

Optimizing Run-Time Assertions for Quantum Circuit Debugging: A Depth–Ancilla Tradeoff

Karl Aaron Rudkowski¹

Abhoy Kole¹

Rolf Drechsler^{1,2}

¹Cyber-Physical Systems, DFKI GmbH, 28359 Bremen, Germany

²Institute of Computer Science, University of Bremen, 28359 Bremen, Germany

{karlaaron, drechsler}@uni-bremen.de, Abhoy.Kole@dfki.de

Abstract—Debugging quantum circuits is inherently challenging due to the probabilistic nature of quantum computation and the inability to directly observe intermediate states without disturbing the computation. Run-time assertion-based debugging is a promising approach to provide partial visibility into quantum programs by validating expected properties during execution. However, existing approaches often incur significant overhead in terms of gate count, circuit depth, and auxiliary qubits, limiting their applicability on resource-constrained quantum hardware.

In this work, we propose a framework for optimizing run-time assertions in quantum circuits using dynamic circuit capabilities. Our approach leverages classically conditioned operations and repeated measurement–reset patterns to implement assertions efficiently while preserving the state of data qubits. Furthermore, we explore the tradeoff between circuit depth and the number of auxiliary qubits by dynamically distributing assertion components across a limited set of ancilla, enabling scalable assertion insertion while balancing hardware constraints such as measurement latency and reset overhead. Experimental evaluation on Quantum Phase Estimation (QPE) circuits demonstrates that our approach significantly reduces auxiliary qubit usage and gate overhead compared to existing methods, while maintaining effective fault detection and localization capabilities.

Index Terms—Quantum Debugging, Quantum Assertions, Dynamic Circuits,

I. INTRODUCTION

Ensuring the correctness of quantum programs is a fundamental challenge in the development of reliable quantum software [1]–[4]. Unlike classical programs, quantum computations are inherently probabilistic and do not allow direct observation of intermediate states without disturbing the computation. As a result, traditional debugging techniques based on state inspection and trace analysis are not directly applicable. While formal methods such as equivalence checking and formal reasoning provide strong guarantees of correctness, they often suffer from scalability limitations for large and complex quantum circuits [5]–[7]. Consequently, testing and debugging remain essential tools for identifying and localizing faults in practical quantum programs [8]–[11].

In the classical domain, decades of research have led to mature and highly effective debugging and verification techniques. In contrast, debugging methodologies for quantum programs are still in their early stages. Existing approaches broadly include statistical techniques, which infer correctness properties from repeated executions and are particularly suitable for noisy intermediate-scale quantum (NISQ) devices, and run-time assertion-based techniques, which validate expected

properties during execution [12]–[15]. While statistical assertions offer low overhead, they often require extensive post-processing and may lack precision. Run-time assertions, on the other hand, provide fine-grained validation and improved fault localization, making them particularly promising for fault-tolerant quantum computing (FTQC) settings. However, the practical deployment of run-time assertions faces significant challenges. Inserting assertion checks into quantum circuits introduces additional overhead in terms of gate count, circuit depth, and auxiliary qubits. These overheads directly impact execution latency, hardware resource requirements, and overall scalability. Therefore, efficient interpretation and realization of run-time assertions are crucial to ensure their applicability on real quantum hardware. In particular, minimizing execution delay, reducing ancillary qubit usage, and balancing tradeoffs between circuit depth and resource consumption are key factors for enabling scalable debugging.

Motivated by these challenges, this work proposes a framework for efficiently interpreting and implementing run-time assertions in quantum circuits. Our approach builds upon prior work on projection-based assertions [13] and systematic state assertion techniques [15], and combines their strengths to derive optimized assertion realizations. Specifically, we make the following contributions:

- We propose a dynamic assertion interpretation that leverages auxiliary qubits and classically conditioned operations to minimize execution delay on quantum hardware.
- We introduce a qubit reuse strategy that significantly reduces the number of auxiliary qubits required for debugging, enabling scalable assertion deployment.
- We demonstrate the effectiveness of our approach using the QPE algorithm as a representative case study.
- We experimentally evaluate the resource requirements of our approach and show that it reduces gate overhead and auxiliary qubit usage compared to existing methods, while enabling a flexible tradeoff between circuit depth and ancilla resources.

