# Relational Verification for Cost-Aware Quantum Program Optimization

**Ziming Zhao[1], Tingting Li[1,2*], Zhaoxuan Li[1,3], Jianwei Yin[1*]**

[1]Zhejiang University, Hangzhou, China
[2]Shanghai Qi Zhi Institute, Shanghai, China
[3]Institute of Information Engineering, CAS, Beijing, China
zhaoziming@zju.edu.cn, litt2020@zju.edu.cn, lizhaoxuan@iie.ac.cn, zjuyjw@cs.zju.edu.cn

## Abstract

Optimizing quantum programs is key to mitigating noise, reducing error-correction overhead, and improving performance on both near-term and fault-tolerant devices. Existing heuristic and learning-based optimizers, however, lack formal guarantees and risk semantic errors in the presence of entanglement and measurement. We present RelOpt, a semantics-preserving optimizer that enforces *relational correctness* between original and optimized programs. RelOpt is built on a lightweight intermediate language (`QCore`) with a relational operational semantics supporting partial-trace equivalence, measurement-distribution preservation, and approximate correctness. Optimization is guided by a multi-objective cost model that considers gate count, circuit depth, and error-correction cost. Only rewrite rules that are formally verified against user-specified contracts are applied. The engine combines symbolic simulation, SMT reasoning, and cost analysis to achieve safe and effective optimizations. On standard benchmarks such as QFT, Grover, and QAOA, RelOpt consistently outperforms Qiskit, `t|ket⟩`, and learning-based optimizers across multiple cost metrics while maintaining formal guarantees. By integrating formal verification with cost-aware compilation, RelOpt establishes a foundation for trustworthy and hardware-adaptive quantum toolchains.

## Introduction

Quantum computing holds the promise of revolutionizing domains such as cryptography, combinatorial optimization, and quantum simulation (Preskill 2018). As quantum hardware advances from the noisy intermediate-scale quantum (NISQ) devices to early fault-tolerant architectures (Campbell, Terhal, and Vuillot 2017; Li and Zhao 2024), the reliability and efficiency of quantum programs become increasingly vital. However, quantum programs are inherently fragile: minor syntactic transformations, such as gate reordering, fusion, or elimination, can induce subtle but catastrophic deviations in entanglement structure, measurement statistics, or coherence (Amy et al. 2013; Hietala et al. 2023). It makes optimization both necessary and perilous.

Existing quantum optimizers including state-of-the-art compilers like Qiskit (Javadi-Abhari et al. 2024) and `t|ket⟩` (Sivarajah et al. 2020), primarily rely on algebraic rewrites and hardware-specific simplifications via heuristics or pattern matching. While effective in reducing circuit size or depth, these methods offer no formal correctness guarantees. Even more concerning, learning-based optimizers such as Quarl (Li et al. 2024b) treat optimization as a black-box process, lacking interpretability or trustworthiness. Unlike classical compilers, which benefit from mature support for symbolic execution (Cadar, Dunbar, and Engler 2008), property-based testing (Claessen and Hughes 2000), and formal verification (Nebut et al. 2006; Guo, Kusano, and Wang 2016), quantum compilation remains largely semantics-agnostic (Murillo et al. 2025).

We advocate for a *semantics-preserving* approach to quantum optimization grounded in *relational verification* (Benton and Leperchey 2005; Barthe et al. 2019). Rather than blindly applying rewrites, we cast optimization as a constrained synthesis problem: given an original program $P$ and a relational correctness contract $(\Phi, \Psi)$ specifying the intended input-output relationship, we seek an optimized program $Q$ such that $\{\Phi\}\ P \sim Q\ \{\Psi\}$ holds (Benton and Leperchey 2005; Li and Unruh 2019; Barthe et al. 2019). Here, $\sim$ denotes a relational correctness judgment, *e.g.,* full-state equivalence, measurement distribution preservation, or partial trace agreement, allowing correctness guarantees to be tailored to the optimization context.

However, realizing this vision introduces several technical challenges. (i) *Semantics modeling:* Quantum programs manipulate probabilistic, entangled, and non-cloneable states. Modeling their relational behavior is nontrivial, especially in the presence of measurement and classical control (Ying 2010; Qiu and Li 2008; Feng and Xu 2023). (ii) *Cost-guided search:* Optimizations must balance competing objectives (*e.g.,* gate count, depth, fault-tolerance cost), which vary significantly across hardware platforms (Cowtan et al. 2019; Sinha et al. 2024; Murali et al. 2019). (iii) *Efficient verification:* Relational verification is typically computationally expensive, supporting it at the rewrite granularity requires lightweight and compositional reasoning mechanisms (Hagedorn et al. 2020; Baader and Nipkow 1998; Gourdin et al. 2023).

In this paper, we propose RelOpt, a semantics-aware quantum optimizer that applies only formally verified rewrites under a user-specified relational correctness contract. RelOpt consists of three main components. (i)

*Corresponding authors.
Copyright © 2026, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

A lightweight quantum intermediate language (`QCore`) equipped with relational operational semantics for compositional correctness reasoning (Feng and Xu 2023; Hua et al. 2023). (ii) A library of verified rewrite rules, each annotated with both cost delta and the corresponding relational specification (Kissinger and van de Wetering 2019; Bauer-Marquart, Leue, and Schilling 2023). (iii) A multi-objective cost model that guides optimization under hardware-specific constraints such as gate fidelity and error-correction overhead (Li, Shi, and Javadi-Abhari 2021; Li et al. 2024b).

This paper makes the following contributions.

- We introduce a *relational operational semantics* for quantum programs, enabling compositional reasoning about correctness notions such as partial trace equivalence, measurement distribution preservation, and approximate fidelity.
- We formulate quantum program optimization as a *relationally constrained synthesis problem*, guided by a tunable, multi-objective cost model that captures gate count, circuit depth, and error correction overhead.
- We design and implement RelOpt, a verified rewrite-based optimizer that applies only correctness-preserving transformations via a greedy search loop supported by symbolic simulation and SMT solving.
- We evaluate RelOpt on a suite of standard quantum algorithm benchmarks and show that it consistently outperforms heuristic and learning-based baselines in cost reduction, while maintaining strong formal correctness guarantees.

## Background and Motivation

Quantum program optimization is essential yet challenging. Unlike classical programs, even small syntactic changes can disturb entanglement or measurement behavior (Ying 2010; Barthe et al. 2019; Feng, Xu, and Long 2013). Compilers must therefore balance semantic correctness with cost efficiency under diverse hardware constraints (Murali et al. 2019; Cowtan et al. 2019).

