Qubrio: High-Performance Quantum Compilation via Multi-Agent LLM Collaboration

Jixuan Ruan[1*] **Zhuo Cui**[1*] **Zhengding Hu**[1] **Zhongkai Yu**[1] **Xiang Fang**[1]
Yue Guan[1] **Jason Ludmir**[2] **Phillip E. Weinberg**[3] **Xiu-Zhe Luo**[3]
Sheng-Tao Wang[3] **Sonia Lopez Alarcon**[4] **Hezi Zhang**[5] **Yufei Ding**[1]
[1]University of California, San Diego [2]Rice University [3]QuEra Computing Inc.
[4]Rochester Institute of Technology [5]University of Wisconsin–Madison

Abstract

Large Language Models (LLMs) offer an adaptable alternative to traditional quantum compilation heuristics, which bottleneck progress by demanding costly manual redesigns whenever hardware evolves. However, naive single-pass LLM approaches fail under the unique challenges introduced by quantum compilation. We introduce Qubrio, a practical agentic framework that significantly reduces heuristic redesigns through a three-level decomposition. By (1) partitioning programs into sequential operation stages, (2) deploying specialized agents for placement, routing and optimization, and (3) using disaggregated feedback for precise error correction, our system reliably navigates complex physical constraints. Beyond serving as a direct compiler, our LLM framework also uncovers novel strategies that facilitate convoy-style shuttling, substantially enhancing hardware concurrency without sacrificing fidelity. We further integrate these strategies into the previous state-of-the-art (SOTA) compiler, recovering a significant fraction of the LLM's performance advantages. On realistic workloads, our compiler achieves significant improvements in hardware runtime and program fidelity over the SOTA baseline PowerMove, seamlessly adapting to new hardware capabilities. Qubrio features an open user interface for practical use, and its code is publicly available at `https://github.com/crosaa/Qubrio`.

1 Introduction

Quantum compilation, the process of translating high-level descriptions of quantum algorithms into hardware-executable instructions, is a key challenge in enabling practical quantum computation [1–5]. Existing quantum compilers [4–7] typically rely on domain-engineered heuristics to handle diverse hardware constraints and capabilities. This reliance, however, creates a significant maintenance burden. Adapting to new hardware features often requires domain experts to redesign heuristics and rewrite substantial portions of the compilation pipeline [1, 4, 7, 8]. This highlights a critical need for compilation frameworks capable of automatically adapting to evolving hardware.

Against this backdrop, LLMs [9, 10] offer a promising solution. Notably, LLMs have demonstrated strong capabilities in program synthesis, code optimization, and reasoning over structured representations [11–14]. This suggests they can learn compilation logic directly from specific hardware descriptions, significantly reducing the need for experts to redesign heuristics. However, whether and how LLMs can work as practical quantum compilers is still largely unexplored.

Empirical observations reveal that naive single-pass LLM compilation fails in practice. We attribute this failure to three orthogonal sources of pressure that any LLM compilation agent must absorb, corresponding to the input, processing, and feedback stages of its operational loop: (i) Input complexity:

[*]Equal contribution.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

A full quantum program exceeds the regime in which current LLMs reason reliably. Current LLMs struggle with long contexts, especially in tasks like quantum compilation where each decision affects all the decisions that follow. (ii) Task heterogeneity: Compilation couples sub-tasks with distinct reasoning modes (e.g., spatial assignment versus sequential trajectory planning). Forcing a single LLM agent to handle these jointly induces cross-task interference, degrading performance on all of them. (iii) Feedback ambiguity: LLMs improve their outputs by iterating on feedback, but feedback that only reports overall performance does not tell the model which constraint was violated and which step caused this violation, leaving the agent without an actionable gradient for repair.

We address the challenges through a three-level decomposition that relieves pressure at each stage of the compilation process. (1) Input decomposition partitions a quantum program into sequential operation stages, so that each stage fits within the context length over which LLMs reason reliably. (2) Task decomposition assigns sub-tasks to specialized agents (placement, routing and optimization) each matched to one reasoning mode, eliminating cross-task interference. (3) Feedback decomposition replaces overall performance scores with structured, multi-dimensional signals: the verifier localizes constraint violations to the offending step, and the simulator disentangles performance into distinct physical error sources, enabling the agent to reason about trade-offs and apply targeted repair.

Furthermore, our LLM framework uncovers novel strategies that facilitate a highly parallel routing pattern, which we term 'convoy-style shuttling'. Unlike previous compilers that route qubits along shortest paths, which frequently induces move conflicts under hardware constraints, the LLM learns to leverage buffer positions to resolve these conflicts and align multiple qubits into synchronized movements. Moreover, the LLM learns to maintain qubits in continuous motion across consecutive routing rounds to minimize associated fidelity penalties introduced by qubit transfer (pick-up and drop) operations. This coordination parallelizes otherwise sequential movement operations, significantly accelerating hardware execution without compromising fidelity. A key insight is that without the bias of a hardcoded heuristic algorithm, the LLM uncovers non-trivial strategies to accelerate hardware execution. This demonstrates our framework's dual value: not just as a practical compiler, but as a systematic tool for discovering novel heuristics.

In summary, our contributions are as follows:

(1) An Adaptable Agentic Compiler Framework: We introduce an agentic quantum compilation framework that significantly reduces the need for manual heuristic rewrites. To overcome the failures of single-pass LLMs, we propose a three-level decomposition framework: partitioning programs into manageable stages, deploying specialized agents for distinct tasks, and providing disaggregated feedback for precise correction and optimization.

(2) LLM-Driven Heuristic Discovery and Convoy-Style Shuttling: Beyond serving as a direct compiler, our framework also uncovers novel strategies that utilize buffer positions and continuous motion to facilitate convoy-style shuttling, substantially enhancing hardware concurrency without sacrificing fidelity. We further integrate these strategies into the classical compiler, recovering a significant fraction of the LLM's performance advantages.

(3) SOTA Efficacy and Hardware Adaptability: Qubrio consistently outperforms the SOTA compiler PowerMove[4], yielding average improvements of 4.7× and 1.3× in hardware runtime and fidelity, and easily adapts to new hardware capabilities.

2 Background

Quantum Compilation Problem. Quantum compilation translates a high-level quantum program, typically represented by a quantum circuit, into executable operations on a quantum processor with hardware constraints [1–5]. Because hardware limits two-qubit interactions to adjacent physical positions, a common strategy is to decompose an n-qubit circuit into a temporal sequence of stages, $\{S_1, \ldots, S_K\}$ [4, 5]. Each stage contains parallelizable two-qubit gates interleaved with single-qubit gates (Fig. 1(a)). To satisfy the hardware constraints, the compiler must solve two coupled optimization tasks per stage: (1) Placement (mapping logical qubits to physical positions) and (2) Routing (moving qubits to adjacent positions for two-qubit gate interaction). Concatenating these local solutions yields a global execution schedule. The goal of the compiler is to generate a valid, executable schedule that minimizes total hardware runtime while maximizing overall fidelity.

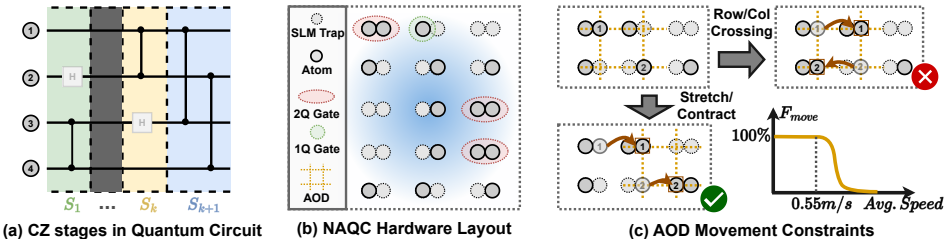


Figure 1: **Quantum circuit structure and neutral atom hardware.** (a) Circuits decompose into sequential stages of parallel CZ interleaved with single-qubit gates. (b) Atoms reside in static SLM and are shuttled into interaction distance for two-qubit gates. (c) Atom movements prohibit row and column crossings, requiring controlled transport velocities to ensure high movement fidelity (F_{move}).

Neutral Atom Quantum Computing. Neutral atom quantum computers (NAQCs) have emerged as one of the leading platforms for scalable quantum computation [15–20], offering an attractive combination of long coherence times [15–18], high-fidelity operations [18, 19, 21–25], and flexible reconfigurability [18, 19]. However, unlocking this potential requires a compiler that can navigate the platform's uniquely demanding routing problem: unlike the localized SWAP operations in superconducting and trapped-ion systems [26, 27], NAQC routing requires global movement planning across the entire qubit array. We instantiate our framework on this platform, as success under its stringent constraints implies broad applicability to simpler routing paradigms. NAQCs execute computations by trapping individual atoms in a 2D grid of static traps generated by a spatial light modulator (SLM) (Fig. 1(b)), and routing them via mobile traps controlled by an acousto-optic deflector (AOD) (Fig. 1(c)) [18, 19]. While single-qubit gates execute in place, executing two-qubit gates requires transferring target atoms from SLMs into AODs and then shuttling them to adjacent traps for interaction [18, 19, 23, 28, 29]. This movement is governed by three critical hardware constraints (Fig. 1(c)): (1) Topological Ordering: The AOD grid can stretch or contract, but rows and columns can never cross [18, 19]. (2) Velocity Bounds: Shuttling speed must be controlled to prevent atom loss [4, 5, 18, 19]. (3) Transfer Penalties: Each SLM–AOD transfer incurs a non-negligible error [18, 19]. Minimizing these transfers often conflicts with minimizing overall routing time, requiring the compiler to carefully navigate this trade-off.

LLM Agents for Structured Reasoning. While LLMs and agentic frameworks [14, 30, 31] have demonstrated strong capabilities in structured reasoning, their naive application to quantum compilation proves inadequate. To reason under strict constraints, general agentic paradigms typically rely on two design patterns: (1) multi-agent decomposition , which allocates subtasks to specialized agents to minimize cross-task interference [11, 32]; and (2) verification-in-the-loop, which couples LLM generation with deterministic checkers to bridge the gap between probabilistic outputs and strict correctness guarantees [13, 33, 34]. However, the unique challenges of quantum compilation render these generic task divisions and standard verify-then-fix loops ineffective, necessitating domain-specific designs.

3 Method

Fig. 2 illustrates the full pipeline, which is explicitly structured around our three-level decomposition strategy. First, to manage input complexity, input decomposition partitions the circuit into manageable CZ stages $\{S_1, \ldots, S_K\}$. Second, to prevent cross-task interference, task decomposition delegates distinct reasoning modes to specialized modules: a Placement Agent (Fig. 2(e)) assigns qubits to trap positions, while a Routing Agent (Fig. 2(f)) generates AOD movement operations. Finally, addressing feedback ambiguity, feedback decomposition is implemented via dedicated verifiers. The Placement Verifier (Fig. 2(b)) and Routing Verifier (Fig. 2(c)) isolate and localize specific constraint violations. Once a stage is successfully placed and routed, an Optimize Agent (Fig. 2(g)) refines the local schedule through a generate–evaluate loop guided by a Hardware Simulator (Fig. 2(d)).