II. BACKGROUND

A) Quantum Computing: Quantum computing exploits principles of quantum mechanics to process information in fundamentally different ways from classical computers. While classical systems use bits with values $\in \{0, 1\}$, quantum computers use qubits, which can exist in $|0\rangle$ and $|1\rangle$ as well

```

1 OPENQASM 2.0;
2 include "qelib1.inc";
3
4 qreg q[3];
5 creg c[2];
6
7 init q[0], q[1], q[2];
8 cp(3*pi/4) q[0], q[2];
9 cp(3*pi/2) q[1], q[2];
10 iqft q[0], q[1];
11
12 measure q[0] -> c[0];
13 measure q[1] -> c[1];

```

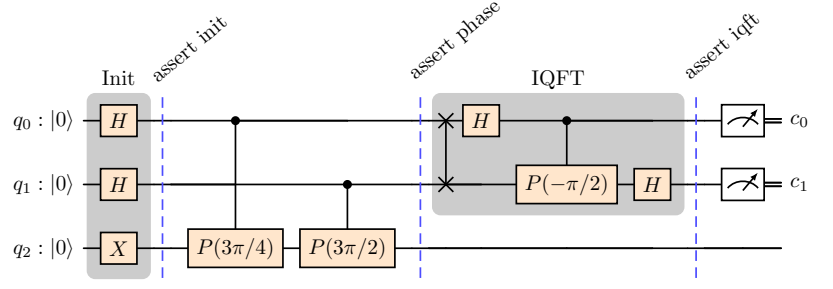
```

gate init a,b,c {
  h a;
  h b;
  x c;
} // Init

gate iqft a,b {
  swap a,b;
  h a;
  cp(-pi/2) a, b;
  h b;
} // IQFT

```

(a)



(b)

Fig. 1: QPE example: (a) QASM program and (b) corresponding 3-qubit circuit representation

as superpositions of states. This enables certain computations to be performed more efficiently than on classical hardware. A key feature of quantum systems is entanglement, where the state of one qubit becomes correlated with another. Another important property is interference, which allows quantum algorithms to amplify correct computational paths while canceling out incorrect ones. Quantum computations are expressed as sequences of gates applied to qubits, forming quantum circuits. Common single-qubit gates include the Pauli gates X , Y , and Z , and H and phase rotations such as $P(\theta)$. Together with a two-qubit entangling gate such as the controlled-NOT (CX) gate, these form a universal gate set for quantum computation [16]. These gates correspond to unitary transformations, and measurement collapses their probabilistic quantum states into classical outcomes. Due to this probabilistic nature, quantum programs often need to be executed multiple times to obtain reliable results. Fig. 1 shows an QASM2.0 program and its corresponding circuit representation for the QPE algorithm.

B) Debugging Quantum Circuit: The probabilistic nature of quantum computation makes debugging quantum circuits more challenging than debugging classical programs. Whereas program states in classical systems can be inspected during execution, quantum states cannot be directly observed without disturbing the computation, as measurement collapses superposition and entanglement. Consequently, incorrect behavior may not manifest deterministically, requiring repeated executions and statistical analysis. Moreover, interference plays a central role: small errors in phase or gate application can significantly alter interference patterns, amplifying incorrect computational paths while suppressing correct ones. Such phase-related bugs are particularly difficult to detect, as they do not affect circuit structure and cannot be directly observed in intermediate states. These challenges motivate the need for indirect debugging techniques that provide partial visibility into program execution. One promising approach is assertion-based debugging, where programmers specify expected properties of a quantum program at specific points in the circuit. In particular, run-time assertions can be categorized based on the quantum properties they capture, including (i) classical, (ii) superposition-based (iii) entanglement assertions, and (iv) reference-state assertions, where a target quantum state is

specified via a reference circuit. As an illustration, Fig. 1b shows the QPE circuit annotated with run-time assertions at multiple locations. After the initialization block, one may assert that the target qubit (q_2) is in state $|1\rangle$, while the counting register (q_1q_0) is in the superposition $|++\rangle$. Similarly, after the controlled phase operations, assertions can validate that the phase information has been correctly encoded, and after the inverse Quantum Fourier Transformation (QFT), classical assertions can verify that the measurement outcome matches the expected phase.

