)* measures the relative entropy between generated and reference measurement distributions, aggregated over evaluated samples. Lower is better.

(3) *Objective (SREV, Success Rate of Expectation Validation)* is a pass-rate metric for task-level objective quality: the fraction of generations whose expectation value lies within a predefined tolerance of the reference optimum, i.e., $\text{SREV} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}[|E(C_i) - E_i^{\star}| \le \epsilon_E]$. Higher is better.

(4) *Utility (HQCR, High-Quality Circuit Rate)* measures the fraction of generations that satisfy the high-quality criterion used in evaluation, combining semantic quality and downstream usefulness. In the threshold-sweep plots, the varied component is the RE quality cutoff (default 0.1, with sweeps shown in Fig. 3a), while the efficiency condition is kept fixed. Higher is better.

(5) *Objective Gap (ΔE, Expectation-Value Error)* measures per-instance absolute objective error, $\Delta E_i = |E(C_i) - E_i^{\star}|$. Lower is better.

Baselines. We compare against four categories of baselines

(1) *Frontier LLM agents:* DeepSeek-V3 [29], GPT-4o [16], and GPT-5 [37], all evaluated under the same four-shot prompting budget. These represent strong general-purpose LLM agents without task-specific reinforcement learning.

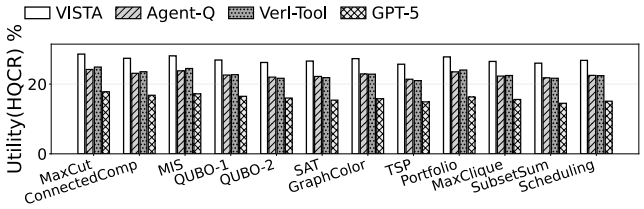


Fig. 2: Per-primitive breakdown of *Utility* (HQCR) across the 12 graph-optimization tasks. VISTA improves semantic quality broadly rather than on only a small subset of primitives.

(2) *Quantum-specific agents:* Agent-Q [21], a state-of-the-art fine-tuned LLM for quantum circuit generation and optimization.
(3) *RL post-training agents:* Cold Start GRPO [40], where the base model is trained directly with GRPO from scratch, without hierarchical verification or structured tool interaction.
(4) *RL agentic tool-use agents:* Verl-Tool [22], an agentic reinforcement learning framework that integrates tool interaction within a streaming training paradigm.

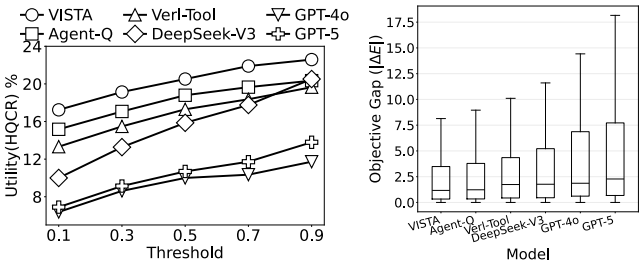
6.2 End-to-End Effectiveness and Robustness

We first evaluate whether VISTA improves end-to-end semantic quality over baselines, and whether these gains are robust.

(i) Overall performance. Table 1 shows that VISTA achieves the strongest overall combination of *Feasibility* (SCR), *Objective* (SREV), *Behavior* (RE), and *Utility* (HQCR) among the evaluated baselines. At Pass@1, VISTA reaches **99.31%** *Feasibility* (SCR), **22.41%** *Objective* (SREV), **11.61** *Behavior* (RE), and **17.24%** *Utility* (HQCR), improving over the strongest post-training baselines by +1.45 points, +2.57 points, −0.70 (5.69% lower), and +1.16 points, respectively. At Pass@10, VISTA reaches **100%** *Feasibility* (SCR), **33.10%** *Objective* (SREV), **8.48** *Behavior* (RE), and **27.24%** *Utility* (HQCR), corresponding to gains of +0.28 points, +1.06 points, −1.74 (17.03% lower), and +2.53 points against the strongest baseline per metric. Overall, verifier-in-the-loop post-training improves not only compilability, but also task-level semantic alignment.

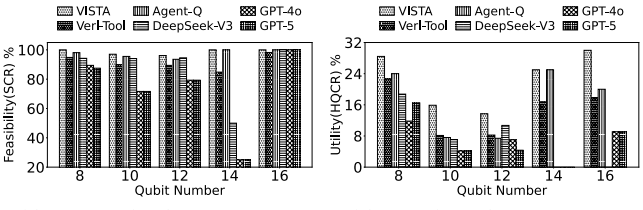
(ii) Per-primitive results. We next test whether these gains are broad or concentrated in only a few tasks. Fig. 2 reports per-primitive performance across the 12 graph-optimization tasks. The gains are broad rather than narrow: VISTA improves *Utility* (HQCR) across most primitives and maintains advantages on harder tasks where prompting-based baselines degrade more sharply. This pattern is consistent with the aggregate *Utility* (HQCR) gains in Table 1, suggesting that the improvement is distributed across primitives rather than driven by a single outlier task.

(iii) Robustness to semantic thresholds. We next test whether the gains persist under alternative semantic views rather than a single threshold. *Utility* (HQCR) uses a default *Behavior* (RE) threshold of 0.1; Fig. 3a sweeps this threshold from 0.1 to 0.9 and shows that VISTA remains consistently superior across the full range, indicating robustness beyond a hand-picked cutoff. We further assess objective-level semantics via $\Delta E = |E(C) - E^{\star}|$. Fig. 3b shows that VISTA reduces ΔE against all baselines, indicating genuinely better optima rather than superficial alignment. Compared with random parameter initialization, VISTA also achieves lower JS divergence



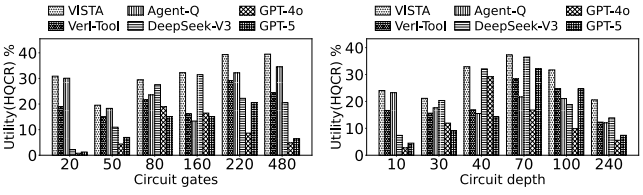
(a) *Utility* (HQCR) under varying *Behavior* (RE) thresholds. **(b)** *Objective Gap* (ΔE) for valid QASMs.

Fig. 3: Robustness of semantic quality under alternative thresholds and objective-gap evaluation. VISTA remains stronger under both *Utility* (HQCR) threshold sweeps and *Objective Gap* (ΔE).



(a) *Feasibility* (SCR) vs. qubit count. **(b)** *Utility* (HQCR) vs. qubit count.

Fig. 4: Scaling robustness in qubit count. VISTA maintains stronger *Feasibility* (SCR) and *Utility* (HQCR) as problem size grows.



(a) *Utility* (HQCR) vs. gate count. **(b)** *Utility* (HQCR) vs. circuit depth.

Fig. 5: Scaling robustness in circuit complexity. VISTA retains stronger *Utility* (HQCR) as gate count and circuit depth increase.

(0.79 vs. 0.95; 16.84% lower) and expectation values closer to the optimum (0.16 vs. 0.36; 55.56% lower), suggesting that it learns meaningful circuit structure and parameter priors rather than merely producing compilable templates.