3.1 Placement Agent

The Placement Agent (Fig. 2(e)) computes the qubit-to-trap mapping $P_{k+1} : Q_{k+1} \to T$ for each CZ stage S_{k+1}. Given the hardware specification and the previous mapping P_k, the agent optimizes P_{k+1} to ensure all CZ pairs stay within interaction distance r while maintaining a separation $> r$ for non-interacting pairs to eliminate crosstalk.

Hierarchical verification and feedback. To resolve mapping violations without causing optimization oscillations, the verifier localizes errors into three structured sets of tuples $(q_i, q_j, P_{k+1}(q_i), P_{k+1}(q_j))$, providing the model with entity-level localization:

- Collision (V_{coll}): Qubits assigned to the same trap, where $P_{k+1}(q_i) = P_{k+1}(q_j)$.
- Adjacency (V_{adj}): CZ pairs $(q_i, q_j) \in S_{k+1}$ erroneously placed at distance $> r$.
- Crosstalk (V_{xtalk}): Non-interacting pairs erroneously placed at distance $\leq r$.

Constraints are addressed via a two-stage reasoning pipeline. (1) Stage 1 prioritizes V_{coll} independently to establish a valid topological baseline. Once $V_{coll} = \emptyset$, (2) Stage 2 jointly reports V_{adj} and V_{xtalk}. This joint reporting is critical: these sets impose opposing spatial pressures—adjacency requires proximity, whereas crosstalk requires separation. Addressing them in isolation causes the model to over-correct in a single direction, leading to oscillatory behavior that hinders convergence. By exposing these *conflicting spatial constraints* simultaneously, the model can navigate the trade-off and identify balanced placements that satisfy all requirements.

3.2 Routing Agent

The Routing Agent (Fig. 2(f)) determines a sequence of M discrete AOD operations $\mathcal{A} = \{a_0, \ldots, a_{M-1}\}$ to shuttle atoms for the $(k+1)$-th stage. We denote the mapping after the t-th routing step as P_{k+1}^t, where P_{k+1}^0 is the initial configuration and P_{k+1}^M is the target placement. Each action a_t transitions the system state from P_{k+1}^t to P_{k+1}^{t+1}.

Spatio-temporal verification and feedback. To ensure valid trajectories, the verifier evaluates each state transition $P_{k+1}^t \xrightarrow{a_t} P_{k+1}^{t+1}$ against AOD hardware constraints.

1) Spatio-temporal violation localization. Routing violations are localized to the offending qubits and the specific step t. The verifier produces two structured violation sets of tuples $(q_i, q_j, P_{k+1}^t, P_{k+1}^{t+1}, t)$:

- Collision (V_{coll}): This violation occurs when multiple qubits are assigned to the same trap position after an operation, i.e., $P_{k+1}^{t+1}(q_i) = P_{k+1}^{t+1}(q_j)$.
- Crossing (V_{cross}): This violation represents a breach of the rigid AOD ordering, occurring if action a_t causes the relative row or column indices of q_i and q_j in P_{k+1}^{t+1} to invert compared to their order in P_{k+1}^t.

2) Temporal localization as causal signal. Incorporating the step index t within each violation tuple provides a critical causal signal for trajectory refinement. For instance, a collision at step $t = 2$ indicates that the initial departure strategy is flawed, whereas a collision at $t = M - 1$ suggests that only the final approach to the target P_{k+1} requires adjustment. Without this temporal context, the model lacks the granularity to distinguish between these distinct failure modes and often defaults to redundant global regenerations. By localizing violations to specific transitions $P_{k+1}^t \xrightarrow{a_t} P_{k+1}^{t+1}$, the agent can surgically modify the offending segments while preserving the valid portions of the routing sequence. The complete violation report, $V = V_{coll} \cup V_{cross}$, is serialized into natural language and appended to the agent's context for iterative refinement. Upon successful verification, the final mapping P_{k+1}^M is established as the initial configuration P_{k+2}^0 for the subsequent gate stage S_{k+2}.

3.3 Optimize Agent

The initial compilation phase ensures architectural correctness but does not guarantee an optimal schedule. The Optimize Agent (Fig. 2(g)) iteratively refines the schedule by sampling candidate modifications and evaluating them against the Hardware Simulator (Fig. 2(d)).

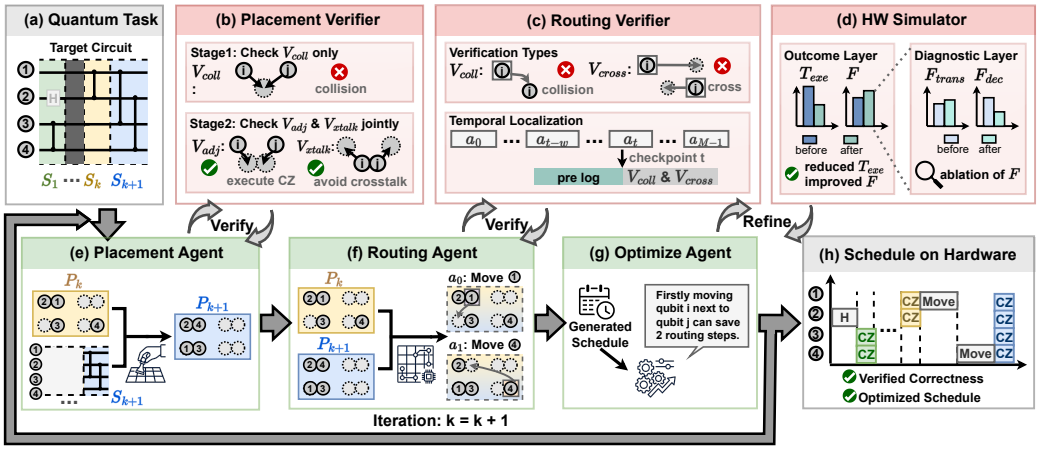


Figure 2: **Multi-agent framework for quantum compilation.** LLM agents (e–g, green) generate placements, routes, and optimizations; deterministic verifiers (b–d, red) enforce constraints and evaluate performance. Agents iteratively refine outputs via structured feedback until all constraints are satisfied, producing a verified, optimized schedule (h).

The pitfall of scalar rewards. The simulator evaluates candidates across four dimensions: hardware runtime (T_{exe}), fidelity (F), transfer penalty (SLM-AOD error), and decoherence (idle error). A conventional approach would collapse these into a single scalar reward. However, T_{exe} and F are coupled via competing physical mechanisms. A faster schedule reduces decoherence but often requires aggressive routing that increases transfer overhead. A scalar reward masks which mechanism dominates, leaving the LLM unable to discern whether a fidelity drop stems from excessive transfers or decoherence. These two failure modes require diametrically opposite corrections.

Outcome-diagnostic decomposition. To resolve this ambiguity, the simulator provides a two-layer feedback structure that exposes the underlying causal chain:

- Outcome Layer: Reports T_{exe} and F to characterize global performance trade-offs.
- Diagnostic Layer: Decouples F into transfer penalty and decoherence components.

This decomposition allows the agent to pinpoint the root cause of performance shifts. For instance, if a faster schedule degrades fidelity, the diagnostic layer can reveal that while decoherence improved, the transfer penalty increased disproportionately. Without this granularity, the LLM might naively attempt to slow down the schedule to improve fidelity. Such a move would further exacerbate decoherence, resulting in the worst possible physical correction. A candidate modification is accepted if and only if it improves at least one of T_{exe} or F without degrading the other. If no candidate in a round meets this criterion, the proposal is rejected and the current best schedule is retained. Crucially, diagnostic feedback from rejected candidates is not discarded. It serves as a negative constraint that narrows the search space for subsequent rounds.

4 Convoy-style Shuttling

Applying the framework described above, we identify a novel routing pattern that substantially improves hardware schedule efficiency, which we call *convoy-style shuttling*. Unlike previous compilers that typically route qubits independently toward their target traps along shortest paths (Fig. 3(b)), our agent groups qubits with spatially compatible trajectories to displace them in a single, coordinated movement (Fig. 3(c)). By analyzing the LLM's thinking logs, we found that the formation of this pattern is primarily driven by two distinct strategies:

(1) Strategic use of buffer positions: Crucially, the agent often utilizes buffer positions to perform subtle local adjustments. By intentionally deviating from individual shortest paths, the agent aligns qubits whose movements would otherwise be incompatible, allowing them to form a shared convoy.

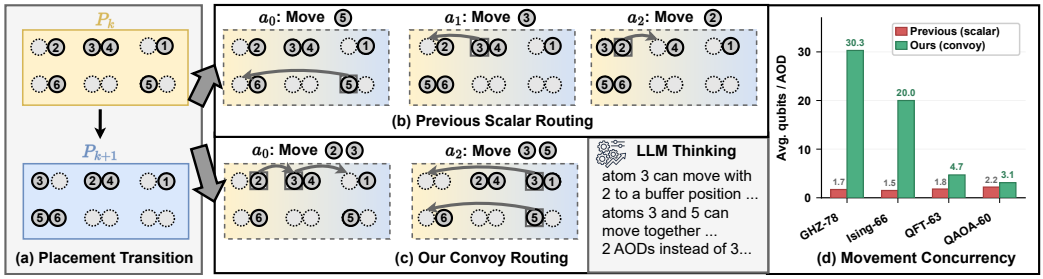


Figure 3: **Convoy routing overview.** (a) Placement transition between two CZ stages. (b) Scalar routing: qubits are moved individually. (c) Convoy routing: multiple qubits are moved in lockstep, reducing hardware runtime and decoherence. (d) Movement concurrency across benchmarks.

Since AOD rows and columns move in tandem, this deliberate synchronization effectively converts idle hardware cycles into concurrent movements without introducing crossing or collision violations.

(2) Minimizing transfer penalties: The agent discovers that maintaining atoms in continuous motion across consecutive AOD rounds avoids the penalties associated with repeated SLM–AOD transfers. Effectively, the agent converts the physical fidelity model into a scheduling constraint that favors sustained convoy membership over naive round-by-round re-optimization.

Together, these two strategies enable a highly coordinated execution that dramatically increases AOD concurrency (Fig. 3(d)), transforming disjointed routing steps into synchronized, penalty-minimizing convoys. Specifically, the gap between convoy and scalar routing is most pronounced on circuits with regular gate structures (GHZ, Ising) where compatible displacement vectors arise naturally. While this advantage reduces on irregular topologies (QFT, QAOA) where spatial locality is scarce, the agent maintains a higher concurrency floor than previous path-finding across all benchmarks.

This discovery is notable in three aspects. (1) The LLM discovers parallelization strategies that facilitate convoy-style shuttling without explicit heuristic guidance. (2) These strategies are non-trivial, requiring joint reasoning over AOD constraints, transfer penalties, and decoherence, a trade-off space that conventional compilers navigate with hand-tuned cost functions. (3) The pattern is directly actionable: the utilization of buffer positions and continuous motion can be extracted as a standalone heuristic for future compilers, illustrating that LLM-driven search can surface optimization opportunities that complement existing domain expertise.

5 Evaluation

5.1 Experiment Setup

LLM configuration. Qubrio does not depend on a specific LLM backbone and uses LLMs only as proposal generators: every candidate placement, route and optimization is accepted only after deterministic verification and simulator evaluation. We evaluate three representative LLM backbones (Qwen-3.5 [35], DeepSeek-V3.2 [36] and GPT-5.4 [37]), spanning closed- and open-source frontier models, to assess backbone sensitivity. Detailed parameters, retry budgets, and run-to-run variance are reported in Appendix F.

