Explainable Quantum Program Repair with Verifiable Proof Traces

Tingting Li[1], **Ziming Zhao**[1*], **Zhaoxuan Li**[2], **Jiongchi Yu**[3], **Xiaofei Yue**[4], **Jianwei Yin**[1*]
[1] School of Software Technology, Zhejiang University
[2] State Key Laboratory of Cyberspace Security Defense, IIE, CAS
[3] Singapore Management University
[4] Beijing Institute of Technology
Correspondence: zhaoziming@zju.edu.cn, zjuyjw@cs.zju.edu.cn

Abstract

Large language models have recently advanced automated program repair, yet most existing approaches provide only post-hoc natural-language explanations that are neither executable nor verifiable. This limitation is especially critical for quantum programs, where correctness hinges on subtle semantic properties such as circuit equivalence and fidelity preservation. We propose *Explainable Quantum Program Repair*, a framework that couples repair generation with machine-checkable executable explanations. Given a buggy quantum circuit, a language model proposes candidate repairs together with structured transformation rationales, which are compiled into proof traces and validated using formal verification backends, including circuit equivalence checking, ZX-calculus reasoning, stabilizer analysis, and quantum simulation. Only repairs whose explanations are fully verified are accepted. Experiments on QASMBench with mutation-generated quantum program bugs demonstrate that our approach achieves competitive repair success while substantially improving semantic precision and explanation faithfulness over baselines that rely on unconstrained or purely natural-language explanations.

1 Introduction

Large language models (LLMs) have recently shown strong capabilities in automated program repair, generating patches that fix bugs across a wide range of programming languages (Jimenez et al., 2024; Fried et al., 2023; Li et al., 2025e,a,f). Alongside improved repair accuracy, many systems now also provide natural-language explanations intended to justify the generated fixes (Ribeiro et al., 2016; Li et al., 2025d). However, such explanations are typically post-hoc, informal, and unverifiable, making it difficult to assess whether they faithfully reflect the actual semantics of the repair (Jacovi and Goldberg, 2020; Wiegreffe and Pinter, 2019). This gap between explanation plausibility and explanation correctness remains a central challenge for explainable AI in code (Jacovi and Goldberg, 2020).

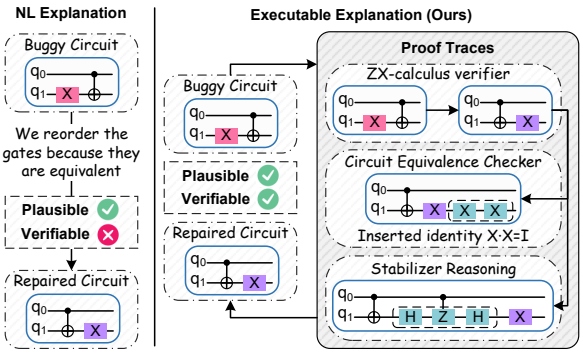

Figure 1: The illustration of executable explanation.

This challenge is particularly acute in the domain of quantum programming. Quantum programs are highly sensitive to subtle semantic constraints, including circuit equivalence under gate reordering, preservation of quantum state fidelity, and compliance with hardware-specific restrictions (Nielsen and Chuang, 2010; Duncan et al., 2020). Small syntactic changes may result in drastically different quantum behavior, while superficially different circuits may be semantically equivalent (Duncan et al., 2020). As a result, explanations that rely solely on natural language are insufficient to establish trust in automated quantum program repair: what matters is not whether an explanation sounds reasonable, but whether it can be formally validated against the semantics of the circuit (Necula, 1997).

In this work, we argue that explanations for program repair should be treated not merely as textual artifacts, but as executable objects (Necula, 1997). We propose *Explainable Quantum Program Repair* in Figure 1, a framework that repairs buggy quantum programs while producing explanations explicitly grounded in verifiable proof traces (Necula, 1997; Jacovi and Goldberg, 2020). Rather than

*Corresponding authors.

Findings of the Association for Computational Linguistics: ACL 2026, pages 30985–30995

asking a language model to "explain" a patch in free-form text, we require explanations to be instantiated as sequences of verifiable transformation steps, such as equivalence rewrites, gate commutation rules, and fidelity-preserving constraints, each of which can be independently checked by formal verification tools (Duncan et al., 2020).

Our approach operates in three stages. First, given a faulty quantum program written in QASM, Qiskit, or Cirq, together with a specification, such as semantic equivalence constraints or an oracle circuit, a large language model generates multiple candidate patches along with structured transformation rationales (Cross et al., 2017; Qiskit Community, 2017; Cirq Developers, 2021). Second, these rationales are compiled into proof obligations and checked using complementary quantum verification backends, including circuit equivalence checking, stabilizer-based reasoning, ZX-calculus rewriting, and quantum simulation (Burgholzer and Wille, 2021; Aaronson and Gottesman, 2004; Duncan et al., 2020). Third, only those candidate repairs whose explanations are fully verified are retained, yielding repaired programs accompanied by faithful, executable explanations (Jacovi and Goldberg, 2020; Necula, 1997).

To evaluate this approach, we base our experiments on QASMBench (Li et al., 2023), a widely used benchmark of quantum circuits covering diverse application domains and circuit structures. Starting from semantically correct reference circuits, we generate buggy programs using mutation-based transformations that systematically violate specific semantic properties, such as gate commutation, parameter consistency, and qubit indexing, while preserving syntactic similarity (Jia and Harman, 2011). This setup provides access to ground-truth semantics and enables precise evaluation of both repair effectiveness and explanation faithfulness, disentangling semantic correctness from superficial plausibility (Jacovi and Goldberg, 2020).