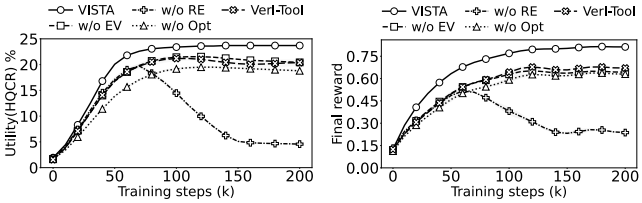
(iv) Scaling robustness. We finally test whether VISTA's advantage persists as problem size grows. In Fig. 4, VISTA retains higher *Feasibility* (SCR) and *Utility* (HQCR) than all baselines across 8–16 qubits, while prompting-based LLMs degrade sharply, especially beyond 14 qubits. This robustness also carries over to structural complexity: Fig. 5 shows that VISTA maintains stronger *Utility* (HQCR) as gate count (20–480) and depth (10–240) increase. These results indicate robustness to scale in both qubit size and circuit structure within the evaluated regime, though they should not be read as a full systems-level scaling study.

6.3 Why Does VISTA Work?

We next isolate which design choices drive VISTA's gains, focusing on hierarchical verified rewards and their effect on optimization. We

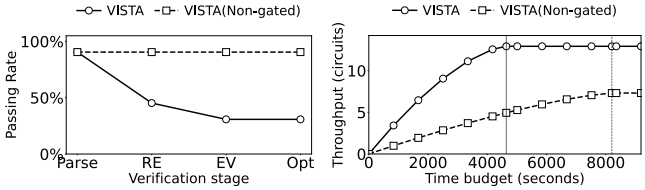
Tab. 1: Vista's Pass@K comparison across baseline categories. Metrics include Feasibility (SCR), Objective (SREV), and Utility (HQCR).

Category	Methods	Pass@1				Pass@10			
		Feasibility SCR↑	*Behavior* RE↓	*Objective* SREV↑	*Utility* HQCR↑	*Feasibility* SCR↑	*Behavior* RE↓	*Objective* SREV↑	*Utility* HQCR↑
LLM Agents	DeepSeek-V3	94.83%	19.20	12.24%	10.00%	98.97%	16.39	26.38%	16.38%
	GPT-5	87.07%	19.94	10.00%	6.90%	90.52%	11.57	27.07%	16.55%
	GPT-4o	87.93%	19.42	9.83%	6.38%	88.79%	14.08	18.62%	12.07%
Quantum-Specific Agent	Agent-Q	97.41%	12.74	18.97%	15.17%	99.65%	10.81	31.55%	23.62%
RL Post-Training	Cold Start GRPO	84.48%	14.32	19.84%	12.41%	95.17%	11.38	27.59%	18.96%
RL Agentic Tool-Use	Verl-Tool	97.86%	12.31	19.62%	16.08%	99.72%	10.22	32.04%	24.71%
Quantum-Specific RL (Ours)	**Vista**	**99.31%**	**11.61**	**22.41%**	**17.24%**	**100%**	**8.48**	**33.10%**	**27.24%**



(a) *Utility* (HQCR) during training. **(b)** Total verified reward during training.

Fig. 6: Training dynamics of hierarchical verified rewards. Vista converges faster and remains more stable than weaker ablations and Verl-Tool.



(a) Stage-wise passing rates under gated evaluation. **(b)** Average verifier time per iteration: gated vs. non-gated.

Fig. 7: Cost-aware gated evaluation improves verifier efficiency by filtering weak candidates before expensive stages.

answer this in two steps: (i) reward ablations under matched compute (Table 2), and (ii) training-dynamics analysis to test whether staged semantic supervision improves convergence.

(i) Reward ablation results. We quantify the contribution of each reward component by removing one element of the hierarchical reward in §4. The ablations follow the four stages: *Feasibility*, *Behavior*, *Objective*, and *Utility*. For each variant, we disable one component while keeping the remainder unchanged, and include a Validity-only sanity baseline that uses only the *Feasibility* reward (valid QASM: 1, invalid QASM: −1). All runs start from the same SFT checkpoint and use matched compute, fixing the verifier, references, and optimization budget to isolate the effect of reward design.

Table 2 shows that *Behavior* is the dominant contributor. Removing it causes the largest drop across all metrics: at Pass@1, *Feasibility* (SCR) falls from 99.31% to 66.38%, *Objective* (SREV) from 22.41% to 5.17%, *Utility* (HQCR) from 17.24% to 5.69%, and *Behavior* (RE) worsens from 11.61 to 24.67. The same pattern holds at Pass@10, with drops of 20.18 points in *Feasibility* (SCR), 17.58 in *Objective* (SREV), 10.34 in *Utility* (HQCR), and 9.78 in *Behavior* (RE). Other stages contribute smaller but still measurable gains: removing *Objective* lowers Pass@10 *Objective* (SREV) by 2.07 points, and removing *Utility* lowers Pass@10 *Utility* (HQCR) by 0.69 points. *Feasibility* alone is insufficient: the Validity-only baseline keeps Pass@10 *Feasibility* (SCR) at 100% but reduces *Utility* (HQCR) from 27.24% to 23.27%. Overall, the stages play different roles, and the full hierarchy performs best jointly.

(ii) Training dynamics and stability. We further test whether hierarchical rewards improve optimization behavior during training. As shown in Fig. 6, over 0–200k steps Vista converges faster and maintains higher *Utility* (HQCR) and total reward than weaker ablations and Verl-Tool. Variants such as `w/o Behavior` also exhibit visible degradation or instability. These dynamics are consistent with the final ablation gaps in Table 2, for example the +10.34 Pass@10 gain in *Utility* (HQCR) over `w/o Behavior`. This supports the role of hierarchical verified rewards in steering optimization toward semantically meaningful regions of the search space.

6.4 Matched-Budget Verifier Efficiency

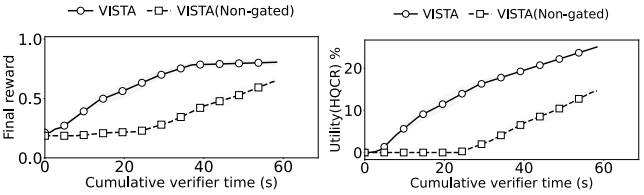
We next isolate the verifier-side benefit of cost-aware gating: whether it reduces verifier cost, whether the saved budget improves quality under matched verifier wall-clock budgets, and where the savings arise within the multi-stage verifier. This section focuses on verifier-only accounting; §6.5 complements it with end-to-end system latency and hardware-backed deployment results.

Verifier efficiency. Fig. 7 reports (a) stage-wise passing rates and (b) verifier time under gated evaluation. Compared with the non-gated variant, Vista reduces average verifier time per iteration from 8.20s to 4.63s, a 1.77× speedup (Fig. 7b). The passing-rate plot explains the gain: many weak candidates are filtered out by early, inexpensive stages, so costly simulation and optimization are reserved for stronger trajectories. This provides direct evidence that progressive semantic screening improves verifier efficiency.

Budget-matched gating analysis. To separate gating effects from differences in total compute, we compare gated and non-gated variants under matched cumulative verifier wall-clock budgets. As shown in Fig. 8, across the 0–60s budget range, the gated design achieves higher final reward and HQCR at every budget level. Using the measured per-iteration times (4.63s gated vs. 8.20s non-gated), a 60s budget corresponds to roughly 13 gated iterations versus about 7 non-gated iterations, explaining the faster quality gains per

Tab. 2: Stage-based reward ablation for Vista. The full model uses Feasibility (Val) + Behavior (RE) + Objective (EV) + Utility (Opt).