Benchmarks. Following the prior work evaluation [4, 5], we select a variety of benchmark programs from QASMBench [38], including QFT, QAOA, GHZ, and Ising, across multiple circuit sizes (up to the order of 10^4 gates). They cover the most widely used algorithmic and state-preparation primitives in near-term quantum computing, spanning a wide range of qubit interaction topologies.

Baselines. We compare against Enola [5] and PowerMove [4], which is the SOTA NA compiler. All compilers are evaluated under the same hardware configurations following prior work [4, 5]. We use PowerMove as the heuristic backbone for our ablation study.

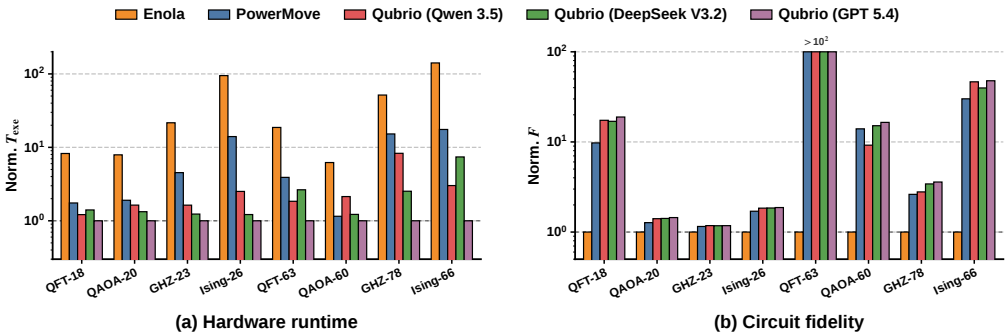


Figure 4: **End-to-end comparison of Qubrio against Enola and PowerMove.** **(a)** Hardware runtime, normalized to Qubrio (GPT-5.4). **(b)** Circuit fidelity, normalized to Enola.

Metrics. We report two primary metrics: (1) *Hardware runtime* (T_{exe}), the end-to-end execution time of the entire circuit on NA hardware, integrating gate durations, atom movement, and SLM–AOD transfer overheads. (2) *Circuit fidelity* (F), the end-to-end probability that a benchmark completes without any logical error, which jointly account for gate fidelity, transfer penalties and decoherence. Both metrics are evaluated using the NA hardware simulator from PowerMove and Enola [4, 5] with identical parameter settings as described in Appendix J. Hardware feasibility is enforced as a hard constraint by the deterministic verifiers. Schedules failing verification are reported as '-'.

5.2 Main Results

Comparison to heuristic compilers. *(1) Hardware runtime.* As shown in Fig. 4(a), Qubrio (GPT-5.4) achieves the lowest hardware runtime across all benchmarks. On ∼20-qubit circuits, Qubrio (GPT-5.4) reduces hardware runtime by an average of 19.1×/3.8× versus Enola/PowerMove. This gap widens at ∼60-qubit circuits, where Qubrio (GPT-5.4) reduces hardware runtime by an average of 30.3×/5.9× versus Enola/PowerMove. The performance advantage stems primarily from convoy-style shuttling, which packs more qubits into each AOD movement round and reduces the total number of rounds (Fig. 3(d)), directly shortening the movement duration. As discussed in Section 4, the variation in performance gap across benchmarks stems from their circuit structural differences. On regular circuits (GHZ, Ising) characterized by local gate interactions, Qubrio (GPT-5.4) achieves an average speedup of $11.4\times$ over PowerMove. On irregular circuits (QFT, QAOA) with non-local and dense interaction patterns, the average speedup reduces to $2.0\times$. Qubrio (DeepSeek-V3.2) and Qubrio (Qwen-3.5) also outperform Enola/PowerMove by an average of 12.7×/2.5× and 10.4×/2.1×. The better performance of both LLM variants suggests that these gains are inherent to the agentic architecture rather than being contingent on the reasoning capabilities of a specific LLM model.

(2) Circuit fidelity. As shown in Fig. 4(b), Qubrio (GPT-5.4) achieves higher fidelity across all benchmarks. The most significant gain is on QFT-63, where massive global routing penalties cause Enola and PowerMove to collapse to 10^{-69} and 10^{-16}. Qubrio recovers this to 10^{-8} by effectively managing dense connectivity through its efficient convoy-style shuttling. Excluding this outlier, Qubrio still improves fidelity by an average of $5.6\times$/$1.30\times$ over Enola/PowerMove (reaching a $9.9\times$ improvement over PowerMove when included). These gains naturally stem from our reduced hardware runtimes and transfer penalties, which minimize the primary sources of hardware errors. Furthermore, Qubrio (DeepSeek-V3.2) and Qubrio (Qwen-3.5) also demonstrate strong performance, improving fidelity by $5.2\times$/$1.22\times$ and $4.9\times$/$1.13\times$ (excluding QFT-63). Across all underlying LLMs, these fidelity improvements match the conclusions drawn from our hardware runtime analysis.

(3) Cost analysis. The financial overhead of LLM-driven compilation is negligible relative to QPU execution expenses and is rapidly amortized by the resulting QPU runtime savings. Specifically, compiling via DeepSeek-V3.2 incurs per-stage costs from \$0.29 (∼20 qubits) to \$0.59 (∼60 qubits). At current NA QPU pricing [39, 40], a standard execution costs roughly \$10. Since QPU usage scales with hardware runtime, the runtime reduction delivered by Qubrio saves a substantial fraction of this cost per execution, so the one-time compilation expense is offset within a few standard executions, after which all subsequent runs yield net savings. By compiling batches of stages in parallel (Appendix I), end-to-end compilation takes a scale-invariant ∼5 minutes. This latency is a

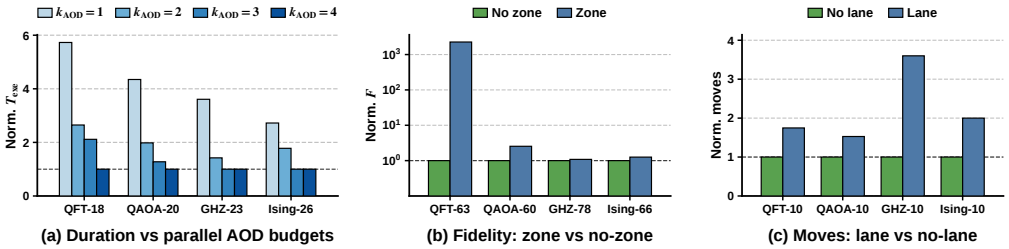


Figure 5: **Adaptability to three hardware capabilities. (a)** Hardware runtime normalized to the $k_{\mathrm{AOD}} = 4$ case across different parallel AOD budgets ($k_{\mathrm{AOD}} \in \{1, 2, 3, 4\}$). **(b)** Fidelity comparison between zoned and non-zoned scenarios, normalized to the non-zoned case.**(c)** Number of atom movements on a lane architecture, normalized to the no-lane case.

one-time cost: the generated routing schedules are reusable, requiring no recompilation for different inputs or local parameter updates [41–43].

Adaptability to three hardware capabilities. Fig. 5 demonstrates Qubrio's adaptability to three hardware capabilities, using DeepSeek-V3.2 as an example backbone model.

(1) Adapt to multi-AOD scenario. Supporting multiple parallel AODs in a heuristic compiler requires extending the router from single-AOD pathfinding to multi-AOD assignment and scheduling, along with rederiving the associated cost model. As shown in Fig. 5(a), a single additional AOD already yields substantial duration reductions across all benchmarks, with further gains as k_{AOD} increases. This demonstrates that the LLM agent can effectively leverage modest hardware additions to substantially improve execution time, without any change to the underlying compilation logic.

(2) Adapt to zoned architecture. Zoned Architectures partition the trap array into spatial regions with distinct operational roles [18]. Previous support requires modifying placement to respect zone boundaries and constraining the router to avoid illegal cross-zone transfers. Fig. 5(b) shows that the fidelity under zoned architectures consistently outperforms that of the non-zoned case across circuits, with a notable $2263\times$ improvement on QFT-63. This gain arises because zone partitioning eliminates atoms' exposure to excitation error during two-qubit gate execution, and the agent exploits this structure to improve fidelity.

(3) Adapt to lane architecture. We further evaluate Qubrio on the recently introduced lane architecture by QuEra [44], which reduces AOD transport calibration overhead by restricting atom movements to a predefined set of calibrated transport lanes. While this design improves hardware control efficiency, the reduced transport flexibility introduces additional routing challenges, as atoms may need multiple hops to reach their destinations. As shown in Fig. 5(c), Qubrio successfully adapts to this more constrained architecture while maintaining reasonable execution performance. Although the number of atom movements increases compared with unrestricted transport, the overhead remains within a manageable range, demonstrating Qubrio's adaptability to hardware architectures with different transport capabilities.

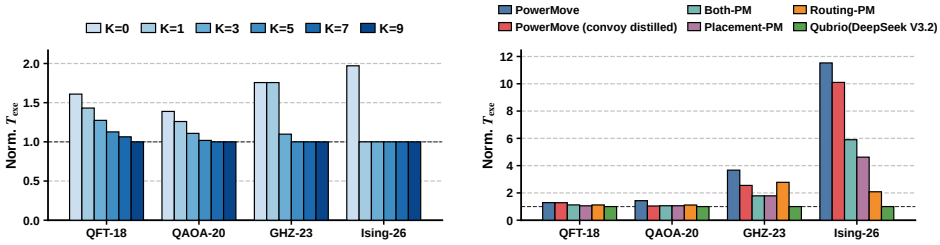
Across these hardware scenarios, adapting Qubrio follows a lightweight and consistent procedure. Specifically, only three components require updates: (1) the hardware description in the LLM prompt, (2) the architecture-specific legality constraints enforced by the verifier, and (3) the hardware parameters and supported operations modeled by the simulator. These changes can be directly derived from the corresponding hardware specifications without modifying the core compilation algorithms, enabling Qubrio to accommodate diverse hardware capabilities with minimal manual effort.

5.3 Ablation Study

Ablation of agent design choices. Table 6(a) isolates the contribution of each architectural decision by comparing four pipeline configurations. (1) Single-Agent, which removes both stage-wise decomposition and the placement-routing separation, fails to produce valid schedules on QFT-18 and QAOA-20, indicating that a single LLM pass overwhelms the context window and conflates distinct reasoning modes. (2) No-Verifier also fails on these two circuits, showing that stage-wise decomposition alone is insufficient without closed-loop constraint correction. On GHZ-23 and

Benchmark	Single-Agent Dur. (µs)	No Verify Dur. (µs)	Merge Placement Routing Dur. (µs)	Multi-Agent Dur. (µs)	Single-Agent Fidelity	No Verify Fidelity	Merge Placement Routing Fidelity	Multi-Agent Fidelity
QFT-18	—	—	38.9k	33.6k	—	—	0.052	0.055
QAOA-20	—	—	5.86k	3.98k	—	—	0.680	0.689
GHZ-23	1.15k	1.14k	968.5	459.0	0.854	0.848	0.857	0.867
Ising-26	1.26k	519.8	704.1	346.4	0.735	0.743	0.742	0.746

(a) Ablation of agent design choices.