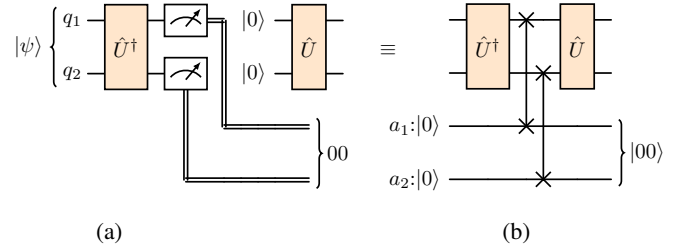


Fig. 2: Reference-state assertion interpretations for two data qubits based on: (a) projection [13], (b) ancilla & SWAP [15]

C) Related Work: Prior work on quantum debugging and validation spans formal verification, statistical debugging, and run-time assertion techniques. Formal approaches such as equivalence checking and quantum Hoare logic provide rigorous correctness guarantees [5]–[7], while empirical studies have investigated the nature and characteristics of quantum software bugs [8]–[10]. Benchmark infrastructures such as Q Bugs [17] and Bugs4Q [18] have been proposed to support reproducible debugging and testing studies. Assertion-based debugging approaches can broadly be classified into statistical and run-time assertions. Statistical assertions validate program behavior through repeated executions and measurement analysis. Huang and Martonosi [12] introduced one of the earliest statistical assertion frameworks for quantum programs, later extended in CUDA-Q by Li *et al.* [19]. Recent work such as Bloq [20] further explored automated assertion generation and fault localization using Bloch-vector-based assertions. Run-time assertion techniques attempt to validate quantum states during execution while preserving program behavior. Zhou

and Byrd [14] proposed early runtime assertion circuits using ancilla qubits. Proq [13] introduced projection-based runtime assertions based on direct measurement and reset of program qubits, leading to increased delay due to measurement and reset operations. In contrast, Liu and Zhou [15] avoids these direct measurements by transferring the state to ancilla qubits using SWAP operations, which can be executed in parallel but introduce additional gate overhead. Figure 2 illustrates these two assertion interpretations for a two-qubit system. Subsequent work explored assertions for debugging and error mitigation [21], efficient runtime assertion evaluation [22], and tooling support for assertion-based debugging [23].

In contrast to prior approaches, our work focuses on efficient interpretation of run-time assertions under realistic hardware constraints by combining low-ancilla assertion strategies with dynamic correction propagation and ancilla reuse techniques.

III. TEST AND DEBUGGING FRAMEWORK

We propose a framework to realize dynamic assertions with efficient interpretations under user-defined constraints. With the dynamic assertions, bugs can be detected, localized, and corrected during the execution of the circuit. For efficient interpretations, we leverage two features of dynamic circuits, (1) classically conditioned gates, which can facilitate the assertion interpretations themselves, and (2) mid-circuit measurements, which enable the reuse of ancilla qubits.

A) Supported Assertions: We consider a reference-state assertion of the form $\text{assert } q_{\text{reg}} == \hat{U}$, which verifies that an n -qubit register q_{reg} is in a target state $|\psi\rangle$. Here, the operator $\hat{U}$ is defined such that $|\psi\rangle = \hat{U}|0\rangle^{\otimes n}$, where $|0\rangle^{\otimes n}$ denotes the n -qubit zero state. Our framework reads and writes QASM code, where $\hat{U}$ can be represented as:

```
gate  $\hat{U}$  q_1, q_2, ..., q_n {
    // operations preparing  $|\psi\rangle$  from  $|0\rangle^{\otimes n}$ 
}
```

B) Assertion Interpretations using Classically Conditioned Gates: Current reference-state assertions with ancilla qubits use SWAP gates to communicate between data and ancilla qubits. However, given that $\hat{U}$ is applied to the data qubits, the ancilla qubits are in $|0 \dots 0\rangle$ before the SWAP. Therefore, errors can be detected using CX gates instead of SWAP gates, as shown in Fig. 3a. Then, we use classically conditioned X gates on the data qubits to ensure that these are in $|0 \dots 0\rangle$ state even in case of an error. These gates are only included in the circuit at runtime, and if a $|1\rangle$ was measured on the corresponding ancilla qubit. With these assertion interpretations, any arbitrary state can be checked, including basic cases such as classic, superposition, or entangled states. However, inserting them into circuits leads to an overhead in gates, depth, and ancilla qubits. The latter two can be addressed by reusing ancilla qubits.

