HornBro: Homotopy-Like Method for Automated Quantum Program Repair

SIWEI TAN, Zhejiang University, China
LIQIANG LU*, Zhejiang University, China
DEBIN XIANG, Zhejiang University, China
TIANYAO CHU, Zhejiang University, China
CONGLIANG LANG, Zhejiang University, China
JINTAO CHEN, Zhejiang University, China
XING HU, Zhejiang University, China
JIANWEI YIN*, Zhejiang University, China

Quantum programs provide exponential speedups compared to classical programs in certain areas, but they also inevitably encounter logical faults. Automatically repairing quantum programs is much more challenging than repairing classical programs due to the non-replicability of data, the vast search space of program inputs, and the new programming paradigm. Existing works based on semantic-based or learning-based program repair techniques are fundamentally limited in repairing efficiency and effectiveness. In this work, we propose HornBro, an efficient framework for automated quantum program repair. The key insight of HornBro lies in the homotopy-like method, which iteratively switches between the classical and quantum parts. This approach allows the repair tasks to be efficiently offloaded to the most suitable platforms, enabling a progressive convergence toward the correct program. We start by designing an implication assertion pragma to enable rigorous specifications of quantum program behavior, which helps to generate a quantum test suite automatically. This suite leverages the orthonormal bases of quantum programs to accommodate different encoding schemes. Given a fixed number of test cases, it allows the maximum input coverage of potential counter-example candidates. Then, we develop a Clifford approximation method with an SMT-based search to transform the patch localization program into a symbolic reasoning problem. Finally, we offload the computationally intensive repair of gate parameters to quantum hardware by leveraging the differentiability of quantum gates. Experiments suggest that HornBro increases the repair success rate by more than 62.5% compared to the existing repair techniques, supporting more types of quantum bugs. It also achieves 35.7× speedup in the repair and 99.9% gate reduction of the patch.

CCS Concepts: • **Software and its engineering** → **Software creation and management**.

Additional Key Words and Phrases: Automated Program Repair, Quantum Computing, Program Verification

*Corresponding authors

Authors' Contact Information: Siwei Tan, Zhejiang University, Hangzhou, China, siweitan@zju.edu.cn; Liqiang Lu, Zhejiang University, Hangzhou, China, liqianglu@zju.edu.cn; Debin Xiang, Zhejiang University, Hangzhou, China, db.xiang@zju.edu.cn; Tianyao Chu, Zhejiang University, Hangzhou, China, tianyao_chu@zju.edu.cn; Congliang Lang, Zhejiang University, Hangzhou, China, langcongliang@zju.edu.cn; Jintao Chen, Zhejiang University, Hangzhou, China, chenjintao@zju.edu.cn; Xing Hu, Zhejiang University, Hangzhou, China, xinghu@zju.edu.cn; Jianwei Yin, Zhejiang University, Hangzhou, China, zjuyjw@cs.zju.edu.cn.

ACM 2994-970X/2025/7-ARTFSE034
https://doi.org/10.1145/3715751

ACM Reference Format:
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. *Proc. ACM Softw. Eng.* 2, FSE, Article FSE034 (July 2025), 23 pages. https://doi.org/10.1145/3715751

1 Introduction

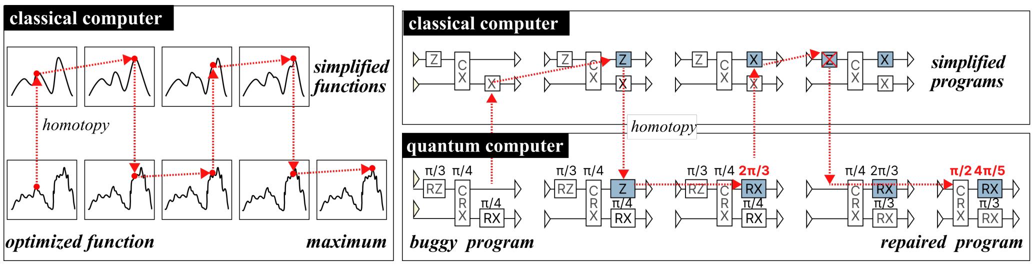
Quantum programs have suggested remarkable speedups in certain domains, such as financial technology [6] and chemistry analysis [11]. However, quantum programs inevitably encounter logical faults caused by human mistakes, necessitating program debugging. Unfortunately, the information representation and operations of quantum programs are different in semantics and behaviors compared to the classical programs [3], which leads to more unresolved bugs. For example, on Stack Overflow [54], only 21% of bugs related to quantum programs are resolved, which is lower (4.1 times less) than that of classical programs. Debugging quantum programs is difficult even for the experts. For example, Qiskit [1] (i.e., the most popular quantum package) is managed by IBM experts, while 26/51 quantum algorithm issues remain unsolved on github. Therefore, compared to manual repair, automated program repair (APR) becomes a natural idea.

Unfortunately, current advanced APR techniques can not be directly transferred to automated quantum program repair (Q-APR). Typical APR techniques heuristically generate and validate patches until the program passes the test [41, 43]. Here, a patch is a program module that can correct the program. However, quantum programs employ a quantum circuit model, exposing an extremely large patch space. For example, a 5-qubit quantum program yields 5.7×10^{15} possible combinations of quantum operations as patches. Moreover, prior APR works use the neural networks [41] or semantic information [43] to facilitate the repair. However, current quantum programming languages lack semantic information, such as type systems and exception systems, as well as sufficient program data to train neural networks effectively.

Validating patches in Q-APR further increases the difficulty. Tomograph on the quantum hardware to obtain the density matrix of the state requires exponential complexity due to quantum collapse. On the other hand, validation on classical computers by simulation also suffers from exponential complexity. For example, simulating a 53-qubit quantum program requires 128 PiB of memory and 20 days on the Summit supercomputer [28]. The number of test inputs to ensure the patch is correct further increases the overhead. For instance, Quito [59] requires 4.8×10^6 test inputs to validate a patch of an 11-qubit quantum lock program.

To alleviate these challenges, several Q-APR methods have adopted large-language model [24] and program synthesis [65] to fix bugs. For example, Guo et al. [24] employs ChatGPT to repair quantum programs. However, the high cost of executing a quantum program inevitably results in limited training data, leading to less than a 20% success rate when repairing quantum faults on Stack Overflow. Li et al. [37] appends large unitary matrices to the buggy program and approximates the correct program. It relies on the synthesis of unitary matrices to executable quantum gates. Thus, for quantum programs with more than eight qubits, even the optimal synthesis algorithms have to take more than one year to output high-quality program [65] or synthesize in minutes but generate thousands of redundant gates [51].

In this work, we propose HornBro, a Q-APR repair framework that shows a high success rate. HornBro employs a homotopy-like method that iteratively switches between classical and quantum computers to locate and repair faults, as shown in Figure 1a. A typical homotopy method solves a complex optimization problem by gradually transforming it into a simpler one and evolving it into the original problem. To do this, it creates a continuous path, namely *homotopy*. Inspired by this, Figure 1b transforms the quantum program into a more simplified, affordable formation to analyze on classical computers. In this manner, HornBro iteratively optimizes the reduced program to



(a) Homotopy optimization (b) Homotopy-like repair of quantum program.

Fig. 1. Homotopy-like method for quantum program repair.

identify the location of coarse-grained patches and then refine the quantum program to capture the fine-grained gate parameters. As a result, it progressively converges towards the correct program.

In the implementation, HornBro starts with an implication assertion pragma to specify the correct behavior of the quantum program. It exploits the orthonormal bases of the programs and generates a test suite with maximum input coverage. To eliminate the high computational overhead, HornBro transforms the patch localization into a symbolic reasoning problem by a Clifford approximation. This approach achieves high accuracy and reduces the complexity of localization from $\mathcal{O}(4^N)$ to approximately $\mathcal{O}(N^5)$. The localization determines the location and operated qubits of the patches. HornBro, then searches for the correct parameters of the patch. By exploiting the differentiability of quantum gates, HornBro offloads the computationally intensive program executions to quantum hardware. The main contributions of this paper are summarized as follows:

- We propose a repair flow of quantum programs, which enables fast and accurate program repair with high generality by a homotopy-like method.
- We introduce a test suite generation method that supports various quantum input encoding schemes, which achieves higher input coverage by combining orthonormal bases of quantum programs.
- We reduce the complexity of the patch localization of Q-APR to a close to a polynomial degree by formulating it as a symbolic reasoning problem and exploiting the differentiability of the quantum gates.

The experiments show that HornBro improves the repair success rate by 93.3% (14.9×) and 62.5% (2.67×) compared to the existing LLM-based technique [24] and synthesis-based technique [51, 65]. Furthermore, HornBro can repair a 20-qubit quantum program using 28.2 minutes and 249 additional gates, while prior synthesis-based methods may require more than 6 months and more than 10^6 additional gates.

2 Background

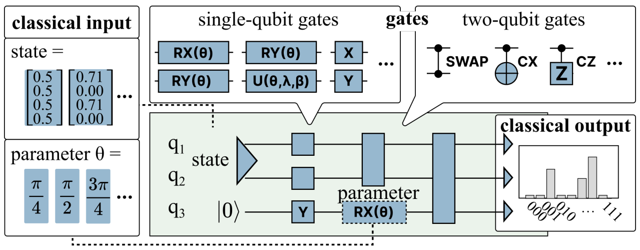
2.1 Quantum Program

Current quantum computers are mainly programmed by the **quantum circuit model**. Figure 2 presents an example of programming a quantum circuit using Python. A quantum program consists of classical inputs (i.e., float values), qubits, quantum gates, and classical outputs (i.e., distributions). The program is initialized as a `QuantumCircuit` instance (line 4). The input of the quantum program can be

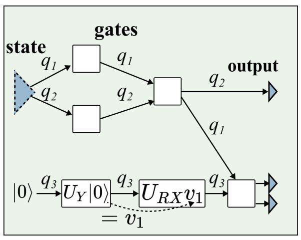
```
1  from qiskit import QuantumCircuit
2  from numpy import pi

3  def execute(state, parameter):          // classical input
4      circuit = QuantumCircuit(3)         // qubit
5      circuit.initialize(state, 0)        // state encoding
6      circuit.ry(pi/2, 2)                 // gate
7      ...
8      circuit.rx(parameter,1)             // gate encoding
9      ...
10     circuit.measure([0,1,2])            // classical output
```

Fig. 2. Example of programming a quantum circuit using Qiskit [1].



(a) Visualization representation of quantum program.



(b) Computational graph model of a quantum program.

Fig. 3. Representations of quantum circuit.

encoded into the initial state of the qubits (line 5) or the parameter of the gate (line 8). The gates are declared by the functions of the class to perform computation. For example, `circuit.RY(pi/2,2)` represents an RY gate with parameter $\pi/2$ operating on qubit 2. The program output is obtained by measuring qubits (line 10). The output is a probability distribution p of measuring different bit-strings, e.g., $p[000] = 10\%$ and $p[111] = 90\%$ for a three-qubit measurement.

