Quantum Assertion Testing Without Mid-Circuit Measurement: Strategies and Lower Bounds

SHENGYUAN YANG and CHARLES YUAN, University of Wisconsin–Madison, USA

1 Introduction

Quantum computation promises powerful advantages for tasks, including factoring, search, and simulation, that are intractable for classical computers. Recent advances in quantum hardware and error correction [2, 4] have raised prospects of realizing this advantage in the near future. However, significant challenges still prevent quantum computing from becoming practical and ubiquitous. One problem is that quantum programs are hard to debug — learning about the state of a quantum computation requires measurement, which could perturb or destroy the state.

To tackle this problem of detecting bugs in quantum programs, researchers have developed sound methods for quantum assertion testing [17, 20–22]. Prior work has studied the expressiveness of individual assertions and their construction as quantum circuits, but a complementary question has been less explored: *how to efficiently check multiple assertions in a quantum program.*

Motivating Example. Consider the program in Fig. 1, which takes as input a qubit x and a classical bit c. It initializes the fresh qubits y and z to $|0\rangle$ and executes a series of quantum logic gates on the states of all three qubits before finally measuring the value of y and z. To test correctness, the program invokes three assertions A_1, A_2, and A_3 about the relationships between x, y, and z.

Critically, the program must test A_1 without disturbing the program state seen by A_3. Thus, it could not simply measure x and y and collapse their superposition. Instead, it tests each assertion A_i whose predicate is the function f_i by invoking a unitary oracle O_i that coherently evaluates f_i into an ancilla qubit:

$$O_i : \sum_x |x\rangle_{\text{prog}} |0\rangle_{\text{anc}} \mapsto \sum_x |x\rangle_{\text{prog}} |f_i(x)\rangle_{\text{anc}},$$

where x denotes the subset of program qubits involved in A_i.

In the example, O_1 performs two CNOT gates targeting the same ancilla controlled by x and y respectively. O_2 and O_3 each perform a CNOT controlled by z or y, each targeting a fresh ancilla.

```
1  x,y := CNOT(x,y);
2  assert x == y;   A1
3  z := H(z);
4  x,z := CNOT(x,z);
5  z := H(z);
6  assert z == 0    A2
7  x,y := CNOT(x,y);
8  y := H(y);
9  if (c) { y := Z(y) }
10 y := H(y)
11 assert y == 1;   A3
12 y,z := CNOT(y,z)
13 r1 := measure(y);
14 r2 := measure(z);
```

Fig. 1. A quantum program with three assertions. Input qubit x is arbitrary; y and z are initialized to $|0\rangle$; c is a classical input bit.

Efficiency. Using three ancillae – one per assertion – and a terminal measurement on them, the program can obtain all assertion outcomes in one run. This strategy generalizes to a *single-run/n-ancillae strategy* for programs with n assertions. An alternative is a *single-ancilla/n-runs* strategy that executes the example program three times. Each run enables one assertion, uses one ancilla, and measures it at the end to obtain the outcome of that assertion.

These two baseline strategies – *single-run/n-ancillae* and *single-ancilla/n-runs* – both achieve a total time-space complexity of $T \times S = O(n)$, where T denotes the number of executions and S the number of qubits used per execution. It is natural to ask whether there exists a strategy that achieves a better overall complexity or different trade-offs between space and time.

For example, if mid-circuit measurement and reset are possible, one can employ a *eagerly-measure-and-reuse* strategy that uses a single run and one ancilla qubit: measure the ancilla for assertion

Authors' Contact Information: Shengyuan Yang; Charles Yuan, University of Wisconsin–Madison, USA, syang686@wisc.edu,charlesyuan@cs.wisc.edu.

A_i and then reset it to $|0\rangle$ for reuse by later assertions. Unfortunately, mid-circuit measurements are slow and relatively difficult to support in current hardware due to latency [7, 8, 24] and noise sensitivity [12–15]. Improving mid-circuit measurement remains a focus in hardware research [18], whereas reducing its occurrence has become a focus in program optimization research [5, 6].