Variant	Components	Pass@1				Pass@10			
		Feasibility SCR↑	*Objective* SREV↑	*Behavior* RE↓	*Utility* HQCR↑	*Feasibility* SCR↑	*Objective* SREV↑	*Behavior* RE↓	*Utility* HQCR↑
Full (Vista)	**Val + RE + EV + Opt**	**99.31%**	**22.41%**	**11.61**	**17.24%**	**100%**	**33.10%**	**8.48**	**27.24%**
w/o EV term	Val + RE + Opt	98.62%	20.69%	11.82	16.38%	100%	31.03%	9.12	26.72%
w/o RE term	Val + EV + Opt	66.38%	5.17%	24.67	5.69%	79.82%	15.52%	18.26	16.90%
w/o Opt term	Val + RE + EV	98.79%	21.90%	11.98	16.90%	100%	32.76%	9.01	26.55%
Validity only	Val	99.13%	18.79%	12.89	14.66%	100%	30.86%	11.27	23.27%



(a) Final reward under matched cumulative verifier wall-clock budget on the simulator.

(b) HQCR under matched cumulative verifier wall-clock budget on the simulator.

Fig. 8: Matched-budget simulator comparison: gating achieves better quality under the same cumulative verifier wall-clock budget.

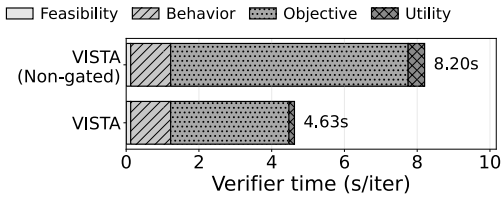


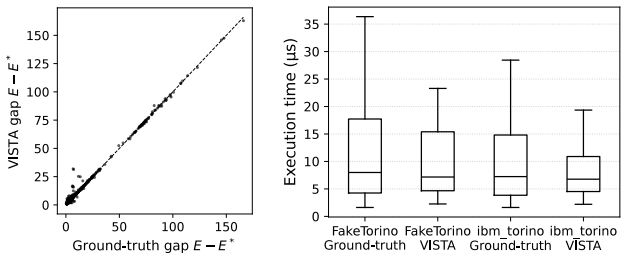
Fig. 9: Average verifier wall-clock time per iteration by stage. Gating avoids expensive stages on weak candidates.

unit time. Gating therefore delivers a better quality–cost trade-off without changing the verifier or reward definition.

Stage-wise verifier cost analysis. Fig. 9 breaks down average verifier wall-clock time per iteration across the four stages: *Feasibility*, *Behavior*, *Objective*, and *Utility*. Relative to the non-gated variant, Vista reduces total verifier time from 8.20s to 4.63s per iteration (1.77× speedup), mainly by stopping weak candidates before expensive late-stage evaluation. The largest savings come from the *Objective* stage, which dominates cost in the non-gated variant but is triggered much less often under gating. Early-stage costs remain similar, indicating that the savings come from selective progression rather than reduced verification fidelity.

6.5 Real Hybrid Classical–Quantum Deployment and End-to-End Latency

We finally evaluate Vista in two real deployment environments. This subsection complements §6.4 with hardware-backed quality results and end-to-end measurements. We first validate circuit quality and execution efficiency on public IBM Quantum backends, then evaluate the full verifier-in-the-loop pipeline on a real hybrid classical–quantum system built from the LUMI supercomputer [9] and the VTT Q50 quantum computer [48].



(a) Optimality-gap agreement between Vista and ground truth.

(b) Scheduled execution duration on the backend.

Fig. 10: Hardware-level validation on public IBM Quantum backends. Vista maintains comparable solution quality while reducing scheduled execution time.

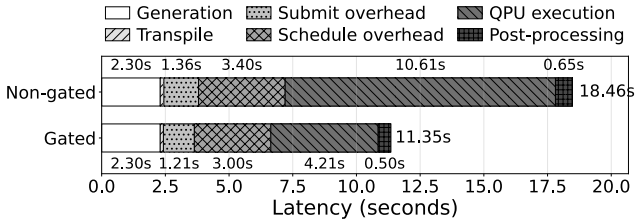
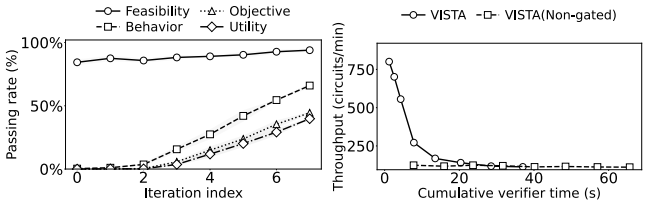


Fig. 11: End-to-end pipeline latency on the LUMI + VTT Q50 hybrid classical–quantum system, including generation, transpilation, submission, scheduling, QPU execution, and post-processing. Gated execution reduces total latency relative to the non-gated variant.

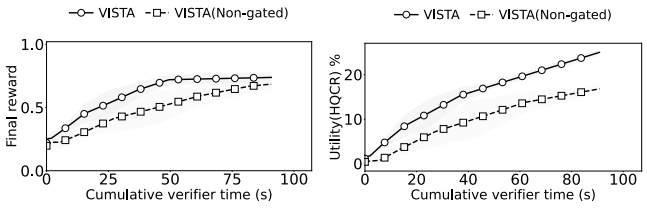
6.5.1 Public IBM Quantum backends: hardware-level quality and efficiency. We evaluate Vista on `ibm_torino` and `FakeTorino` under a *same-device* constraint: both ground-truth and Vista circuits are transpiled with identical Qiskit settings (fixed seed, same routing/layout/optimization level) and restricted to the same n-qubit subdevice (physical qubits $0, \ldots, n-1$) [17, 20]. `FakeTorino` provides a controlled comparison under matched compilation conditions. As shown in Fig. 10, Vista closely matches ground truth in optimality gap while producing shorter scheduled durations, yielding average speedups of 1.97× on `ibm_torino` and 2.12× on `FakeTorino`.

6.5.2 LUMI + VTT Q50: end-to-end hybrid deployment. We next evaluate the full verifier-in-the-loop pipeline on a real hybrid classical–quantum system in which LLM inference, RL post-training, and reward computation run on the LUMI supercomputer [9], while physical quantum execution is provided by the VTT Q50 superconducting quantum computer [48, 49]. Fig. 11 shows that gating reduces mean



(a) Stage-wise pass rates across iterations. **(b)** Iteration progress versus cumulative verifier budget.

Fig. 12: Verifier-loop diagnostics on the LUMI + VTT Q50 hybrid classical–quantum system. Gating sharpens stage-wise filtering and improves throughput under budget constraints.



(a) Final reward under matched effective verifier wall-clock budget.

(b) *Utility* (HQCR) under matched effective verifier wall-clock budget.

Fig. 13: Matched-budget comparison on the LUMI + VTT Q50 hybrid system. Gating achieves better quality than the non-gated variant.