**Semantic Fragility.** Quantum programs evolve through superposition and entanglement, so local changes such as gate reordering or elimination can cause large semantic deviations (Bauer-Marquart, Leue, and Schilling 2023; Kissinger and van de Wetering 2019). For example:

```
H[q]; H[q]; measure[q]
```

Classical simplification uses $H \cdot H = I$, but if $q$ is entangled or its measurement controls later computation, this rewrite alters the final output (Nielsen and Chuang 2010; Qiu and Li 2008). On real hardware, internal states are inaccessible, correctness must be defined by the distribution of classical outputs (Hietala et al. 2023):

$$\forall x \in \text{Outcomes}, \quad \Pr_P[x] = \Pr_Q[x].$$

In practice, weaker relational criteria are often used: matching only selected qubits, bounding trace distance by $\epsilon$, or preserving partial entanglement structure (Li and Unruh 2019; Wang and Zhang 2023; Feng and Xu 2023; Benton and Leperchey 2005). These relaxed notions allow safe but more aggressive optimization.

**Formal Guarantees.** Current compilers such as Qiskit (Javadi-Abhari et al. 2024) and `t|ket⟩` (Sivarajah et al. 2020) use rule-based rewrites without formal guarantees, while learning-based systems such as Quarl (Li et al. 2024b) apply black-box search with little interpretability. Both approaches risk semantic regressions from operations like cross-entanglement reordering, unguarded fusion, or inlining of control-dependent subroutines, where locally valid changes lead to globally incorrect behavior. Quantum optimization must be both *semantics-aware* and *formally verified*. Each transformation $P \rightarrow Q$ is justified by a relational correctness judgment $\{\Phi\}\ P \sim Q\ \{\Psi\}$, meaning that inputs related by $\Phi$ produce outputs related by $\Psi$. This approach treats optimization as a semantics-preserving refinement rather than a heuristic rewrite. Relational correctness restricts rewrites to those with provable guarantees, adapts correctness notions to applications or hardware, and supports modular reasoning across multiple passes, making it essential for trustworthy quantum compilers.

**Design Drivers for Verified Quantum Optimization.** Because quantum programs are fragile and current compilers lack guarantees, our framework is driven by two needs: balancing hardware-specific cost trade-offs, and supporting tunable correctness contracts for safe, aggressive optimization (Li, Shi, and Javadi-Abhari 2021; Sinha et al. 2024). Thus, beyond correctness, compilation must optimize for hardware metrics (Cowtan et al. 2019; Bauer-Marquart, Leue, and Schilling 2023): *gate count*, which drives noise; *circuit depth*, constrained by coherence times; and *error-correction cost*, dominant in fault-tolerant architectures where each logical operation may require thousands of physical gates (Campbell, Terhal, and Vuillot 2017). These objectives may conflict, *e.g.,* reducing depth sometimes increases gate count, while lowering error-correction cost could require non-local transformations unsuitable for NISQ devices (Li et al. 2024b; Murillo et al. 2025; Li et al. 2024a). Optimizers must therefore be both cost-aware and hardware-adaptive. To enable this safely, we introduce a *relational correctness hierarchy* that controls the admissible transformations (Li and Unruh 2019; Wang and Zhang 2023). (i) *Full-state equivalence* allows only unitary-preserving rewrites. (ii) *Measurement equivalence* permits transformations that preserve measured output distributions, such as reordering and partial fusions. (iii) *Approximate correctness* admits trace-lossy rewrites when the deviation is bounded by a tunable threshold $\epsilon$ (Barthe et al. 2019). Relaxed contracts expand optimization opportunities but require precise semantic tracking.

**Motivating Example.** Optimizing a QFT subroutine in a variational algorithm (Li, Shi, and Javadi-Abhari 2021) illustrates these trade-offs: strict equivalence restricts optimizations to exact unitary rewrites, measurement-only correctness allows broader transformations, and depth-prioritized cost models may prefer trace-preserving rewrites that reduce latency at the expense of global fidelity. These scenarios motivate combining *relational correctness*, *multi-objective cost modeling*, and *formal rule verification*.

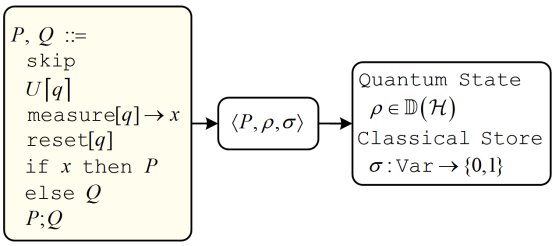


Figure 1: Syntax and state configuration of `QCore`.

## Relational Semantics for Quantum Programs

Semantics-preserving optimization requires a formal foundation that can express behavioral relationships between program variants. We must reason not only about single executions, but about the correctness of two programs relative to each other under approximate, partial, or context-specific equivalence. To this end, we introduce `QCore`, a lightweight intermediate language with a relational operational semantics that relates program behaviors on related inputs. These semantics enable compositional, context-aware verification of rewrites and form the basis of our framework.

**Language Syntax and State Model.** `QCore` is minimal yet expressive, supporting unitary operations, measurement, classical control, and sequential composition, while remaining amenable to symbolic reasoning. Figure 1 shows the syntax and execution model:

$$
\begin{aligned}
P, Q ::= \ & \texttt{skip} && \triangleright \text{no-op} \\
& \mid U[q] && \triangleright \text{unitary on qubit } q \\
& \mid \texttt{measure}[q] \to x && \triangleright \text{measurement with binding} \\
& \mid \texttt{reset}[q] && \triangleright \text{reset } q \text{ to } |0\rangle \\
& \mid \texttt{if } x \texttt{ then } P \texttt{ else } Q && \triangleright \text{conditional} \\
& \mid P; Q && \triangleright \text{sequential composition}
\end{aligned}
$$

Programs execute on configurations $\langle P, \rho, \sigma \rangle$, where $\rho$ is the quantum state (density matrix over $\mathcal{H}$) and $\sigma$ is the classical store mapping variables $x \in X$ to $\{0, 1\}$. This separation of quantum and classical states enables modular reasoning about hybrid programs with measurement and control. It also provides the foundation for defining relational correctness judgments over program executions.

**Relational Correctness Judgments.** We model correctness as a relation between *two* programs on related inputs, rather than a property of a single program. This relational view allows reasoning about exact, partial, or approximate equivalence between their observable behaviors.

**Definition** (Relational Correctness)**.** For binary relations $\Phi$ and $\Psi$ over quantum-classical configurations, $\{\Phi\}\ P \sim Q\ \{\Psi\}$ holds if for all input configurations $(\rho, \sigma)$ and $(\rho', \sigma')$ satisfying $\Phi$, the (possibly probabilistic) executions of $P$ and $Q$ terminate and induce final output distributions whose supports are related by $\Psi$.