Figure 3 presents the visual representation and the mathematical formulation of a quantum circuit. The programming language describes a quantum circuit represented in a sequential view in Figure 3a. Each qubit is displayed as a horizontal line, and quantum gates operate on qubits from left to right. Gates in the same layer can be executed in parallel. There are multiple types of gates. Single-qubit gates operate on one qubit, such as the RX, RY, X, and Y gates. Two-qubit gates, such as the SWAP and CZ gates, operate on two qubits. Some gates have parameters; for example, the RX gate has a parameter θ, while the X gate has no parameters.

The computation of the quantum circuit can be represented by a **computational graph model** shown in Figure 3b. The nodes are quantum states or gates, and the label of the edges represents the operated qubits of the gates. In the program, N qubits represent a vector with 2^N complex elements. **Quantum gates** represent a unitary matrix of size $2^N \times 2^N$. When a gate operates on a qubit, the computation is performed as a matrix-vector multiplication:

$$v_2 = Uv_1. \tag{1}$$

v_1 is input into the next gate operation. The unitary matrix U of a gate is determined by the type of this gate and its parameters. For example, a Y gate and a RX gate in Figure 3a operate the qubit 3 with initial state $|0\rangle$. The computation of the Y gate is formulated as $v_2 = U_Y|0\rangle$, where $v_1 = |0\rangle$ is the simplified representation of state vector $[1, 0]^\top$. Specifically,

$$v_2 = U_Y|0\rangle = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ i \end{bmatrix}.$$

The computation of the RX gate is formulated as $v_3 = U_{RX}v_2$:

$$v_3 = U_{RX}v_2 = \begin{bmatrix} \cos(\theta) & -i\sin(\theta/2) \\ -i\sin(\theta/2) & \cos(\theta/2) \end{bmatrix} \begin{bmatrix} 0 \\ i \end{bmatrix} = \begin{bmatrix} -i\, sin(\theta/2) \\ cos(\theta/2) \end{bmatrix},$$

where $\theta \in [0, 2\pi]$ is the gate parameter of the RX gate. Notations used in the paper are listed in Table 1.

2.2 Bugs of Quantum Program

Bugs occur during the writing, compilation, and execution of quantum programs. Many works have summarized the patterns of quantum bugs [10, 66, 67]. In some cases, quantum bugs also include errors in classical software such as quantum compilers [2, 57] and quantum programming

Table 1. Notations used in the following sections.

Notation	Meaning
$\dagger$	Conjugate transpose that transposes a matrix and applies complex conjugation to its elements.
$\top$	Transpose that transform a $N \times M$ dimension matrix to $M \times N$ dimension.
tr	Trace operator that sums the diagonal elements of a matrix.
v	State vector of qubits that store information in the quantum program.
$\|x\rangle$	Basis state x, where x is a bitstring. For example, for a two-qubit system, $\|01\rangle = [0, 1, 0, 0]^\top$.
p	Measured probability distribution of qubits. The probability of each basis state is the square of its magnitude in the state vector.
g	Quantum gate, which can be I, X, Y, $RX(\theta)$ gates, etc.
U	Unitary matrix of quantum gate. The operation of this gate on the state vector is represented as Uv
t	Test input using state- or gate-encoding scheme.
P_θ	Quantum program with parameter θ.

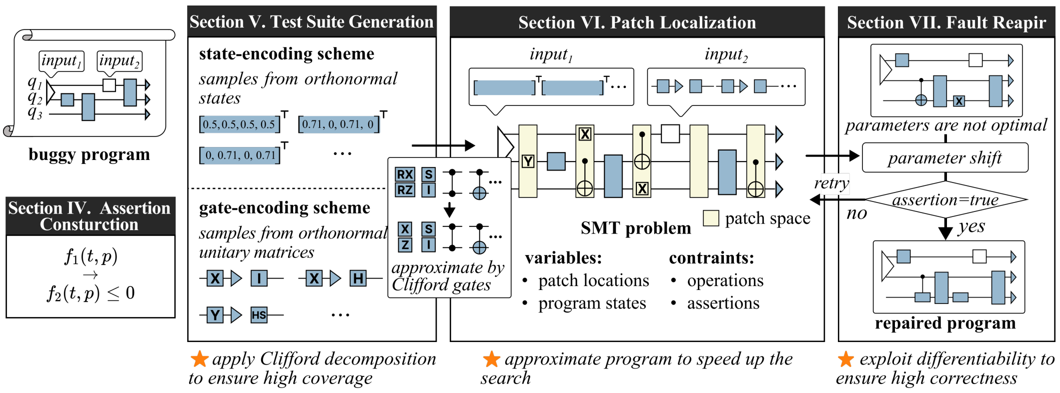

input1
input2
q1
q2
q3
buggy program
Section IV. Assertion Construction
$f_1(t,p) \to f_2(t,p) \le 0$
Section V. Test Suite Generation
state-encoding scheme
samples from orthonormal states
[0.5, 0.5, 0.5, 0.5]ᵀ
[0.71, 0, 0.71, 0]ᵀ
[0, 0.71, 0, 0.71]ᵀ
...
gate-encoding scheme
samples from orthonormal unitary matrices
X I X H Y HS
RX RZ S I X Z S I
approximate by Clifford gates
★ apply Clifford decomposition to ensure high coverage
Section VI. Patch Localization
input1
input2
Y X X
SMT problem
patch space
variables:
• patch locations
• program states
contraints:
• operations
• assertions
★ approximate program to speed up the search
Section VII. Fault Reapir
parameters are not optimal
parameter shift
retry
assertion=true
no
yes
repaired program
★ exploit differentiability to ensure high correctness

Fig. 4. Overview of HornBro.

languages [1]. This paper focuses on the bugs in the quantum program (circuit). Huang et al. [29] summarize the six types of bugs in quantum circuits:

- *Bug type 1:* The program starts with an unexpected state.
- *Bug type 2:* The gates operate on the incorrect qubits
- *Bug type 3:* The type of the gate is incorrect.
- *Bug type 4:* The number of iterations is incorrect.
- *Bug type 5:* The uncomputation is incorrect, leading to side effects in the garbage collection.
- *Bug type 6:* Unexpected entanglement are created by two-qubit gates.

Bug types 1-4 also occur in the classical programs, while debugging them on quantum programs suffers fromhigh complexity due to quantum collapse. Bug types 5-6 only exist in quantum programs. We will discuss how HornBro automatically defends these six types of bugs.

3 Overview

Figure 4 presents the workflow of HornBro, which mainly consists of four stages.
Stage 1: Assertion Construction. Given a buggy quantum program, an assertion is set up to define the expected behavior of the program via an implication formation. The antecedent of the implication specifies the range of the input space, and the consequent gives the expected relationship between the program input and program runtime state.

Stage 2: Test Suite Generation. HornBro then automatically samples test cases. For inputs encoded into the qubit state and the gate parameters, HornBro employs different sampling methods via the orthonormal decomposition of quantum states and gates, aiming to cover the maximum input space. Furthermore, we propose extending the repair to those untested inputs.
Stage 3: Patch Localization on the Classical Computer. HornBro formulates the search for candidate multi-location patches as a MAX-SMT problem [33]. The variables of the problem represent the possible locations of the patches and the qubit states during the execution. The constraints of the problem consist of the assertion and operations on qubit states after applying different gates. To accelerate space exploration, HornBro approximates the quantum program by replacing the universal quantum gate set with a Clifford gate set. It exhibits polynomial complexity as the operations are formulated as simple Boolean expressions.
Stage 4: Fault Repair on the Quantum Computer. HornBro exploits the differentiability of quantum gates to fix errors. The repair is conducted by minimizing the objective function derived from the assertion, where the gradients of the gate parameters are calculated on the simulator or the quantum computer. If the assertion is satisfied after the repair, HornBro will output the repaired program. If the assertion fails, HornBro will return to the patch localization to gradually approach the correct program.

Consideration of the computational hardware. Offloading the patch localization on the classical computer and the fault repair on the quantum computer is necessary for HornBro. Specifically, the patch localization requires profiling the intermediate state, which is efficiently performed on the classical computer by our approximation technique. However, profiling will destroy the program due to quantum collapse on the quantum computer. On the other hand, the repair requires numerous program executions to obtain program outputs under different gate parameters. 11Each exuection suffers from exponential complexity on the classical computer. Obtaining only output on the quantum computer has a constant computational overhead.

4 Assertion Construction

HornBro employs a user-defined assertion as the optimization objective for repair.

Definition 1 (Assertion). *An assertion is defined as an implication that involves the program input and output:*

$$assert \equiv \; f_1(t,p) \rightarrow [f_2(t,p) \leq 0]\,, \tag{2}$$

where t is a test case, and p is the measured probability distribution of the program output. $f_1(t,p)$ and $f_2(t,p) \leq 0$ must adhere to the SMT formulation as polynomial inequalities. Besides, $f_2(t,p)$ is defined as a polynomial objective function.

The assertion means that if input t and output p satisfy constraint $f_1(t,p)$, they should also satisfy constraint $f_2(t,p) \leq 0$. Programmers can define multiple assertions to elaborately describe the program functionality. HornBro can aggregate them for a single repair. Theoretically, p can be the arbitrary measured probability distribution derived from the runtime state. In the following sections, we regard p as the program output for simplicity. Figure 5 uses Grover's algorithm [23] as an example, which is a

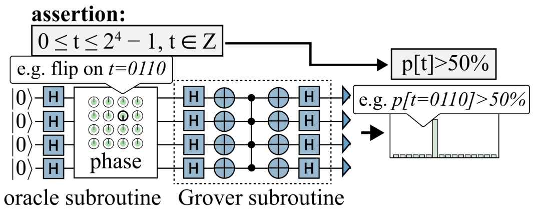


Fig. 5. Example of the Grover's algorithm.