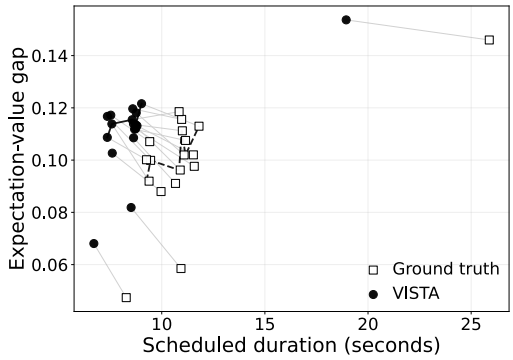


Fig. 14: Instance-level quality–efficiency trade-off on the VTT Q50 quantum computer. Vista typically matches solution quality while requiring less scheduled hardware time.

end-to-end latency from 18.46 s to 11.35 s, a 1.63× speedup, mainly by reducing quantum-side execution and queueing overhead.

On the same platform, Fig. 12a and Fig. 12b show that gating sharpens stage-wise filtering and improves iteration throughput per unit cumulative verifier budget. Under matched effective verifier wall-clock budgets, Fig. 13 further shows that gating achieves higher final reward and higher *Utility* (HQCR) across the full 0–60 s range, reaching the same quality threshold earlier than the non-gated baseline.

Finally, Fig. 14 compares scheduled hardware duration against expectation-value gap for individual hardware-executed instances. Across most cases, Vista matches or improves expectation-value gap while requiring shorter scheduled duration.

7 Related Work

Agentic tool use and LLM-based quantum programming. Our work relates to agentic systems that learn from tool or execution feedback and to LLM-based quantum programming assistants. Prior work has used executable feedback for RL-based code generation [25] and self-reflection for agent improvement [42]. In quantum computing, studies have explored Qiskit-oriented code assistance [19, 47], quantum-verifiable post-training [11], and prompting-based synthesis [6]. In contrast, we target OpenQASM 3.0 [8] and study verifier-in-the-loop RL under staged semantic evaluation.

LLMs for quantum design. Language models have also been used for higher-level quantum design tasks such as ansatz design [15, 27, 45] and quantum experiment design [4]. These studies show that LLMs can assist broader quantum workflows, but they do not address staged verifier feedback, hierarchical reward construction, or budget-aware control of expensive verification pipelines.

Transformer and GPT-style quantum models. Related work has further applied transformer and GPT-style models to quantum circuit prediction and generation, including measurement prediction [13], circuit generation for electronic-structure problems [35], OpenQASM 2.0 circuit generation [3, 26], and QAOA circuit generation for MaxCut [44]. Compared with these approaches, our focus is not a stronger standalone generator, but a verifier-in-the-loop training system that organizes staged semantic feedback through hierarchical verified rewards and budget-aware verifier scheduling.

8 Conclusion

We presented Vista, a verifier-in-the-loop agentic RL framework for quantum program synthesis, instantiated for OpenQASM-based quantum optimization. We showed that practical verifier-in-the-loop training in this setting must address what semantic signals to optimize, how to shape them into stable learning objectives, and when to spend verification budget across a staged verifier. Vista addresses these through hierarchical verified reward optimization and budget-aware verifier scheduling. Across quantum optimization tasks, Vista improves executability and semantic quality over prompting-based and post-training baselines, while demonstrating deployment on real hybrid classical–quantum systems. Future work includes exploring whether a similar staged-verifier perspective applies to non-quantum domains with expensive external checks, such as tool-verified code generation or program repair, without claiming such instantiations as validated contributions here.

Acknowledgments

This work was supported by the Research Council of Finland (Grant Nos. 362729 and 358877), Business Finland (Grant No. 169/31/2024), and the Finnish Ministry of Education and Culture's Doctoral Education Pilot programs AI-DOC (Decision No. VN/3137/2024-OKM-6) and QDOC (Decision No. VN/3137/2024-OKM-4). The authors acknowledge EuroHPC JU computing resources under Project EHPC-REG-2025R02-367 and access to the Finnish Quantum-Computing Infrastructure (FiQCI), including the VTT Q50 quantum computer under Project 462001123.

References

[1] Amira Abbas, Andris Ambainis, Brandon Augustino, Andreas Bärtschi, Harry Buhrman, Carleton Coffrin, Giorgio Cortiana, et al. 2024. Challenges and opportunities in quantum optimization. *Nature Reviews Physics* 6, 12 (Dec. 2024), 718–735. doi:10.1038/s42254-024-00770-9 Publisher: Nature Publishing Group.

[2] Amazon Web Services. 2026. Amazon Braket Pricing. https://aws.amazon.com/braket/pricing/. Accessed: 2026-04-29.

[3] Boran Apak, Medina Bandic, Aritra Sarkar, and Sebastian Feld. 2024. KetGPT – Dataset Augmentation of Quantum Circuits Using Transformers. In *Computational Science – ICCS 2024*, Leonardo Franco, Clélia de Mulatier, Maciej Paszynski, Valeria V. Krzhizhanovskaya, Jack J. Dongarra, and Peter M. A. Sloot (Eds.). Springer Nature Switzerland, Cham, 235–251.

[4] Sören Arlt, Haonan Duan, Felix Li, Sang Michael Xie, Yuhuai Wu, and Mario Krenn. 2024. Meta-Designing Quantum Experiments with Language Models. *arXiv preprint arXiv:2406.02470* (2024).

[5] Endre Boros and Peter L. Hammer. 2002. Pseudo-Boolean optimization. *Discrete Applied Mathematics* 123, 1 (2002), 155–225. doi:10.1016/S0166-218X(01)00341-9

[6] Charlie Campbell, Hao Mark Chen, Wayne Luk, and Hongxiang Fan. 2025. Enhancing LLM-based Quantum Code Generation with Multi-Agent Optimization and Quantum Error Correction. *arXiv preprint arXiv:2504.14557* (2025).

[7] M. Cerezo, Andrew Arrasmith, Ryan Babbush, Simon C. Benjamin, Suguru Endo, Keisuke Fujii, Jarrod R. McClean, Kosuke Mitarai, Xiao Yuan, Lukasz Cincio, and Patrick J. Coles. 2021. Variational quantum algorithms. *Nature Reviews Physics* 3, 9 (Aug. 2021), 625–644. doi:10.1038/s42254-021-00348-9

[8] Andrew W. Cross, Ali Javadi-Abhari, Thomas Alexander, Niel de Beaudrap, Lev S. Bishop, et al. 2022. OpenQASM 3: A broader and deeper quantum assembly language. *ACM Transactions on Quantum Computing* 3, 3 (Sept. 2022), 1–50. doi:10.1145/3505636 arXiv:2104.14722 [quant-ph].

[9] CSC – IT Center for Science and LUMI Consortium. 2026. LUMI: One of the most powerful supercomputers in the world. https://csc.fi/en/our-expertise/high-performance-computing/lumi-supercomputer/. Accessed: 2026-04-29.