We assess our method along three primary dimensions: (i) repair success measured by the ability to produce a valid repair whose semantics are equivalent to the original circuit, (ii) minimality of code changes, and (iii) explanation faithfulness measured by verifier acceptance of executable proof traces. Our experiments show that enforcing verifiable explanations substantially improves explanation faithfulness while maintaining competitive repair success, revealing a trade-off that remains invisible when explanations are evaluated solely at the natural-language level.

In summary, we make the following contributions.

- We introduce the problem of explainable quantum program repair, formalizing explanation faithfulness as a machine-checkable property grounded in circuit semantics.
- We present a verification-driven repair framework that tightly couples language-model generation with formal quantum verification, transforming explanations into proof traces.
- We provide a benchmark and evaluation protocol that disentangles explanation plausibility from faithfulness, enabling principled analysis of explainability in language-model-based code repair systems.

2 Problem Formulation

We formalize the task of explainable quantum program repair as repairing a faulty quantum program while producing an explanation whose correctness can be mechanically verified (Necula, 1997; Hietala et al., 2021; Tao et al., 2022; Li et al., 2026). In contrast to prior work that treats explanations as informal descriptions, we explicitly require explanations to be grounded in executable proof traces (Hietala et al., 2021; Tao et al., 2022; Xu et al., 2022).

2.1 Quantum Programs and Specifications

Let P denote a quantum program represented as a quantum circuit, expressed in a concrete language such as QASM, Qiskit, or Cirq (Cross et al., 2017; Qiskit Community, 2017; Cirq Developers, 2021; Li et al., 2025b). A program P operates on a fixed number of qubits and induces a quantum operation $[\![P]\!]$, which may be characterized as a unitary transformation or a quantum channel, depending on the presence of measurements and noise (Nielsen and Chuang, 2010; Selinger, 2004; D'Hondt and Panangaden, 2006). We assume that a program is accompanied by a specification $\mathcal{S}$, which may take one or more of the following forms: (i) a test suite consisting of input-output constraints (Weimer et al., 2009; Nguyen et al., 2013; Long and Rinard, 2016), (ii) an oracle program P^* defining the intended semantics, or (iii) semantic constraints such as circuit equivalence up to global phase or fidelity thresholds (Burgholzer and Wille, 2021; Nielsen and Chuang,

2010). A program P is considered *correct* with respect to $\mathcal{S}$ if it satisfies the specification under the chosen verification backend (Ying, 2011; D'Hondt and Panangaden, 2006).

2.2 Quantum Program Repair

Given a buggy quantum program P_b that violates the specification $\mathcal{S}$, the goal of quantum program repair is to produce a repaired program P_r that:

$$[\![P_r]\!] \models \mathcal{S}. \tag{1}$$

In addition to functional correctness, we are interested in repairs that minimally deviate from the original program (Nguyen et al., 2013; Long and Rinard, 2016). We therefore associate each repair with a distance function $\mathrm{dist}(P_b, P_r)$, which may be instantiated as syntactic edit distance or gate-level modifications (Weimer et al., 2009).

2.3 Executable Explanations

We define an *explanation* as a structured object that justifies why a repaired program P_r is a valid fix for P_b. Crucially, explanations are not free-form text, but sequences of verifiable transformation steps (Hietala et al., 2021; Tao et al., 2022; Xu et al., 2022). Formally, an explanation E is a finite sequence:

$$E = \langle t_1, t_2, \ldots, t_k \rangle, \tag{2}$$

where each transformation t_i maps a quantum program to another program:

$$P_i = t_i(P_{i-1}), \quad P_0 = P_b, \quad P_k = P_r. \tag{3}$$

Each transformation t_i is associated with a rule drawn from a predefined set $\mathcal{R}$, including but not limited to: gate commutation rules, circuit rewrite rules, equivalence-preserving substitutions, and constraint-preserving optimizations (Duncan et al., 2020; Xu et al., 2022). Each rule is equipped with a verifier that can mechanically check whether the claimed step is valid under the corresponding proof obligation (Hietala et al., 2021; Tao et al., 2022).

2.4 Explanation Faithfulness

We define explanation faithfulness as the property that an explanation can be fully verified by an independent checker (Hietala et al., 2021).

Definition 1 (Faithful Explanation). An explanation E for a repair from P_b to P_r is faithful if and only if all steps in E are accepted by their corresponding verifiers and the final repaired program P_r satisfies the specification S:

$$\forall i \in \{1, \ldots, k\}, \ \mathrm{Verify}(t_i, P_{i-1}, S) = \texttt{true}, \tag{4}$$

and

$$P_r \models S. \tag{5}$$

Faithfulness is a binary property determined by the verifier and is independent of the linguistic plausibility or human readability of the explanation (Jacovi and Goldberg, 2020). This distinguishes faithful explanations from post-hoc natural-language rationales, which may appear convincing but lack semantic grounding (Qi et al., 2015).

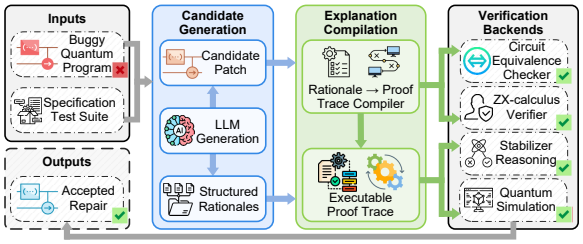

Figure 2: The overview of proposed framework.

2.5 Problem Definition

We now define the task studied in this paper.