Contributions. In this work, we investigate time-space trade-offs for quantum assertion testing in the setting without mid-circuit measurement. To begin, we formalize the possible levels of learnable information about a quantum program that uses multiple assertions, including deciding whether any assertion fails (ExistFail) or outputting the index of the first failing assertion (FirstFail).

Our first main contribution is a *single-run, $O(\log n)$-ancilla* strategy that solves the FirstFail problem, improving over $O(n)$ from the two naive baselines. This result is tight; we prove a lower bound stating that the total complexity for FirstFail satisfies $T \times S = \Omega(\log n)$. Moreover, our result implies that the *eagerly-measure-and-reuse* strategy does not yield $O(n)$ total advantage over strategies without mid-circuit measurement. This advantage is, surprisingly, only $O(\log n)$.

In the near term, these contributions improve our ability to test quantum programs on hardware platforms where mid-circuit measurement is costly or unsupported. More broadly, our work initiates the study of practical time-space tradeoffs for quantum program testing and paves way toward testing more general assertions and more complex programs that will be needed for applications.

2 Related Work

Prior work has extensively studied assertions in quantum programming from the angles of expressiveness and realizability [17, 20–22]. For example, the *statistical assertions* introduced by Ref. [17] measure repeated executions of a program to test properties such as whether a qubit takes a specific value or whether two qubits are entangled. This idea generalizes to *runtime assertion circuits* [21], which write each assertion outcome into an ancilla and measure this ancilla rather than destructively measuring the program state. *Approximate assertions* [22] further extend assertions with support for checking arbitrary pure states and some mixed states, including set membership tests. Finally, *projection-based runtime assertions* [20] provide subspace-membership assertions with theoretical guarantees, subsuming the types of assertions described earlier [17, 21].

Prior work has discussed programs with multiple assertions but has not extensively studied their time–space trade-offs. Typically, prior work handles multiple assertions by either (i) performing mid-circuit measurement [20], or (ii) effectively performing either the *single-ancilla/n-runs* approach as in Ref. [17] or the *single-run/n-ancillae* approach as in Refs. [21, 22]. By contrast, we study the time-space trade-offs of checking multiple assertions under practical constraints such as lack of mid-circuit measurement. This question is orthogonal to expressiveness of assertions. Our study thus applies to, and increases the applicability of, existing frameworks for assertion testing.

In addition to assertions, researchers have adapted classical techniques for software testing to quantum programs that cover multiple aspects of test design and evaluation [3, 9–11, 16, 19, 23, 25–28]. These techniques pose further opportunities to study efficiency as we do in this work.

3 Formalizing Problems in Quantum Assertion Testing

In this section, we formalize the concept of quantum assertion testing by specifying (i) a program model, (ii) a family of quantum assertion-testing problems for programs with multiple assertions, and (iii) an instrumentation and strategy model for checking them.

Program Model. Let P be a quantum program acting on a *program register* $|\rangle_{\text{prog}}$. We adapt the Q-While language defined in Ref. [29], with the syntax for commands:

$$S ::= \textbf{skip} \mid (c_1, ..., c_n) := (t_1, ..., t_n) \mid (q_1, ..., q_n) := U(q_1, ..., q_n) \mid c := \textbf{measure}(q)$$
$$\mid S1; S2 \mid \textbf{if } \varphi \textbf{ then } S_1 \textbf{ else } S2 \mid \textbf{while } \varphi \textbf{ do } S \textbf{ end}$$

Here, c, t are classical variables/terms, q is a qubit, U is a unitary operation, and **measure** denotes a measurement returning a classical outcome. The guards φ in **if** and **while** are classical.

Assertion. We consider n program points for assertions, indexed by $[n] \triangleq \{1, 2, ..., n\}$. For each $i \in [n]$, assertion A_i is a predicate on the instantaneous state of the program qubits in prog involved in A_i. We write the set of all assertions $\mathcal{A} \triangleq \{A_i \mid i \in [n]\}$, and abstract the semantics of each A_i by a function $f_i : \mathcal{H}_{\text{prog}} \to \mathbb{B}$ that produces the conceptual classical outcome of the assertion: $f_i(x) = 0$ denotes *pass* and $f_i(x) = 1$ denotes *fail*[1]. Collectively, these outcomes define the *assertion outcome vector* $F \in \{0,1\}^n$ with $F_i \triangleq f_i(x^{(i)})$, where $x^{(i)}$ is the state of prog immediately before A_i.