[10] Cirq Developers. 2025. *Cirq*. Zenodo. doi:10.5281/ZENODO.4062499

[11] Nicolas Dupuis, Adarsh Tiwari, Youssef Mroueh, David Kremer, Ismael Faro, and Juan Cruz-Benito. 2025. Quantum Verifiable Rewards for Post-Training Qiskit Code Assistant. *arXiv preprint arXiv:2508.20907* (2025). arXiv:2508.20907 [quant-ph] doi:10.48550/arXiv.2508.20907

[12] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. 2014. A Quantum Approximate Optimization Algorithm. *arXiv preprint arXiv:1411.4028* (2014). doi:10.48550/arXiv.1411.4028

[13] David Fitzek, Yi Hong Teoh, Hin Pok Fung, Gebremedhin A. Dagnew, Ejaaz Merali, M. Schuyler Moss, Benjamin MacLellan, and Roger G. Melko. 2024. RydbergGPT. *arXiv preprint arXiv:2405.21052* (may 2024). doi:10.48550/arXiv.2405.21052

[14] Harper R. Grimsley, Sophia E. Economou, Edwin Barnes, and Nicholas J. Mayhall. 2019. An adaptive variational algorithm for exact molecular simulations on a quantum computer. *Nature Communications* 10, 1 (July 2019), 3007. doi:10.1038/s41467-019-10988-2

[15] Yaswitha Gujju, Romain Harang, and Tetsuo Shibuya. 2025. LLM-Guided Ansätze Design for Quantum Circuit Born Machines in Financial Generative Modeling. arXiv:2509.08385 [quant-ph]

[16] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. 2024. GPT-4o System Card. *arXiv preprint arXiv:2410.21276* (2024).

[17] IBM. 2026. IBM Quantum Platform. https://www.ibm.com/quantum Accessed: 2026-04-29.

[18] IBM. 2026. IBM Quantum: Products and services (Quantum compute access plans). https://www.ibm.com/quantum/products. Accessed: 2026-04-29.

[19] IBM Quantum. 2025. Qiskit Code Assistant. https://quantum.cloud.ibm.com/docs/en/guides/qiskit-code-assistant Accessed: 2026-04-29.

[20] Ali Javadi-Abhari, Matthew Treinish, Kevin Krsulich, Christopher J. Wood, Jake Lishman, Julien Gacon, Simon Martiel, Paul D. Nation, Lev S. Bishop, Andrew W. Cross, Blake R. Johnson, and Jay M. Gambetta. 2024. Quantum computing with Qiskit. arXiv:2405.08810 [quant-ph]

[21] Linus Jern, Valter Uotila, Cong Yu, and Bo Zhao. 2025. Agent-Q: Fine-Tuning Large Language Models for Quantum Circuit Generation and Optimization. In *2025 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE.

[22] Dongfu Jiang, Yi Lu, Zhuofeng Li, Zhiheng Lyu, Ping Nie, Haozhe Wang, Alex Su, Hui Chen, Kai Zou, Chao Du, et al. 2025. VerlTool: Towards Holistic Agentic Reinforcement Learning with Tool Use. *arXiv preprint arXiv:2509.01055* (2025).

[23] Richard M. Karp. 1972. *Reducibility among Combinatorial Problems*. Springer US, Boston, MA, 85–103. doi:10.1007/978-1-4684-2001-2_9

[24] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles* (Koblenz, Germany) *(SOSP '23)*. Association for Computing Machinery, New York, NY, USA, 611–626. doi:10.1145/3600006.3613165

[25] Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven C. H. Hoi. 2022. CodeRL: Mastering Code Generation through Pretrained Models and Deep Reinforcement Learning. *arXiv preprint arXiv:2207.01780* (2022). arXiv:2207.01780 [cs.LG] doi:10.48550/arXiv.2207.01780

[26] Ang Li, Samuel Stein, Sriram Krishnamoorthy, and James Ang. 2023. QASMBench: A Low-Level Quantum Benchmark Suite for NISQ Evaluation and Simulation. *ACM Transactions on Quantum Computing* 4, 2, Article 10 (Feb. 2023), 26 pages. doi:10.1145/3550488

[27] Zhiding Liang, Jinglei Cheng, Rui Yang, Hang Ren, Zhixin Song, Di Wu, Xuehai Qian, Tongyang Li, and Yiyu Shi. 2023. Unleashing the Potential of LLMs for Quantum Computing: A Study in Quantum Architecture Design. arXiv:2307.08191 (July 2023). doi:10.48550/arXiv.2307.08191 arXiv:2307.08191 [quant-ph].

[28] J. Lin. 1991. Divergence measures based on the Shannon entropy. *IEEE Transactions on Information Theory* 37, 1 (1991), 145–151. doi:10.1109/18.61115

[29] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024. DeepSeek-V3 Technical Report. *arXiv preprint arXiv:2412.19437* (2024).

[30] Ilya Loshchilov and Frank Hutter. 2017. Decoupled Weight Decay Regularization. *arXiv preprint arXiv:1711.05101* (2017).

[31] Andrew Lucas. 2014. Ising formulations of many NP problems. *Frontiers in Physics* 2 (Feb. 2014). doi:10.3389/fphy.2014.00005

[32] Jarrod R. McClean, Sergio Boixo, Vadim N. Smelyanskiy, Ryan Babbush, and Hartmut Neven. 2018. Barren plateaus in quantum neural network training landscapes. *Nature Communications* 9, 1 (Nov. 2018). doi:10.1038/s41467-018-07090-4

[33] Jarrod R McClean, Nicholas C Rubin, Kevin J Sung, Ian D Kivlichan, Xavier Bonet-Monroig, Yudong Cao, Chengyu Dai, E Schuyler Fried, Craig Gidney, Brendan Gimby, Pranav Gokhale, Thomas Häner, Tarini Hardikar, Vojtěch Havlíček, Oscar Higgott, Cupjin Huang, Josh Izaac, Zhang Jiang, Xinle Liu, Sam McArdle, Matthew Neeley, Thomas O'Brien, Bryan O'Gorman, Isil Ozfidan, Maxwell D Radin, Jhonathan Romero, Nicolas P D Sawaya, Bruno Senjean, Kanav Setia, Sukin Sim, Damian S Steiger, Mark Steudtner, Qiming Sun, Wei Sun, Daochen Wang, Fang Zhang, and Ryan Babbush. 2020. OpenFermion: the electronic structure package for quantum computers. *Quantum Science and Technology* 5, 3 (jun 2020), 034014. doi:10.1088/2058-9565/ab8ebc

[34] Microsoft. [n. d.]. *Azure Quantum Development Kit*. https://github.com/microsoft/qsharp

[35] Kouhei Nakaji, Lasse Bjørn Kristensen, Jorge A. Campos-Gonzalez-Angulo, Mohammad Ghazi Vakili, Haozhe Huang, Mohsen Bagherimehrab, Christoph Gorgulla, FuTe Wong, Alex McCaskey, Jin-Sung Kim, Thien Nguyen, Pooja Rao, and Alan Aspuru-Guzik. 2024. The generative quantum eigensolver (GQE) and its application for ground state search. *arXiv preprint arXiv:2401.09253* (jan 2024). doi:10.48550/arXiv.2401.09253

[36] Michael A. Nielsen and Isaac L. Chuang. 2010. *Quantum computation and quantum information* (10th anniversary ed ed.). Cambridge University Press, Cambridge ; New York.

[37] OpenAI. 2025. *Introducing GPT-5*. https://openai.com/index/introducing-gpt-5/ Accessed: 2026-04-29.