This formulation subsumes strict semantic equivalence (same unitary action), partial equivalence (matching output distributions on selected qubits), and approximate correctness (trace distance within $\epsilon$). By treating correctness as a relation, it enables flexible, formal reasoning about diverse optimization scenarios.

| Rule | Description |
|---|---|
| **[R-Skip]** | $\{\Phi\}\ \texttt{skip} \sim \texttt{skip}\ \{\Phi\}$ |
| **[R-Unitary]** | $\{\Phi\}\ U[q] \sim U[q]\ \{\Phi'\}$ where $\Phi'$ reflects updated states: $\rho_1 = U\rho U^\dagger$, $\rho_2 = U\rho' U^\dagger$ |
| **[R-Measure]** | $\{\Phi\}\ \texttt{measure}[q] \to x \sim \texttt{measure}[q] \to x\ \{\Psi\}$ where $\Psi$ relates the post-measurement distributions |
| **[R-If]** | $\{\Phi\}\ \texttt{if } x \texttt{ then } P \texttt{ else } P' \sim \texttt{if } x \texttt{ then } Q \texttt{ else } Q'\ \{\Psi\}$ assuming $\sigma(x) = \sigma'(x)$ and $\{\Phi\}\ P \sim Q\ \{\Psi\}$ and $\{\Phi\}\ P' \sim Q'\ \{\Psi\}$ |
| **[R-Seq]** | $\{\Phi\}\ P_1; P_2 \sim Q_1; Q_2\ \{\Psi\}$ if $\{\Phi\}\ P_1 \sim Q_1\ \{\Theta\}$ and $\{\Theta\}\ P_2 \sim Q_2\ \{\Psi\}$ |

Table 1: Core relational transition rules in `QCore`.

**Relational Operational Semantics.** We define a small-step operational semantics on *pairs* of configurations, lifting single-program transitions to the relational level. This semantics model the stepwise evolution of two programs on related quantum-classical states. Table 1 summarizes the main relational rules for `QCore`. Each rule specifies how structurally similar constructs (*e.g.,* unitaries, measurements, conditionals) map related inputs to related outputs. The rules are compositional, reasoning about individual steps extends to sequences and branches. This relational semantics enables modular verification, each rewrite rule can be proven correct in isolation and safely composed into end-to-end verified optimization pipelines.

**Example Relational Specifications.** The relational semantics supports different choices of precondition $\Phi$ and post-condition $\Psi$, corresponding to practical correctness contracts (Feng and Xu 2023). *Full-state equivalence* requires final quantum states to match exactly:

$$\Psi_{\text{full}}(\rho, \rho') := \rho = \rho'.$$

*Measurement distribution equivalence* ensures indistinguishable classical outputs:

$$\Psi_{\text{meas}}(\rho, \rho') := \forall x.\ \Pr_P[x] = \Pr_Q[x].$$

*Partial trace agreement* preserves only a subset $S$ of qubits:

$$\Psi_{\text{trace}}(\rho, \rho') := \text{Tr}_{\bar{S}}[\rho] = \text{Tr}_{\bar{S}}[\rho'].$$

*Approximate correctness* tolerates bounded deviation:

$$\Psi_\epsilon(\rho, \rho') := \|\rho - \rho'\|_{\text{tr}} \leq \epsilon.$$

These specifications enable different trade-offs. Strict contracts allow only unitary-preserving rewrites, while relaxed ones admit more aggressive optimizations as long as the required observable behavior is preserved. This flexibility is essential in practice: for example, variational algorithms often only require measurement distribution preservation on a subset of qubits, and simulation-heavy workloads can tolerate small trace-distance deviations to gain significant performance improvements.

**Semantic Properties.** Our relational semantics satisfy key meta-theoretical properties that ensure sound reasoning in optimization pipelines.

**Theorem** (Compositionality)**.** If $\{\Phi\}\ P_1 \sim Q_1\ \{\Theta\}$ and $\{\Theta\}\ P_2 \sim Q_2\ \{\Psi\}$, then

$$\{\Phi\}\ P_1; P_2 \sim Q_1; Q_2\ \{\Psi\}.$$

Verified rewrites are correct when composed sequentially.

**Theorem** (Transitivity)**.** If $\{\Phi\}\ P \sim Q\ \{\Theta\}$ and $\{\Theta\}\ Q \sim R\ \{\Psi\}$, then

$$\{\Phi\}\ P \sim R\ \{\Psi\}.$$

Correctness propagates across multiple transformations.

**Theorem** (Congruence)**.** Relational correctness is preserved under syntactic restructuring (*e.g.,* associativity of sequencing, conditional flattening).

These properties allow the optimizer to reason modularly: once each transformation is verified, the composition remains correct by construction, guaranteeing end-to-end semantic fidelity.

## Multi-Objective Cost Optimization

Semantic correctness is essential, but program performance is equally critical, especially on resource-constrained quantum hardware. Unlike classical compilation, where objectives such as runtime or memory dominate, quantum compilation must balance diverse and often conflicting hardware factors: gate noise, decoherence, limited qubit connectivity, and fault-tolerance overhead. These factors make quantum compilation inherently multi-objective.

**Modeling Cost.** Compilation involves balancing key resource dimensions (Cowtan et al. 2019; Bauer-Marquart, Leue, and Schilling 2023): *gate count*, dominated by costly two-qubit gates such as CNOT (Arute et al. 2019; Martiel and De Brugière 2022); *circuit depth*, which affects the probability of finishing execution before decoherence; and *error-correction cost*, which becomes dominant when logical operations require thousands of physical gates (Campbell, Terhal, and Vuillot 2017). These objectives often conflict: gate fusions can reduce gate count but elongate critical paths, while flattening circuits to reduce depth can add ancillas or non-local interactions. Optimizers must therefore use tunable, multi-objective cost models that respect correctness. We model cost as a weighted sum:

$$\text{Cost}_{\boldsymbol{\lambda}}(P) = \alpha{\cdot}\text{GateCount}(P){+}\beta{\cdot}\text{Depth}(P){+}\gamma{\cdot}\text{ECCost}(P),$$

where $\boldsymbol{\lambda} = (\alpha, \beta, \gamma)$ encodes user- or hardware-specified priorities. Gate counts may be weighted by gate type (*e.g.,* CNOT $=$ 2, T $=$ 1), $\text{Depth}(P)$ measures the critical path length, and $\text{ECCost}(P)$ estimates the cost of fault-tolerant execution. This formulation generalizes to additional metrics, *e.g.,* T-depth or ancilla usage (Amy et al. 2013; Bocharov, Roetteler, and Svore 2015):

$$\text{Cost}_{\boldsymbol{\lambda}}(P) = \sum_i \lambda_i \cdot \mathcal{M}_i(P).$$

By tuning $\boldsymbol{\lambda}$, the model adapts to different backends: NISQ devices may prioritize low depth, *e.g.,* $(1, 2, 0)$, while fault-tolerant architectures assign more weight to error-correction overhead, *e.g.,* $(1, 1, 10)$. Simulation environments may prefer fewer gates or topology-aware layouts. This parameterization guides the optimizer to explore cost-performance trade-offs while maintaining semantic guarantees.