Problem Family. We identify the following problems of interest: (1) ExistFail: Decide whether $\exists\, i \in [n]$ where $F_i = 1$; (2) FindOne: Output any index i such that $F_i = 1$, or output $\perp$ if none. (3) FirstFail: Output $\min\{i \mid F_i = 1\}$, or output $\perp$ if none. (4) TopK: Output the k smallest indices in $\{i \mid F_i = 1\}$, or all that exist if fewer than k. (5) Count: Output $c = \sum_{i=1}^{n} F_i$, i.e., the number of failing assertions. (6) ListAll: Output the full set $\{i \mid F_i = 1\}$.

Instrumentation. Given P and $\mathcal{A}$, an *instrumentation* $\mathcal{I}$ specifies a *mask* $\mathcal{M}$ — a subset of $[n]$ indicating which assertions are enabled — and an execution $\mathcal{I}(P)$ that checks assertions:

(i) *Initialization.* The program register $|\rangle_{\text{prog}}$ is initialized to P's standard input state, along with an additional ancilla pool $|\rangle_{\text{anc}}$ with size s initialized to all $|0\rangle$.

(ii) *Oracle exposure and processing.* When $\mathcal{I}(P)$ reaches the program point of A_i for $i \in \mathcal{M}$, it invokes the unitary oracle O_i to access the assertion predicate f_i:

$$O_i : \sum_x |x\rangle_{\text{prog}} |c\rangle_\alpha \mapsto \sum_x |x\rangle_{\text{prog}} |c \oplus f_i(x)\rangle_\alpha \tag{1}$$

which computes $f_i(x) \in \{0,1\}$ into a designated qubit α in register anc. After oracle invocations, $\mathcal{I}(P)$ may apply arbitrary unitaries U_i acting on anc, with the constraint that U_i preserves the measurement distribution of prog. In formal terms, it requires that for all joint states $\rho_{\text{prog,anc}}$, it holds that (ρ for density matrix, I for identity matrix):

$$\text{Tr}_{\text{anc}} \left[(I \otimes U_i)\, \rho_{\text{prog,anc}}\, (I \otimes U_i)^\dagger \right] = \text{Tr}_{\text{anc}}(\rho_{\text{prog,anc}})$$

(iii) *Terminal measurement.* At the end of execution after $\mathcal{I}(P)$ finishes the computation on prog, it measures all ancillae in the computational basis, producing a classical bit string $r \in \{0,1\}^s$.

Strategy. Given program P and assertions $\mathcal{A}$, a strategy $\mathcal{S}$ specifies t instrumentations $(\mathcal{I}_1, ..., \mathcal{I}_t)$ to check $\mathcal{A}$ within t rounds of program execution. At the k+1-th round, based on the history $H_k \triangleq \big((\mathcal{I}_1, r_1), ..., (\mathcal{I}_k, r_k)\big)$, the strategy selects the next instrumentation $\mathcal{I}_{k+1} = \mathcal{S}(H_k)$. After t rounds of execution, $\mathcal{S}$ performs classical post-processing V on $(r_1, r_2, ..., r_t)$ to output an answer to a target problem. We evaluate $\mathcal{S}$ by four worst-case costs over all histories it may realize:

- *Time (runs)*: $T(\mathcal{S}) \triangleq$ # of instrumented executions of P.
- *Space (ancillae)*: $S(\mathcal{S}) \triangleq \max_t\{\#\text{ancillae allocated by } \mathcal{I}_t\}$.
- *Gate (operations)*: $G(\mathcal{S}) \triangleq$ total # additional quantum gates used over all $\mathcal{I}_t$.
- *Oracle calls*: $\mathcal{O}(\mathcal{S}) \triangleq$ total # oracle calls over all $\mathcal{I}_t$.