[38] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O'Brien. 2014. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications* 5 (July 2014), 4213. doi:10.1038/ncomms5213

[39] Martin Rinard. 2003. Acceptability-Oriented Computing. In *Companion of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*.

[40] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. arXiv:2402.03300 [cs.CL] https://arxiv.org/abs/2402.03300

[41] Rahul Sharma, Eric Schkufza, Bertrand Churchill, and Alex Aiken. 2015. Conditionally Correct Superoptimization. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*.

[42] Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language Agents with Verbal Reinforcement Learning. *arXiv preprint arXiv:2303.11366* (2023). arXiv:2303.11366 [cs.AI] doi:10.48550/arXiv.2303.11366

[43] Robert S. Smith, Michael J. Curtis, and William J. Zeng. 2017. A Practical Quantum Instruction Set Architecture. arXiv:1608.03355 [quant-ph]

[44] Ilya Tyagin, Marwa H. Farag, Kyle Sherbert, Karunya Shirali, Yuri Alexeev, and Ilya Safro. 2025. QAOA-GPT: Efficient Generation of Adaptive and Regular Quantum Approximate Optimization Algorithm Circuits. arXiv:2504.16350 (April 2025). doi:10.48550/arXiv.2504.16350 arXiv:2504.16350 [quant-ph].

[45] Kento Ueda and Atsushi Matsuo. 2025. Optimizing Ansatz Design in Quantum Generative Adversarial Networks Using Large Language Models. arXiv:2503.12884 (March 2025). doi:10.48550/arXiv.2503.12884 arXiv:2503.12884 [quant-ph].

[46] Valter Uotila, Julia Ripatti, and Bo Zhao. 2025. Higher-Order Portfolio Optimization with Quantum Approximate Optimization Algorithm. In *2025 IEEE International Conference on Quantum Computing and Engineering (QCE)*, Vol. 01. 01–12. doi:10.1109/QCE65121.2025.00244

[47] Sanjay Vishwakarma, Francis Harkins, Siddharth Golecha, Vishal Sharathchandra Bajpe, Nicolas Dupuis, Luca Buratti, David Kremer, Ismael Faro, Ruchir Puri, and Juan Cruz-Benito. 2024. Qiskit HumanEval: An Evaluation Benchmark For Quantum Code Generative Models. In *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*, Vol. 1. IEEE, 1169–1176.

[48] VTT Technical Research Centre of Finland. 2026. Quantum computers: VTT Q50. https://www.vttresearch.com/en/technology-infrastructures/quantum-computers. Accessed: 2026-04-29.

[49] VTT Technical Research Centre of Finland. 2026. Quantum computing services & innovation. https://www.vttresearch.com/en/ourservices/quantum-computing. Accessed: 2026-04-29.

[50] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. 2025. Qwen3 Technical Report. arXiv:2505.09388 [cs.CL] https://arxiv.org/abs/2505.09388

[51] Zebo Yang, Maede Zolanvari, and Raj Jain. 2023. A Survey of Important Issues in Quantum Computing and Communications. *IEEE Communications Surveys & Tutorials* 25, 2 (2023), 1059–1094. doi:10.1109/COMST.2023.3254481

[52] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, et al. 2023. PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel. arXiv:2304.11277 [cs.DC]

A Additional Details

A.1 Background on OpenQASM and Quantum Optimization

OpenQASM as a quantum circuit representation. OpenQASM (Open Quantum Assembly Language) [8] is a low-level language for expressing quantum circuits and operations. It serves as a hardware-facing intermediate representation: quantum programs written in higher-level software stacks can be compiled into OpenQASM for execution, exchange, and optimization across different backends. OpenQASM is supported by major quantum software frameworks, including Qiskit [20], Cirq [10], Microsoft's QDK [34], and Rigetti's Forest [43]. This makes it a practical representation for studying synthesis problems that must bridge quantum software and hardware.

Quantum circuits and parameterized optimization. Quantum computation operates on qubits, where a single qubit is represented by a unit vector in a two-dimensional Hilbert space and may exist in a superposition $\alpha|0\rangle + \beta|1\rangle$ with $\alpha, \beta \in \mathbb{C}$ and $|\alpha|^2 + |\beta|^2 = 1$ [36]. Multi-qubit states are formed through tensor products, and computation proceeds by applying unitary gates. Rotation gates such as $R_x(\theta)$, $R_y(\theta)$, and $R_z(\theta)$, together with entangling gates such as CNOT and standard gates such as Hadamard and CZ, form universal gate sets for quantum computing [51]. By composing such gates into parameterized circuits $U(\theta)$, one obtains the basic framework used by variational quantum algorithms.

Optimization problems as Hamiltonian minimization. A major application of near-term quantum computing is combinatorial optimization [1]. Classical problems such as QUBO and HUBO [5, 31] can be mapped into Hamiltonian minimization problems by rewriting them in spin form and encoding them with Pauli operators. The optimization objective is then to prepare a parameterized state $U(\theta)|0\rangle$ whose expectation value

$$\langle 0|U^\dagger(\theta)HU(\theta)|0\rangle$$

approximates the ground-state energy of the Hamiltonian H.

Representative variational methods. Hybrid quantum–classical methods optimize circuit parameters iteratively. QAOA alternates between a cost Hamiltonian and a mixer ansatz [12]. VQE uses expressive or hardware-efficient ansatzes to minimize the target energy [38]. Adaptive VQE further constructs circuits incrementally from a gate pool [14]. These methods motivate our synthesis setting: the generated artifact is not only a circuit topology, but also a parameterized program whose semantic quality depends jointly on structure, parameters, and objective evaluation.

A.2 Additional Details of Hierarchical Verified Reward

This appendix provides technical details omitted from §4, including the theoretical motivation for distributional alignment, the detailed qubit-mismatch correction, the optimization-progress interpretation, and the integration with GRPO.

Reward rationale and theoretical motivation. For the quantum optimization tasks considered in this work, the objective is to produce a circuit whose measurement distribution concentrates on low-energy bitstrings, equivalently minimizing the expected energy

$$J(C) := \langle\psi_C|H|\psi_C\rangle = \sum_z p_C(z)\,E(z).$$

Here, C denotes a quantum circuit that prepares the state $|\psi_C\rangle = U_C|0\rangle^{\otimes n}$, and $p_C(z) = |\langle z|\psi_C\rangle|^2$ is the distribution obtained by measuring $|\psi_C\rangle$ in the computational basis. For the optimization problems considered in this work, the cost Hamiltonian H is diagonal in the computational basis and can be written as

$$H = \sum_{z\in\{0,1\}^n} E(z)\,|z\rangle\langle z|,$$

where $E(z)$ is the classical objective value associated with bitstring z.

Lemma 1 (Distributional alignment bounds energy error). Let p_C and $p_{C^\star}$ denote the measurement distributions induced by a generated circuit C and a reference circuit $C^\star$, respectively. If H is diagonal in the computational basis, then

$$\left|J(C) - J(C^\star)\right| \le \|E\|_\infty\,\|p_C - p_{C^\star}\|_1 \le \|E\|_\infty\sqrt{2\ln 2\cdot \mathrm{JS}(p_C\,\|\,p_{C^\star})},$$

where $\|E\|_\infty = \max_z |E(z)|$.