**Semantics-Preserving Optimization Strategy.** We formulate verified quantum compilation as a constrained optimization problem: minimize a cost function while preserving relational correctness.

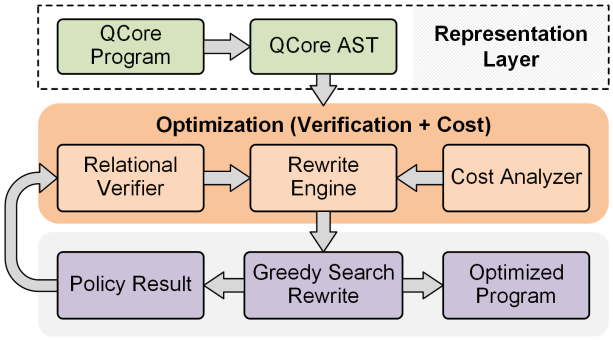


Figure 2: RelOpt architecture and component interaction.

**Definition** (Relationally Constrained Optimization)**.** Given a source program $P_0$, a relational correctness specification $(\Phi, \Psi)$, and a weighted cost function $\text{Cost}_{\boldsymbol{\lambda}}$, find a program $Q$ such that

$$\{\Phi\}\ P_0 \sim Q\ \{\Psi\} \quad \text{and} \quad \text{Cost}_{\boldsymbol{\lambda}}(Q) = \min_{Q'} \text{Cost}_{\boldsymbol{\lambda}}(Q'),$$

where $Q'$ ranges over all programs satisfying the same correctness contract.

This formulation separates correctness from optimization: correctness is enforced declaratively, while search explores the space of valid rewrites. Exact search is infeasible due to the combinatorial space of candidate programs. We therefore use a greedy rewrite-driven strategy: the program is transformed incrementally using verified, cost-reducing rewrites. Each rewrite rule is a tuple

$$r = (\texttt{lhs}, \texttt{rhs}, \Phi_r, \Psi_r, \delta_r),$$

where `lhs` and `rhs` are `QCore` patterns, $\{\Phi_r\}\ \texttt{lhs} \sim \texttt{rhs}\ \{\Psi_r\}$ encodes the verified semantic guarantee, and

$$\delta_r = \text{Cost}_{\boldsymbol{\lambda}}(\texttt{rhs}) - \text{Cost}_{\boldsymbol{\lambda}}(\texttt{lhs})$$

is the estimated cost change. At each step, the optimizer enumerates all applicable rewrites, checks their validity, and applies the one with maximal cost reduction. This process repeats until no valid cost-reducing rewrite remains, producing a program that satisfies the correctness contract and is locally optimal under the chosen cost model. Global correctness follows from the compositional nature of relational reasoning. Each step $P_i \to P_{i+1}$ is statically verified as

$$\{\Phi_i\}\ P_i \sim P_{i+1}\ \{\Psi_i\}.$$

By transitivity, the sequence $P_0 \sim P_1 \sim \cdots \sim P_n$ implies

$$\{\Phi\}\ P_0 \sim P_n\ \{\Psi\}.$$

Thus, local verified transformations combine into a globally sound optimization pipeline. In the process, each rewrite carries a formal correctness judgment ensuring that the final program preserves the semantics of the original.

**Expressiveness and Complexity.** Our cost model flexibly captures diverse optimization objectives. It supports cost-equivalent searches (*e.g.,* minimizing depth with fixed gate count) and Pareto-front exploration via scalarization or constraints. By adjusting the weight vector $\boldsymbol{\lambda}$, the optimizer can

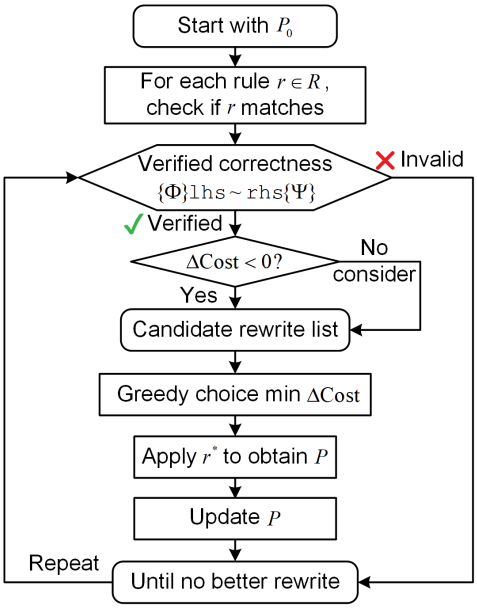

Figure 3: RelOpt rewrite loop.

specialize to different backends, from NISQ devices to fault-tolerant architectures (Preskill 2018; Campbell, Terhal, and Vuillot 2017; Li et al. 2025c,a). The optimization problem remains NP-hard, reducible from classical circuit minimization (Gupta, Agrawal, and Jha 2006). Even with a fixed set of rewrites, finding a global optimum is intractable. Our greedy rewrite-driven approach balances scalability and correctness: it is not globally optimal but scales well and guarantees formally verified transformations suitable for practical compilation pipelines.

## A Verified Rewrite-Based Framework

RelOpt is a semantics-preserving quantum optimizer that generates efficient program variants through a sequence of formally verified rewrites. It combines relational verification, symbolic cost modeling, and correctness-aware rewrite search, ensuring that each optimized program remains behaviorally consistent with the original under a user-specified correctness relation.

**System Architecture.** Figure 2 illustrates the architecture of RelOpt, which is organized around a verification-centric optimization core. The frontend first translates a `QCore` program into a `QCore` AST, forming the *representation layer* on which all subsequent analyses and transformations operate. At the center of the system is the *optimization layer*, which tightly integrates relational verification with cost-aware decision making. The *rewrite engine* serves as the coordination hub: it proposes candidate rewrites on the current AST and interacts with both the *relational verifier* and the *cost analyzer*. The verifier checks whether a candidate rewrite satisfies its relational correctness contract $(\Phi, \Psi)$ using symbolic simulation and SMT-based reasoning, while the cost analyzer evaluates its impact under a weighted cost model parameterized by the user-defined vector $\boldsymbol{\lambda}$. Only rewrites that are verified as correct and deemed cost-effective are admitted for application. Rewrite selection and application are driven by a *greedy search rewrite* strategy. Based on the verification outcomes and cost evaluations, the rewrite engine updates the current program and records the corresponding *policy result*, which captures the selected rewrite decisions. This process iterates until no further verified rewrite can improve the objective, yielding the final *optimized program*.