(b) Optimization convergence. (c) Ablation of LLM vs. classical modules

Figure 6: **Ablation studies isolating architectural contributions.**

Ising-26 where both do converge, their schedules are 2.4× longer than the full pipeline. (3) Merge Placement Routing, which retains the verifier but merges placement and routing into a single agent, succeeds on all four circuits, demonstrating that the verifier is the critical ingredient for reliability. (4) Multi-Agent, which decouples placement and routing into separate agents, further reduces duration by an average of 1.6× across the board while maintaining comparable fidelity. These results confirm that agent decomposition and the verification mechanism are complementary: the verifier ensures correctness, while decomposition allows agents to reason effectively within focused scopes.

Sensitivity to optimization rounds. Fig. 6(b) shows schedule duration as the number of optimizer turns K is swept over $\{0, 1, 3, 5, 7, 9\}$, min-normalized to K=9. Without optimization (K=0), schedules are on average 1.7× longer. The fifth round (K=5) captures the bulk of the improvement. This convergence suggests that the disaggregated feedback structure enables the LLM to identify high-impact modifications early rather than requiring many rounds of trial and error.

Ablation of LLM vs. heuristic modules. Each LLM module contributes to the overall pipeline performance. As shown in Fig. 6(c), progressively replacing LLM agents with heuristic PowerMove counterparts across five configurations, while retaining the verifier, increases hardware runtime. The LLM advantage is moderate on irregular circuits (QFT-18, QAOA-20) and grows on regular structures, reaching an 11.5× improvement on Ising-26. Furthermore, the routing stage benefits more from LLM reasoning than the placement stage. Comparing the single-replacement variants shows that using a heuristic for routing degrades performance more than using a heuristic for placement on three of the four benchmarks. This is likely because sequential trajectory planning under coupled AOD constraints involves a large combinatorial search space, where the LLM's ability to evaluate multi-step trade-offs provides an advantage over heuristic search. Finally, the discovered convoy-style shuttling strategies demonstrate practical transferability. Integrating these strategies into the PowerMove baseline recovers 38% of the total runtime reduction. The remaining performance gap aligns with our expectations: while hardcoded rules effectively establish coarse-grained parallel routing patterns, they lack the flexibility to handle granular, circuit-specific decisions.

Contribution of LLM Reasoning. To evaluate the contribution of LLM reasoning, we replace both LLM agents in Qubrio with simulated annealing (SA), a standard combinatorial optimizer [45], while retaining the same verifiers and hardware simulator. Fig. 7 compares SA under two simulator-evaluation budgets: SA (1×), which matches Qubrio's evaluation budget, and SA (10,000×), at which SA converges across all benchmarks. We sweep the initial acceptance probability following [45] and report the best configuration for each benchmark.

At the matched evaluation budget, SA achieves only limited improvements over its initial schedule. Even with 10,000× more evaluations, Qubrio consistently achieves lower hardware execution time and higher circuit fidelity across all benchmarks. These results indicate that Qubrio's improvements cannot be attributed solely to additional search or simulator evaluations. Instead, they highlight the

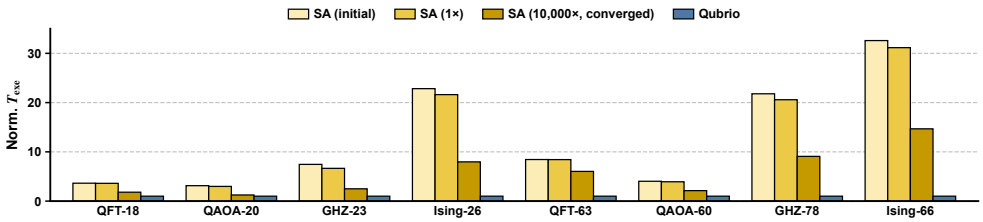


Figure 7: **Comparison with Simulated Annealing Across Different Evaluation Budgets.**

value of LLM-guided reasoning in navigating the coupled placement and routing constraints, where conventional search-based optimization can become trapped in poor local optima [46, 47].

6 Related Work

Quantum Compilation. Existing frameworks address qubit routing across diverse platforms, including superconducting devices [27, 48–54], photonics [55–57], ion traps [26, 58–60], and multi-chip systems [61–64]. Among these, NA architectures present a uniquely challenging compilation target [4–7, 65–67]. NA qubits are shuttled via AODs restricted by non-crossing constraints [18, 19], creating a globally coupled trajectory planning problem governed by velocity limits and transfer-decoherence trade-offs. Current NA compilers [4, 5] navigate this space using hand-engineered heuristics, meaning adaptations for hardware features like parallel AODs require redesigning routing logic and cost models [4].

Machine Learning for Quantum Compilation. Recent ML advances in quantum compilation, such as RL for SWAP routing [68–70] and GNNs for qubit allocation [71, 72], are tailored for superconducting platforms where routing is local on fixed graphs and single-step errors are not strictly fatal. In contrast, NA hardware introduces a non-local, highly constrained routing paradigm. A single AOD operation shifts multiple atoms simultaneously, enforcing rigid no-crossing constraints across all trajectories. This global mechanical linkage invalidates standard ML: RL suffers from prohibitively sparse rewards since a single collision breaks the entire trajectory, while GNNs' local message-passing fails to capture non-local AOD row and column dependencies. Therefore, NA routing demands a paradigm capable of global reasoning and complex constraint satisfaction rather than step-by-step local optimization. While LLMs possess the requisite advanced reasoning for such tasks, existing quantum LLM research sidesteps hardware constraints to focus on algorithmic circuit generation [73–75] instead of compilation onto hardware. No existing ML-based method has demonstrated end-to-end compilation for NA hardware, and state-of-the-art NA compilers rely mostly on hand-engineered heuristics [4, 5].

7 Conclusion

We presented Qubrio, an agentic framework for NA quantum compilation. By deploying specialized agents grounded by deterministic verifiers and disentangled simulator feedback, Qubrio resolves core LLM bottlenecks in quantum compilation. Consequently, Qubrio delivers average improvements of 4.7× and 1.3× in hardware runtime and fidelity over the SOTA compiler. The framework also seamlessly adapts to parallel AODs, zoned architectures and the laned architecture without algorithmic changes and discovers novel strategies that facilitate convoy-style shuttling, substantially enhancing hardware concurrency without sacrificing fidelity. Current evaluations focus on NA hardware. Future work could be empirically validating this framework on other quantum platforms.

8 Acknowledge

This work was supported in part by the U.S. National Science Foundation (NSF) under Grants No. 2552449, 2534655, 2547483, 2427109, and 2422169, and the U.S. Department of Energy (DOE) under Award No. DE-SC0026692. We also thank QuEra Computing for their valuable guidance on hardware-related aspects of this work.

References

[1] Chenghong Zhu, Xian Wu, Zhaohui Yang, Jingbo Wang, Anbang Wu, Shenggen Zheng, and Xin Wang. Quantum compiler design for qubit mapping and routing: A cross-architectural survey of superconducting, trapped-ion, and neutral atom systems. *arXiv preprint arXiv:2505.16891*, 2025.

[2] Marco Maronese, Lorenzo Moro, Lorenzo Rocutto, and Enrico Prati. Quantum compiling. In *Quantum Computing Environments*, pages 39–74. Springer, 2022.

[3] Ge Yan, Wenjie Wu, Yuheng Chen, Kaisen Pan, Xudong Lu, Zixiang Zhou, Yuhan Wang, Ruocheng Wang, and Junchi Yan. Quantum circuit synthesis and compilation optimization: Overview and prospects. *arXiv preprint arXiv:2407.00736*, 2024.

[4] Jixuan Ruan, Xiang Fang, Hezi Zhang, Ang Li, Travis Humble, and Yufei Ding. Powermove: Optimizing compilation for neutral atom quantum computers with zoned architecture. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 163–178, 2025.

[5] Daniel Bochen Tan, Wan-Hsuan Lin, and Jason Cong. Compilation for dynamically field-programmable qubit arrays with efficient and provably near-optimal scheduling. *arXiv e-prints*, pages arXiv–2405, 2024.

[6] Daniel Bochen Tan, Dolev Bluvstein, Mikhail D Lukin, and Jason Cong. Compiling quantum circuits for dynamically field-programmable neutral atoms array processors. *Quantum*, 8:1281, 2024.

[7] Hanrui Wang, Pengyu Liu, Daniel Bochen Tan, Yilian Liu, Jiaqi Gu, David Z Pan, Jason Cong, Umut A Acar, and Song Han. Atomique: A quantum compiler for reconfigurable neutral atom arrays. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pages 293–309. IEEE, 2024.

[8] Medina Bandic, Sebastian Feld, and Carmen G Almudever. Full-stack quantum computing systems in the nisq era: algorithm-driven and hardware-aware compilation techniques. In *2022 design, automation & test in Europe conference & exhibition (DATE)*, pages 1–6. IEEE, 2022.

[9] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[10] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[11] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, et al. Metagpt: Meta programming for a multi-agent collaborative framework. In *The twelfth international conference on learning representations*, 2023.

[12] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nat.*, 625(7995):468–475, January 2024. URL `http://dblp.uni-trier.de/db/journals/nature/nature625.html#RomeraParedesBNBKDREWFKF24`.

[13] Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023. URL `https://arxiv.org/abs/2303.11366`.

[14] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[15] Sepehr Ebadi, Tout T Wang, Harry Levine, Alexander Keesling, Giulia Semeghini, Ahmed Omran, Dolev Bluvstein, Rhine Samajdar, Hannes Pichler, Wen Wei Ho, et al. Quantum phases of matter on a 256-atom programmable quantum simulator. *Nature*, 595(7866):227–232, 2021.

[16] TM Graham, Y Song, J Scott, C Poole, L Phuttitarn, K Jooya, P Eichler, X Jiang, A Marra, B Grinkemeyer, et al. Multi-qubit entanglement and algorithms on a neutral-atom quantum computer. *Nature*, 604(7906):457–462, 2022.

[17] Jonathan Wurtz, Alexei Bylinskii, Boris Braverman, Jesse Amato-Grill, Sergio H Cantu, Florian Huber, Alexander Lukin, Fangli Liu, Phillip Weinberg, John Long, et al. Aquila: Quera's 256-qubit neutral-atom quantum computer. *arXiv preprint arXiv:2306.11727*, 2023.

[18] Dolev Bluvstein, Simon J Evered, Alexandra A Geim, Sophie H Li, Hengyun Zhou, Tom Manovitz, Sepehr Ebadi, Madelyn Cain, Marcin Kalinowski, Dominik Hangleiter, et al. Logical quantum processor based on reconfigurable atom arrays. *Nature*, 626(7997):58–65, 2024.

[19] Dolev Bluvstein, Harry Levine, Giulia Semeghini, Tout T Wang, Sepehr Ebadi, Marcin Kalinowski, Alexander Keesling, Nishad Maskara, Hannes Pichler, Markus Greiner, et al. A quantum processor based on coherent transport of entangled atom arrays. *Nature*, 604(7906): 451–456, 2022.