Interpretation. Reducing the Jensen–Shannon divergence between measurement distributions bounds the deviation in expected energy, which motivates distributional alignment as a lower-variance surrogate for the optimization objective.

Detailed qubit-mismatch correction. In practice, the generated QASM may specify a different number of qubits than the reference QASM, making direct distribution comparison ill-posed due to dimension mismatch. Let n_{gen} and n_{gt} be the qubit counts of the generated and reference circuits, respectively, and let $k = \min(n_{\text{gen}}, n_{\text{gt}})$. We define a qubit-mismatch correction

$$R_{\text{qm}} = \mathrm{clip}_{[-0.2,0]}\big(\alpha + \beta\,\Delta n + \gamma\,a_{\text{extra}} + \eta\,e_{\text{cross}}\big), \qquad \Delta n = |n_{\text{gen}} - n_{\text{gt}}|,$$

where a_{extra} counts *active* extra qubits beyond the first k, and e_{cross} counts multi-qubit gates that entangle extra wires with core wires. To ensure fairness, we compute the JS-based comparison on the first k qubits via marginalization and down-scale its contribution according to the mismatch severity.

For the expectation-value and optimization-progress terms, we pad the problem Hamiltonian with identities so that its width matches the generated circuit, preserving comparability while preventing reward artifacts through ancillary qubits.

Detailed reward construction. For each generated trajectory τ_i, corresponding to a complete OpenQASM program, we define the terminal reward as

$$R(\tau_i) = \begin{cases} -1, & R_{\text{syn}}(\tau_i) = -1, \\ w_2\, s_2(\tau_i) + w_3\, s_3(\tau_i) + w_4\, R_{\text{opt}}(\tau_i), & R_{\text{syn}}(\tau_i) = 1, \end{cases}$$

where $R_{\text{syn}}(\tau_i) \in \{-1, 1\}$ is the Feasibility reward, $s_2(\tau_i)$ is the Behavior-stage score, $s_3(\tau_i)$ is the Objective-stage score, and $R_{\text{opt}}(\tau_i)$ is the Utility-stage score. The weights $w_2, w_3, w_4 \geq 0$ control the relative contribution of the later semantic stages.

Behavior-stage score. Prior work [21] evaluated circuit quality using relative entropy, i.e., Kullback–Leibler divergence. In this work, we use the Jensen–Shannon distance because it is bounded in $[0, 1]$ and empirically more stable for RL training. The Jensen–Shannon distance is

$$d_{\text{JS}}(p, q) = \sqrt{\frac{\text{JS}(p \,\|\, q)}{\log 2}}, \tag{5}$$

where

$$\text{JS}(p \,\|\, q) = \tfrac{1}{2} D_{\text{KL}}(p \,\|\, m) + \tfrac{1}{2} D_{\text{KL}}(q \,\|\, m), \qquad m = \tfrac{1}{2}(p + q).$$

We define the Behavior-stage score as

$$s_2(\tau_i) = 1 - d_{\text{JS}}(p, q),$$

optionally corrected by the qubit-mismatch term described above.

Objective-stage score. For a feasible circuit, we simulate the circuit and compute the expectation value of the problem-specific Hamiltonian, denoted by E_{gen}. Let E_{min} and E_{max} denote the minimum and maximum eigenvalues of the Hamiltonian. We use min–max normalization to obtain

$$f_{\text{max}}^{\text{min}}(E_{\text{gen}}) := \frac{E_{\text{gen}} - E_{\text{min}}}{E_{\text{max}} - E_{\text{min}}}, \tag{6}$$

and define the Objective-stage score as

$$s_3(\tau_i) := 1 - f_{\text{max}}^{\text{min}}(E_{\text{gen}}) = 1 - \frac{E_{\text{gen}} - E_{\text{min}}}{E_{\text{max}} - E_{\text{min}}}. \tag{7}$$

Utility-stage score. A practical workflow often continues local optimization from the generated circuit. Let n denote the number of optimization steps required to reach the stopping criterion, and let E_{opt} denote the expectation value after local optimization. We define

$$R_{\text{opt}}(\tau_i) := \frac{1}{1 + n} + \left(1 - f_{\text{max}}^{\text{min}}(E_{\text{opt}})\right), \tag{8}$$

where $f_{\text{max}}^{\text{min}}$ is defined in Equation 6. Under standard local convergence assumptions, the number of optimization steps required to reach a target energy threshold is monotone in the initial suboptimality, making optimization progress a proxy for initialization quality.

Integration with GRPO.. Following the GRPO formulation used in our system, for a given input prompt x we sample a group of G trajectories $\{\tau_i\}_{i=1}^{G}$ from the current policy $\pi_\theta(\cdot \mid x)$. Each trajectory τ_i receives a terminal reward $R(\tau_i)$ defined above. We compute the group-normalized advantage as

$$\hat{A}_i = \frac{R(\tau_i) - \mu}{\sigma}, \qquad \mu = \frac{1}{G}\sum_{j=1}^{G} R(\tau_j), \quad \sigma = \sqrt{\frac{1}{G}\sum_{j=1}^{G}\big(R(\tau_j) - \mu\big)^2 + \varepsilon}. \tag{9}$$

The policy is then updated using the GRPO clipped surrogate objective. Let $a_{i,t}$ denote the t-th token in trajectory τ_i, and define the importance ratio

$$r_{i,t}(\theta) = \frac{\pi_\theta(a_{i,t} \mid a_{i,<t}, x)}{\pi_{\text{ref}}(a_{i,t} \mid a_{i,<t}, x)}.$$

Since the reward is terminal, the same advantage $\hat{A}_i$ is assigned to all tokens in τ_i. The resulting GRPO objective is

$$\mathcal{L}_{\text{GRPO}}(\theta) = \mathbb{E}_x\left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|\tau_i|}\sum_{t=1}^{|\tau_i|} \min\Big(r_{i,t}(\theta)\hat{A}_i,\ \text{clip}(r_{i,t}(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_i\Big)\right]. \tag{10}$$

In summary, the hierarchical verified reward provides a stage-structured scalar signal for each generated OpenQASM trajectory. This signal is normalized within each prompt-level group and directly optimized through the GRPO objective.

A.3 Additional Details of Budget-Aware Verifier Scheduling

This appendix provides additional details for the budget-aware verifier scheduling policy described in §5, including the stage-wise escalation logic, implementation-oriented thresholding, and pseudocode-style summary for reproducibility.

Stage-wise scheduling objective. Given a candidate artifact a, the verifier exposes a partial report after each stage,

$$\rho_{\leq k} = \{(s_i, c_i)\}_{i=1}^{k},$$

where s_i is the observed stage-i semantic signal and c_i is the cost of obtaining that signal. The scheduling problem is to decide, after each stage, whether deeper evaluation is warranted:

$$\pi_{\text{gate}}(\rho_{\leq k}) \in \{\texttt{stop}, \texttt{continue}\}.$$

In our implementation, this policy is realized using threshold-based escalation over partial verifier outcomes. We do not claim that this policy is globally optimal; rather, it is a practical approximation to budget-aware verifier allocation for staged semantic evaluation.

Concrete escalation logic. In the quantum instantiation used in this paper, the verifier proceeds through four semantic stages: Feasibility, Behavior, Objective, and Utility.