Algorithm 1: Rewrite Optimization in RelOpt

**Input:** Initial program $P_0$, cost weights $\lambda$, rewrite ruleset $R$
**Output:** Optimized program $P$

$P \leftarrow P_0$
**repeat**
  $A \leftarrow$ all rewrite rules $r \in R$ applicable to $P$
  **for all** $r = (\texttt{lhs}, \texttt{rhs}, \Phi, \Psi, \delta) \in A$ **do**
    Identify program context $C[\cdot]$ that $P = C[\texttt{lhs}]$
    Use verifier to check $\{\Phi\}\texttt{lhs} \sim \texttt{rhs}\{\Psi\}$
    Estimate cost change
    $\delta_r \leftarrow \text{Cost}(C[\texttt{rhs}]) - \text{Cost}(P)$
  **end for**
  Select $r^* \in A$ with lowest $\delta_r$
  **if** $\delta_{r^*} < 0$ **then**
    Apply rewrite: $P \leftarrow C[\texttt{rhs}]$
  **end if**
**until** no further cost-improving rewrite is found
**return** $P$

**Formal Structure of Rewrite Rules.** As mentioned above, each RelOpt rewrite rule is a tuple $r = (\texttt{lhs}, \texttt{rhs}, \Phi, \Psi, \delta)$, where `lhs` and `rhs` are `QCore` fragments. The relation $\{\Phi\}\,\texttt{lhs} \sim \texttt{rhs}\,\{\Psi\}$ specifies the correctness contract, and $\delta$ is the estimated cost change from the parameterized cost model. Applying a rule replaces `lhs` with `rhs` wherever its preconditions hold, guaranteeing that the correctness contract is preserved. The optimizer uses $\delta$ to prioritize rewrites with larger cost reductions. Consider the *Hadamard Cancellation* example

$$\begin{aligned} \texttt{lhs} &= \texttt{H[q]; H[q]} \\ \texttt{rhs} &= \texttt{skip} \\ \Phi = \Psi &= \text{full-state equivalence on } q \\ \delta &= -2 \cdot \alpha \end{aligned}$$

This rule cancels consecutive Hadamards on the same qubit, preserving the state exactly and reducing gate count. For the *Commutative Swap under Measurement Equivalence*

$$\begin{aligned} \texttt{lhs} &= \texttt{CNOT[q1,q2]; T[q1]} \\ \texttt{rhs} &= \texttt{T[q1]; CNOT[q1,q2]} \\ \Phi &= \text{no entanglement or measurement dependency on } q_2 \\ \Psi &= \text{measurement-distribution equivalence} \\ \delta &= 0 \end{aligned}$$

This rule reorders gates under measurement-preserving semantics, changing syntax without altering outputs.

**Rewrite-Guided Optimization Algorithm.** Algorithm 1 shows the core loop of RelOpt, which performs a greedy search over verified rewrites. At each iteration, the optimizer

| Program | Gate Count | | | | Circuit Depth | | | | ECCost | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Qiskit | t\|ket⟩ | Quarl | RelOpt | Qiskit | t\|ket⟩ | Quarl | RelOpt | Qiskit | t\|ket⟩ | Quarl | RelOpt |
| QFT(8) | 448 | 426 | 420 | **324** | 92 | 89 | 88 | **73** | 901 | 881 | 872 | **612** |
| Grover(4) | 312 | 301 | 295 | **260** | 71 | 68 | 69 | **60** | 713 | 708 | 702 | **510** |
| RCA(8) | 1012 | 980 | 998 | **832** | 213 | 204 | 202 | **178** | 2080 | 2030 | 2023 | **1602** |
| QAOA-3 | 678 | 659 | 662 | **618** | 112 | 107 | 109 | **98** | 1376 | 1340 | 1359 | **1125** |
| ModMul(3) | 2155 | 1988 | 1980 | **1742** | 502 | 476 | 471 | **409** | 4553 | 4388 | 4321 | **3677** |
| PE(3) | 852 | 807 | 821 | **688** | 141 | 129 | 133 | **112** | 1784 | 1670 | 1709 | **1322** |

Table 2: Multi-objective cost comparison across optimizers.

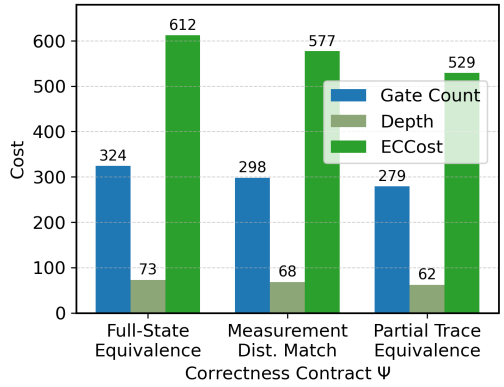


Figure 4: Correctness contract.

enumerates candidate rewrites, retains only those that satisfy their relational correctness contracts and reduce cost, and applies the most beneficial rule. This process repeats until no further cost-reducing verified rule remains. Figure 3 illustrates this control flow: candidate patterns are matched, verified for correctness, filtered by cost improvement, and applied iteratively. The loop terminates once no valid transformation can further reduce cost.

**Verification Techniques and Correctness Guarantee.** RelOpt verifies candidate rewrites with a combination of symbolic simulation and constraint solving. For unitary code, it translates `QCore` fragments to matrices and checks $U_{\texttt{lhs}} = U_{\texttt{rhs}}$ (up to global phase), sufficient for full-state equivalence. For rewrites with measurements, it compares symbolic output distributions under the chosen $\Psi$ (*e.g.,* marginal equality or partial-trace agreement) using the Born rule. For rewrites with classical control or non-unitary operations (*e.g.,* branching, reset), it encodes transitions as logical constraints and discharges them with an SMT solver (*e.g.,* Z3). In this pipeline, symbolic simulation, then distribution comparison, and finally SMT reasoning; a rewrite is accepted only if all checks succeed.

**Theorem** (Global Correctness Invariant)**.** Given a rewrite sequence $P_0 \to P_1 \to \cdots \to P_n$ where each step satisfies $\{\Phi\}\ P_i \sim P_{i+1}\ \{\Psi\}$, the transitivity of relational correctness implies

$$\{\Phi\}\ P_0 \sim P_n\ \{\Psi\}.$$

Thus, the final program $P_n$ is guaranteed to remain semantically related to the source $P_0$ under the specified contract $(\Phi, \Psi)$. Because every transformation is verified and relational correctness composes transitively, RelOpt ensures end-to-end soundness, making it a reliable backend for quantum compilers across different correctness regimes.

| Rewrite Type | Symbolic Simulation | SMT-based Reasoning | Avg. Time |
|---|---|---|---|
| Gate Fusion | 4.2 | – | ∼4.2 |
| Gate Commutation | 5.1 | 12.0 | ∼8.6 |
| Measurement Reorder | 7.8 | 21.7 | ∼14.7 |
| Conditional Folding | 10.4 | 33.5 | ∼22.0 |

Table 3: Average verification time per rewrite ($ms$).