C) Reusing Ancilla Qubits: In previous work, measurements are either made directly on the data qubits [13], or each assertion location in the algorithm is assigned a separate set of ancilla qubits [15]. Each ancilla qubit is then measured once,

at the end. The former incurs significant execution delay due to the relatively high latency of measurement and reset operations compared to standard quantum gates, while the latter mitigates this by using a maximum number of ancilla qubits. Though involving auxiliary qubits is preferable to lower the depth, quantum computers only have a very limited number of qubits. Therefore, precisely these ancilla qubits are the reason the latter approach also does not scale well. We propose to reuse a limited set of ancilla qubits, thereby finding a balance between the execution delay and ancilla qubit requirements. An extreme example of this is shown in Fig. 3b for a two-qubit reference-state assertion where only one ancilla qubit is available. After each measurement, we reset the ancilla qubit so that the next assertion is not influenced by a possible error. If an error is detected, it can be identified using the classical bits, because each measurement only writes to one classical bit.

Algorithm 1 Distributing assertion interpretations across a limited number of ancilla qubits

Input: QASM *file* with assertion statements

Input: limit for new ancilla qubits a_{max}

Output: QASM file for circuit with inserted ancilla qubits and assertion interpretations

```
1:  $\text{ancillas} \leftarrow 0$ 
2:  $\text{assertions} \leftarrow \text{scanQASM}(\text{file})$ 
3: for all  $a \in \text{assertions}$  do
4:   for all  $q \in a.\text{qubits}$  do
5:     if  $\text{ancillas} + 1 \leq a_{\text{max}}$  then
6:        $a.\text{ancilla}[q] \leftarrow \text{ancillas}$ 
7:        $\text{ancillas} \leftarrow \text{ancillas} + 1$ 
8:     else
9:        $a.\text{ancilla}[q] \leftarrow \text{lru\_ancilla}()$ 
10:  for all  $a \in \text{assertions}$  do
11:     $\text{addCircuit}(a)$ 
```

For a general user-defined number of ancilla qubits, we propose an algorithm that evenly distributes the assertion interpretations across the available ancilla qubits. For the interpretation shown in Fig. 3a, the gates involving the ancilla qubits can be split into subcomponents. Before and after the subcomponents, $\hat{U}^\dagger$ and $\hat{U}$ are executed on the data qubits, respectively. Each subcomponent implicates one ancilla and one data qubit, and consists of a CX , a measurement and a classically conditioned X gate. They are independent of each other, both in their gates and in their measurement outcomes. These observations mean that assertion interpretations can be distributed evenly at the subcomponent-level. Each subcomponent is assigned to one ancilla qubit in a round-robin fashion, always choosing the least recently used ancilla qubit. Fig. 3c illustrates this approach for an n -qubit assertion with two available ancilla qubits. Algorithm 1 describes how our framework distributes assertion interpretations. First, we scan the QASM code for all assertion statements. Then, we assign ancilla qubits to the assertions. For each assertion, we iterate through all associated data qubits. For each data qubit, we determine if we can still insert a new ancilla qubit, and use that

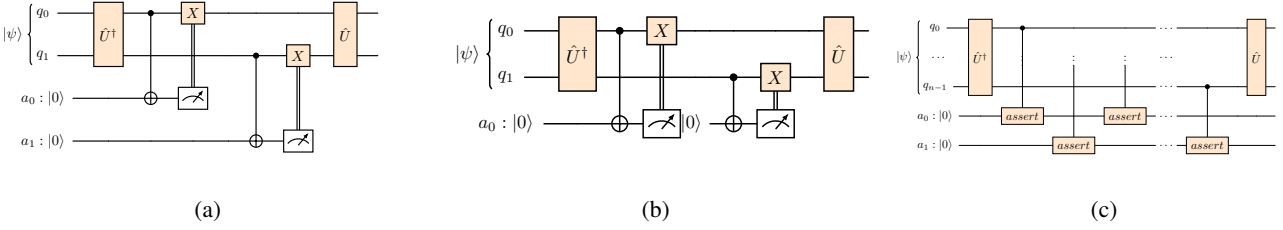


Fig. 3: Reference-state assertion interpretation for two qubits (a) using classically conditioned gates and (b) reusing one ancilla qubit; (c) Distributing n -qubit assertion across two ancilla qubits

if possible. If that is not possible, we reuse the least recently used ancilla qubit. Finally, we insert the interpretations of the assertions. In our algorithm, associating the assertions with ancilla qubits (lines 3-9) is done in $\mathcal{O}(m \times n)$ for m assertions and n data qubits. Because of the separation into subcomponents that each require one ancilla qubit, all available ancilla qubits are used. Further, our approach enables an even distribution not just for the subcomponents of a single assertion, but for multiple assertions as well, without necessitating any distinction between them.