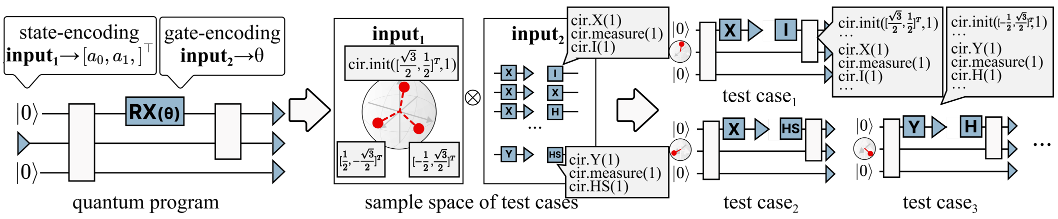


Fig. 6. Process to generate test suite.

popular quantum algorithm in pattern matching [58] and recommendation system [12]. We apply a module test to verify Grover's algorithm, consisting of an oracle and a Grover's subroutine. An input bit-string t is encoded into the gate parameters to flip the phase of its corresponding basis state $|t\rangle$. In a correct Grover's program, this input bit-string is expected to have a higher measured probability (e.g., $p[t] \geq 50\%$). For example, when 0110 is the input, the phase of the basis state $|0110\rangle$ is flipped in the oracle. Consequently, the probability of obtaining 0110 is around 76% in the output. The assertion is defined as:

$$\underbrace{\left(0 \leq t \leq 2^4 - 1,\ t \in Z\right)}_{f_1(t,p)} \rightarrow \underbrace{\left(50\% - p[t]\right)}_{f_2(t,p)} \leq 0, \tag{3}$$

$p[t]$ is the probability of bit-string t in the output. $\left(0 \leq t \leq 2^4 - 1,\ t \in Z\right)$ is the range when the number of qubits is 4.

5 Test Suite Generation

HornBro automatically generates a test suite by sampling the input space. *The key feature of these sampled inputs is that their execution results can be generalized to the rest of untested inputs, bringing a high tolerance of HornBro to overfitting.* The inputs of quantum programs can be encoded into the initial state of qubits (refer to the *state-encoding scheme*) or the gate parameters (refer to the *gate-encoding scheme*). The states and gate parameters are represented as vectors and unitary matrices, respectively. Accordingly, HornBro adopts different sampling strategies:

- For inputs using the state-encoding scheme, test cases are sampled from the orthonormal state space S_{sample} prepared by Bravyi et al. [8].
- For inputs using the gate-encoding scheme, the quantum gate is replaced with one measurement followed by a gate. The measurement operator is sampled from the basis set $O_{\text{sample}} = \{I, X, Y\}$. The gate is sampled from the gate set $G_{\text{sample}} = \{I, X, H, HS\}$.

As a result, these two encoding schemes orthogonally form the input sample space, which can be regarded as the tensor-product ⊗ between the state space and the gate space. To the best of our knowledge, prior test suite generation methods [25, 56, 56, 59, 60] can only generate test cases for the state-encoding scheme. Consequently, prior Q-APR methods [24, 37] have a limited generality to repair programs.

Figure 6 visualizes an example of our test suite generation flow. *input*$_1$ uses the state-encoding scheme, and *input*$_2$ uses the gate-encoding scheme. The sampling space of *input*$_1$ consists of 3 basic states. The sampling space of *inputs*$_2$ includes 12 combinations of measurement and gate operations. Thus, the total number of test cases is 36. For example, in the first test case, *input*$_1$ is set to $|0\rangle$, and the gate of *input*$_2$ features a measurement on the X basis and an I gate.

Proposition 1 (Generality of test cases). *For inputs using the state-encoding or gate-encoding schemes, we define test cases $T = [t_1, t_2, \cdots]$ and the resultant outputs $TP = [tp_1, tp_2, \cdots]$. Then, for* *any input represented as a linear combination of test cases:*

$$input = \sum\nolimits_{t_i \in T} \alpha_i t_i \ , \tag{4}$$

where α_i is a complex value, the output p under this input is:

$$p = \sum\nolimits_{tp_i \in TP} \alpha_i tp_i \ . \tag{5}$$

The generality of the state-encoding scheme stems from the isomorphism provided by Tan et al. [56]. The generality of the gate-encoding scheme can refer to the circuit cutting theory for distributed quantum computing [47]. Based on the generality of the sample space, the assertion validation is expected to ensure that the assertion is supposed to be satisfied for any linear combination of test cases, thereby maximizing input space coverage.

6 Patch Localization on Classical Computer

6.1 Clifford-Based Approximation

Simulating quantum behaviors on classical computers suffers from high computational overhead, making the validation of possible patches overwhelming. On the other hand, validation on quantum computers is inflexible due to quantum collapse. *To speed up validation, we adopt a homotopy-like method that approximates the quantum program using Clifford gates, which makes it easier to simulate on the classical computer.* The Clifford gate set is defined as:

Definition 2 (Clifford gate set). *Clifford gate set is a subset of universal quantum gates, which includes 24 types of single-qubit gates and 3 types of two-qubit gates.*

In the approximation, for each quantum gate g in the quantum program, we compute the distance between the unitary matrices of this gate U_g and each gate in the Clifford gate set U_c using the Hilbert-Schmidt test [46]:

$$1 - \frac{|Tr(U_g^\dagger \ U_c)|}{d^2}, \tag{6}$$

where d is the dimension of the unitary matrices. Tr is the trace operator that sums the diagonal elements in the matrix. † denotes the conjugate transpose, which transposes the matrix and applies complex conjugation to its elements. Each quantum gate g is replaced with the nearest Clifford gate in the approximated program.

We choose the Clifford gate set as it contains the most commonly used two-qubit gates and single-qubit gates. Moreover, in current quantum algorithms, more than 60% of the gates are Clifford gates [7, 14, 32, 42]. When the quantum program consists of only Clifford gates, its N-qubit state can be represented as an $N \times (2N+1)$-dimension stabilizer table instead of a 2^N-dimension state vector. A stabilizer table is:

$$\left[\begin{array}{ccc|ccc|c} x_{0,0} & \cdots & x_{0,N-1} & z_{0,0} & \cdots & z_{0,N-1} & r_0 \\ x_{1,0} & \cdots & x_{1,N-1} & z_{1,0} & \cdots & z_{1,N-1} & r_1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{N-1,0} & \cdots & x_{N-1,N-1} & z_{N-1,0} & \cdots & z_{N-1} & r_{N-1} \end{array}\right], \tag{7}$$

where elements $x_{i,j}$, $z_{i,j}$, and r_i are all boolean. $x_{i,j}$ and $z_{i,j}$ represents whether the i-th stabilizer generator has the Pauli X and Z components on the j-th qubit or not. r_i determines whether the sign of the i-th stabilizer generator is +1 or −1. There are $2N^2 + N$ elements in the stabilizer table of a N-qubit program.

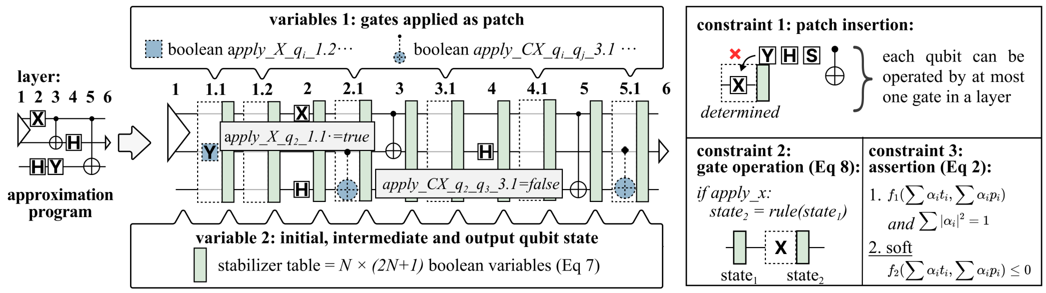

variables 1: gates applied as patch
boolean apply_X_q_i_1.2···
boolean apply_CX_q_i_q_j_3.1 ···
layer:
1 2 3 4 5 6
approximation program
1 1.1 1.2 2 2.1 3 3.1 4 4.1 5 5.1 6
apply_X_q_2_1.1·=true
apply_CX_q_2_q_3_3.1=false
variable 2: initial, intermediate and output qubit state
stabilizer table = N × (2N+1) boolean variables (Eq 7)
constraint 1: patch insertion:
determined
each qubit can be operated by at most one gate in a layer
constraint 2: gate operation (Eq 8):
if apply_x: state_2 = rule(state_1)
state_1 state_2
constraint 3: assertion (Eq 2):
1. f_1(∑α_i t_i, ∑α_i p_i) and ∑|α_i|² = 1
2. soft f_2(∑α_i t_i, ∑α_i p_i) ≤ 0

Fig. 7. The SMT-based formulation of patch localization.

Clifford quantum gates can be computed as boolean operations instead of matrix-vector multiplications. For example:

- Applying X gate on qubit i: $r_k \oplus = z_{k,i}$ for each row k.
- Applying H gate on qubit i: $x_{k,i} \leftrightarrow z_{k,i}$ and $r_k \oplus = x_{k,i} z_{k,i}$ for each row k. (8)
- Apply $CNOT$ gate on qubits i and j: $r_k \oplus = x_i z_j (x_j \oplus z_i \oplus 1)$, $x_{k,j} \oplus = x_{k,i}$ and $z_{k,i} \oplus = x_{k,j}$ for each row k.

$\oplus$ is the XOR operation. Operations of all Clifford gates can be computed with no more than polynomial complexity [18].

The probability of measurement results can also be calculated from the stabilizer table. For a bitstring $x = b_1 b_2 \cdots b_N$, the probability of measuring it on the stabilizer table in Eq (7) is

$$P(b_1 b_2 \cdots b_N) = \begin{cases} 0, & \text{if } \vee_{i=1}^{N} (\wedge_{j=1}^{N} \neg x_{i,j} \wedge (\oplus_{j=1}^{N} b_j z_{i,j} \oplus r_i)) \\ \frac{1}{2^d}, & \text{else} \end{cases}$$

where $x_{i,j}$, $z_{i,j}$ and r_i are elements of the stabilizer table in Eq (7). d is the rank of $x_{i,j}$ components in the table, which is

$$d = \sum_{i=1}^{N} \neg(\vee_{c \in \{0,1\}^i} (\wedge_{j=1}^{N} (\oplus_{k=1}^{i-1} c_k x_{k,j} \oplus x_{i,j})))$$

6.2 SMT-Based Search

HornBro adopts a gate-level patch to fix the quantum program.

Definition 3 (Patch). *A patch is a quantum gate that is inserted into the quantum program. The patches in the localization stage use the Clifford gate set, which is transformed into parameterized gates in the fault repair stage.*

HornBro formulates patch localization as a MAX-SMT problem. As shown in Figure 7, the problem consists of two types of variables and three types of constraints. We use *layers* to represent the execution order of quantum gates (e.g., layers 1, 2, $\cdots$ in Figure 7), where gates in the same layer are executed in parallel. The two types of variables include:

- *Variable for patch insertion.* HornBro appends auxiliary layers (e.g., layers 1.1, 1.2) after each layer to place the patches. The number of auxiliary layers is user-specified, depending on the expected search space. We use boolean variables to represent whether a quantum gate operates on certain qubits in an auxiliary layer. For example, variable $apply\_CX\_q_1\_q_2\_1.1 = true$ means that a CX gate operates on qubits 1 and 2 in layer 1.1.