## Evaluation

**Experimental Setup.** We implement RelOpt in `OCaml`, using Z3 for constraint-based verification and `NumPy` for symbolic matrix analysis. The rewrite library contains 42 statically verified transformations annotated with relational contracts and cost deltas. Experiments are run on a 32-core Intel Xeon Gold (2.6 GHz) with 128 GB RAM under Ubuntu 22.04. Verification runs single-threaded, while the rewrite application is lightly parallelized. Unless stated otherwise, we use balanced weights $\boldsymbol{\lambda} = (1, 1, 1)$, equally emphasizing gate count, depth, and error-correction cost. We also explore alternative weightings in the adaptivity experiments.

**Benchmarks and Baselines.** We evaluate on quantum programs from Qiskit's library and RevLib, covering arithmetic circuits (RCA-8, ModMul-3), search and sampling (Grover-4, QAOA-depth-3), and transforms (QFT-8, PE-3). We compare RelOpt with Qiskit (Javadi-Abhari et al. 2024) (optimization level 3), `t|ket⟩` (Sivarajah et al. 2020), and the learning-based optimizer Quarl (Li et al. 2024b).

**RQ1: Cost Reduction Comparison.** Table 2 compares RelOpt with state-of-the-art optimizers under a balanced cost model $\boldsymbol{\lambda} = (1, 1, 1)$. Across all benchmarks and metrics (gate count, circuit depth, and ECCost), RelOpt achieves the lowest cost. It reduces gate count by 15-30% relative to Qiskit and by over 10% compared to Quarl and `t|ket⟩` on arithmetic-heavy programs such as `RCA(8)` and `ModMul(3)`. RelOpt also produces shallower circuits, important for NISQ devices (*e.g.,* `Grover(4)`, `QAOA-3`), and consistently lowers ECCost through cost-aware, verified rewrites. While baselines rely on pattern matching (Qiskit, `t|ket⟩`) or black-box reinforcement learning (Quarl), RelOpt uses relational reasoning to safely apply optimizations that may relax full-state equivalence but preserve the correctness specification $\Psi$. This broader, verified transformation space enables more effective optimizations without compromising trustworthiness.

**RQ2: Verification Overhead.** Table 3 reports the average time to verify a single rewrite under different rule types and reasoning backends. Most rewrites complete within 25 ms. Simple cases such as local gate fusion or commutation re-

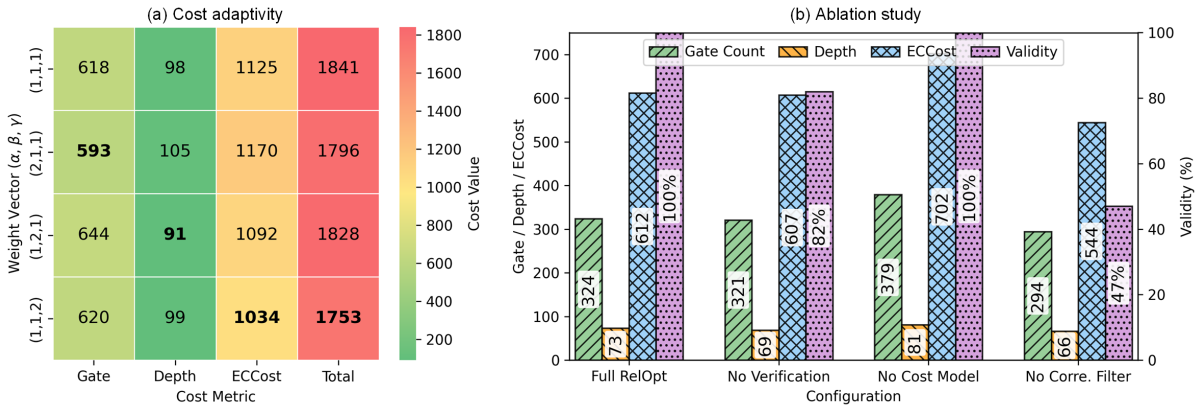


Figure 5: Cost adaptivity and ablation study.

quire only symbolic simulation, while rules involving measurement or classical control trigger SMT-based reasoning, yet still finish within tens of milliseconds. Overall, verification adds less than 10% to total compilation time, showing that relational correctness checking is practical and scales well for both interactive and batch optimization.

**RQ3: Impact of Correctness Specification.** To study how the correctness contract affects optimization, we vary $\Psi$ for the `QFT(8)` benchmark and report the resulting costs in Figure 4. Relaxing $\Psi$ from full-state equivalence to observable equivalence, and further to partial state preservation, allows RelOpt to apply more aggressive rewrites. Weaker contracts expand the space of semantically valid transformations, enabling significant cost reductions while preserving the intended behavior. This flexibility is valuable, where measurement-only correctness often suffices to achieve large performance gains without compromising functionality.

**RQ4: Adaptivity to Cost Priorities.** To test adaptivity, we run RelOpt on `QAOA-3` with different weight vectors $\boldsymbol{\lambda} = (\alpha, \beta, \gamma)$, as shown in Figure 5(a). RelOpt adjusts rewrite choices based on cost priorities: emphasizing gate count ($\alpha = 2$) reduces gates at the expense of ECCost, while prioritizing fault-tolerance ($\gamma = 2$) minimizes ECCost and overall cost with a slight increase in depth. This ability to adapt cost objectives enables deployment-specific optimization strategies to be encoded directly into the compiler.

**RQ5: Ablation Study.** We assess the impact of each RelOpt component through ablation experiments on `QFT(8)`, in Figure 5(b). Removing verification slightly improves cost but risks unsound rewrites. Disabling the cost model causes inefficient transformations and a higher overall cost. Eliminating the correctness filter yields the largest cost reduction but breaks correctness guarantees. These trends hold across small and medium circuits, confirming that verification, correctness filtering, and cost modeling are all essential for achieving safe and effective optimizations.