For example, the baseline *single-run/n-ancillae* strategy consists of a single instrumentation $\mathcal{I}_1$ with $\mathcal{M}_1 = [n]$ and n fresh ancillas in anc. At each assertion point A_i, the oracle O_i writes $f_i(x)$ into the i-th qubit of anc; a terminal measurement of anc reveals all F_i. This yields $T = 1$ and $S = n$.

[1]In this work, we focus on assertions whose runtime outcomes are *classical*, i.e., *deterministic* (e.g., A_1 and A_2 in Fig. 1) or *classically probabilistic* (e.g., A_3 in Fig. 1), which simplifies the formalism and facilitates reasoning. A more general assertion may be in a superposition of passing and failing. To cover such cases, we can define an assertion as failing if the state has any amplitude on the failure subspace, and assume efficient sampling to detect failure with high probability.

Meanwhile, the *single-ancilla/n-runs* baseline comprises the instrumentations $(\mathcal{I}_1, \ldots, \mathcal{I}_n)$, where $\mathcal{I}_k$ sets $\mathcal{M}_k = \{k\}$ and allocates a single ancilla. Run k invokes O_k at A_k, writes $f_k(x)$ into the ancilla, and measures it at termination to reveal F_k. This yields $T = n$ and $S = 1$.

Both baselines invoke the oracles n times and perform no additional processing beyond oracle calls, so $G = 0$ and $\mathcal{O} = n$ for both. Each baseline directly solves ListAll and, with suitable classical post-processing V, also solves the other five problems.

4 A Single-Run, $O(\log N)$-Ancillae Strategy for FirstFail

We present a strategy that solves FirstFail using $\lceil \log_2(n+1) \rceil + 1$ ancillae in a single instrumented run. The core idea is to maintain an m-qubit *index register* idx (with $m = \lceil \log_2(n+1) \rceil$), together with a single *fail flag* fail, all initialized to zero. At each assertion A_i, the run performs:

(1) *Oracle invocation.* Invoke O_i (c.f. unitary 1) to compute the outcome of A_i into fail. Here fail is always fresh before each O_i's invocation, as it will be reset to $|0\rangle$ by step (3).

(2) *Index update.* Apply a unitary U_i on idx, controlled on fail $= 1$, which swaps the two basis states $|0^m\rangle$ and $|\mathrm{bin}(i)\rangle$ of idx (where $\mathrm{bin}(i)$ is the m-bit binary encoding of i). Thus, if A_i fails and idx $= |0^m\rangle$, it updates to $|\mathrm{bin}(i)\rangle$. For later failing assertions A_j with $j > i$, since idx $\neq |0^m\rangle$, U_j acts as the identity. To build the quantum circuits of U_i, fix a *Gray path* from 0^m to $z = \mathrm{bin}(i) \in \{0,1\}^m$ of length $\ell = \mathrm{wt}(z)$ (Hamming weight), flipping bits 0 to 1 from *right to left*:

$$0^m = v_0^{(i)} \to v_1^{(i)} \to \cdots \to v_\ell^{(i)} = z.$$

Then define the palindromic gate sequence[2]:

$$U_i \;=\; G_1^{(i)} G_2^{(i)} \cdots G_{\ell-1}^{(i)} G_\ell^{(i)} G_{\ell-1}^{(i)} \cdots G_2^{(i)} G_1^{(i)},$$

where each $G_k^{(i)}$ is a multi-controlled NOT on the unique target bit that differs between $v_{k-1}^{(i)}$ and $v_k^{(i)}$, with controls on (i) fail $= 1$ and (ii) all non-target bits of idx fixed to match $v_{k-1}^{(i)}$[3].

(3) *Uncomputation.* Apply O_i again to uncompute fail to $|0\rangle$, for reuse at the next assertion.

At the end of execution, idx $= |\mathrm{bin}(i^*)\rangle$, where i^* is the smallest failing index, or $|0^m\rangle$ if none fail.

Example. Consider $n = 10$ and $F = 0010001010$ (i.e., A_3, A_7, and A_9 fail). We use $\lceil \log_2(n+1) \rceil + 1 = 5$ ancillae: one for fail and four for idx. For $i \notin \{3, 7, 9\}$, fail $= |0\rangle$ so U_i is inactive. For $i \in \{3, 7, 9\}$, the Gray paths and corresponding circuits of U_i appear in Fig. 2.