[20] Dolev Bluvstein, Alexandra A Geim, Sophie H Li, Simon J Evered, J Ataides, Gefen Baranes, Andi Gu, Tom Manovitz, Muqing Xu, Marcin Kalinowski, et al. Architectural mechanisms of a universal fault-tolerant quantum computer. *arXiv preprint arXiv:2506.20661*, 2025.

[21] Simon J. Evered, Muqing Xu, Sophie H. Li, Alexandra A. Geim, J. Pablo Bonilla Ataides, Marcin Kalinowski, Dolev Bluvstein, Nishad Maskara, Christian Kokail, Markus Greiner, Vladan Vuletić, and Mikhail D. Lukin. High-fidelity entangling gates and nonlocal circuits with neutral atoms, 2026. URL `https://arxiv.org/abs/2604.25987`.

[22] Harry Levine, Dolev Bluvstein, Alexander Keesling, Tout T Wang, Sepehr Ebadi, Giulia Semeghini, Ahmed Omran, Markus Greiner, Vladan Vuletić, and Mikhail D Lukin. Dispersive optical systems for scalable raman driving of hyperfine qubits. *Physical Review A*, 105(3): 032618, 2022.

[23] Harry Levine, Alexander Keesling, Giulia Semeghini, Ahmed Omran, Tout T Wang, Sepehr Ebadi, Hannes Bernien, Markus Greiner, Vladan Vuletić, Hannes Pichler, et al. Parallel implementation of high-fidelity multiqubit gates with neutral atoms. *Physical review letters*, 123(17):170503, 2019.

[24] Zhuo Fu, Peng Xu, Yuan Sun, Yang-Yang Liu, Xiao-Dong He, Xiao Li, Min Liu, Run-Bing Li, Jin Wang, Liang Liu, et al. High-fidelity entanglement of neutral atoms via a rydberg-mediated single-modulated-pulse controlled-phase gate. *Physical Review A*, 105(4):042430, 2022.

[25] Simon J Evered, Dolev Bluvstein, Marcin Kalinowski, Sepehr Ebadi, Tom Manovitz, Hengyun Zhou, Sophie H Li, Alexandra A Geim, Tout T Wang, Nishad Maskara, et al. High-fidelity parallel entangling gates on a neutral-atom quantum computer. *Nature*, 622(7982):268–272, 2023.

[26] Chenghong Zhu, Xian Wu, Jingbo Wang, and Xin Wang. S-sync: Shuttle and swap co-optimization in quantum charge-coupled devices. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, pages 271–284, 2025.

[27] Gushu Li, Yufei Ding, and Yuan Xie. Tackling the qubit mapping problem for nisq-era quantum devices. In *Proceedings of the twenty-fourth international conference on architectural support for programming languages and operating systems*, pages 1001–1014, 2019.

[28] Dieter Jaksch, Juan Ignacio Cirac, Peter Zoller, Steve L Rolston, Robin Côté, and Mikhail D Lukin. Fast quantum gates for neutral atoms. *Physical Review Letters*, 85(10):2208, 2000.

[29] E Urban, Todd A Johnson, T Henage, L Isenhower, DD Yavuz, TG Walker, and M Saffman. Observation of rydberg blockade between two atoms. *Nature Physics*, 5(2):110–114, 2009.

[30] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

[31] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[32] Qian Chen, Cong Xin, Yang Cheng, Chen Weize, Yusheng Su, Juyuan Xu, Liu Zhiyuan, and Sun Maosong. Communicative agents for software development. *arXiv preprint arXiv: 2307.07924 v4 [cs. SE]*, 2023.

[33] Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug, 2023. URL `https://arxiv.org/abs/2304.05128`.

[34] Pranjal Aggarwal, Bryan Parno, and Sean Welleck. Alphaverus: Bootstrapping formally verified code generation through self-improving translation and treefinement. *arXiv preprint arXiv:2412.06176*, 2024.

[35] Qwen Team. Qwen3.5-omni technical report, 2026. URL `https://arxiv.org/abs/2604.15804`.

[36] DeepSeek-AI, Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenhao Xu, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Erhang Li, Fangqi Zhou, Fangyun Lin, Fucong Dai, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Hao Li, Haofen Liang, Haoran Wei, Haowei Zhang, Haowen Luo, Haozhe Ji, Honghui Ding, Hongxuan Tang, Huanqi Cao, Huazuo Gao, Hui Qu, Hui Zeng, Jialiang Huang, Jiashi Li, Jiaxin Xu, Jiewen Hu, Jingchang Chen, Jingting Xiang, Jingyang Yuan, Jingyuan Cheng, Jinhua Zhu, Jun Ran, Junguang Jiang, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Kexin Huang, Kexing Zhou, Kezhao Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Wang, Liang Zhao, Liangsheng Yin, Lihua Guo, Lingxiao Luo, Linwang Ma, Litong Wang, Liyue Zhang, M. S. Di, M. Y Xu, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Panpan Huang, Peixin Cong, Peiyi Wang, Qiancheng Wang, Qihao Zhu, Qingyang Li, Qinyu Chen, Qiushi Du, Ruiling Xu, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runqiu Yin, Runxin Xu, Ruomeng Shen, Ruoyu Zhang, S. H. Liu, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaofei Cai, Shaoyuan Chen, Shengding Hu, Shengyu Liu, Shiqiang Hu, Shirong Ma, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, Songyang Zhou, Tao Ni, Tao Yun, Tian Pei, Tian Ye, Tianyuan Yue, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjie Pang, Wenjing Luo, Wenjun Gao, Wentao Zhang, Xi Gao, Xiangwen Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaokang Zhang, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xingyou Li, Xinyu Yang, Xinyuan Li, Xu Chen, Xuecheng Su, Xuehai Pan, Xuheng Lin, Xuwei Fu, Y. Q. Wang, Yang Zhang, Yanhong Xu, Yanru Ma, Yao Li, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Qian, Yi Yu, Yichao Zhang, Yifan Ding, Yifan Shi, Yiliang Xiong, Ying He, Ying Zhou, Yinmin Zhong, Yishi Piao, Yisong Wang, Yixiao Chen, Yixuan Tan, Yixuan Wei, Yiyang Ma, Yiyuan Liu, Yonglun Yang, Yongqiang Guo, Yongtong Wu, Yu Wu, Yuan Cheng, Yuan Ou, Yuanfan Xu, Yuduan Wang, Yue Gong, Yuhan Wu, Yuheng Zou, Yukun Li, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehua Zhao, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhixian Huang, Zhiyu Wu, Zhuoshu Li, Zhuping Zhang, Zian Xu, Zihao Wang, Zihui Gu, Zijia Zhu, Zilin Li, Zipeng Zhang, Ziwei Xie, Ziyi Gao, Zizheng Pan, Zongqing Yao, Bei Feng, Hui Li, J. L. Cai, Jiaqi Ni, Lei Xu, Meng Li, Ning Tian, R. J. Chen, R. L. Jin, S. S. Li, Shuang Zhou, Tianyu Sun, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xinnan Song, Xinyi Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, Dongjie Ji, Jian Liang, Jianzhong Guo, Jin Chen, Leyi Xia, Miaojun Wang, Mingming Li, Peng Zhang, Ruyi Chen, Shangmian Sun, Shaoqing Wu, Shengfeng Ye, T. Wang, W. L. Xiao, Wei An, Xianzu Wang, Xiaowen Sun, Xiaoxiang Wang, Ying Tang, Yukun Zha, Zekai Zhang, Zhe Ju, Zhen Zhang, and Zihua Qu. Deepseek-v3.2: Pushing the frontier of open large language models, 2025. URL `https://arxiv.org/abs/2512.02556`.

[37] OpenAI. Introducing GPT-5.4. https://openai.com/index/introducing-gpt-5-4/, March 2026. Accessed: 2026-05-04.

[38] Ang Li, Samuel Stein, Sriram Krishnamoorthy, and James Ang. Qasmbench: A low-level quantum benchmark suite for nisq evaluation and simulation. *ACM Transactions on Quantum Computing*, 4(2):1–26, 2023.

[39] Darrell Teegarden, Allison Casey, F. Gino Serpa, Patrick Becker, Asmita Brahme, Saanvi Kataria, and Paul Lopata. Three months in the life of cloud quantum computing, 2026. URL https://arxiv.org/abs/2601.09943.

[40] The Quantum Insider. What is the price of a quantum computer in 2025? https://thequantuminsider.com/2025/12/08/what-is-the-price-of-a-quantum-computer-in-2025/, 2025. Accessed: 2026-04-29.

[41] Yilun Zhao, Yu Chen, Kaiyan Chang, He Li, Bing Li, Yinhe Han, and Ying Wang. A system architecture for low latency multiprogramming quantum computing. *arXiv preprint arXiv:2601.01158*, 2026.

[42] Anna M Krol, Koen Mesman, Aritra Sarkar, and Zaid Al-Ars. Efficient parameterised compilation for hybrid quantum programming. In *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, volume 2, pages 103–111. IEEE, 2023.

[43] Aniket S Dalvi, Jacob Whitlow, Marissa D'Onofrio, Leon Riesebos, Tianyi Chen, Samuel Phiri, Kenneth R Brown, and Jonathan M Baker. One-time compilation of device-level instructions for quantum subroutines. In *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*, volume 1, pages 873–884. IEEE, 2024.

[44] Xiu-Zhe Luo, Phillip Weinberg, and Shengtao Wang. Building the software stack for fault-tolerant neutral atom computers. Bloqade Blog, QuEra Computing, August 2026. URL https://bloqade.quera.com/blog/building-the-software-stack-for-fault-tolerant-neutral-atom-computers/.

[45] Emile Aarts. Simulated annealing and boltzmann machines. *IEEE Control Syst. Mag.*, 9:31–37, 1989.

[46] Zhaohui Yang, Kai Zhang, Xinyang Tian, Xiangyu Ren, Yingjian Liu, Yunfeng Li, Dawei Ding, Jianxin Chen, and Yuan Xie. Unifying qubit routing across diverse quantum isas via canonical representation. In *2026 ACM/IEEE 53rd Annual International Symposium on Computer Architecture (ISCA)*, pages 2302–2317. IEEE, 2026.

[47] Henry Zou, Matthew Treinish, Kevin Hartman, Alexander Ivrii, and Jake Lishman. Lightsabre: A lightweight and enhanced sabre algorithm. *arXiv preprint arXiv:2409.08368*, 2024.

[48] 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, et al. Quantum computing with qiskit. *arXiv preprint arXiv:2405.08810*, 2024.

[49] Lukas Burgholzer, Yannick Stade, Tom Peham, and Robert Wille. Mqt core: The backbone of the munich quantum toolkit (mqt). *Journal of Open Source Software*, 10(108):7478, 2025.

[50] Alexander Cowtan, Silas Dilkes, Ross Duncan, Alexandre Krajenbrink, Will Simmons, and Seyon Sivarajah. On the qubit routing problem. *arXiv preprint arXiv:1902.08091*, 2019.

[51] Alwin Zulehner, Alexandru Paler, and Robert Wille. An efficient methodology for mapping quantum circuits to the ibm qx architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(7):1226–1236, 2018.