Problem 1 (Explainable Quantum Program Repair). Given a buggy quantum program P_b and a specification $\mathcal{S}$, produce a pair (P_r, E) such that: (i) P_r satisfies $\mathcal{S}$, (ii) E is a faithful explanation of the repair process from P_b to P_r, and (iii) $\mathrm{dist}(P_b, P_r)$ is minimized. This formulation explicitly couples repair correctness with explanation verifiability, enabling principled evaluation of both dimensions (Qi et al., 2015; Hietala et al., 2021; Tao et al., 2022).

3 Proposed Method

We present a framework for explainable quantum program repair that tightly couples large language model generation with formal verification. The key design principle is to separate *generation* from *validation*: language models are used to propose candidate repairs and structured explanations, while independent verifiers determine their correctness and faithfulness. Figure 2 provides an overview of the framework.

3.1 Overview Pipeline

Given a buggy quantum program P_b and a specification $\mathcal{S}$, our framework proceeds in three stages: (i) candidate generation with structured rationales, (ii) explanation-to-proof compilation and verification,

and (iii) selection of faithful repairs. The output is a repaired program P_r paired with an explanation E whose correctness is mechanically verified.

3.2 Candidate Generation

We prompt a large language model to generate a set of candidate repairs $\{P_r^j\}_{j=1}^N$ for the buggy program P_b. Unlike standard program repair prompting (Xia and Zhang, 2024; Xia et al., 2023), we require the model to also produce a *structured rationale* for each candidate (Wei et al., 2022; Nye et al., 2021; Chen et al., 2023).

A structured rationale is a sequence of transformation steps expressed in a constrained schema, where each step specifies: (i) the transformation type (*e.g.,* gate commutation, substitution, decomposition), (ii) the target circuit region, and (iii) the intended semantic justification (*e.g.,* equivalence preservation). This schema serves two purposes. First, it reduces the ambiguity of free-form explanations. Second, it enables subsequent compilation into formal proof obligations. Importantly, the language model is not required to ensure correctness of the rationale; it only proposes candidate explanations that are later verified.

3.3 Compilation to Proof Traces

Each structured rationale is deterministically compiled into an executable proof trace. Given a rationale $R = \langle r_1, \ldots, r_k \rangle$, we construct a sequence of transformations $E = \langle t_1, \ldots, t_k \rangle$ as defined in § 2. Each transformation t_i is mapped to a rule in a predefined rule set $\mathcal{R}$. Examples include: gate commutation rules validated by algebraic equivalence, rewrite rules validated by ZX-calculus normalization (Kissinger and van de Wetering, 2020; van de Wetering, 2020), and circuit substitutions validated by simulation-based equivalence checking.

This compilation step is purely syntactic and does not involve the language model. If a rationale cannot be compiled due to schema violations or unsupported transformations, it is discarded.

3.4 Verification Backends

To validate proof traces, we employ multiple complementary verification backends (Hietala et al., 2021; Tao et al., 2022; Burgholzer and Wille, 2021; Aaronson and Gottesman, 2004; Kissinger and van de Wetering, 2020). Each backend implements a verifier $\mathrm{Verify}(t_i, P_{i-1}, \mathcal{S})$ for a subset of transformation rules. Specifically, we use: (i) circuit equivalence checkers for unitary subcircuits, (ii) stabilizer-based reasoning for Clifford circuits, (iii) ZX-calculus rewriting for general circuit equivalence, and (iv) quantum simulation to estimate fidelity when exact equivalence is intractable. A transformation is accepted only if the corresponding verifier returns `true`. Explanations that fail verification at any step are rejected in their entirety.

3.5 Faithful Repair Selection

After verification, we obtain a filtered set of candidate repairs whose explanations are faithful. From this set, we select the final repair P_r according to a ranking function that prioritizes: (i) satisfaction of the specification $\mathcal{S}$, (ii) minimal edit distance from P_b, and (iii) brevity of the proof trace.

This selection strategy ensures that faithfulness is treated as a hard constraint, while minimality and simplicity are optimized secondarily. Formally, among all candidate repairs whose explanations are faithful, we select the final repair by solving:

$$(P_r, E) = \arg\min_{(P,E)} \ \alpha \cdot \mathrm{dist}(P_b, P) + \beta \cdot |E| \quad \text{s.t. } P \models S \text{ and } E \text{ is faithful} \tag{6}$$

where $|E|$ denotes the length of the executable proof trace.

Algorithm 1 summarizes the full procedure. Our framework deliberately places the burden of correctness on verification rather than on the language model. This design allows us to leverage the generative strengths of LLMs while avoiding reliance on their implicit reasoning. Moreover, by enforcing faithfulness as a hard constraint, we ensure that explanations are not merely plausible, but semantically grounded and independently checkable.

4 Experimental Evaluation

We conduct a comprehensive experimental evaluation to assess effectiveness, robustness, and explainability of our approach to quantum circuit repair.

4.1 Experimental Setup

Benchmark Circuits. Our benchmark circuits are based on QASMBench (Li et al., 2023), a widely used benchmark of quantum circuits spanning diverse application domains and circuit structures. QASMBench provides a collection of syntactically valid and semantically correct quantum circuits expressed in QASM and compatible Python-based circuit representations. We select 500 circuits from QASMBench (*e.g.,* Amplitude Estimation,

Algorithm 1 Explainable Program Repair

Require: Buggy program P_b, specification $\mathcal{S}$
Ensure: Repaired program P_r and faithful explanation E

```
Generate candidate repairs and rationales {(P_r^j, R^j)}_{j=1}^N using an LLM
for each candidate (P_r^j, R^j) do
    Compile R^j into proof trace E^j
    if compilation fails then
        discard candidate
    else
        Verify each step in E^j
        if any step fails then
            discard candidate
        else
            Verify whether P_r^j ⊨ S
            if P_r^j ⊭ S then
                discard candidate
            end if
        end if
    end if
end for
Select (P_r, E) from remaining candidates minimizing distance and proof length
return (P_r, E)
```

GHZ State, Grover, Quantum Approximation Optimization Algorithm, Quantum Fourier Transformation, Variational Quantum Eigensolver *etc.*), covering small to large-scale circuits with 2-130 qubits (some instances are in Figure 3), including both Clifford-dominant circuits and circuits with substantial non-Clifford content, such as parameterized rotation gates and T gates. Smaller circuits allow precise semantic equivalence checking, while larger and deeper circuits expose scalability limits and trade-offs between repair coverage and explanation faithfulness.