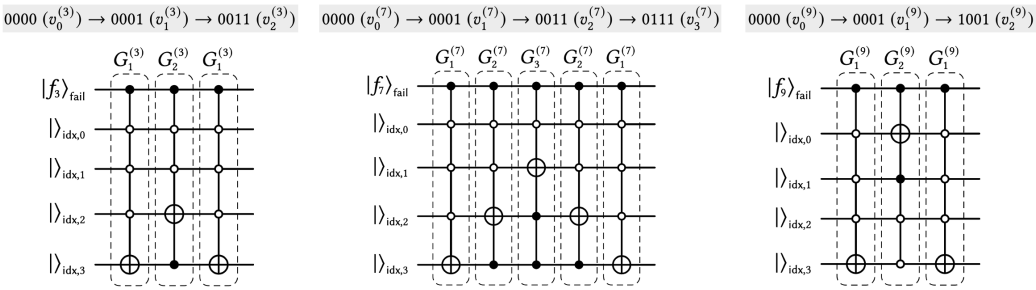

Fig. 2. Gray path and corresponding quantum circuits of U_i, for $i = 3$ (left), $i = 7$ (middle) and $i = 9$ (right).

Starting from idx $= |0000\rangle$, at A_3, the first two gates of U_3 fire and advance idx to $|0011\rangle$. At A_7, with idx $= |0011\rangle$ (which equals $v_2^{(7)}$), the fired gates are the two $G_2^{(7)}$—the first flipping $|0011\rangle$ to $|0001\rangle$, and the second $G_2^{(7)}$ restoring $|0001\rangle$ to $|0011\rangle$—thus U_7's net effect is identity on $|0011\rangle$. At A_9, since

[2]Similar constructions have been developed for unitary synthesis to compute a two-level unitary matrix that acts nontrivially on two or fewer vector components [1] .

[3]$G_k^{(i)}$ implements the controlled transposition between the basis states $v_{k-1}^{(i)}$ and $v_k^{(i)}$, and acts as the identity elsewhere.

idx = $|0011\rangle$ does not match any node on the $i = 9$ Gray path, no gate in U_9 fires[4]. At the end of the run, measuring idx gives $0011 = \mathrm{bin}(3)$, identifying A_3 as the first failing assertion correctly.

Cost. The time and space cost is $T = 1$ and $S = \Theta(\log n)$. The oracle unitary O_i is called twice for each i, hence $\mathcal{O} = 2n$. The total gates used over all U_i is computed as:

$$G = \sum_{i=1}^{n} \big(2\,\ell(i) - 1\big), \quad \text{where } \ell(i) = \mathrm{wt}\big(\mathrm{bin}(i)\big)$$

A convenient worst-case upper bound is (as if all nonzero m-bit strings occurred in $\{\mathrm{bin}(i) \mid i \in [n]\}$):

$$G \le \sum_{\ell=1}^{m} \binom{m}{\ell} (2\ell - 1) = (m-1)\cdot 2^m + 1 = \Theta(m \cdot 2^m) = \Theta(n \log n).$$

5 Lower Bounds for FirstFail

Theorem 5.1 (Lower bound for time–space product). *For any program P with n assertions, every strategy $\mathcal{S}$ that solves* FirstFail *satisfies $S \times T = \Omega(\log n)$.*

Proof. There are n+1 possible answers to FirstFail ($\perp$ or one of n assertions). Across T rounds, at most S fresh classical bits are revealed per round, so $\mathcal{S}$ produces a transcript $\mathrm{tr} \in \{0,1\}^{S\times T}$ upon finishing. Zero-error correctness in the worst case requires that distinct answers induce distinct transcripts, hence $|\{0,1\}^{S\times T}| \ge n$+1, yielding $2^{S\times T} \ge n + 1$, which implies $S \times T \ge \log(n + 1)$. □

Theorem 5.2 (Lower bound tightness). *The lower bound $S \times T = \Omega(\log(n))$ is tight.*