[52] Bochen Tan and Jason Cong. Optimal layout synthesis for quantum computing. In *Proceedings of the 39th International Conference on Computer-Aided Design*, pages 1–9, 2020.

[53] Santanu Pattanayak. *Quantum machine learning with Python: Using Cirq from Google research and IBM Qiskit*. Springer, 2021.

[54] Seyon Sivarajah, Silas Dilkes, Alexander Cowtan, Will Simmons, Alec Edgington, and Ross Duncan. t| ket>: a retargetable compiler for nisq devices. *Quantum Science & Technology*, 6(1): 014003, 2021.

[55] Hezi Zhang, Anbang Wu, Yuke Wang, Gushu Li, Hassan Shapourian, Alireza Shabani, and Yufei Ding. Oneq: A compilation framework for photonic one-way quantum computation. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, pages 1–14, 2023.

[56] Hezi Zhang, Jixuan Ruan, Hassan Shapourian, Ramana Rao Kompella, and Yufei Ding. Oneperc: A randomness-aware compiler for photonic quantum computing. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 738–754, 2024.

[57] Hezi Zhang, Jixuan Ruan, Dean Tullsen, Yufei Ding, Ang Li, and Travis S Humble. Oneadapt: Adaptive compilation for resource-constrained photonic one-way quantum computing. *arXiv preprint arXiv:2504.17116*, 2025.

[58] Prakash Murali, Dripto M Debroy, Kenneth R Brown, and Margaret Martonosi. Architecting noisy intermediate-scale trapped ion quantum computers. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 529–542. IEEE, 2020.

[59] Jixuan Ruan, Hezi Zhang, Xiang Fang, Ang Li, Wesley C Campbell, Eric Hudson, David Hayes, Hartmut Haeffner, Travis Humble, Jens Palsberg, et al. Trapsimd: Simd-aware compiler optimization for 2d trapped-ion quantum machines. *arXiv preprint arXiv:2504.17886*, 2025.

[60] Bao Bach, Ilya Safro, and Ed Younis. Efficient compilation for shuttling trapped-ion machines via the position graph architectural abstraction. *arXiv preprint arXiv:2501.12470*, 2025.

[61] Sergey Bravyi, Oliver Dial, Jay M Gambetta, Darío Gil, and Zaira Nazario. The future of quantum computing with superconducting qubits. *Journal of Applied Physics*, 132(16), 2022.

[62] Jianwei Wang, Damien Bonneau, Matteo Villa, Joshua W Silverstone, Raffaele Santagati, Shigehito Miki, Taro Yamashita, Mikio Fujiwara, Masahide Sasaki, Hirotaka Terai, et al. Chip-to-chip quantum photonic interconnect by path-polarization interconversion. *Optica*, 3(4): 407–413, 2016.

[63] Hezi Zhang, Keyi Yin, Anbang Wu, Hassan Shapourian, Alireza Shabani, and Yufei Ding. Mech: Multi-entry communication highway for superconducting quantum chiplets. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 699–714, 2024.

[64] Anbang Wu, Hezi Zhang, Gushu Li, Alireza Shabani, Yuan Xie, and Yufei Ding. Autocomm: A framework for enabling efficient communication in distributed quantum programs. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1027–1041. IEEE, 2022.

[65] Hanrui Wang, Daniel Bochen Tan, Pengyu Liu, Yilian Liu, Jiaqi Gu, Jason Cong, and Song Han. Q-pilot: Field programmable qubit array compilation with flying ancillas. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, pages 1–6, 2024.

[66] Bochen Tan, Dolev Bluvstein, Mikhail D Lukin, and Jason Cong. Qubit mapping for reconfigurable atom arrays. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, pages 1–9, 2022.

[67] Wan-Hsuan Lin, Daniel Bochen Tan, and Jason Cong. Reuse-aware compilation for zoned quantum architectures based on neutral atoms, 2024. URL `https://arxiv.org/abs/2411.11784`.

[68] Matteo G Pozzi, Steven J Herbert, Akash Sengupta, and Robert D Mullins. Using reinforcement learning to perform qubit routing in quantum compilers. *ACM Transactions on Quantum Computing*, 3(2):1–25, 2022.

[69] Wei Tang, Yiheng Duan, Yaroslav Kharkov, Rasool Fakoor, Eric Kessler, and Yunong Shi. Alpharouter: Quantum circuit routing with reinforcement learning and tree search. In *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*, volume 1, pages 930–940. IEEE, 2024.

[70] Hoang M Ngo, Tamer Kahveci, and My T Thai. Fiddle: Reinforcement learning for quantum fidelity enhancement. *ACM Transactions on Quantum Computing*, 7(1):1–28, 2025.

[71] Travis LeCompte, Fang Qi, Xu Yuan, Nian-Feng Tzeng, M Hassan Najafi, and Lu Peng. Graph neural network assisted quantum compilation for qubit allocation. In *Proceedings of the Great Lakes Symposium on VLSI 2023*, pages 415–419, 2023.

[72] Sokea Sang, Leanghok Hour, and Youngsun Han. Learning-optimized qubit mapping and reuse to minimize inter-core communication in modular quantum architectures, 2026. URL `https://arxiv.org/abs/2506.09323`.

[73] Linus Jern, Valter Uotila, Cong Yu, and Bo Zhao. Agent-q: fine-tuning large language models for quantum circuit generation and optimization. In *2025 IEEE International Conference on Quantum Computing and Engineering (QCE)*, volume 1, pages 1621–1632. IEEE, 2025.

[74] Zhiding Liang, Jinglei Cheng, Rui Yang, Hang Ren, Zhixin Song, Di Wu, Xuehai Qian, Tongyang Li, and Yiyu Shi. Unleashing the potential of llms for quantum computing: A study in quantum architecture design. *arXiv preprint arXiv:2307.08191*, 2023.

[75] Pranav Sinha, Sumit Kumar Jha, and Sunny Raj. Circuit partitioning using large language models for quantum compilation and simulations. *arXiv preprint arXiv:2505.07711*, 2025.

[76] Haotian Ye, Haowei Lin, Jingyi Tang, Yizhen Luo, Caiyin Yang, Chang Su, Rahul Thapa, Rui Yang, Ruihua Liu, Zeyu Li, et al. Evaluation-driven scaling for scientific discovery. *arXiv preprint arXiv:2604.19341*, 2026.

A Agent Prompts and Output Schemas

This appendix summarizes the prompt templates used by Qubrio's LLM agents and connects the abstract agent descriptions in the main text to their implementation. The implementation contains four prompt templates: initial placement generation, stage placement generation, AOD transition routing, and schedule optimization. These templates correspond to the three agents described in the main text: the Placement Agent, Routing Agent, and Optimize Agent. Across all templates, the LLM is used as a proposal generator; generated placements, routes, and optimized schedules are accepted only after deterministic validation.

Placement Agent. Qubrio uses two placement prompts. The first generates the initial qubit-to-trap mapping before any routing is required, while the second generates the target layout for each subsequent CZ stage. Both prompts output a complete logical-to-physical mapping and are checked by the Placement Verifier before routing begins.

Initial placement. The initial placement prompt specifies the hardware grid width, height, and number of qubits. Trap coordinates are integer lattice points, each physical trap can host at most one atom, and each gate site consists of a left trap and a right trap defined by

$$\text{group_id}(x, y) = \text{HEIGHT} \cdot y + \lfloor x/2 \rfloor.$$

The prompt provides the first CZ stage and asks the model to assign every qubit to exactly one valid trap coordinate. The main optimization instruction is dense clustering: atoms should be packed into a compact rectangular or square block, rather than spread randomly across the entire grid, to reduce future movement distance.

The required output schema is:

```
{
  "initial_layout": {
    0: (x, y),
    1: (x, y),
    ...
  }
}
```

Stage placement. The stage placement prompt receives the current layout, the next CZ stage, and a lookahead list of future CZ stages. For the immediate stage, each CZ pair must occupy the same gate site, with one atom in the left trap and one atom in the right trap. This same-gate-site condition is the implementation-level realization of the interaction-distance constraint used in the main text. The prompt also allows optional prefetching: qubits that interact in future stages may be moved closer in the current stage, as long as the immediate CZ constraints remain satisfied.

The prompt enforces complete assignment, trap capacity, and valid CZ-pair placement constraints.

The required output schema is:

```
{
  "target_layout": {
    0: (x, y),
    1: (x, y),
    ...
  }
}
```

Routing Agent. The Routing Agent receives the current layout and the target layout produced by the Placement Agent. It outputs a sequence of AOD move rounds that physically route atoms to their target traps. Each move round is a set of simultaneous atom movements.

The routing prompt specifies two hard physical constraints. First, the capacity constraint requires that, at the end of each round, no physical trap contains more than one atom, accounting for atoms that leave and arrive in the same round. Second, the no-crossing symbolic rule requires that, within a single AOD round, the strict relative ordering of moving atoms along both the x and y axes must not flip.

The required output schema is:

```
{
  "aod_move_groups": [
    [ (q, (x_src, y_src), (x_dst, y_dst)), ... ],
    [ (q, (x_src, y_src), (x_dst, y_dst)), ... ],
    ...
  ]
}
```

Optimize Agent. The Optimize Agent refines an already valid AOD schedule. It receives the current layout, the target layout, vacant traps, hardware dimensions, and an unoptimized list of AOD move groups. The model is instructed not to change the final target layout, but only to improve the route used to reach it.

The prompt enforces the same hard physical constraints as the Routing Agent: no crossing along either axis, valid trap occupancy at arrival, and unique destinations within each round.

For iterative optimization, the prompt receives two-layer feedback from the Hardware Simulator. The outcome layer reports hardware runtime T_{exe} and circuit fidelity F. The diagnostic layer decomposes fidelity into transfer fidelity and coherence fidelity, allowing the model to distinguish whether a candidate schedule is limited by excessive SLM–AOD transfers or excessive idle decoherence. A candidate is accepted only if it improves at least one of T_{exe} or F without degrading the other.

The required output is a JSON array of optimized AOD move groups:

```
[
  [ [qubit_id, [start_x, start_y], [end_x, end_y]], ... ],
  ...
]
```

B Deterministic Verifiers

Qubrio uses LLMs as proposal generators and relies on deterministic verifiers to enforce hardware validity. This appendix describes the implementation-level checks corresponding to the Placement Verifier and Routing Verifier in the main text. Invalid outputs are rejected and converted into feedback for subsequent refinement attempts.

Placement Verifier. The Placement Verifier is implemented by `diagnose_layout` in `verify.py`. Given a hardware instance, a candidate layout, and the current CZ stage, it diagnoses placement violations as

$$\{V_{\text{coll}}, V_{\text{adj}}, V_{\text{xtalk}}, \text{fatal}\}.$$

The hardware layout follows the split-trap model: for an n-qubit program, the verifier uses

$$\text{WIDTH} = 2\lceil\sqrt{n}\rceil, \qquad \text{HEIGHT} = \lceil\sqrt{n}\rceil.$$

A coordinate (x, y) is valid only if

$$0 \le x < \text{WIDTH}, \qquad 0 \le y < \text{HEIGHT}.$$

Each gate site is defined by

$$\text{group_id}(x, y) = \text{HEIGHT} \cdot y + \lfloor x/2 \rfloor,$$

so the two traps with x coordinates $2k$ and $2k + 1$ at the same row belong to the same gate site.