This algorithm influences the overall depth of the circuit depending on the number of ancilla qubits a_{max} . If a_{max} is smaller than the number of data qubits n , our algorithm reuses ancilla qubits within one assertion. In this case, one ancilla qubit a_i is mapped to multiple data qubits $\{q_j, q_k, \dots\}$, and the CX , measure, reset subcomponent for q_k can only be executed after the subcomponent for q_j is done. Therefore, the longest path through one assertion goes over the ancilla qubit that is mapped to the most data qubits. If, however, $a_{max} \geq n$ holds, our framework interprets one assertion by mapping one ancilla qubit to exactly one data qubit. Within this assertion, each ancilla qubit executes the same number of gates, resulting in the minimum local depth. For $a_{max} > n$, this local depth does not change because one n qubit assertion can only use at most n ancilla qubits. Finally, reusing ancilla qubits across multiple assertions $eq_0, eq_1, \dots$ does not influence the overall depth. On the data qubits, the last gates for eq_0 that depend on the ancilla qubits are the CX before the measurement, and the classically controlled X after. Following, $\hat{U}_0$, U_1 , and $U_1^\dagger$ are executed before the next involvement of the ancilla qubits for eq_1 . This sequence has a depth ≥ 3 . On each ancilla qubit, only one reset is executed. Consequently, the longest path between two assertions goes over the data qubits. Therefore, our framework reaches the lowest possible depth for an n qubit circuit using n ancilla qubits. Even for multiple assertions, $a_{max} > n$ would not lead to a lower depth, because neither the local depth of each assertion, nor the path between the assertions would improve.

IV. EVALUATION

For assessing our approach, we evaluate the (1) resource requirements in comparison to existing methods, (2) tradeoff between circuit depth and ancilla resources, and (3) effectiveness for localizing bugs. We use the QPE algorithm that

already served as a motivating example above. We consider circuits with 3 to 14 data qubits, and insert reference-state assertions in the three positions shown in Fig. 1b. The first assertion checks the initial state preparation, where the n^{th} qubit should be in the state $|1\rangle$, and all others in $|+\rangle$. The second and third assertion check that the controlled phase operations and the inverse QFT, respectively, implement the expected calculations. Our framework uses the unitary gates $\{H, P, X, CX\}$ as well as the non-unitary gates $\{\text{mid-circuit measurement, reset, classically controlled gate}\}$. The circuits are simulated in Qiskit [24] version 2.4.1 on the Aer simulator version 0.17.1 for 1000 shots each. All experiments are conducted on a AMD Ryzen 7 PRO 5850U with 48GB RAM.

A) Baseline Comparison: To assess the influence of our proposed framework on the assertion efficiency, we compare it to the two approaches proposed in [15] in terms of the costs and depth of the assertions in the 12 QPE circuits. The first of the two reference approaches realizes assertions with multiple ancilla qubits and $SWAP$ gates, the second with one ancilla qubit and a negative-controlled MCX gate. Both approaches use a fresh set of ancilla qubits for each assertion. For our framework, we configured the user-defined ancilla qubit limit to 1, which is the smallest possible limit. All non-unitary gates are treated as single qubit operations (denoted as U), including the classically conditioned X gates. This represents the case that all assertions fail and have to be corrected before the $\hat{U}$. For the multi-ancilla baseline [15], each $SWAP$ gate is decomposed to two CX gates. For the single-ancilla baseline [15], we use the standard Qiskit [24] decomposition for MCX gates. While additional ancilla qubits can improve the depth of the MCX decomposition, we use a version without ancilla qubits to represent an assertion interpretation using a single ancilla. Figure 4 shows the results, with each of the four diagrams illustrating one of the circuit characteristics (a) number of ancilla qubits, (b) circuit depth, and number of (c) U and (d) CX gates. For each characteristic, three graphs show how our framework compares to the two baselines for QPE circuits with a rising number of data qubits.

When comparing the results of the three approaches, our framework is comparable to the multi-ancilla baseline, with differences mostly below 15%. Our framework uses more U gates (baseline up to 22.78% less), but fewer CX gates (baseline up to 30% more). Regarding the CX gates, both our framework and the baseline require at least n CX gates