- **Feasibility stage.** Every generated candidate is first checked for executability and structural validity. If parsing or basic structural checks fail, verification terminates immediately and the candidate is rejected from deeper evaluation.
- **Behavior stage.** If the candidate is feasible, the verifier computes the Behavior-stage signal based on distributional alignment. This stage acts as the first semantic screen.
- **Objective stage.** Objective evaluation is invoked only when the earlier-stage report indicates that the candidate is sufficiently promising. In practice, this is implemented using a threshold on the Behavior-stage score.
- **Utility stage.** Utility evaluation is the most expensive stage because it requires downstream local refinement. It is therefore reserved for candidates that have already demonstrated sufficiently strong semantic evidence from the earlier stages.

Threshold-based policy form. A generic threshold realization of the gating policy can be written as

$$\pi_{\text{gate}}(\rho_{\le k}) = \begin{cases} \texttt{continue}, & \text{if } g_k(\rho_{\le k}) \ge \delta_k, \\ \texttt{stop}, & \text{otherwise}, \end{cases}$$

where $g_k(\rho_{\le k})$ is a stage-specific scoring function over the partial verifier report and δ_k is the corresponding escalation threshold. In our setting, the first useful threshold is applied after the Behavior stage, since Feasibility is always executed.

Implementation-oriented instantiation. A concrete realization consistent with our system is

$$\texttt{continue to Objective} \iff s_2(\tau_i) \ge \delta_{\text{beh}},$$

where $s_2(\tau_i)$ is the Behavior-stage score and δ_{beh} is a predefined escalation threshold.

Utility evaluation can then be written as

$$\texttt{continue to Utility} \iff \left(s_2(\tau_i) \ge \delta'_{\text{beh}}\right) \vee \left(s_3(\tau_i) \ge \delta_{\text{obj}}\right),$$

where $s_3(\tau_i)$ is the Objective-stage score, and δ'_{beh} and δ_{obj} are thresholds used to decide whether downstream refinement is worth its cost. Intuitively, this rule allows Utility evaluation to proceed for candidates that are either already behaviorally strong or have demonstrated sufficiently good task-level quality.

Why the thresholds are stage-aligned. The threshold policy is designed to follow the semantics of the verifier stages rather than impose an external priority rule. Feasibility is mandatory because deeper semantics are undefined for invalid programs. Behavior is the first screening signal because it is bounded, comparatively stable, and cheaper than downstream optimization-aware evaluation. Objective is more task-specific and more expensive, so it is evaluated selectively. Utility is the most deployment-specific and expensive signal, so it is invoked only when the earlier stages already justify that cost.

Suggested pseudocode structure. For clarity, the scheduling logic can be summarized as follows:

Input: generated candidate τ_i
1. Run Feasibility checks. If infeasible, stop.
2. Compute Behavior-stage score $s_2(\tau_i)$.
3. If $s_2(\tau_i) < \delta_{\text{beh}}$, stop.
4. Compute Objective-stage score $s_3(\tau_i)$.
5. If neither $s_2(\tau_i) \ge \delta'_{\text{beh}}$ nor $s_3(\tau_i) \ge \delta_{\text{obj}}$, stop.
6. Run Utility-stage evaluation and compute $R_{\text{opt}}(\tau_i)$.
Output: partial or full verifier report ρ.

Threshold robustness and calibration. The exact threshold values are implementation parameters and should be treated as part of the scheduling policy rather than as universal constants. Sensitivity analysis or a quality-versus-cost frontier can be used to show that the policy is not brittle and that it exposes a controllable trade-off between verifier budget and semantic supervision strength.

A.4 Training Data, Ground Truth, and Hyperparameters

Training data. We utilize the training data from [21], one of the most extensive available datasets of quantum circuits in QASM format covering 12 central optimization problem primitives on graphs [23], many of whose abstract formulations appear in [31]. The full dataset contains 13,914 samples, which we split into training and validation sets using a 96/4 ratio. The key characteristics are parameterized circuits with optimized parameters, problem Hamiltonians, and the smallest and largest eigenvalues of the Hamiltonians. The dataset is constructed by generating random graph instances and solving each instance using QAOA, VQE, or adaptive VQE under noiseless simulation. We retain only instances for which optimization converges.

Ground-truth circuits and reward reference. As reference solutions, we use *ground-truth* circuits generated offline using QAOA, VQE, and adaptive VQE with numerically optimized parameters for each problem instance [21]. These circuits are obtained by minimizing the problem Hamiltonian within a fixed variational ansatz (1–4 layers for QAOA/VQE or a finite gate pool for adaptive VQE) using classical optimizers, and are exported in OpenQASM 3.0 format. While these solutions achieve high-probability correct measurements under noiseless simulation, they are not guaranteed to be globally optimal: optimization is restricted to a chosen ansatz family, may converge to local minima, and does not exhaustively search the space of all possible circuit structures or parameterizations. Consequently, the ground-truth serves as a strong heuristic baseline rather than a provably optimal quantum circuit.

Training settings. We fine-tune a 4B model with GRPO on 128×AMD MI250X GPUs using FSDP [52]. The average training time is 48 hours. Rollouts are executed with vLLM [24] using a temperature of 0.7 and top-p sampling of 0.8. The model is initialized from a supervised fine-tuned checkpoint following Agent-Q [21]. Generated circuits are validated both through simulator-based verification during training and through real-device execution on a 5-qubit superconducting quantum processor.

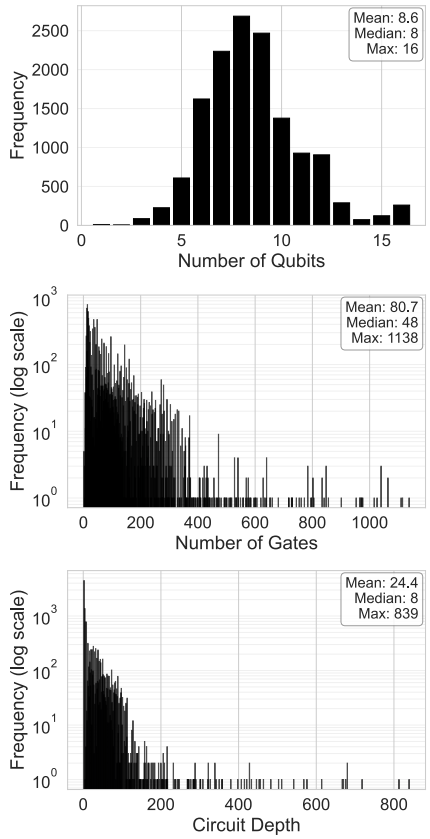


Fig. 15: Dataset statistics: distribution of (a) number of qubits, (b) number of gates, and (c) circuit depth.

Tab. 3: Training settings for tool-augmented agentic RL with a 4B SFT model.

Component	Configuration
Base model	Qwen3-4B-Instruct-2507
SFT setup	Quantum Datasets [21]
SFT learning rate	2×10^{-5}
Rollout Num	16
Temperature	0.7
top_p	0.8
top_k	-1
Token limits	Prompt: 1024; Response: 9216; Observation: 2048; Action: 8192
Optimizer	AdamW [30]
Learning rate	1×10^{-6}
Epochs	10
Batch size	128