The Placement Verifier first checks fatal layout errors, including missing qubits, unknown qubits, and out-of-bound coordinates. It then checks exact trap collisions, where two logical qubits are assigned to the same physical trap. In addition, if more than two atoms occupy the same gate site, all such pairs are reported as collision-class violations. For the current CZ stage, V_{adj} contains CZ pairs that are not assigned to the same gate site, while V_{xtalk} contains non-CZ pairs that occupy the same gate site. Each categorized violation is localized as

$$(q_i, q_j, P(q_i), P(q_j)).$$

Thus, the interaction-distance and crosstalk constraints in the main text are instantiated by same-gate-site checks in the implementation.

The placement feedback loop is implemented in `_format_placement_feedback` in `incremental_compiler.py`. It follows the two-stage policy described in the main text. If any collision exists, the feedback exposes only V_{coll}, so the model first establishes a valid trap assignment. Once collisions are resolved, the feedback jointly reports V_{adj} and V_{xtalk}, since these two constraints impose opposing spatial pressures. A placement is accepted only when all violation sets and fatal errors are empty.

Routing Verifier. The Routing Verifier checks AOD movement rounds. Each AOD round is represented as a list of moves

$$(q, (x_{\mathrm{src}}, y_{\mathrm{src}}), (x_{\mathrm{dst}}, y_{\mathrm{dst}})).$$

The online checker is implemented by `check_aod_round` in `verify.py`. For each round, it enforces: (i) the round is non-empty; (ii) no qubit moves more than once in the same round; (iii) every moved qubit exists in the current layout; (iv) each source coordinate matches the current layout; (v) each destination coordinate is within bounds; and (vi) after the round, each physical trap contains at most one atom. The occupancy check accounts for atoms that leave and arrive in the same round, so vacate-then-occupy moves are permitted when they do not violate final occupancy.

The Routing Verifier also enforces the AOD no-crossing constraint using symbolic ordering. For every pair of moving atoms in the same round, it compares their relative order along the x and y axes before and after the move. A crossing violation is reported if a strict order flips, i.e., $<$ becomes $>$ or $>$ becomes $<$. This check is applied pairwise among all moving atoms in the round.

The implementation also provides a structured diagnostic routine, `diagnose_routing` in `verify.py`, which localizes routing failures as

$$\{V_{\mathrm{coll}}, V_{\mathrm{cross}}, \mathrm{fatal}\}.$$

Here, V_{coll} contains trap-capacity violations after a movement step, and V_{cross} contains AOD ordering violations. Each categorized routing violation is localized as

$$(q_i, q_j, P^t, P^{t+1}, t),$$

where t is the offending movement step. Fatal errors include empty rounds, duplicate movers, unknown qubits, source mismatches, and out-of-bound destinations.

In the compilation loop, routing outputs are parsed and checked in `generate_transition` in `incremental_compiler.py`. The loop uses `check_aod_round` for online validation and target-layout consistency, while `diagnose_routing` provides the structured step-level diagnosis corresponding to the violation sets described in the main text. After all AOD rounds pass validation, the compiler checks that every qubit reaches the target layout produced by the Placement Agent. If any round fails verification or the final layout does not match the target layout, the route is rejected and the error is returned to the LLM as refinement feedback.

Optimize Agent validation and acceptance. The Optimize Agent refines an already valid AOD schedule. Candidate optimized schedules are parsed in `optimize_aod_moves` in `incremental_compiler.py`. Each candidate is replayed from the current layout, and every optimized AOD round is checked by the same deterministic Routing Verifier, `check_aod_round`. The compiler then verifies that the replayed final layout exactly matches the target layout. Therefore, optimization is not allowed to change the target placement; it may only change the route used to reach that placement.

After physical validation, the candidate is evaluated by the Hardware Simulator. The acceptance rule is implemented by `_accept_candidate`: a candidate is accepted only if it improves at least one of hardware runtime or circuit fidelity without degrading the other. Invalid candidates are rejected before simulation. Valid but non-improving candidates are rejected by the acceptance rule. When full feedback mode is enabled, rejected candidates are included in the subsequent optimization context together with outcome-level metrics and diagnostic metrics.

Parsing and retry handling. LLM outputs are parsed before verification. Placement and routing outputs are extracted as Python dictionaries using `extract_python_dict` in

`incremental_compiler.py`, then parsed with `ast.literal_eval`. Initial placement must contain the `initial_layout`, stage placement must contain `target_layout`, and routing must contain `aod_move_groups`. The Optimize Agent output is parsed as a JSON array of AOD move groups.

The compiler uses bounded retry loops. Initial placement, stage placement, and routing use an inner retry limit of 10 by default. Optimization uses 5 attempts by default. The full transition loop also has an outer retry limit of 5. When parsing or verification fails, the compiler appends the error message and, when available, a history of failed attempts to the next LLM call. If the retry limit is exhausted, the corresponding compilation stage fails rather than accepting an invalid schedule.

Full-schedule sanity check. The function `verify_output` in `verify.py` provides a full-schedule sanity check. It validates the initial layout, parses each schedule item, replays all AOD move groups with `check_aod_round`, and audits each CZ stage using `audit_groups_pre_cz`. The CZ audit checks that each executed CZ stage matches the input stage and that any two atoms sharing a gate site correspond to an intended CZ pair. The verifier also checks that the schedule contains the expected number of CZ stages and that the final layout remains within bounds.

C Broader Impacts

Qubrio has positive societal impacts by reducing the manual effort required to adapt quantum compilers to evolving hardware, potentially accelerating research in quantum computing and making compiler development more accessible. Potential negative impacts include the possibility that more automated and efficient compilation could accelerate dual-use quantum-computing applications in the future, as well as privacy and reproducibility concerns when using commercial LLM APIs.

D LLM Configuration and Execution Details

Qubrio employs a model-agnostic architecture with a unified API to support different LLM backbones. In all experiments, the LLM functions solely as a proposal generator; all outputs require deterministic verification and simulator evaluation prior to acceptance.

To assess the impact of the model backbone, we evaluate Qwen-3.5 [35], DeepSeek-V3.2 [36] and GPT-5.4 [37] using their API snapshots accessed in March 2026. By default, the models use temperature 0.7 and top-p 1.0. Verifier-guided correction is capped at 10 rounds for placement and routing, and optimization is capped at 5 rounds.

E Compute resources

All verifier and hardware-simulator experiments were run on a CPU server with an Intel Xeon CPU with 32 cores and 256 GB of RAM. LLM backbones were accessed through external APIs, so no local GPU was required for the reported API-based experiments. Compilation time and per-stage API cost are reported in Section 5.

F Raw Data for Main Results

Table 1 reports the hardware execution time T_{exe}, end-to-end circuit fidelity F, and average LLM retry counts triggered by the Placement and Routing Verifiers. We evaluate Qubrio across LLM backbones and compare it against two deterministic heuristic baselines, Enola and PowerMove. The results are obtained from $N = 6$ independent runs for each benchmark and reported as mean values.

Overall, Qubrio achieves favorable execution time and fidelity compared with the heuristic baselines, with particularly notable runtime improvements on regularly structured circuits. These circuits offer greater opportunities for convoy-style shuttling, allowing compatible displacement vectors to be coordinated and reducing execution overhead. The verifier retry counts also remain within the per-stage budget of 10 across all benchmarks. All runs successfully converge to schedules that pass deterministic verification, demonstrating the effectiveness of the proposed framework.

For Ising-66, we observe relatively large runtime variability across independent runs. This behavior is primarily associated with its repeated local interaction patterns, where each qubit interacts with

Table 1: Per-benchmark hardware runtime, circuit fidelity, and LLM retry counts. Error bars indicate mean ± one standard deviation, computed from N=6 independent runs for each benchmark.

Benchmark	Model	T_{exe} (μs)		F		Average retries	
		mean	±std	mean	±std	Placement	Routing
QFT-18	Enola	197,129.8	0	3.26×10^{-3}	0	N/A	N/A
	PowerMove	41,769.7	0	3.18×10^{-2}	0	N/A	N/A
	Qubrio (Qwen-3.5)	29,007.5	3,553	5.66×10^{-2}	7.7×10^{-4}	4.00	3.17
	Qubrio (DS-V3.2)	33,569.1	2,954	5.51×10^{-2}	5.5×10^{-4}	1.17	3.00
	Qubrio (GPT-5.4)	23,895.6	1,903	6.16×10^{-2}	4.6×10^{-4}	1.00	3.00
QAOA-20	Enola	23,641.9	0	0.4864	0	N/A	N/A
	PowerMove	5,690.3	0	0.6186	0	N/A	N/A
	Qubrio (Qwen-3.5)	4,887.3	809	0.6844	0.021	2.66	3.00
	Qubrio (DS-V3.2)	3,975.3	576	0.6889	0.019	1.33	4.00
	Qubrio (GPT-5.4)	2,989.5	173	0.7041	0.018	1.00	3.00
GHZ-23	Enola	8,037.5	0	0.7377	0	N/A	N/A
	PowerMove	1,684.0	0	0.8509	0	N/A	N/A
	Qubrio (Qwen-3.5)	607.0	59	0.8699	0.003	1.00	3.00
	Qubrio (DS-V3.2)	459.0	39	0.8670	0.003	1.00	3.00
	Qubrio (GPT-5.4)	371.8	62	0.8687	0.001	1.00	3.00
Ising-26	Enola	27,019.2	0	0.4035	0	N/A	N/A
	PowerMove	3,992.7	0	0.6868	0	N/A	N/A
	Qubrio (Qwen-3.5)	715.1	127	0.7432	0.004	1.33	3.00
	Qubrio (DS-V3.2)	346.4	61	0.7460	0.003	1.17	3.00
	Qubrio (GPT-5.4)	284.7	145	0.7548	0.005	1.00	2.00
QFT-63	Enola	1,329,226.1	0	3.95×10^{-69}	0	N/A	N/A
	PowerMove	277,720.0	0	8.02×10^{-16}	0	N/A	N/A
	Qubrio (Qwen-3.5)	131,229.4	24,934	9.39×10^{-10}	1.9×10^{-10}	3.33	3.00
	Qubrio (DS-V3.2)	188,228.8	35,763	1.00×10^{-10}	2.1×10^{-11}	2.50	3.00
	Qubrio (GPT-5.4)	71,201.3	13,528	1.13×10^{-8}	2.3×10^{-9}	1.00	2.00
QAOA-60	Enola	73,590.2	0	0.0106	0	N/A	N/A
	PowerMove	13,647.6	0	0.1486	0	N/A	N/A
	Qubrio (Qwen-3.5)	25,320.8	5,791	0.0979	0.024	2.66	3.00
	Qubrio (DS-V3.2)	14,475.4	2,967	0.1610	0.040	1.50	3.17
	Qubrio (GPT-5.4)	11,847.6	2,129	0.1752	0.041	1.33	3.00
GHZ-78	Enola	21,754.9	0	0.1720	0	N/A	N/A
	PowerMove	6,424.8	0	0.4507	0	N/A	N/A
	Qubrio (Qwen-3.5)	3,500.3	54	0.4798	0.009	2.50	3.66
	Qubrio (DS-V3.2)	1,066.1	20	0.5882	0.011	1.66	3.00
	Qubrio (GPT-5.4)	421.9	6	0.6176	0.011	1.00	3.00
Ising-66	Enola	77,380.5	0	0.0100	0	N/A	N/A
	PowerMove	9,610.0	0	0.2998	0	N/A	N/A
	Qubrio (Qwen-3.5)	1,655.1	921	0.4629	0.046	7.00	3.33
	Qubrio (DS-V3.2)	4,060.5	2,010	0.3957	0.040	4.00	3.00
	Qubrio (GPT-5.4)	548.4	251	0.4756	0.041	1.00	3.00

multiple nearby qubits in succession. Routing decisions that minimize immediate movement costs may lead to unfavorable atom placements for subsequent interactions, requiring additional corrective movements. Consequently, different early routing choices can produce substantially different shuttle schedules and execution times. This suggests that the observed variability stems from the sensitivity of routing decisions to future interactions, rather than verifier instability.