- *Variables for qubit state.* For the approximated program, the qubit state after each layer can be represented as a stabilizer table in Equation 7 with $N \times (2N+1)$ variables.

Three types of constraints are defined to specify the relations between variables:

- *Constraints among patches.* In each layer, a qubit can be operated by, at most, one gate due to the restriction of the quantum circuit model.
- *Constraints for gate operations.* If a quantum gate is applied, i.e., $gate\_type\_q_i\_q_j\_layer = true$, the qubit state in the next layer is determined by its former layer according to the boolean operations in Equation 8.
- *Constraints for the assertion.* The localization aims to find patches that satisfy the assertion. The assertion is added as two constraints:

$$1.\ f_1(\sum \alpha_i t_i, \sum \alpha_i p_i) \text{ for any } \sum |\alpha_i|^2 = 1,$$
$$2.\ f_2(\sum \alpha_i t_i, \sum \alpha_i p_i) \leq 0.$$

  This specification ensures the patches make the assertion true under the test cases and their linear combinations. Constraint 2 is soft because the approximated program can only identify the most likely patches.

The SMT problem is put into the Z3 solver [16]. By finding the true assignment related to patch insertion, we can obtain the potential locations and the types of gates that can correct the quantum program.

7 Fault Repair on Quantum Computer

Patch localization employs an approximated quantum program to identify the patches. The homotopy-like method is then performed on the exact program with the patches. It further fine-tunes the gate parameters to approach the correct program. *This fine-tuning can be performed on quantum computers by leveraging the differentiability of quantum gates, eliminating computational-intensive simulation.*

Property 1 (Differentiability of quantum gate). *Let P_θ be a parameterized quantum program. Gate parameters $\theta = [\theta[1], \theta[2], \cdots]$ are real-valued. P_θ is differentiable with respect to θ, meaning that for each parameter $\theta[i]$, the partial derivative $\frac{\partial P_\theta}{\partial \theta[i]}$ exists and is continuous. Mathematically, this means:*

$$\frac{\partial P_\theta}{\partial \theta[i]} = \lim_{\epsilon \to 0} \frac{P_{\theta+\epsilon \hat{e}_i} - P_\theta}{\epsilon},$$

where $\hat{e}_i$ is the unit vector in the i-th direction.

Any quantum program that incorporates parameterized quantum gates, such as RX, RY, and RZ gates, can be considered a parameterized program P_θ Besides, the Clifford gates in the patches can be converted into parameterized gates according to the equivalence. For example, the X Clifford gate can be converted into a parameterized RX gate with the default parameter π. The optimization aims to minimize the objective function f_2 of the assertion subject to the constraint f_1:

$$\arg\min_\theta f_2(t, P_\theta(t))$$
$$\text{subject to } f_1(t, P_\theta(t)).$$

We adopt the parameter shift method to compute the gradients of the gate parameters [62]. This approach allows the quantum program to run on quantum computers, eliminating the need for computationally intensive simulations. Algorithm 1 presents the overall repair flow. The algorithm updates the gate parameters over L epochs (lines 1-15). In each epoch, the algorithm generates a set

Algorithm 1: Fault Repair

```
input  : quantum program $P_\theta$, learning rate $\ell$, number of epochs $L$
output : repaired quantum program
for $epoch = 1$ to $L$ do
    $X \leftarrow$ linear combinations of test cases                //generate training data
    $X \leftarrow \{x \mid x \in X,\ f_1(x, P_\theta(x)) = True\}$   //filter training data
    $\frac{\partial P_\theta}{\partial \theta} \leftarrow [0, 0, \cdots, 0]$   //initialize gradients
    for $i = 1$ to $length(\theta)$ do
        $\theta_+,\ \theta_- \leftarrow \theta,\ \theta$
        $\theta_+[i] \leftarrow \theta[i] + \frac{\pi}{2}$             //add perturbation $\frac{\pi}{2}$
        $\theta_-[i] \leftarrow \theta[i] - \frac{\pi}{2}$             //add perturbation $-\frac{\pi}{2}$
        $\frac{\partial P_\theta}{\partial \theta[i]} \leftarrow f_1(X, P_{\theta_+}(X)) - f_1(X, P_{\theta_-}(X))$   //compute gradients
    end
    $\theta \leftarrow \theta - \ell \frac{\partial P_\theta}{\partial \theta}$   //update parameters
    if $f_2(X, P_\theta(X)) \leq 0$ then
        return $P$
    end
end
```

of test inputs X by randomly computing the linear combinations of the test cases (line 2). Programs with these inputs are executed to obtain outputs $P_\theta(x)$. The test inputs are filtered to retain only those that satisfy the constraint f_1 (line 3). The gradient is first initialized to be 0 (line 4). Then, for each parameter, we add perturbations $+\pi/2$ and $-\pi/2$ and compute the gradient between the perturbations (lines 6-9). After the gradient is updated, if the assertion holds under the test cases, the algorithm outputs the repaired quantum program (lines 12-14).

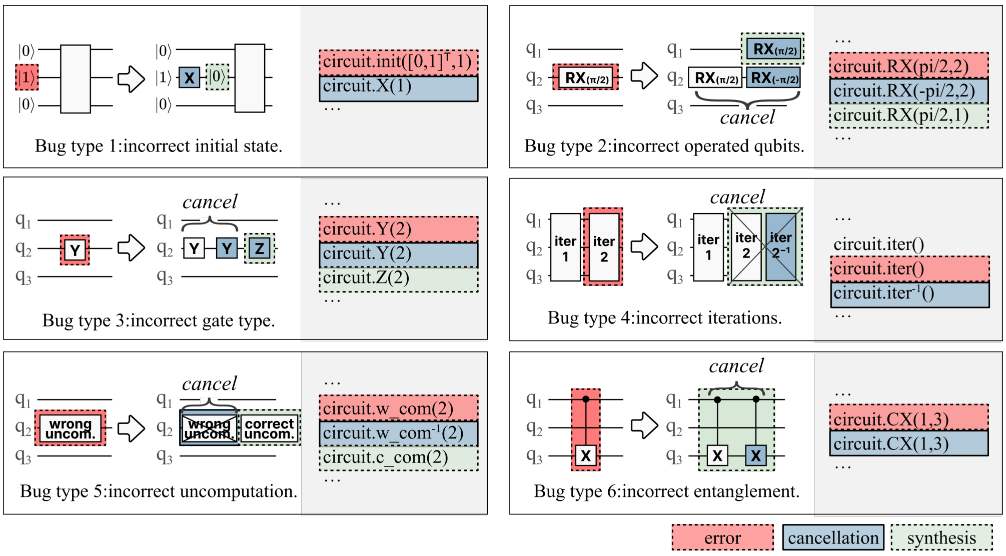
Example of Repairs. HornBro can repair programs containing erroneous gates or modules, achieved via a *cancel-and-synthesize* mechanism. For each incorrect quantum gate, this mechanism involves two steps. The first step cancels the wrong gate by inserting an inverse gate afterward. The second step inserts gates to perform the correct computation. Using the examples in Figure 8, we demonstrate how HornBro defends against six types of quantum bugs, as summarized by Huang et al. [29].

Bug type 1: Repair of incorrect initial state. A program program may start with an incorrect initial state. For example, the initial state $|010\rangle$ is incorrect, which is expected to be $|000\rangle$. HornBro can correct the initial state by adding quantum gates to transform $|010\rangle$ into $|000\rangle$.

Bug type 2: Repair of incorrectly operated qubits. HornBro also can correct gates on incorrect qubits by the cancel-and-synthesize mechanism. For example, if an RX gate with a parameter of $\pi/2$ is mistakenly applied, It can be canceled by applying an inverse RX gate with a parameter of $-\pi/2$. The correct gate is then applied to the correct qubit.

Bug type 3: Repair of incorrect gate types. The repair of the incorrect gate type is the same as the incorrectly operated qubits.

Bug type 4: Repair of incorrect iterations. Quantum programs usually include iterative quantum modules. If a program executes more iterations than expected, HornBro can cancel the redundant iterations. If the program performs fewer iterations than expected, HornBro can use auxiliary layers to synthesize the iterative modules.


Bug type 1:incorrect initial state.
Bug type 2:incorrect operated qubits.
Bug type 3:incorrect gate type.
Bug type 4:incorrect iterations.
Bug type 5:incorrect uncomputation.
Bug type 6:incorrect entanglement.
cancel
wrong uncom.
correct uncom.
iter 1
iter 2
iter 2⁻¹
error
cancellation
synthesis

```
circuit.init([0,1]ᵀ,1)
circuit.X(1)
...
```

```
...
circuit.RX(pi/2,2)
circuit.RX(-pi/2,2)
circuit.RX(pi/2,1)
...
```

```
...
circuit.Y(2)
circuit.Y(2)
circuit.Z(2)
...
```

```
...
circuit.iter()
circuit.iter()
circuit.iter⁻¹()
...
```

```
...
circuit.w_com(2)
circuit.w_com⁻¹(2)
circuit.c_com(2)
...
```

```
...
circuit.CX(1,3)
circuit.CX(1,3)
...
```

Fig. 8. Defense of HornBro to different types of bugs.

Bug type 5: Repair of incorrect uncomputation. Uncomputation erases information from reusable qubits. A correct uncomputation eliminates any entanglement between these qubits and others. For incorrect uncomputation, HornBro cancels it and synthesizes a correct module.
Bug type 6: Repair of incorrect entanglement. The incorrect entanglement occurs when the wrong two-qubit gates are applied. HornBro can repair it by canceling these gates.

8 Complexity Analysis

The overall complexity of HornBro is polynomial relative to the number of qubits and gates. Without the optimization, prior methods suffer neither from high complexity [37] nor low success rate [24] Specifically, the computational complexity of HornBro primarily arises from the patch localization and repair processes, both of which have polynomial complexity. We define the buggy quantum program as containing N qubits and M quantum gates. The number of auxiliary layers is set to K, and the number of epochs in the parameter shift is set to L.

In patch localization, K auxiliary layers contain $\mathcal{O}(KN^2)$ variables for gate insertions. A buggy quantum program usually has numerous solutions. For this type of problem, the average search complexity is $\mathcal{O}(N_{var}\log(N_{var}))$ [50], where N_{var} is the number of variables. Validation using Clifford simulation incurs a complexity of $\mathcal{O}(MN^2)$ [18]. The overall complexity is therefore $\mathcal{O}(KMN^4log(KN^2))$. In fault repair, in each epoch, without time-consuming simulation, the complexity of parameter shifts on a quantum computer is $\mathcal{O}(2M)$. Parameter updates incur complexity of $\mathcal{O}(M)$. There are L epochs, making complexity $\mathcal{O}(3LM)$.