Bug Generation via Mutation. To construct buggy circuits, we adopt a mutation-based strategy inspired by prior work on quantum fault localization and automated repair. For each original circuit C^*, we generate buggy variants by applying one mutation operator at a time, yielding a buggy circuit C_b. We consider the following mutation categories. (i) *Gate Deletion*: removing a randomly selected gate; (ii) *Gate Insertion*: inserting an identity-equivalent or non-identity gate; (iii) *Gate Reordering*: permuting adjacent non-commuting gates; (iv) *Parameter Perturbation*: modifying continuous gate parameters; (v) *Qubit Index Mutation*: altering target or control qubit indices. This process results in a total of 5,000 buggy circuits (10 mutations per original circuit), ensuring controlled and reproducible ground truth for evaluation.

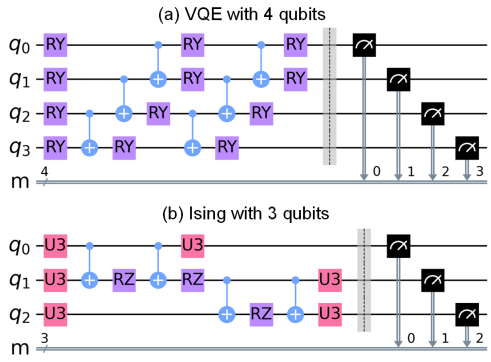

Figure 3: Some instances of quantum circuits.

Table 1: Comparison of baselines and our approach.

Method	Repair Capability	Quantum-aware	Natural-Language Explanation	Uses Task / Test Info	Explanation Verifiable	Faithfulness Enforced
LLM-Repair	✓	✗	✗	✗	✗	✗
LLM+NL-Explain	✓	✗	✓	✗	✗	✗
LLM-QAPR	✓	✓	◐	✓	✗	✗
Syn-QSD	✓	✓	✗	◐	✗	✗
Syn-FAST	✓	✓	✗	◐	✗	✗
HornBro	✓	✓	✗	✓	◐	◐
Ours	✓	✓	✓	✓	✓	✓

✓: Yes, ◐: Partial, ✗: No

Correctness Criterion. A repaired circuit C_r is considered *correct* if it is semantically equivalent to the original circuit C^* from which the buggy circuit was derived. Semantic equivalence is defined up to global phase and verified using the same formal backends employed for explanation validation, including circuit equivalence checking, ZX-calculus rewriting, stabilizer reasoning, and simulation-based fidelity estimation. Importantly, the original circuit C^* is used *only* as an oracle for evaluation and is not accessible to the repair algorithm.

Baselines. We compare against the following representative baselines. (i) *HornBro* (Tan et al., 2025): a state-of-the-art automated quantum program repair framework based on homotopy-like optimization; (ii) *LLM-QAPR* (Guo et al., 2024): an LLM-based quantum repair approach without explanation constraints; (iii) *Syn-QSD* (Li et al., 2024): a synthesis-based repair method using unitary decomposition; (iv) *Syn-FAST* (Younis et al., 2021): a heuristic synthesis-based repair technique optimized for speed; (v) *LLM-Repair* (Xia and Zhang, 2024): a pure language-model-based repair method without explanations; (vi) *LLM+NL-Explain* (Tian et al., 2022): LLM-based repair augmented with free-form natural-language explanations. Table 1 summarizes the key differences among all baselines

Table 2: Overall repair performance on QASMBench with mutation-generated bugs.

Method	Succ. (%)	Pre. (%)	Edit Dist.	Faith. (%)	Time (s)
HornBro	74.6	82.3	0.41	-	38.2
Syn-QSD	61.2	79.5	0.56	-	182.4
Syn-FAST	58.7	75.1	0.49	-	96.8
LLM-QAPR	72.9	68.4	0.33	-	12.7
LLM-Repair	75.3	65.9	0.31	-	9.8
LLM+NL-Exp.	74.8	66.7	0.32	18.6	11.1
Ours	**76.1**	**96.1**	**0.28**	**94.8**	21.3

and our approach. Unless otherwise specified, all LLM-based methods use the same backbone model, `GPT-4o-mini`, with identical decoding parameters. To examine the robustness of our approach across different language models, we further evaluate representative LLM backbones in § 4.6.

Metrics. We evaluate each approach using the following metrics. (i) *Repair Success Rate*: percentage of buggy circuits for which at least one generated candidate repair is semantically equivalent to the original circuit; (ii) *Semantic Precision*: percentage of accepted repairs that are semantically equivalent to the original circuit; (iii) *Edit Distance*: normalized gate-level edit distance between C_b and C_r; (iv) *Explanation Faithfulness*: percentage of explanations fully verified by formal backends; (v) *Repair Time*: average time to produce a valid repair.