## Related Work

Quantum program optimization is critical for both NISQ (Li et al. 2025b) and fault-tolerant systems. Rule-based compilers such as Qiskit (Javadi-Abhari et al. 2024) and `t|ket⟩` (Sivarajah et al. 2020) use syntactic rewrites but lack correctness guarantees. Learning-based methods like Quarl (Li et al. 2024b) and Synthetiq (Paradis et al. 2024) search for rewrite sequences but remain black boxes without interpretability. RelOpt instead verifies each rewrite against a relational contract, combining correctness with cost-aware optimization. Verification approaches such as QWIRE (Paykin, Rand, and Zdancewic 2017), Quantum Hoare Logic (Ying 2012; Dai and Ying 2024), Weakest Preconditions (D'hondt and Panangaden 2006), and Separation Logic (Zhou et al. 2021) focus on single programs. RelOpt extends these to *relational verification*, proving that optimizations preserve behavior through symbolic simulation, distribution checks, and SMT solving, building on relational logics (Barthe, Crespo, and Kunz 2011; Barthe et al. 2019) without manual proofs. Classical compilers such as TVM (Chen et al. 2018), MLIR (Lattner et al. 2021), and Relay (Chen, Moreau, and et al. 2018) handle multi-objective optimization; quantum tools like ScaffCC (JavadiAbhari et al. 2015) and OpenQL (Khammassi et al. 2021) model costs but assume correctness. Certified compilers (CompCert (Leroy 2009), CakeML (Chlipala 2010)) and quantum systems (QWIRE, CoqQ (Hietala et al. 2021)) provide formal guarantees but do not address cost-driven rewriting. RelOpt aims for a practical middle ground. Moreover, we will explore integration and scalability on other frameworks such as MindSpore Quantum (Xu et al. 2024).

## Conclusion and Future Work

In this paper, we present RelOpt, a relationally verified optimization framework for quantum programs that unifies correctness preservation, cost-aware compilation, and architecture sensitivity. By combining formally validated rewrite rules with a tunable multi-objective cost model, RelOpt offers provable guarantees over a spectrum of semantic contracts, including full equivalence, measurement-level consistency, and partial trace preservation. Extensive experiments demonstrate its ability to outperform existing heuristics and learning-based optimizers across key cost metrics, without sacrificing correctness. Our work positions RelOpt as a principled bridge between formal verification and practical compiler design in quantum computing. Future directions include certified mechanization, approximate correctness reasoning, and integration into existing toolchains to support scalable, trustworthy quantum software development.

## Acknowledgments

This research was supported by the Shanghai Qi Zhi Institute Innovation Program (No. SQZ202318) and the CPS-Yangtze Delta Region Industrial Innovation Center of Quantum and Information Technology-MindSpore Quantum Open Fund.

## References

Amy, M.; et al. 2013. A Meet-in-the-Middle Algorithm for Fast Synthesis of Depth-Optimal Quantum Circuits. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 32(6): 818–830.

Arute, F.; Arya, K.; Babbush, R.; Bacon, D.; Bardin, J. C.; Barends, R.; Biswas, R.; Boixo, S.; Brandao, F. G.; Buell, D. A.; et al. 2019. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779): 505–510.

Baader, F.; and Nipkow, T. 1998. *Term rewriting and all that*. Cambridge University Press.

Barthe, G.; Crespo, J. M.; and Kunz, C. 2011. Relational verification using product programs. In *International Symposium on Formal Methods*, 200–214. Springer.

Barthe, G.; Hsu, J.; Ying, M.; Yu, N.; and Zhou, L. 2019. Relational proofs for quantum programs. *Proceedings of the ACM on Programming Languages*, 4(POPL): 1–29.

Bauer-Marquart, F.; Leue, S.; and Schilling, C. 2023. symQV: automated symbolic verification of quantum programs. In *International Symposium on Formal Methods*, 181–198. Springer.

Benton, N.; and Leperchey, B. 2005. Relational Reasoning in a Nominal Semantics for Storage. In *TLCA*, volume 3461 of *Lecture Notes in Computer Science*, 86–101. Springer.

Bocharov, A.; Roetteler, M.; and Svore, K. M. 2015. Efficient synthesis of universal repeat-until-success quantum circuits. *Physical review letters*, 114(8): 080502.

Cadar, C.; Dunbar, D.; and Engler, D. R. 2008. KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs. In *OSDI*, 209–224. USENIX Association.

Campbell, E. T.; Terhal, B. M.; and Vuillot, C. 2017. Roads towards fault-tolerant universal quantum computation. *Nature*, 549(7671): 172–179.

Chen, T.; Moreau, T.; and et al. 2018. Relay: A High-Level IR for Deep Learning. https://tvm.apache.org/docs/dev/relay.html.

Chen, T.; et al. 2018. {TVM}: An automated {End-to-End} optimizing compiler for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 578–594.

Chlipala, A. 2010. A verified compiler for an impure functional language. *ACM Sigplan Notices*, 45(1): 93–106.

Claessen, K.; and Hughes, J. 2000. QuickCheck: a lightweight tool for random testing of Haskell programs. In *ICFP*, 268–279. ACM.

Cowtan, A.; Dilkes, S.; Duncan, R.; Krajenbrink, A.; Simmons, W.; and Sivarajah, S. 2019. On the Qubit Routing Problem. In *TQC*, volume 135 of *LIPIcs*, 5:1–5:32. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Dai, A.; and Ying, M. 2024. QReach: A Reachability Analysis Tool for Quantum Markov Chains. In *International Conference on Computer Aided Verification*, 520–532. Springer.

D'hondt, E.; and Panangaden, P. 2006. Quantum weakest preconditions. *Mathematical Structures in Computer Science*, 16(3): 429–451.

Feng, G.; Xu, G.; and Long, G. 2013. Experimental realization of nonadiabatic holonomic quantum computation. *arXiv preprint arXiv:1302.0384*.

Feng, Y.; and Xu, Y. 2023. Verification of nondeterministic quantum programs. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 789–805.

Gourdin, L.; Bonneau, B.; Boulmé, S.; Monniaux, D.; and Bérard, A. 2023. Formally verifying optimizations with block simulations. *Proceedings of the ACM on Programming Languages*, 7(OOPSLA2): 59–88.

Guo, S.; Kusano, M.; and Wang, C. 2016. Conc-iSE: Incremental symbolic execution of concurrent software. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, 531–542.

Gupta, P.; Agrawal, A.; and Jha, N. K. 2006. An algorithm for synthesis of reversible logic circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(11): 2317–2330.

Hagedorn, B.; Lenfers, J.; Koehler, T.; Qin, X.; Gorlatch, S.; and Steuwer, M. 2020. Achieving high-performance the functional way: a functional pearl on expressing high-performance optimizations as rewrite strategies. *Proceedings of the ACM on Programming Languages*, 4(ICFP): 1–29.

Hietala, K.; Rand, R.; Hung, S.-H.; Wu, X.; and Hicks, M. 2021. A verified optimizer for quantum circuits. *Proceedings of the ACM on Programming Languages*, 5(POPL): 1–29.

Hietala, K.; Rand, R.; Li, L.; Hung, S.; Wu, X.; and Hicks, M. 2023. A Verified Optimizer for Quantum Circuits. *ACM Trans. Program. Lang. Syst.*, 45(3): 18:1–18:35.