9 Experiment

9.1 Experiment Setup

Questions. Our experiments were designed to answer the following research questions:

- **RQ 1:** Is HornBro effective in repairing the bugs in the quantum programs?
- **RQ 2:** Is HornBro efficient in repairing the bugs in the quantum programs?
- **RQ 3:** How techniques of HornBro contribute to the improvement?
- **RQ 4:** Is HornBro sensitive to the parameter setting?

Table 2. Benchmarking quantum programs used in the evaluation.

Input	#Qubit	Benchmark	#Gate	Benchmark	#Gate
State-encoding scheme	5–20	Quantum Fourier Transformation (QFT) [14]	23–241	Greenberger-Horne-Zeilinger State (GHZ) [22]	11–41
	5–20	Quantum Walk (QW) [42]	459–19948	Amplitude Estimation (AE) [7]	37–307
	5–20	Deutsch-Jozsa (DJ) [19]	18–78	Quantum Phase Estimation (QPE) [32]	26–258
	5–20	Grover's algorithm (GROVER) [23]	234–10530		
Gtate-encoding scheme	5–20	Variational Quantum Eigensolver (VQE) [49]	29–470	Quantum Approximation Optimization Algorithm (QAOA) [68]	31–121
	5–20	Vehicle Routing [4]	25–323	Quantum Neural Network (QNN) [9]	90–1260

Implementation details. HornBro is implemented in Python, enabling QAPR to work with multiple quantum programming packages, including Qiskit [1], Pennylane [5], and Q# [55]. HornBro leverages the Z3 solver to solve the Max-SMT problem and uses Jax for gradient computation.
Benchmarks. We prepared three benchmarks for evaluation. Two of them are open-source, namely Bug4Q [67] and QBugs [10]. After excluding bugs in quantum compilers and languages, Bug4Q and QBugs contain 39 and 200 instances, respectively. The third benchmark was constructed by mutating correct implementations of 11 typical quantum algorithms in Table 2. The mutation was performed by randomly deleting, adding, or replacing universal quantum gates in the program. We generated 5- to 20-qubit versions for each quantum algorithm, which were compiled to meet the hardware constraints of current IBM and Google quantum computers. The times of mutations of each program range from 1 to 10. Overall, the third benchmark contains 2200 buggy programs.
Baselines. We compared HornBro with three baselines, including the ChatGPT-based technique [24] (referred to *LLM-QAPR*) and the synthesis-based technique [37], and a basic method (referred to *Basic-QAPR*). The basic method follows the test suite generation, localization and repair routine, while it randomly generates test cases and locates faults through mutation-based analysis. We used GPT-4o for LLM-QAPR. We followed the prompts provided in the original paper and allowed each buggy program to be processed up to three times by ChatGPT. For the synthesis-based method, we employed two state-of-the-art synthesis techniques, QFAST [65] and QSD [51], to facilitate quantum program synthesis, which we refer to as *Syn-QFAST* and *Syn-QSD*, respectively.
Metrics. We evaluated the success rate, repair time, and the number of gates of the patches for both HornBro and the baselines. All software experiments were performed on an Ubuntu 22.04 LTS server with two 64-core AMD EPYC CPUs and 1.6 TB of DDR5 RAM.

9.2 RQ1: Effectiveness of HornBro

Table 3 presents the success rate of quantum program repair of HornBro, LLM-QAPR [24], Syn-QSD [51], and Syn-FAST [65]. Overall, HornBro achieves a 100% success rate while maintaining efficiency in both time and computational resources, as it can solve all the six types of quantum bugs and repair programs with both state- and gate-encoding schemes. It improves the repairing success rate by 93.3%, 62.5%, and 62.5% compared to LLM-QAPR, Syn-QSD, and Syn-FAST, respectively. LLM-QAPR fails to repair quantum programs when the number of qubits increases due to the limited training data for quantum programs in ChatGPT. Additionally, Syn-QSD and Syn-Fast show the same success rate as HornBro in a 5-qubit programs. However, as they suffer from high computational and memory complexity in synthesis, they scale only up to fewer than ten qubits. For example, Syn-FAST requires more than three days for a15-qubit program. The synthesis-based approaches cannot repair the program using gate-encoding schemes, as the synthesizer only supports unparameterized unitary matrices.

Table 3. Comparison of success rate (in %) using different Q-APR techniques.

	Bug4Q [67]				QBugs [10]				Mute-Alg			
#Qubit	2	3	4	5	5	10	15	20	5	10	15	20
#Instance	16	12	5	6	50	50	50	50	500	500	500	500
HornBro	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
LLM-QAPR [24]	56.3	41.6	40.0	33.3	27.4	3.4	0.2	0.0	21.5	1.6	0.0	0.0
Syn-QSD [51]	100.0	100.0	100.0	100.0	100.0	100.0	t-out	m-out	63.4	60	t-out	m-out
Syn-FAST [65]	100.0	100.0	100.0	100.0	100.0	100.0	t-out	m-out	63.4	60	t-out	m-out

"t-out" means the method is requires more than three days. "m-out" means the methods more than 1.6Tb memory.

9.3 RQ2: Efficiency of HornBro

Table 4. Efficiency of HornBro and Syn-QSD over programs with different number of qubits.

Benchmark		Bug4Q [67]				QBugs [10]				Mute-Alg			
#Qubit		2	3	4	5	5	10	15	20	5	10	15	20
Time (s)	**HornBro**	**0.017**	**0.099**	**0.377**	**1.100**	**1.094**	**37.25**	**141.9**	**1691**	**5.448**	**36.75**	**135.8**	**1693**
	Syn-QSD [51]	0.018	0.334	0.362	0.533	0.495	1286	-	-	287	1313	-	-
	Speedup(×)	1.05	3.37	0.96	0.48	0.45	34.5	-	-	52.3	35.7	-	-
#Gate	**HornBro**	**6**	**8**	**112**	**18**	**16**	**63**	**198**	**249**	**17**	**67**	**215**	**274**
	Syn-QSD [51]	3	26	175	862	841	1062343	-	-	819	1188709	-	-
	Reduction(%)	-100	69.2	36.0	97.9	98.1	99.9	-	-	98.0	99.9	-	-

We evaluated the efficiency of HornBro on buggy quantum programs and compared it with Syn-QSD, as shown in Table 4. We recorded the repair time and the number of gates introduced in the patches. A longer repair time indicates greater computational complexity, while a higher number of inserted gates indicates lower efficiency of the repaired program.

In terms of repair time, HornBro can repair 3-qubit programs in 0.099 seconds and 20-qubit programs in 28.2 minutes, achieving a speedup ranging from 3.37× to 52.3× over Syn-QSD. Additionally, HornBro outperforms Syn-QSD in large-scale programs with more than ten qubits. Specifically, Syn-QSD requires iterative singular value decomposition to decompose large unitary matrices into smaller ones, which involves expotential complexity, whereas the Clifford approximation in HornBro reduces the complexity to polynomial time.

In terms of the number of gates, HornBro generates patches with approximately 18 gates for 5-qubit programs and reduces the number of gates by 99.9% for 10-qubit programs relative to Syn-QSD. Additionally, the number of gates generated by HornBro grows at a slower rate than in Syn-QSD, resulting in patches with only hundreds of gates for 20-qubit programs. Notably, the Clifford approximation also helps reduce the number of quantum gates, as a polynomial number of gates can efficiently explore the search space of the Clifford state.

9.4 RQ3: Effects of Techniques in Each Stages

Evlauation of the test suite generation. We compared the test suite generation of HornBro with three baselines, including Quito [59], QuSBT [61], and a randomized method. Table 5 presents the test suite generation time and the success rate achieved with each test suite. Quito constructs

Table 5. Evaluation of different test suite generation methods. For each number of qubits, 50 test cases were generated using each method.

#Qubit	#Error	Time of test suite generation(s)				Success rate (%)			
		HornBro	Quito [59]	QuSBT [61]	Random	**HornBro**	Quito [59]	QuSBT [61]	Random
5	2	2.00	1.31	1.92	1.22	**100.0**	70.5	100.0	67.7
	4	1.74	1.50	1.88	2.16	**100.0**	71.9	80.7	75.2
	6	1.84	1.79	2.05	2.00	**100.0**	59.6	78.6	61.7
10	2	9.65	9.81	9.79	6.77	**74.5**	30.6	52.4	48.4
	4	14.60	8.41	9.66	3.02	**62.9**	27.1	51.6	43.5
	6	16.24	14.07	14.01	8.37	**56.9**	11.2	30.8	10.1
15	2	29.61	19.45	28.52	15.00	**43.3**	0.0	1.5	0.6
	4	43.28	47.33	44.81	17.20	**25.8**	0.0	0.7	0.5
	6	164.96	77.65	111.31	12.10	**19.9**	0.0	0.9	0.3

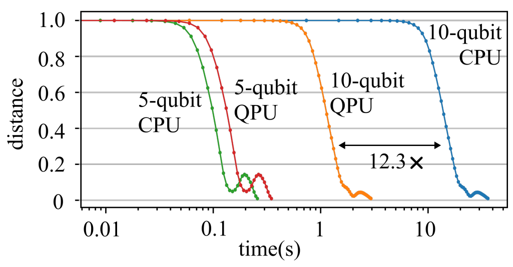
test suites using basis states, while QuSBT generates test cases using a genetic algorithm. The randomized method samples test suites randomly from the complex number field. Overall, HornBro achieves a generation time comparable to Quito and QuSBT. The randomized method requires slightly less time, as it generates test cases in parallel. Moreover, with a similar generation time, HornBro improves the repair success rate by 34.7%, 20.7%, and 30.6% on average compared with those achieved by Quito, QuSBT, and the randomized method, respectively. This observation aligns with Proposition 1, which suggests that the superposition of state-encoding holds more generally for any linear combination of test cases.

Table 6. Evaluation of the SMT-based localization method.

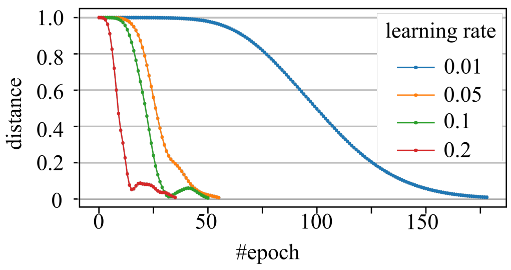
#Qubit		5		10		15		20	
		HornBro	Local Search	HornBro	Local Search	HornBro	Local Search	HornBro	Local Search
Success rate (%)		100.0	93.4	100.0	67.2	100.0	56.3	100.0	21.1
Time (s)	Localization	0.99	0.10	35.07	2.35	106.8	30.50	1572	562.0
	Repair	0.10	0.20	2.18	3.89	35.13	57.20	118.8	201.2

Ablation study of the SMT-based localization. In Table 6, we compared the success rate and localization time of the SMT-based localization method (applied by HornBro) with that of a local search method. HornBro achieves a 100% success rate for all tested 5- to 20-qubit quantum programs, while the success rate of the local-search method decays dramatically. This is because errors in the programs become more distant and less explicit as the quantum program scales up, which can hardly be revealed by local information. Therefore, the 2.88 $\times$ longer localization time in HornBro is justified, as the SMT-based localization method exhibits polynomial-time complexity and remains scalable.