4.2 Effectiveness Evaluations

Table 2 summarizes the overall repair performance on QASMBench with mutation-generated bugs. Our approach achieves the highest repair success rate among all LLM-based methods, demonstrating that enforcing executable explanations does not reduce repair effectiveness when multiple candidate repairs are generated and verified. More importantly, our method substantially outperforms all baselines in semantic precision. With a precision of 96.1%, most accepted repairs are semantically equivalent to the original circuits, whereas LLM-based baselines frequently produce spurious repairs that alter circuit semantics. In addition, our approach yields the lowest average edit distance, indicating that verified repairs tend to make minimal modifications to the buggy circuits. Finally, our method is the only approach that consistently produces faithful explanations. Executable explanations achieve a verification acceptance rate of 94.8%, compared to only 18.6% for natural-language explanations. Although verification introduces additional overhead, the overall repair time remains practical and significantly lower than that of search-based baselines.

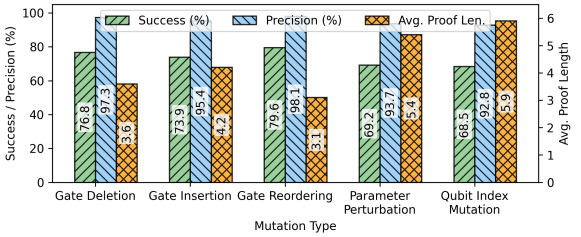

Figure 4: Performance of ours by mutation category.

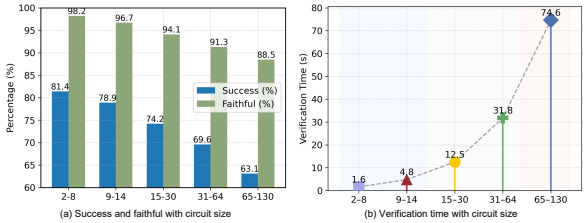

Figure 5: Scalability analysis with different circuit size.

4.3 Results by Mutation Type

Figure 4 reports the performance of our approach across different mutation categories. We observe that gate reordering and gate deletion mutations are repaired most effectively, achieving both high success rates and high semantic precision. These mutations often correspond to well-understood equivalence and commutation rules, which can be captured by short and easily verifiable proof traces. In contrast, parameter perturbation and qubit index mutations are more challenging. Repairing these bugs typically requires longer proof traces and more complex semantic reasoning, resulting in slightly lower success rates and precision. Nevertheless, even for these harder mutation types, our method maintains semantic precision above 92%, indicating that explanation verification remains effective across diverse bug categories. Overall, the results suggest that the complexity of the required proof trace correlates with mutation difficulty, and that executable explanations provide a robust mechanism for reasoning about a wide range of quantum circuit errors.

4.4 Ablation Study

We conduct an ablation study to evaluate the contribution of individual verification components. As shown in Table 3, removing any verification backend leads to a noticeable drop in explanation faithfulness, confirming that no single verifier is sufficient to support diverse quantum circuit structures. Among the components, ZX-calculus reasoning plays the most critical role. Removing ZX-calculus causes the largest decrease in faithfulness, indicating its importance for reasoning about circuit equivalence and rewrite-based repairs. Stabilizer

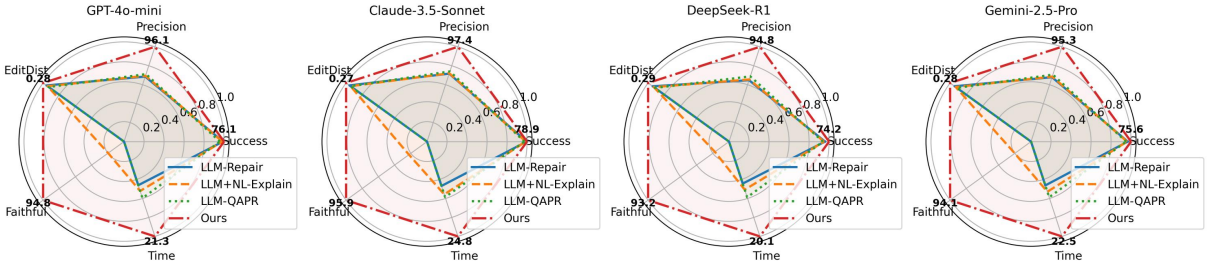

Figure 6: Impact of different LLM backbones.

Table 3: Ablation results of verification backends.

Configuration	Success (%)	Faithful (%)
Full System	76.1	94.8
w/o ZX-calculus	71.7	81.5
w/o Stabilizer Reasoning	73.1	84.2
w/o Simulation	74.9	86.9

reasoning and simulation-based verification also contribute substantially, particularly for Clifford-dominant circuits and parameterized gates, respectively. In addition to reducing faithfulness, removing verification components also slightly lowers repair success. This suggests that verification not only filters incorrect repairs but also guides the selection of reliable candidates when multiple repair options are available. Overall, the results highlight the necessity of combining complementary verification backends to achieve robust and faithful explainable quantum program repair.

4.5 Scalability Analysis

To assess the scalability of our approach, we analyze repair effectiveness and verification overhead as circuit size increases. We group benchmark circuits by qubit count and report average repair success rate, explanation faithfulness, and verification time per repair. Figure 5 shows that our approach scales effectively to medium-sized circuits while maintaining high explanation faithfulness. For circuits with up to 14 qubits, verification overhead remains modest and repair success exceeds 78%. As circuit size increases beyond this range, explanation verification gradually becomes the dominant cost, reflecting the inherent complexity of semantic equivalence checking in large quantum circuits. Despite the increased verification cost, explanation faithfulness remains consistently high across all scales, staying above 88% even for circuits with more than 65 qubits. Repair success degrades gracefully rather than collapsing, indicating that multi-candidate generation combined with verification continues to be effective under larger search spaces. Overall, these results demonstrate

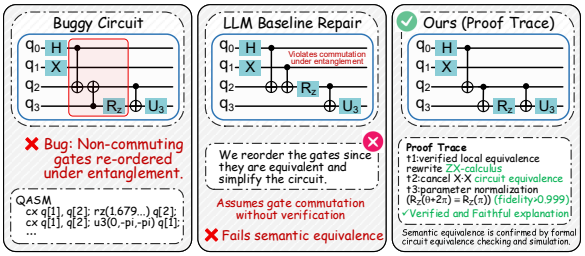

Figure 7: The case study of proposed framework.

that executable explanations introduce a principled and predictable scalability trade-off. While verification limits throughput for very large circuits, it provides strong semantic guarantees that are difficult to obtain through test-based or unconstrained repair methods. This trade-off is inherent to faithful explainability and highlights the importance of balancing coverage and trustworthiness.