Hua, F.; Wang, M.; Li, G.; Peng, B.; Liu, C.; Zheng, M.; Stein, S. A.; Ding, Y.; Zhang, E. Z.; Humble, T. S.; and Li, A. 2023. QASMTrans: A QASM Quantum Transpiler Framework for NISQ Devices. In *SC Workshops*, 1468–1477. ACM.

Javadi-Abhari, A.; Treinish, M.; Krsulich, K.; Wood, C. J.; Lishman, J.; Gacon, J.; Martiel, S.; Nation, P. D.; Bishop, L. S.; Cross, A. W.; Johnson, B. R.; and Gambetta, J. M. 2024. Quantum computing with Qiskit. *CoRR*, abs/2405.08810.

JavadiAbhari, A.; Patil, S.; Kudrow, D.; Heckey, J.; Lvov, A.; Chong, F. T.; and Martonosi, M. 2015. ScaffCC: Scalable compilation and analysis of quantum programs. *Parallel Computing*, 45: 2–17.

Khammassi, N.; et al. 2021. OpenQL: A portable quantum programming framework for quantum accelerators. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 18(1): 1–24.

Kissinger, A.; and van de Wetering, J. 2019. PyZX: Large Scale Automated Diagrammatic Reasoning. In *QPL*, volume 318 of *EPTCS*, 229–241.

Lattner, C.; Amini, M.; Bondhugula, U.; Cohen, A.; Davis, A.; Pienaar, J.; Riddle, R.; Shpeisman, T.; Vasilache, N.; and Zinenko, O. 2021. MLIR: Scaling compiler infrastructure for domain specific computation. In *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, 2–14. IEEE.

Leroy, X. 2009. Formal verification of a realistic compiler. *Communications of the ACM*, 52(7): 107–115.

Li, G.; Shi, Y.; and Javadi-Abhari, A. 2021. Software-hardware co-optimization for computational chemistry on superconducting quantum processors. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 832–845. IEEE.

Li, T.; and Zhao, Z. 2024. Moirai: Optimizing quantum serverless function orchestration via device allocation and circuit deployment. In *2024 IEEE International Conference on Web Services (ICWS)*, 707–717. IEEE.

Li, T.; et al. 2024a. Qust: Optimizing quantum neural network against spatial and temporal noise biases. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.

Li, T.; et al. 2025a. Empowering quantum serverless circuit deployment optimization via graph contrastive learning and learning-to-rank co-designed approaches. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*, 9250–9258.

Li, T.; et al. 2025b. Fortuna: Towards Efficient Selection of High-Fidelity Link for Quantum Network in the Wild. In *IEEE INFOCOM 2025-IEEE Conference on Computer Communications*, 1–10. IEEE.

Li, T.; et al. 2025c. Task-Driven Device Fingerprinting for Quantum Cloud Platforms via Modeling QNN Outcomes under Noise. *IEEE Transactions on Information Forensics and Security*.

Li, Y.; and Unruh, D. 2019. Quantum relational Hoare logic with expectations. *arXiv preprint arXiv:1903.08357*.

Li, Z.; Peng, J.; Mei, Y.; Lin, S.; Wu, Y.; Padon, O.; and Jia, Z. 2024b. Quarl: A Learning-Based Quantum Circuit Optimizer. *Proc. ACM Program. Lang.*, 8(OOPSLA1): 555–582.

Martiel, S.; and De Brugière, T. G. 2022. Architecture aware compilation of quantum circuits via lazy synthesis. *Quantum*, 6: 729.

Murali, P.; Baker, J. M.; Javadi-Abhari, A.; Chong, F. T.; and Martonosi, M. 2019. Noise-Adaptive Compiler Mappings for Noisy Intermediate-Scale Quantum Computers. In *ASPLOS*, 1015–1029. ACM.

Murillo, J. M.; García-Alonso, J.; Moguel, E.; Barzen, J.; Leymann, F.; Ali, S.; Yue, T.; Arcaini, P.; Pérez-Castillo, R.; de Guzmán, I. G. R.; Piattini, M.; Ruiz-Cortés, A.; Brogi, A.; Zhao, J.; Miranskyy, A. V.; and Wimmer, M. 2025. Quantum Software Engineering: Roadmap and Challenges Ahead. *ACM Trans. Softw. Eng. Methodol.*, 34(5): 154:1–154:48.

Nebut, C.; Fleurey, F.; Le Traon, Y.; and Jezequel, J.-M. 2006. Automatic test generation: A use case driven approach. *IEEE Transactions on Software Engineering*, 32(3): 140–155.

Nielsen, M. A.; and Chuang, I. L. 2010. *Quantum computation and quantum information*. Cambridge university press.

Paradis, A.; Dekoninck, J.; Bichsel, B.; and Vechev, M. 2024. Synthetiq: Fast and versatile quantum circuit synthesis. *Proceedings of the ACM on Programming Languages*, 8(OOPSLA1): 55–82.

Paykin, J.; Rand, R.; and Zdancewic, S. 2017. QWIRE: a core language for quantum circuits. *ACM SIGPLAN Notices*, 52(1): 846–858.

Preskill, J. 2018. Quantum Computing in the NISQ era and beyond. *Quantum*, 2: 79.

Qiu, D.; and Li, L. 2008. An overview of quantum computation models: quantum automata. *Frontiers of Computer Science in China*, 2(2): 193–207.

Sinha, N.; Rostami, P.; El Rahman Shabayek, A.; Kacem, A.; and Aouada, D. 2024. Multi-objective hardware aware neural architecture search using hardware cost diversity. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 8032–8039.

Sivarajah, S.; Dilkes, S.; Cowtan, A.; Simmons, W.; Edgington, A.; and Duncan, R. 2020. t— ket¿: a retargetable compiler for NISQ devices. *Quantum Science and Technology*, 6(1): 014003.

Wang, Q.; and Zhang, Z. 2023. Fast quantum algorithms for trace distance estimation. *IEEE Transactions on Information Theory*, 70(4): 2720–2733.

Xu, X.; et al. 2024. MindSpore Quantum: a user-friendly, high-performance, and AI-compatible quantum computing framework. *arXiv:2406.17248*.

Ying, M. 2010. Foundations of Quantum Programming (Extended Abstract). In *APLAS*, volume 6461 of *Lecture Notes in Computer Science*, 16–20. Springer.

Ying, M. 2012. Floyd–hoare logic for quantum programs. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 33(6): 1–49.

Zhou, L.; Barthe, G.; Hsu, J.; Ying, M.; and Yu, N. 2021. A quantum interpretation of bunched logic & quantum separation logic. In *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 1–14. IEEE.