Ablation study of using quantum computers. Figure 9a presents the convergence of fault repair on a quantum processing unit (QPU) and a classical processing unit (CPU). The distance between the unitary matrix of the repaired quantum program and the unitary matrix of the correct program is calculated by Equation 6. The execution time on the quantum computer is estimated based on gate latency data from the IBM Kyiv platform [20]. The quantum computer achieves a 12.3$\times$ speedup over the classical computer in repair time. This is attributed to the fact that the



(a) Ablation study of using quantum computers.



(b) Distance curve under different learning rates.

Fig. 9. Evaluation of repairing process for ablation study and hyper-parameter.

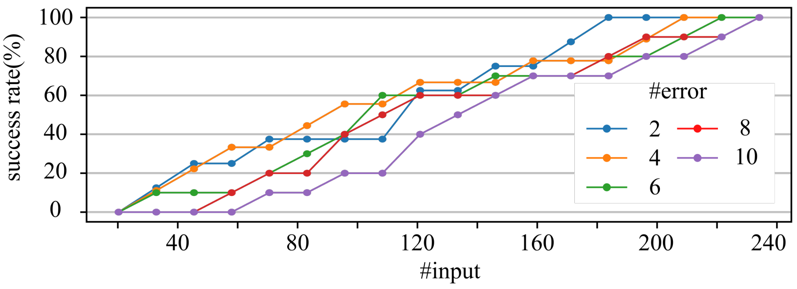
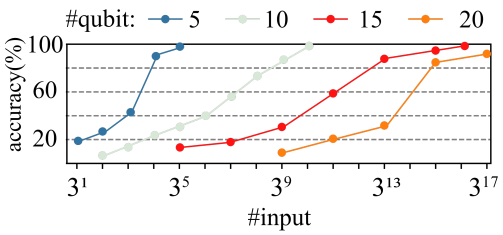
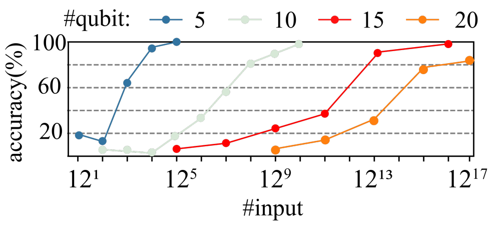


Fig. 10. Success rate under different numbers of inputs.



(a) State-encoding scheme.



(b) Gate-encoding scheme.

Fig. 11. Approximation accuracy varied with the number of inputs in different encoding schemes.

classical computer suffers from exponential complexity to simulate the quantum program, while the quantum computer only requires linear complexity. The improvement in the use of quantum computers has become significant in large-scale programs. The experiment suggests that fault repair on quantum computers is highly beneficial for large-scale quantum programs.

9.5 RQ4: Effects of Hyper-parameters

Evaluation of learning rate. We tested the quantum program repair process with different learning rates in Figure 9b. The unitary matrix of the repaired program converges in significantly fewer epochs as the learning rate increases, thereby accelerating the quantum program repair process. However, when the learning rate reaches 0.1, the convergence behavior becomes unstable, potentially leading to repair failures. Hence, the optimal learning rate for HornBro may be 0.05.

Evaluation of the size of the test suite. We also investigated the number of inputs that can achieve the optimal repair success rate in Figure 10 for 5-qubit quantum programs with the state-encoding scheme. For quantum programs with 2 to 10 errors, the number of inputs required for effective debugging falls within the range of 180 to 240. Meanwhile, we analyzed the approximation accuracy of programs under varying numbers of inputs for both state-encoding and gate-encoding schemes, as shown in Figure 11. The repair success rate is upper-bounded by the approximation accuracy. Based on the experimental data, the optimal number of test cases grows exponentially with the input size of the quantum program.

10 Related Work

10.1 Quantum Program Repair

Debugging a quantum program typically requires techniques to generate test suites, verify results, and repair the program by patches. When it comes to testing suite generation, QSharpCheck [25] generates test cases using property-based testing. QuSBT [60] and Quito [59] utilize oracles to generate initial states and optimize measurement overhead by adaptively adjusting the number of shots. These techniques necessitate traversing the entire quantum input space to ensure global correctness, usually resulting in millions of input tests for programs with less than 13 qubits. MorphQPV [56] exploits isomorphism, a mathematical characteristic of quantum programs, to generate test cases with high efficiency. However, it only supports quantum programs using the state-encoding input scheme. HornBro supports both state- and gate-encoding schemes.

When it comes to verification, current techniques for quantum programs include deductive verification [13, 64, 69] and runtime assertions [29, 35, 38]. Deductive verification is primarily conducted by extending Hoare logic [64, 69] and semantic models [48], which require human experts to identify inductive invariants, thereby restricting the level of automation. Runtime assertion is a lightweight technique that can be implemented on both quantum and classical computers. The technique by Huang et al.[29], along with current repair techniques [24, 37], validate assertions on classical computers. Typical techniques on quantum computers use projection [35] and non-destructive measurement [38]; however, these techniques can only perform equality comparisons due to the limited expressiveness of quantum programs. These techniques can only compare the measured probability distributions rather than the exact state vectors. Compared to the prior techniques, HornBro supports flexible comparisons and higher coverage of bug types by approximation function and can obtain the state vector of a qubit on a classical computer.

In terms of the repair, Luo et al. [39] discussed the challenges of the Automatic Quantum Program Repair (Q-APR) and the characteristics of quantum bugs, such as distinct patterns and higher computational overhead. Unfortunately, prior works are not able to overcome these challenges. Guo et al. [24] examined the use of ChatGPT for Q-APR, indicating a low success rate of repair due to the insufficiency of its training dataset. Li et al. [37] adopted quantum program synthesis, which suffers from low scalability due to the high computational overhead of calculating the unitary matrix. HornBro enables efficient Q-APR with a high success rate by leveraging the isomorphism and Clifford similarity inherent in quantum programs.

10.2 Automated Program Repair

Automated Program Repair (APR) is a prominent field in classical software engineering, as it can significantly reduce human labor [30, 31]. ProveNFix [53] uses future conditions to express the expected behavior of a program. It leverages temporal properties to address bugs related to memory usage and other issues. APR can also be performed by large language models [21, 41]. These techniques typically use iterative pipelines with well-designed prompts to fix bugs. For example, Toggle [27] uses separate, fine-tuned models for bug localization and fixes with distinct prefixes and suffixes. HornBro is the first work to deploy a typical APR workflow for quantum program repair. HornBro addresses the inefficiencies and overfitting issues in Q-APR by redesigning patch generation, patch localization, and patch validation techniques.

APR techniques have also been developed for specific domains, such as smart contracts [45, 52], memory leakage [26, 34] and industrial software [44]. For instance, SmartFix [52] combines statistical models of prior and posterior knowledge to repair smart contracts. APR can also be applied to neural networks [15, 36, 40, 63]. Neural networks are similar to quantum programs, as both are differentiable and involve matrix multiplications. However, quantum programs have an exponentially large input space. Additionally, quantum computers cannot inspect intermediate states due to quantum collapse.

11 Conclusion

We propose HornBro for the automated quantum program repair. HornBro adopts a homotopy-like method, which iteratively switches between the classical and quantum parts to progressively converge toward the correct program. HornBro enables an implication assertion pragma and develops a Clifford approximation method to transform the patch localization into a symbolic reasoning problem. HornBro leverages the differentiability of quantum gates to repair the program. Experiments suggest that HornBro increases the repair success rate by more than 62.5% compared to the existing techniques [24, 37].

12 Data and Code Availability

HornBro is publicly available on [17], where all experimental results can be reproduced.

Acknowledgements

This work was supported by National Natural Science Foundation of China (No.62472374) and the National Key Research and Development Program of China (No. 2023YFF0905200). This work was also funded by Zhejiang Pioneer (Jianbing) Project (No. 2023C01036).

Appendix

Below are the assertions of quantum algorithms in Table 2 :
Quantum Fourier Transformation (QFT). The input of QFT is an initial state. QFT computes the discrete Fourier transform of this state. The assertion for QFT is defined as follows:

$$assert \equiv (t, 0 \le t < 2^n, t \in Z) \rightarrow \sum_{k=0}^{k=2^n-1} (2^n p[k] - (e^{2\pi i/2^n})^{2tk})^2 \le 0 \tag{9}$$

Where $p[k]$ is the measured probability of any output bitstring k.
Greenberger-Horne-Zeilinger (GHZ) State. GHZ has no input. It entangles qubits in the ground state. The assertion is defined as:

$$assert \equiv (t = 0) \rightarrow (p[0] - \frac{1}{2})^2 (p[2^n - 1] - \frac{1}{2})^2 \le 0 \tag{10}$$

since GHZ state is the entangled quantum state between state $|00 \cdots 0\rangle$ and state $|11 \cdots 1\rangle$.
Quantum Walk (QW). Discreate-time QW is a quantum analog of classical random walks. Its input is an initial state with amplitudes α and β. An additional coin qubit is used to decide the probability of moving left and right. For quantum walk unitary, the assertion aims to ensure the amplitudes propagate across qubits.

$$assert \equiv (t, 0 \le t < 2^{n-1}, |c\rangle = \alpha|0\rangle + \beta|1\rangle) \rightarrow (p[t-1] - \alpha^2)^2 (p[t+1] - \beta^2)^2 \le 0 \tag{11}$$

$|c\rangle$ is the state of the coin qubit, and t is the current position.

Amplitude Estimation (AE). The input of AE is a gate parameter that prepares a qubit to the amplitude α on $|1\rangle$. AE estimates the amplitude of a specific quantum state. The output bitstring with more than 50% probability should be close to the $\alpha 2^n$. Thus, the assertion is

$$assert \equiv (\alpha \in [0,1]) \longrightarrow (50\% - p[\lfloor \alpha 2^n \rfloor]) \leq 0 \tag{12}$$

Phase Estimation (PE). AE estimates the phase of a given quantum state. Its input θ is the estimated phase. The output bitstring with more than $4/\pi^2$ probability should be close to the $\theta 2^n$. Thus, the assertion is

$$assert \equiv (\theta \in [0,2\pi]) \longrightarrow (4/\pi^2 - p[\lfloor \theta 2^n \rfloor]) \leq 0 \tag{13}$$

Deutsch-Jozsa (DJ) Algorithm. The input of DJ is a function that maps a bitstring to a binary value, $f(x) : \{0,1\}^n -> \{0,1\}$, encoded into the gate parameters. DJ distinguishes constant functions from balanced functions. A constant function means that the outputs of the function are all 0s or all 1s. A balanced function means the output on half set of inputs is 1 and another half is 0:

$$assert \equiv f \longrightarrow (p[0] - \frac{\sum_t (-1)^{f(t)}}{2^n})^2 \leq 0 \tag{14}$$

Here, f is the input function, and $p[0]$ is the probability of output bitstring $00\cdots0$

Variational Quantum Eigensolver (VQE). VQE is used to approximate ground-state energies. The assertion can be:

$$(\text{Input Hamiltonian} \quad H) \ \longrightarrow \ \left(E_{\text{VQE}}(p) - E_{\min}\right) \leq 0, \tag{15}$$

where H is any input Hamiltonian, E_{VQE} is the measured energy calculated from output probability p, and Emin is the theoretical ground-state energy of Hamiltonian H.