4.6 Effect of Different LLM Backbones

Figure 6 reports the impact of different language model backbones on all LLM-based repair methods. Across baselines, stronger language models such as Claude-3.5-Sonnet generally achieve higher repair success and semantic precision, reflecting improved code generation and reasoning capabilities. However, for unconstrained LLM-based methods, improvements in model quality do not translate into reliable semantic correctness, as precision remains limited and unverified explanations persist. In contrast, our approach consistently maintains high semantic precision and explanation faithfulness across all evaluated models. Even when instantiated with weaker or more compact backbones, executable explanations effectively filter spurious repairs, preserving semantic correctness. Using stronger models primarily improves repair success and reduces proof complexity, rather than altering the core reliability guarantees. These results indicate that the effectiveness of our framework does not depend on a specific language model. Instead, verification-driven filtering plays a dominant role in ensuring correctness, while advances in language modeling further enhance coverage and efficiency.

4.7 Case Study

To illustrate how executable explanations improve repair reliability, we present a representative case study from the QASMBench benchmark. Figure 7 shows a buggy quantum circuit together with repairs generated by an unconstrained LLM baseline and by our approach. In this example, the bug is introduced by reordering two non-commuting gates, which subtly alters the circuit semantics. The LLM+NL-Explain baseline generates a syntactically valid repair accompanied by a plausible natural-language explanation, claiming that the reordered gates are equivalent. However, formal equivalence checking reveals that the repaired circuit is not semantically equivalent to the original one, despite the explanation appearing reasonable. In contrast, our approach generates a repair whose explanation is compiled into an executable proof trace. Each transformation step is independently verified using ZX-calculus rewriting and circuit equivalence checking. Only after all proof obligations are discharged is the repair accepted. This example highlights how executable explanations prevent plausible but incorrect repairs from being accepted, and demonstrates the role of proof traces in bridging the gap between explanation plausibility and semantic correctness.

5 Related Work

Our work relates to several lines of research spanning program repair, explainable AI for code, and quantum software engineering.

Program Repair with Language Models. Recent advances in large language models have enabled data-driven approaches to automated program repair, demonstrating strong performance on classical programming languages (Xia et al., 2023; Fan et al., 2023; Ribeiro, 2023). These methods typically generate candidate patches via sequence-to-sequence modeling or in-context learning, sometimes augmented with test execution or search (Chen et al., 2021; Lutellier et al., 2020; Ye et al., 2022). However, most prior work focuses exclusively on producing correct patches, without addressing the faithfulness or verifiability of explanations (Xia et al., 2023; Fan et al., 2023). In contrast, our work explicitly couples repair generation with executable explanations, treating explanation correctness as a first-class objective.

Explainability for Code Generation. Explainable AI for code has attracted increasing attention, with approaches ranging from natural-language rationales to intermediate representations such as execution traces or symbolic states (DeYoung et al., 2020; Shin et al., 2018; Li et al., 2025c). While natural-language explanations can improve perceived interpretability, prior studies have shown that such explanations are often unfaithful to the underlying model behavior (Serrano and Smith, 2019; Pruthi et al., 2020). Our work aligns with recent calls for faithful explanations, but differs in that we operationalize faithfulness through formal verification, yielding explanations that are independently checkable rather than merely human-interpretable.

Faithfulness and Verifiable Explanations. Several recent works propose defining explanation faithfulness in terms of alignment with model internals or causal influence (Ross et al., 2017; DeYoung et al., 2020; Zhao et al., 2026). Our approach is complementary: rather than analyzing the internal reasoning of the language model, we require explanations to justify the semantic validity of the output program. This shifts the focus from explaining the model to explaining the artifact it produces, a perspective well-suited to program synthesis and repair.

Quantum Program Repair and Verification. Quantum program repair and verification have been studied using rule-based rewriting, symbolic reasoning, and equivalence checking (Paykin et al., 2017; Kissinger and van de Wetering, 2020). These methods typically rely on handcrafted rules and are not designed to interface with language models or produce human-consumable explanations. We build on these verification techniques as backends, but repurpose them as explanation validators rather than repair engines. This design allows us to leverage mature quantum verification tools while retaining the flexibility of language-model generation.

6 Discussion and Conclusion

We introduced a framework for explainable quantum program repair that treats explanations as executable, verifiable objects rather than post-hoc natural-language narratives. By coupling language-model-based generation with formal verification, our approach produces repairs accompanied by proof traces whose semantic correctness can be independently checked. Our results demonstrate that enforcing explanation verifiability substantially improves faithfulness while maintaining competitive repair success, highlighting a fundamental gap be-

tween explanation plausibility and correctness. Although our evaluation focuses on quantum programs, the underlying idea of executable explanations is broadly applicable to other domains where semantic properties can be mechanically verified. Overall, this work advocates for grounding explanations in verifiable semantics as a principled path toward more trustworthy and interpretable AI systems for code and beyond.