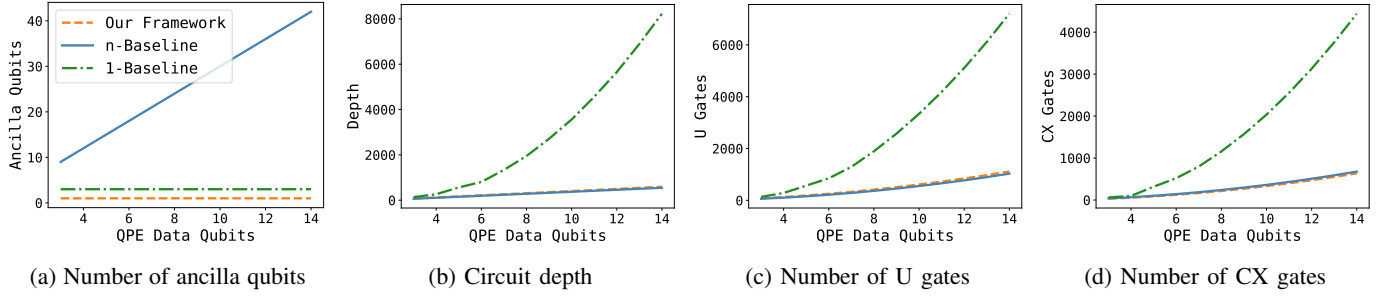


Fig. 4: Comparison of 12 QPE circuits, in which assertions were inserted using (1) our framework with one ancilla qubit overall, (2) a baseline with n ancilla qubit per assertions [15], and (3) a baseline with 1 ancilla qubit per assertion [15]

for one n qubit assertion. However, for n SWAP gates, the baseline uses n additional CX gates, resulting in $3n$ more CX gates overall. Regarding the U gates, our framework realizes one n qubit assertion with n classically controlled X gates and n resets. For the three assertions, this results in $6n$ more U gates. This shift away from CX gates is beneficial for the robustness of the circuit, because CX gates are more error prone. Nonetheless, the differences in gate counts have no drastic influence on the circuit depth. The baseline's longest path is around 10% shorter, which is a direct result of the overall higher gate count in our framework. However, the multi-ancilla baseline requires more ancilla qubits for this comparable circuit performance. For one n qubit assertion, n ancillas are necessary, resulting in $3n$ for three assertions. In our case study, this leads to up to 42 ancilla qubits in addition to 14 data qubits. Our framework implements the three assertions with exactly one ancilla qubit. In this regard, it is comparable to the single-ancilla baseline. This baseline implements the three assertions with one ancilla qubit each, resulting in 3 ancilla qubits overall. However, it relies on costly MCX gates, resulting in higher gate counts and circuit depth. Regarding the gates, the baseline needs up to 546.90% more U gates, and up to 599.06% more CX gates. Regarding the depth, up to 1289.36% more gates are on the longest path.

B) Number of Ancilla Qubits: In quantum circuits, the tradeoff between the number of ancilla qubits and the depth is a common design aspect. With a user-defined number of ancilla qubits, the circuits generated by our framework can be further improved, without scaling in relation to the number of data qubits or assertions. To investigate how this user-defined limit influences the circuit depth, we consider the QPE circuit with seven data qubits, and configure it with an increasingly large ancilla limit. Fig. 5a illustrates the depth of the generated circuits. The depth of the multi-ancilla baseline for the seven data qubit QPE circuit (239) is added for reference.

The general trend is that a higher number of ancilla qubits leads to a lower depth. As discussed in Section III, for $a_{max} < 7$, ancilla qubits have to be reused within a single assertion, which leads to delays during this assertion. For $a_{max} = 7$, ancilla qubits are only reused across assertions, while each assertion maps one ancilla to exactly one data qubit. The depth of this circuit corresponds with the one observed for