Quantum Approximation Optimization Algorithm (QAOA). QAOA has no input. It solves combinatorial optimization problems. The assertion is:

$$(\text{Input problem instance } P) \ \longrightarrow \ \left(C_{\text{QAOA}}(p) - C_{\text{optimal}}\right) \leq 0, \tag{16}$$

where C_{QAOA} is the measured cost function value, and C_{optimal} is the optimal cost.

Vehicle Routing (VR). The assertion for vehicle routing algorithm is the same as QAOA,

$$(\text{Input problem instance } P) \ \longrightarrow \ \left(C_{\text{VR}}(p) - C_{\text{optimal}}\right) \leq 0, \tag{17}$$

where C_{VR} is the measured cost function value, and C_{optimal} is the optimal cost.

Quantum Neural Network (QNN). QNN follows the gate-encoding scheme. For QNN on a classification problem, the assertion is:

$$assert \equiv (\text{data belongs to label } t, 0 \leq t < 2^n, t \in Z) \longrightarrow (50\% - p[t]) \leq 0 \tag{18}$$

References

[1] Gadi Aleksandrowicz, Thomas Alexander, Panagiotis Barkoutsos, Luciano Bello, Yael Ben-Haim, David Bucher, F Jose Cabrera-Hernández, Jorge Carballo-Franquis, Adrian Chen, and Chun-Fu Chen. 2019. Qiskit: An open-source framework for quantum computing. *Accessed on: Mar* 16. https://doi.org/10.5281/zenodo.2562111

[2] Matthew Amy, Martin Roetteler, and Krysta M Svore. 2017. Verified compilation of space-efficient reversible circuits. In *International Conference on Computer Aided Verification (CAV)*. Springer, 3–21. https://doi.org/10.1007/978-3-319-63390-9_1

[3] Zahra Ashktorab, Justin D Weisz, and Maryam Ashoori. 2019. Thinking too classically: research topics in human-quantum computer interaction. In *CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3290605.3300486

[4] Utkarsh Azad, Bikash K. Behera, Emad A. Ahmed, Prasanta K. Panigrahi, and Ahmed Farouk. 2023. Solving Vehicle Routing Problem Using Quantum Approximate Optimization Algorithm. *IEEE Transactions on Intelligent Transportation Systems* 24, 7. https://doi.org/10.1109/TITS.2022.3172241

[5] Ville Bergholm, Josh Izaac, Maria Schuld, Christian Gogolin, Shahnawaz Ahmed, Vishnu Ajith, M Sohaib Alam, Guillermo Alonso-Linaje, B AkashNarayanan, Ali Asadi, et al. 2018. Pennylane: Automatic differentiation of hybrid quantum-classical computations. *arXiv preprint arXiv:1811.04968*. https://doi.org/10.48550/arXiv.1811.04968

[6] Carsten Blank, Daniel K Park, and Francesco Petruccione. 2021. Quantum-enhanced analysis of discrete stochastic processes. *npj Quantum Information* 7, 1, 126.

[7] Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. 2002. Quantum amplitude amplification and estimation. https://doi.org/10.1090/conm/305/05215

[8] Sergey Bravyi and Dmitri Maslov. 2021. Hadamard-Free Circuits Expose the Structure of the Clifford Group. *IEEE Transactions on Information Theory* 67, 7, 4546–4563. https://doi.org/10.1109/TIT.2021.3081415

[9] Michael Broughton, Guillaume Verdon, Trevor McCourt, Antonio J. Martinez, Jae Hyeon Yoo, Sergei V. Isakov, Philip Massey, Ramin Halavati, Murphy Yuezhen Niu, Alexander Zlokapa, Evan Peters, Owen Lockwood, Andrea Skolik, Sofiene Jerbi, Vedran Dunjko, Martin Leib, Michael Streif, David Von Dollen, Hongxiang Chen, Shuxiang Cao, Roeland Wiersema, Hsin-Yuan Huang, Jarrod R. McClean, Ryan Babbush, Sergio Boixo, Dave Bacon, Alan K. Ho, Hartmut Neven, and Masoud Mohseni. 2021. TensorFlow Quantum: A Software Framework for Quantum Machine Learning. https://doi.org/10.48550/arXiv.2003.02989 arXiv:2003.02989 [quant-ph]

[10] José Campos and André Souto. 2021. QBugs: A Collection of Reproducible Bugs in Quantum Algorithms and a Supporting Infrastructure to Enable Controlled Quantum Software Testing and Debugging Experiments. In *2021 IEEE/ACM 2nd International Workshop on Quantum Software Engineering (Q-SE)*. Association for Computing Machinery, New York, NY, USA, 28–32. https://doi.org/10.1109/Q-SE52541.2021.00013

[11] Yudong Cao, Jonathan Romero, Jonathan P Olson, Matthias Degroote, Peter D Johnson, Mária Kieferová, Ian D Kivlichan, Tim Menke, Borja Peropadre, Nicolas PD Sawaya, et al. 2019. Quantum chemistry in the age of quantum computing. *Chemical reviews* 119, 19, 10856–10915. https://doi.org/10.1021/acs.chemrev.8b00803

[12] Indranil Chakrabarty, Shahzor Khan, and Vanshdeep Singh. 2017. Dynamic Grover search: Applications in recommendation systems and optimization problems. *Quantum Information Processing* 16, 1–21. https://doi.org/10.1007/s11128-017-1600-4

[13] Yu-Fang Chen, Kai-Min Chung, Ondřej Lengál, Jyun-Ao Lin, Wei-Lun Tsai, and Di-De Yen. 2023. An automata-based framework for verification and bug hunting in quantum circuits. *International Conference on Programming Language Design and Implementation (PLDI)* 7, PLDI, 1218–1243. https://doi.org/10.1145/3591270

[14] D. Coppersmith. 2002. An approximate Fourier transform useful in quantum factoring. https://doi.org/10.48550/arXiv.quant-ph/0201067 arXiv:quant-ph/0201067

[15] Calsi Davide, Li, Duran Matias, Laurent Thomas, Zhang Xiao-Yi, Arcaini Paolo, Ishikawa Fuyuki, and Ventresque Anthony. 2024. SSBSE Summary of Adaptive Search-based Repair of Deep Neural Networks. In *ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. Association for Computing Machinery, New York, NY, USA, 1229–1241. https://doi.org/10.1145/3583131.3590477

[16] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 337–340. https://doi.org/10.1007/978-3-540-78800-3_24

[17] Siwei Tan Debin Xiang. 2024. *HornBro: Homotopy-like Method for Automated Quantum Program Repair.* https://doi.org/10.5281/zenodo.14288140

[18] Jeroen Dehaene and Bart De Moor. 2003. Clifford group, stabilizer states, and linear and quadratic operations over GF (2). *Physical Review A* 68, 4, 042318. https://doi.org/10.1103/PhysRevA.68.042318

[19] David Deutsch and Richard Jozsa. 1992. Rapid solution of problems by quantum computation. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences* 439.

[20] IBM Quantum Devices. 2024. ibmq quito. https://quantum-computing.ibm.com/services/resources

[21] Emily First, Markus N. Rabe, Talia Ringer, and Yuriy Brun. 2023. Baldur: Whole-Proof Generation and Repair with Large Language Models. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. Association for Computing Machinery, New York, NY, USA, 1229–1241. https://doi.org/10.1145/3611643.3616243

[22] Tobias Fritz. 2012. Beyond Bell's theorem: correlation scenarios. *New Journal of Physics* 14, 10, 103001. https://doi.org/10.1088/1367-2630/14/10/103001

[23] Lov K. Grover. 1996. A Fast Quantum Mechanical Algorithm for Database Search. In *Annual ACM Symposium on Theory of Computing (STOC)*. Association for Computing Machinery, New York, NY, USA, 212–219. https://doi.org/10.1145/237814.237866

[24] Xiaoyu Guo, Jianjun Zhao, and Pengzhan Zhao. 2024. On Repairing Quantum Programs Using ChatGPT. In *Proceedings of the 5th ACM/IEEE International Workshop on Quantum Software Engineering (Q-SE 2024)*. Association for Computing Machinery, New York, NY, USA, 9–16. https://doi.org/10.1145/3643667.3648223

[25] Shahin Honarvar, Mohammad Reza Mousavi, and Rajagopal Nagarajan. 2020. Property-based Testing of Quantum Programs in Q#. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*. Association for Computing Machinery, New York, NY, USA, 430–435. https://doi.org/10.1145/3387940.3391459

[26] Seongjoon Hong, Junhee Lee, Jeongsoo Lee, and Hakjoo Oh. 2020. SAVER: Scalable, Precise, and Safe Memory-Error Repair. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. Association for Computing Machinery, New York, NY, USA, 271–283. https://doi.org/10.1145/3377811.3380323

[27] Soneya Binta Hossain, Nan Jiang, Qiang Zhou, Xiaopeng Li, Wen-Hao Chiang, Yingjun Lyu, Hoan Nguyen, and Omer Tripp. 2024. A deep dive into large language models for automated bug localization and repair. *Proc. ACM Softw. Eng.* 1, FSE, Article 66, 23 pages. https://doi.org/10.1145/3660773

[28] Cupjin Huang, Fang Zhang, Michael Newman, Junjie Cai, Xun Gao, Zhengxiong Tian, Junyin Wu, Haihong Xu, Huanjun Yu, Bo Yuan, et al. 2020. Classical simulation of quantum supremacy circuits. *arXiv preprint arXiv:2005.06787*. https://doi.org/10.48550/arXiv.2005.06787

[29] Yipeng Huang and Margaret Martonosi. 2019. Statistical assertions for validating patterns and finding bugs in quantum programs. In *International Symposium on Computer Architecture (ISCA)*. Association for Computing Machinery, New York, NY, USA, 541–553. https://doi.org/10.1145/3307650.3322213