Limitations

Our approach makes several design choices that naturally define its current scope. First, the effectiveness of executable explanations depends on the availability and scalability of quantum verification backends. For large or highly non-Clifford circuits, verification may become computationally expensive, which can limit throughput or lead to conservative rejection of candidate repairs. Second, while verification ensures correctness, the generation of structured rationales remains dependent on the language model. Incomplete or underspecified rationales may prevent otherwise valid repairs from being accepted, suggesting opportunities for improving rationale generation through more structured prompting or iterative refinement. Third, our evaluation focuses on small- to medium-scale quantum circuits, reflecting common practice in current quantum software development; extending the approach to larger circuits and hardware-aware constraints is an important direction for future work. Finally, although we study executable explanations in the context of quantum program repair, exploring applicability to other code-related and decision-making domains remains an open question.

References

Scott Aaronson and Daniel Gottesman. 2004. Improved simulation of stabilizer circuits. *Physical Review A*, 70(5):052328.

Lukas Burgholzer and Robert Wille. 2021. QCEC: A JKQ tool for quantum circuit equivalence checking. *Softw. Impacts*, 7:100051.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*.

Zimin Chen, Steve Kommrusch, Michele Tufano, Louis-Noël Pouchet, Denys Poshyvanyk, and Martin Monperrus. 2021. Sequence-to-sequence learning for end-to-end program repair. *IEEE Transactions on Software Engineering*, 47(9):1943–1959.

Cirq Developers. 2021. Cirq.

Andrew W. Cross, Lev S. Bishop, John A. Smolin, and Jay M. Gambetta. 2017. Open quantum assembly language. arXiv:1707.03429.

Jay DeYoung, Sarthak Jain, Nazneen Fatema Rajani, Eric Lehman, Caiming Xiong, Richard Socher, and Byron C. Wallace. 2020. ERASER: A benchmark to evaluate rationalized NLP models. In *ACL*, pages 4443–4458. Association for Computational Linguistics.

Ellie D'Hondt and Prakash Panangaden. 2006. Quantum weakest preconditions. *Mathematical Structures in Computer Science*, 16(3):429–451.

Ross Duncan, Aleks Kissinger, Simon Perdrix, and John van de Wetering. 2020. Graph-theoretic simplification of quantum circuits with the ZX-calculus. *Quantum*, 4:279.

Zhiyu Fan, Xiang Gao, Martin Mirchev, Abhik Roychoudhury, and Shin Hwei Tan. 2023. Automated repair of programs from large language models. In *Proceedings of the 45th International Conference on Software Engineering (ICSE)*, pages 1469–1481.

Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Scott Yih, Luke Zettlemoyer, and Mike Lewis. 2023. Incoder: A generative model for code infilling and synthesis. In *International Conference on Learning Representations (ICLR)*.

Xiaoyu Guo, Jianjun Zhao, and Pengzhan Zhao. 2024. On repairing quantum programs using chatgpt. In *Q-SE@ICSE*, pages 9–16. ACM.

Kesha Hietala, Robert Rand, Shih-Han Hung, Xiaodi Wu, and Michael Hicks. 2021. A verified optimizer for quantum circuits. *Proceedings of the ACM on Programming Languages*, 5(POPL):37:1–37:29.

Alon Jacovi and Yoav Goldberg. 2020. Towards faithfully interpretable nlp systems: How should we define and evaluate faithfulness? In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 4198–4205.

Yue Jia and Mark Harman. 2011. An analysis and survey of the development of mutation testing. *IEEE Transactions on Software Engineering*, 37(5):649–678.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. 2024. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations (ICLR)*.

Aleks Kissinger and John van de Wetering. 2020. PyZX: Large scale automated diagrammatic reasoning. In *Electronic Proceedings in Theoretical Computer Science (EPTCS)*, volume 318, pages 229–242.

Ang Li, Samuel Stein, Sriram Krishnamoorthy, and James Ang. 2023. QASMBench: A low-level QASM benchmark suite for NISQ evaluation and simulation. *ACM Transactions on Quantum Computing*, 4(2):1–26.

Tingting Li, Liqiang Lu, Ziming Zhao, Ziqi Tan, Siwei Tan, and Jianwei Yin. 2025a. Qust: Optimizing quantum neural network against spatial and temporal noise biases. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 44(4):1434–1447.

Tingting Li, Ziming Zhao, Liqiang Lu, Siwei Tan, and Jianwei Yin. 2025b. Empowering quantum error traceability with moe for automatic calibration. In *DATE*, pages 1–7. IEEE.

Tingting Li, Ziming Zhao, and Jianwei Yin. 2025c. Autofid: Adaptive and noise-aware fidelity measurement for quantum programs via circuit graph analysis. In *ASE*, pages 2631–2643. IEEE.

Tingting Li, Ziming Zhao, and Jianwei Yin. 2025d. Empowering quantum serverless circuit deployment optimization via graph contrastive learning and learning-to-rank co-designed approaches. *IJCAI 2025*.

Tingting Li, Ziming Zhao, and Jianwei Yin. 2025e. Fortuna: Towards efficient selection of high-fidelity link for quantum network in the wild. In *INFOCOM*, pages 1–10. IEEE.

Tingting Li, Ziming Zhao, and Jianwei Yin. 2025f. Task-driven device fingerprinting for quantum cloud platforms via modeling QNN outcomes under noise. *IEEE Trans. Inf. Forensics Secur.*, 20:11249–11263.

Tingting Li, Ziming Zhao, and Jianwei Yin. 2026. Adaptive fidelity estimation for quantum programs with graph-guided noise awareness. In *AAAI*, pages 695–703. AAAI Press.