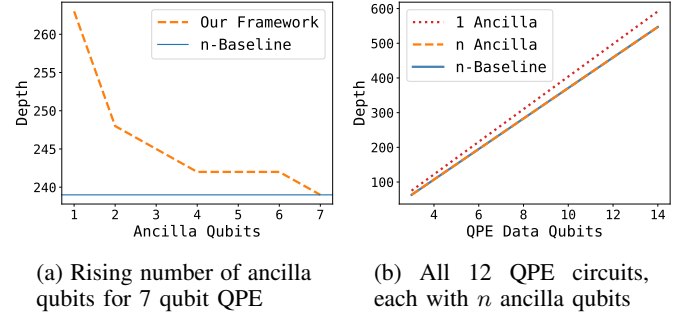


Fig. 5: Influence of the number of ancilla qubits on the depth

the multi-ancilla baseline. Even for $a_{max} = 21$, which is the number of ancilla qubits used by the multi-ancilla baseline, the depth of the circuit generated by our framework would remain unchanged. This is because, as discussed above, additional ancilla qubits beyond the $a_{max} = n$ threshold can improve neither the local depth of a single assertion, nor the global depth of multiple assertions. With this optimal number of ancilla qubits, our framework always generates circuits with the same depth as the multi-ancilla baseline. Fig. 5b shows this comparison for all 12 QPE circuits. The depth for our framework with one ancilla qubit, as shown in Fig. 4b, is added as reference. As can be seen in the graph, our framework can keep up with the multi-ancilla baseline in terms of circuit depth by using only n ancilla, i.e. $2n$ fewer than the baseline. However, even with $< n$ ancilla qubits, the circuit depth remains comparable. It is worth noting that, as seen in Fig. 5a, the biggest change occurs when going from one ancilla qubit to two. For the seven qubit QPE, the longest path could overall be shortened by 24 gates, of which this one additional ancilla qubit already accounts for 15. This suggests that, even with a suboptimal number of ancilla qubits, our framework can use these to drastically improve the circuit performance.

C) Fault Localization: To assess the effectiveness of our approach for bug localization, we consider two of the most common types of faults [4], wrong gate usage and wrong matrix computation. In our evaluation, wrong gate usage is represented by switching control and target of an CX gate, and by omitting H or X gates. Wrong matrix computations are represented by wrong phase arguments to the P gates. For

TABLE I: Fault localisation for QPE with 7 data qubits

Fault Type	Number of Faults	Identified	Missed	Success
CX controls	41	41	0	100.00%
H / X gate	15	15	0	100.00%
P gate argument	60	38	23	63.33%
	116	94	23	81.03%

these two types, we consider all 116 possible error locations in the QPE circuit with seven data qubits. We examine one fault at a time by inserting the assertions into the faulty circuit, and simulating this circuit. Table I shows the results. It lists the fault types, the number of faults inserted into the QPE circuit for each type, and how many of these inserted faults were and were not identified, both in absolute numbers and in percent. Our framework succeeds in finding an overall 81.03% of the faults. The wrong gate usage is always identified. The wrong phase gate argument can only be identified if it has a measurable effect on the result of the circuit. Phase-sensitive assertions that can detect state differences up to a phase would be an interesting endeavor for future work.

V. CONCLUSION

In this paper, we propose a framework for efficiently interpreting and implementing run-time assertions in dynamic quantum circuits. This consists of (1) a dynamic assertion interpretation strategy that leverages auxiliary qubits and classically conditioned gates, and (2) a reuse strategy for the auxiliary qubits. We evaluate our approach using the QPE algorithm as a representative case study, and show that our framework uses up to 41 qubits less than existing methods, and up to 599.06% fewer gates. Our qubit reuse strategy proved effective in minimizing the circuit depth under user-defined constraints, and over half of the possible improvement manifested with just 2 auxiliary qubits. In a fault localization study, our approach identified 81.03% of the bugs.

ACKNOWLEDGMENTS

This work was partly funded by the German Federal Ministry of Research, Technology and Space (BMFT) through the EASEPROFIT (grant no. 16KIS2127), EQeS (grant no. 13N17378), and ExaVerse (grant no. 01IW25003) projects, and by the German Research Foundation (DFG) through the CONAD-QC project (grant no. 559888852). This research was conducted within the scope of the DFG Priority Programme 2514: Quantum Software, Algorithms and Systems.