[30] Jiajun Jiang, Yingfei Xiong, Hongyu Zhang, Qing Gao, and Xiangqun Chen. 2018. Shaping program repair space with existing patches and similar code. In *SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*. Association for Computing Machinery, New York, NY, USA, 298–309. https://doi.org/10.1145/3213846.3213871

[31] Dongsun Kim, Jaechang Nam, Jaewoo Song, and Sunghun Kim. 2013. Automatic patch generation learned from human-written patches. In *2013 35th International Conference on Software Engineering (ICSE)*. Association for Computing Machinery, New York, NY, USA, 802–811. https://doi.org/10.1109/ICSE.2013.6606626

[32] A. Yu. Kitaev. 1995. Quantum measurements and the Abelian Stabilizer Problem. arXiv:quant-ph/9511026 [quant-ph] https://arxiv.org/abs/quant-ph/9511026

[33] Mark W Krentel. 1986. The Complexity of Optimization Problems. In *Annual ACM symposium on Theory of computing (STOC)*. Association for Computing Machinery, New York, NY, USA, 69–76. https://doi.org/10.1016/0022-0000(88)90039-6

[34] Junhee Lee, Seongjoon Hong, and Hakjoo Oh. 2018. Memfix: static analysis-based repair of memory deallocation errors for c. In *ACM Joint meeting on European software engineering conference and symposium on the foundations of software engineering (ESEC/FSE)*. Association for Computing Machinery, New York, NY, USA, 95–106. https://doi.org/10.1145/3236024.3236079

[35] Gushu Li, Li Zhou, Nengkun Yu, Yufei Ding, Mingsheng Ying, and Yuan Xie. 2020. Projection-based runtime assertions for testing and debugging quantum programs. *Proceedings of the ACM on Programming Languages (OOPSLA)* 4, 1–29. https://doi.org/10.1145/3428218

[36] Tianlin Li, Yue Cao, Jian Zhang, Shiqian Zhao, Yihao Huang, Aishan Liu, Qing Guo, and Yang Liu. 2024. RUNNER: Responsible UNfair NEuron Repair for Enhancing Deep Neural Network Fairness. In *IEEE/ACM International Conference on Software Engineering (ICSE)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3597503.3623334

[37] Yuechen Li, Hanyu Pei, Linzhi Huang, Beibei Yin, and Kai-Yuan Cai. 2024. Automatic Repair of Quantum Programs via Unitary Operation. *ACM Transactions on Software Engineering and Methodology (TOSEM)*. https://doi.org/10.1145/3664604

[38] Ji Liu and Huiyang Zhou. 2021. Systematic Approaches for Precise and Approximate Quantum State Runtime Assertion. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 179–193. https://doi.org/10.1109/HPCA51647.2021.00025

[39] Junjie Luo, Pengzhan Zhao, Zhongtao Miao, Shuhan Lan, and Jianjun Zhao. 2022. A Comprehensive Study of Bug Fixes in Quantum Programs. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 1239–1246. https://doi.org/10.1109/SANER53432.2022.00147

[40] Jianan Ma, Pengfei Yang, Jingyi Wang, Youcheng Sun, Cheng-Chao Huang, and Zhen Wang. 2024. VeRe: Verification Guided Synthesis for Repairing Deep Neural Networks. In *IEEE/ACM International Conference on Software Engineering (ICSE)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3597503.3623332

[41] Xiangxin Meng, Xu Wang, Hongyu Zhang, Hailong Sun, Xudong Liu, and Chunming Hu. 2023. Template-based Neural Program Repair. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. 1456–1468. https://doi.org/10.1109/ICSE48619.2023.00127

[42] David A Meyer. 1996. From quantum cellular automata to quantum lattice gases. *Journal of Statistical Physics* 85, 551–574. https://doi.org/10.1007/BF02199356

[43] Manish Motwani and Yuriy Brun. 2023. Better automatic program repair by using bug reports and tests together. In *IEEE/ACM International Conference on Software Engineering (ICSE)*. IEEE, 1225–1237. https://doi.org/10.1109/ICSE48619.2023.00109

[44] Keigo Naitou, Akito Tanikado, Shinsuke Matsumoto, Yoshiki Higo, Shinji Kusumoto, Hiroyuki Kirinuki, Toshiyuki Kurabayashi, and Haruto Tanno. 2018. Toward introducing automated program repair techniques to industrial software development. In *Proceedings of the 26th Conference on Program Comprehension*. Association for Computing Machinery, New York, NY, USA, 332–335. https://doi.org/10.1145/3196321.3196358

[45] Tai D. Nguyen, Long H. Pham, and Jun Sun. 2021. SGUARD: Towards Fixing Vulnerable Smart Contracts Automatically. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1215–1229. https://doi.org/10.1109/SP40001.2021.00057

[46] Michael A. Nielsen and Isaac L. Chuang. 2011. *Quantum Computation and Quantum Information: 10th Anniversary Edition* (10th ed.). Cambridge University Press, USA.

[47] Tianyi Peng, Aram W Harrow, Maris Ozols, and Xiaodi Wu. 2020. Simulating large quantum circuits on a small quantum computer. *Physical review letters* 125, 15, 150504. https://doi.org/10.1103/PhysRevLett.125.150504

[48] Yuxiang Peng, Mingsheng Ying, and Xiaodi Wu. 2022. Algebraic reasoning of Quantum programs via non-idempotent Kleene algebra. In *International Conference on Programming Language Design and Implementation (PLDI)*. Association for Computing Machinery, New York, NY, USA, 657–670. https://doi.org/10.1145/3519939.3523713

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

[50] Robert Robere, Antonina Kolokolova, and Vijay Ganesh. 2018. The proof complexity of SMT solvers. In *International Conference on Computer Aided Verification (CAV)*. Springer, 275–293.

[51] V.V. Shende, S.S. Bullock, and I.L. Markov. 2006. Synthesis of quantum-logic circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 25, 6, 1000–1010. https://doi.org/10.1109/TCAD.2005.855930

[52] Sunbeom So and Hakjoo Oh. 2023. Smartfix: Fixing vulnerable smart contracts by accelerating generate-and-verify repair using statistical models. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. Association for Computing Machinery, New York, NY, USA, 185–197. https://doi.org/10.1145/3611643.3616341

[53] Yahui Song, Xiang Gao, Wenhua Li, Wei-Ngan Chin, and Abhik Roychoudhury. 2024. ProveNFix: Temporal Property-Guided Program Repair. *ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)* 1, 226–248. https://doi.org/10.1145/3643737

[54] Stack Exchange Inc. 2023. Stack Overflow - Where Developers Learn, Share, & Build Careers. https://stackoverflow.com/

[55] Krysta Svore, Alan Geller, Matthias Troyer, John Azariah, Christopher Granade, Bettina Heim, Vadym Kliuchnikov, Mariia Mykhailova, Andres Paz, and Martin Roetteler. 2018. Q# enabling scalable quantum computing and development with a high-level dsl. In *Proceedings of the real world domain specific languages workshop 2018*. Association for Computing Machinery, New York, NY, USA, 1–10. https://doi.org/10.1145/3183895.3183901

[56] Siwei Tan, Debin Xiang, Liqiang Lu, Junlin Lu, Qiuping Jiang, Mingshuai Chen, and Jianwei Yin. 2024. MorphQPV: Exploiting Isomorphism in Quantum Programs to Facilitate Confident Verification. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. Association for Computing Machinery, New York, NY, USA, 671–688. https://doi.org/10.1145/3620666.3651360

[57] 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 *ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI)*. Association for Computing Machinery, New York, NY, USA, 641–656. https://doi.org/10.1145/3519939.3523431

[58] Hiroyuki Tezuka, Kouhei Nakaji, Takahiko Satoh, and Naoki Yamamoto. 2022. Grover search revisited: Application to image pattern matching. *Physical Review A* 105, 3, 032440. https://doi.org/10.1103/PhysRevA.105.032440

[59] Xinyi Wang, Paolo Arcaini, Tao Yue, and Shaukat Ali. 2021. Quito: a Coverage-Guided Test Generator for Quantum Programs. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. Association for Computing Machinery, New York, NY, USA, 1237–1241. https://doi.org/10.1109/ASE51524.2021.9678798

[60] Xinyi Wang, Paolo Arcaini, Tao Yue, and Shaukat Ali. 2022. Generating failing test suites for quantum programs with search. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. Association for Computing Machinery, New York, NY, USA, 47–48. https://doi.org/10.1145/3520304.3534067

[61] Xinyi Wang, Paolo Arcaini, Tao Yue, and Shaukat Ali. 2022. QuSBT: search-based testing of quantum programs. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*. Association for Computing Machinery, New York, NY, USA, 173–177. https://doi.org/10.1145/3510454.3516839

[62] David Wierichs, Josh Izaac, Cody Wang, and Cedric Yen-Yu Lin. 2022. General parameter-shift rules for quantum gradients. *Quantum* 6, 677. https://doi.org/10.22331/q-2022-03-30-677

[63] He Ye and Martin Monperrus. 2024. ITER: Iterative Neural Repair for Multi-Location Patches. In *IEEE/ACM International Conference on Software Engineering (ICSE)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3597503.3623337

[64] Mingsheng Ying. 2012. Floyd–hoare logic for quantum programs. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 33, 6, 1–49. https://doi.org/10.1145/2049706.2049708

[65] Ed Younis, Koushik Sen, Katherine 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)*. IEEE, 232–243. https://doi.org/10.48550/arXiv.2003.04462

[66] Pengzhan Zhao, Jianjun Zhao, and Lei Ma. 2021. Identifying Bug Patterns in Quantum Programs. In *2021 IEEE/ACM 2nd International Workshop on Quantum Software Engineering (Q-SE)*. Association for Computing Machinery, New York, NY, USA, 16–21. https://doi.org/10.1109/Q-SE52541.2021.00011

[67] Pengzhan Zhao, Jianjun Zhao, Zhongtao Miao, and Shuhan Lan. 2021. Bugs4Q: A Benchmark of Real Bugs for Quantum Programs. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. Association for Computing Machinery, New York, NY, USA, 1373–1376. https://doi.org/10.1109/ASE51524.2021.9678908

[68] Leo Zhou, Sheng-Tao Wang, Soonwon Choi, Hannes Pichler, and Mikhail D. Lukin. 2020. Quantum Approximate Optimization Algorithm: Performance, Mechanism, and Implementation on Near-Term Devices. *Phys. Rev. X (PRX)* 10. Issue 2. https://doi.org/10.1103/PhysRevX.10.021067

[69] Li Zhou, Nengkun Yu, and Mingsheng Ying. 2019. An applied quantum Hoare logic. In *International Conference on Programming Language Design and Implementation (PLDI)*. Association for Computing Machinery, New York, NY, USA, 1149–1162. https://doi.org/10.1145/3314221.3314584

Received 2024-09-13; accepted 2025-01-14