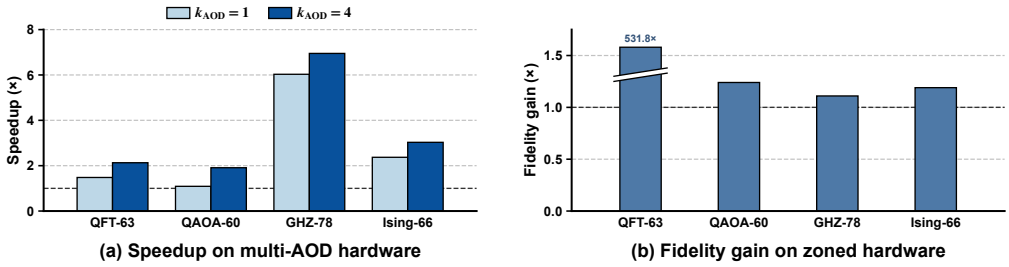


Figure 8: **Adaptability Comparison with PowerMove.**

G Adaptability Comparison with PowerMove

We further evaluate Qubrio's adaptability under multi-AOD and multi-zone hardware configurations, as shown in Fig. 8(a) and (b), respectively. We compare Qubrio against PowerMove using its official adaptations for both scenarios. These adaptations involve algorithmic modifications to nine core modules and changes to more than 90% of the source files. In contrast, adapting Qubrio requires only tens of lines of declarative hardware specifications and constraint updates, without modifying its core algorithms. Across both scenarios, Qubrio consistently outperforms PowerMove, demonstrating its ability to accommodate different hardware configurations with substantially less manual algorithm-design effort.

H Comparison with Concurrent LLM-Based Compilation

We further discuss the differences between Qubrio and the concurrent work SimpleTES [76], which represent two complementary approaches to LLM-assisted compilation. SimpleTES uses an LLM to generate a placement program, while its routing and evaluation components remain hand-written and fixed. Once generated, the resulting compiler processes circuits without further LLM involvement. This can be viewed as a form of distillation that incorporates LLM-generated strategies into a reusable compiler, avoiding repeated LLM inference. In contrast, Qubrio directly integrates specialized LLM agents into the compilation process, enabling more flexible, circuit-specific decisions.

We also empirically compare the two approaches using the zoned architecture evaluation setup provided by SimpleTES, with identical benchmark circuits, hardware layouts, storage-site spacing, hardware parameters, simulation models, and metric accounting. As shown in Fig. 9, Qubrio consistently achieves lower hardware execution time and higher circuit fidelity across all benchmarks. These results are consistent with the distinction between the two approaches: while distilling LLM-generated strategies into fixed compiler components reduces inference overhead, retaining LLM reasoning during compilation allows more flexible optimization tailored to individual circuits.

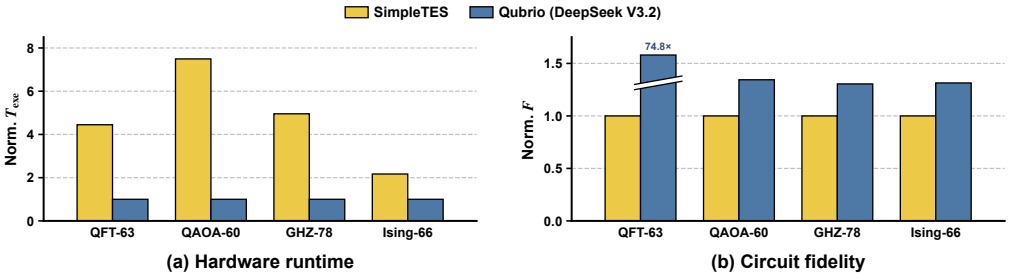


Figure 9: **Comparison with SimpleTES [76].**

I Parallel Compilation Strategy

Qubrio adopts a two-phase parallel compilation strategy to exploit concurrency while respecting the dependency between placement and routing. In the first phase, circuit stages are partitioned

into groups, which are processed in parallel. Within each group, stage placements are generated sequentially, using the placement from the preceding stage as a warm start to improve coordination. We empirically set the group size to six, balancing parallelism and placement quality, with less than 10% performance degradation compared with fully sequential processing. In the second phase, once all stage placements are available, routing tasks for different stages are executed independently and in parallel. Consequently, the overall compilation latency is determined by the slowest placement group followed by the slowest routing task, rather than growing linearly with the number of circuit stages.

J Hardware Simulator Configuration and Evaluation Protocol

Hardware setting. Unless otherwise specified, we evaluate each n-qubit program on a neutral-atom architecture with one AOD array and an SLM grid of size $2 \times \lceil\sqrt{n}\rceil \times \lceil\sqrt{n}\rceil$. The layout consists of $\lceil\sqrt{n}\rceil \times \lceil\sqrt{n}\rceil$ gate sites, with two SLM traps hosted at each site. Adjacent gate sites are separated by an inter-site pitch of $15\,\mu\mathrm{m}$, while the two traps within the same gate site are separated by $1.5\,\mu\mathrm{m}$. Consequently, the physical footprint scales as $16.5\lceil\sqrt{n}\rceil \times 15\lceil\sqrt{n}\rceil\,\mu\mathrm{m}^2$, where the horizontal pitch $16.5\,\mu\mathrm{m}$ accounts for both the inter-site spacing and the intra-site trap separation.

Metrics. We evaluate compiler performance using two primary metrics: hardware runtime (T_{exe}) and circuit fidelity (f_{out}).

Hardware runtime. T_{exe} denotes the end-to-end hardware runtime of the compiled circuit on neutral-atom hardware. It is computed by summing the durations of all scheduled single-qubit gates, two-qubit gates, atom movements, and SLM–AOD transfers:

$$T_{\mathrm{exe}} = T_{1\mathrm{Q}} + T_{\mathrm{CZ}} + T_{\mathrm{move}} + T_{\mathrm{transfer}}.$$

We compute T_{exe} by summing the durations of all scheduled gate, movement, and transfer segments in the compiler-generated execution schedule.

Circuit fidelity. Circuit fidelity captures the end-to-end reliability of the compiled schedule. Following PowerMove [4], we factor the output fidelity into gate, spectator, transfer, and decoherence terms:

$$f_{\mathrm{out}} = f_1^{g_1} \cdot f_2^{g_2} \cdot f_{\mathrm{spec}}^{\sum_{i=1}^{S} n_i} \cdot f_{\mathrm{trans}}^{N_{\mathrm{trans}}} \cdot \prod_q \left(1 - \frac{T_q}{T_2}\right).$$

Here, f_1 and f_2 denote the fidelities of single-qubit gates and CZ gates, respectively, and g_1 and g_2 are their corresponding operation counts. The spectator term $f_{\mathrm{spec}}^{\sum_i n_i}$ accounts for errors on non-interacting qubits exposed during Rydberg pulses, where S is the number of Rydberg pulses and n_i is the number of spectator qubits during the i-th pulse. The transfer term $f_{\mathrm{trans}}^{N_{\mathrm{trans}}}$ models the cumulative fidelity loss from N_{trans} SLM–AOD transfers. Finally, the product over qubits models decoherence, where T_q is the total elapsed time experienced by qubit q and T_2 is the coherence time. The hardware parameters used in our simulator are summarized in Table 2.

	R_z Gate	$R_{x/y}$ Gate	CZ Gate	Spectator	Transfer
Fidelity	99.9%	99.9%	99.5%	99.75%	99.9%
Duration	$1\,\mu$s	$3\,\mu$s	270 ns	270 ns	$15\,\mu$s

	Qubit Movement
Fidelity	$\approx 100\%$ when $a < 2750\,\mathrm{m}\cdot\mathrm{s}^{-2}$
Duration	e.g., $100\,\mu$s for $27.5\,\mu$m; $200\,\mu$s for $110\,\mu$m
Coherence Time	1.5 s

Table 2: Neutral-atom hardware parameters used in the simulator [4, 5, 18, 19].

Baseline configuration. All baselines are evaluated under the same hardware layout, timing model, and fidelity model described above. We use the simulator and default hardware parameters from PowerMove [4] and Enola [5], and apply the same post-compilation evaluation script to all generated schedules. For Qubrio, LLM-generated schedules are accepted only after passing deterministic placement and routing verification; invalid schedules are recorded as '-'.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction clearly state the paper's main contributions and scope, including the agentic compiler design, convoy-style shuttling discovery, and empirical improvements over PowerMove. These claims are supported by the framework description in Sections 3–4 and the experimental results and ablations in Section 5.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper discusses the limitations in the Section 7, noting that current evaluations are limited to NA hardware and that empirically evaluating the framework on other quantum platforms remain future work.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: The paper does not include formal theoretical results, theorems, or proofs.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The paper describes the workflow in detail. It also provides the code link, and the appendix includes the prompts used by Qubrio.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The code is publicly available at `https://github.com/crosaa/Qubrio`, with instructions to reproduce the main results.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The paper specifies the experimental setting in Section 5.1, including the evaluated LLM backbones, benchmark circuits, baseline compilers, hardware configurations, and evaluation metrics. Additional details such as parameters, retry budgets, variance, and simulator settings are provided in the appendices.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: The paper reports run-to-run variance for the evaluated LLM backbones in Appendix F, providing appropriate information about experimental variability. The main results focus on average hardware runtime and fidelity comparisons, with variance details included in the appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.

- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Details regarding the compute resources used for our experiments are provided in Appendix E.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification:The research conforms to the NeurIPS Code of Ethics. It does not involve human subjects, private data, surveillance data, or unsafe model release.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

Justification: The paper discusses broader impacts, highlighting the benefits of reducing manual compiler redesign and accelerating quantum computing research. Conversely, it also addresses potential risks, including the dual-use nature of quantum applications and the reliance on commercial LLM APIs.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper does not release high-risk models or scraped datasets.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The paper credits the existing tools and models used by Qubrio. It also follows their licenses and terms of use.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.

- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The paper releases new code, prompts, and simulation data. These assets are documented with usage instructions and reproduction details.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing. It also does not involve research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

Justification: The paper does not involve human subjects. Therefore, IRB approval is not applicable.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: LLM usage is a core component of the proposed method and is described throughout the paper. The paper also explains that LLM outputs are used as proposals and are accepted only after deterministic verification and simulator-based evaluation.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.