Yuechen Li, Hanyu Pei, Linzhi Huang, Beibei Yin, and Kai-Yuan Cai. 2024. Automatic repair of quantum programs via unitary operation. *ACM Trans. Softw. Eng. Methodol.*, 33(6):154.

Fan Long and Martin C. Rinard. 2016. Automatic patch generation by learning correct code. In *POPL*, pages 298–312. ACM.

Thibaud Lutellier, Hung Viet Pham, Lawrence Pang, Yitong Li, Moshi Wei, and Lin Tan. 2020. Coconut: Combining context-aware neural translation models using ensemble for program repair. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, pages 101–114.

George C. Necula. 1997. Proof-carrying code. In *Conference Record of POPL'97: The 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 106–119. ACM Press.

Hoang Duong Thien Nguyen, Dawei Qi, Abhik Roychoudhury, and Satish Chandra. 2013. Semfix: Program repair via semantic analysis. In *Proceedings of the 35th International Conference on Software Engineering (ICSE)*, pages 772–781.

Michael A. Nielsen and Isaac L. Chuang. 2010. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press.

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Augustus Odena. 2021. Show your work: Scratchpads for intermediate computation with language models. arXiv preprint. *Preprint*, arXiv:2112.00114.

Jennifer Paykin, Robert Rand, and Steve Zdancewic. 2017. Qwire: a core language for quantum circuits. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL)*, pages 846–858.

Danish Pruthi, Mansi Gupta, Bhuwan Dhingra, Graham Neubig, and Zachary C. Lipton. 2020. Learning to deceive with attention-based explanations. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 4782–4793.

Zichao Qi, Fan Long, Sara Achour, and Martin C. Rinard. 2015. An analysis of patch plausibility and correctness for generate-and-validate patch generation systems. In *ISSTA*, pages 24–36. ACM.

Qiskit Community. 2017. Qiskit: An open-source framework for quantum computing.

Francisco Ribeiro. 2023. Large language models for automated program repair. In *SPLASH Companion*, pages 7–9. ACM.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "why should i trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1135–1144.

Andrew Slavin Ross, Michael C. Hughes, and Finale Doshi-Velez. 2017. Right for the right reasons: Training differentiable models by constraining their explanations. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2662–2670.

Peter Selinger. 2004. Towards a quantum programming language. *Mathematical Structures in Computer Science*, 14(4):527–586.

Sofia Serrano and Noah A. Smith. 2019. Is attention interpretable? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 2931–2951.

Richard Shin, Illia Polosukhin, and Dawn Song. 2018. Improving neural program synthesis with inferred execution traces. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 8931–8940.

Siwei Tan, Liqiang Lu, Debin Xiang, Tianyao Chu, Congliang Lang, Jintao Chen, Xing Hu, and Jianwei Yin. 2025. Hornbro: Homotopy-like method for automated quantum program repair. *Proceedings of the ACM on Software Engineering*, 2(FSE):734–756.

Runzhou Tao, Yunong Shi, Jianan Yao, Xupeng Li, Ali Javadi-Abhari, Andrew W. Cross, Frederic T. Chong, and Ronghui Gu. 2022. Giallar: Push-button verification for the qiskit quantum compiler. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI)*, pages 641–656.

Haoye Tian, Xunzhu Tang, Andrew Habib, Shangwen Wang, Kui Liu, Xin Xia, Jacques Klein, and Tégawendé F. Bissyandé. 2022. Is this change the answer to that problem?: Correlating descriptions of bug and code changes for evaluating patch correctness. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 59:1–59:13.

John van de Wetering. 2020. Zx-calculus for the working quantum computer scientist. arXiv preprint. *Preprint*, arXiv:2012.13966.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 24824–24837.

Westley Weimer, Thanh Vu Nguyen, Claire Le Goues, and Stephanie Forrest. 2009. Automatically finding patches using genetic programming. In *Proceedings of the 31st International Conference on Software Engineering (ICSE)*, pages 364–374.

Sarah Wiegreffe and Yuval Pinter. 2019. Attention is not not explanation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*.

Chunqiu Steven Xia, Yuxiang Wei, and Lingming Zhang. 2023. Automated program repair in the era of large pre-trained language models. In *Proceedings of the 45th International Conference on Software Engineering (ICSE)*, pages 1482–1494.

Chunqiu Steven Xia and Lingming Zhang. 2024. Automated program repair via conversation: Fixing 162 out of 337 bugs for $0.42 each using ChatGPT. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, pages 819–831.

Mingkuan Xu, Zikun Li, Oded Padon, Sina Lin, Jessica Pointing, Auguste Hirth, Henry Ma, Jens Palsberg, Alex Aiken, Umut A. Acar, and Zhihao Jia. 2022. Quartz: Superoptimization of quantum circuits. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI)*, pages 625–640.

He Ye, Matias Martinez, and Martin Monperrus. 2022. Neural program repair with execution-based backpropagation. In *Proceedings of the 44th International Conference on Software Engineering (ICSE)*, pages 1506–1518.

Mingsheng Ying. 2011. Floyd–hoare logic for quantum programs. *ACM Transactions on Programming Languages and Systems*, 33(6):19:1–19:49.

Ed Younis, Koushik Sen, Katherine A. Yelick, and Costin Iancu. 2021. QFAST: Conflating search and numerical optimization for scalable quantum circuit synthesis. In *IEEE International Conference on Quantum Computing and Engineering (QCE)*, pages 232–243.

Ziming Zhao, Tingting Li, Zhaoxuan Li, and Jianwei Yin. 2026. Relational verification for cost-aware quantum program optimization. In *AAAI*, pages 14414–14422. AAAI Press.