REFERENCES

- [1] S. Ali, T. Yue, and R. Abreu, “When Software Engineering Meets Quantum Computing,” *Commun. ACM*, vol. 65, no. 4, p. 84–88, Mar. 2022.
- [2] B. Weder, J. Barzen, F. Leymann, M. Salm, and D. Vietz, “The Quantum Software Lifecycle,” in *Proceedings of the 1st ACM SIGSOFT International Workshop on Architectures and Paradigms for Engineering Quantum Software*, 2020, p. 2–9.
- [3] N. C. Leite Ramalho, H. Amario de Souza, and M. Lordello Chaim, “Testing and Debugging Quantum Programs: The Road to 2030,” *ACM Trans. Softw. Eng. Methodol.*, vol. 34, no. 5, pp. 155:1–155:46, May 2025.
- [4] E. Rosa and R. Santiago, “Schrödinger’s Bug: A Survey on Quantum Software Debugging,” *Quantum Information Processing*, vol. 25, no. 3, p. 96, Mar. 2026.
- [5] L. Burgholzer and R. Wille, “Advanced Equivalence Checking for Quantum Circuits,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 9, pp. 1810–1824, 2021.
- [6] M. Amy, “Towards Large-scale Functional Verification of Universal Quantum Circuits,” *arXiv preprint arXiv:1805.06908[quant-ph]*, 2019.
- [7] J. Liu, B. Zhan, S. Wang, S. Ying, T. Liu *et al.*, “Formal Verification of Quantum Algorithms Using Quantum Hoare Logic,” in *Computer Aided Verification*, 2019, pp. 187–207.
- [8] Y. Huang and M. Martonosi, “QDB: From Quantum Algorithms Towards Correct Quantum Programs,” *arXiv preprint arXiv:1811.05447 [cs.PL]*, 2018.
- [9] M. Paltenghi and M. Pradel, “Bugs in Quantum Computing Platforms: an Empirical Study,” *Proc. ACM Program. Lang.*, vol. 6, no. OOPSLA1, Apr. 2022.
- [10] N. Quetschlich and O. Di Matteo, “An Experience-Based Classification of Quantum Bugs in Quantum Software,” *Computing*, vol. 107, no. 10, p. 193, Oct. 2025.
- [11] S. A. Metwalli and R. Van Meter, “Testing and Debugging Quantum Circuits,” *IEEE Transactions on Quantum Engineering*, vol. 5, pp. 1–15, 2024.
- [12] Y. Huang and M. Martonosi, “Statistical Assertions For Validating Patterns and Finding Bugs in Quantum Programs,” in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019, p. 541–553.
- [13] G. Li, L. Zhou, N. Yu, Y. Ding, M. Ying, and Y. Xie, “Projection-Based Runtime Assertions for Testing and Debugging Quantum Programs,” *Proc. ACM Program. Lang.*, vol. 4, no. OOPSLA, pp. 150:1–150:29, Nov. 2020.
- [14] J. Liu, G. T. Byrd, and H. Zhou, “Quantum Circuits for Dynamic Runtime Assertions in Quantum Computation,” in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, p. 1017–1030.
- [15] J. Liu and H. Zhou, “Systematic Approaches for Precise and Approximate Quantum State Runtime Assertion,” in *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021, pp. 179–193.
- [16] M. Nielsen and I. Chuang, *Quantum Computation and Quantum Information*. Cambridge Univ. Press, Oct 2000.
- [17] J. Campos and A. Souto, “Q Bugs: A Collection of Reproducible Bugs in Quantum Algorithms and a Supporting Infrastructure to Enable Controlled Quantum Software Testing and Debugging Experiments,” in *IEEE/ACM 2nd International Workshop on Quantum Software Engineering (Q-SE)*, 2021, pp. 28–32.
- [18] P. Zhao, Z. Miao, S. Lan, and J. Zhao, “Bugs4Q: A Benchmark of Existing Bugs to Enable Controlled Testing and Debugging Studies for Quantum Programs,” *Journal of Systems and Software*, vol. 205, p. 111805, Nov. 2023.
- [19] J. Li, E. Rubinshtein, and M. Martonosi, “Statistical Assertions for Debugging Quantum Circuits and States in CUDA-Q,” in *IEEE International Conference on Quantum Computing and Engineering (QCE)*, vol. 01, 2025, pp. 2074–2081.
- [20] N. H. Oldfield, C. Laaber, and S. Ali, “Bloch Vector Assertions for Debugging Quantum Programs,” *arXiv preprint arXiv:2506.18458[quant-ph]*, 2025.
- [21] P. Li, J. Liu, Y. Li, and H. Zhou, “Exploiting Quantum Assertions for Error Mitigation and Quantum Program Debugging,” in *IEEE 40th International Conference on Computer Design (ICCD)*, 2022, pp. 124–131.
- [22] D. Rovara, L. Burgholzer, and R. Wille, “A Framework for the Efficient Evaluation of Runtime Assertions on Quantum Computers,” *arXiv preprint arXiv:2505.03885[quant-ph]*, 2025.
- [23] J. Veganzones, D. Alamo, A. Cuzzocrea, and P. Bringas, “AssertsQ: A Quantum Assertion Tool for Quantum Software Debugging,” in *Proceedings of the 1st International Conference on Quantum Software*. SCITEPRESS - Science and Technology Publications, 2025, pp. 49–60.
- [24] A. Javadi-Abhari, M. Treinish, K. Krsulich, C. J. Wood *et al.*, “Quantum computing with Qiskit,” *arXiv preprint arXiv:2405.08810 [quant-ph]*, 2024.