Proof. Sec. 4 gives a strategy to solve FirstFail using $\lceil \log(n + 1) \rceil + 1$ ancillae in one run. □

Theorem 5.3 (Gate cost lower bound). *Every strategy that solves* FirstFail *with $S \times T = \Theta(\log n)$ has a gate cost of $G = \Omega(n \log n)$, i.e. it uses additional $\Omega(n \log n)$ non-oracle gates.*

Proof sketch. Under $S \times T = \Theta(\log n)$, any strategy $\mathcal{S}$ that solves FirstFail effectively encodes the answer in the transcript $\mathrm{tr} \in \{0,1\}^{S\times T}$ it produces. This encoding necessarily requires flipping ancillae bits to produce distinct classical bit-strings. Because flipping one bit requires at least one gate, correctness for all $n + 1$ possible outcomes demands $\Omega((n + 1) \times (S \times T))$ total bit flips across the circuits, which implies $G = \Omega(n \cdot (S \times T)) = \Omega(n \log n)$. □

Conjecture 5.4 (Fixed-scheme time-space lower bound). *Any strategy that is* fixed-scheme*, i.e. uses the same non-oracle gate pattern at every assertion point, solves* FirstFail *in time $S \times T = \Omega(n)$.*

Proof Sketch. By definition, a fixed scheme applies the same unitary U at every assertion point. Correctness for FirstFail requires U to effect a different transformation when encountering the first failure as opposed to any later failure, but injectivity forbids collapsing distinct input states to the same output state. Thus, U must leave a persistent witness state of the first update that survives all subsequent assertions unless the strategy allows linear space cost. But leaving such a witness forces the use of $\Omega(1)$ new ancilla per enabled assertion within a run. Therefore, covering all n assertions across T runs still yields an overall complexity of $S \times T = \Omega(n)$. □

6 Conclusion and Future Work

In this work, we initiated the study of time-space trade-offs for checking multiple assertions in quantum programs without mid-circuit measurements. For FirstFail, we presented a *single-run, $O(\log n)$-ancillae* strategy and proved a tight bound $T \times S = \Theta(\log n)$, thereby showing that the effective advantage of mid-circuit measurement is at most $O(\log n)$, not $O(n)$.

Looking ahead, we plan to (i) develop strategies and lower bounds for the other problems defined in Section 3, and (ii) extend our framework to assertions whose outcomes may be in quantum superposition, where efficient sampling is required to provide high-probability guarantees.

[4]Note that idx remains both unchanged for $i = 7$ and $i = 9$, but with different reasons: for $i = 7$, *two gates fire but cancel* within the palindromic gate sequence; for $i = 9$, *no gate fires* because the current idx is not on that Gray path.

References

[1] Alfred V. Aho and Krysta M. Svore. 2003. Compiling Quantum Circuits using the Palindrome Transform. arXiv:quant-ph/0311008 [quant-ph] https://arxiv.org/abs/quant-ph/0311008

[2] Google Quantum AI. 2025. Quantum error correction below the surface code threshold. *Nature* 638, 8052 (2025), 920–926.

[3] Shaukat Ali, Paolo Arcaini, Xinyi Wang, and Tao Yue. 2021. Assessing the Effectiveness of Input and Output Coverage Criteria for Testing Quantum Programs. In *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*. 13–23. doi:10.1109/ICST49551.2021.00014

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

[5] Yanbin Chen, Innocenzo Fulginiti, and Christian B. Mendl. 2024. Reducing Mid-Circuit Measurements via Probabilistic Circuits . In *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE Computer Society, Los Alamitos, CA, USA, 952–958. doi:10.1109/QCE60285.2024.00114

[6] Yanbin Chen, Innocenzo Fulginiti, and Christian B. Mendl. 2025. Optimization Framework for Reducing Mid-circuit Measurements and Resets. In *Computational Science – ICCS 2025 Workshops: 25th International Conference, Singapore, Singapore, July 7–9, 2025, Proceedings, Part V* (Singapore, Singapore). Springer-Verlag, Berlin, Heidelberg, 150–164. doi:10.1007/978-3-031-97570-7_13

[7] A. D. Córcoles, Maika Takita, Ken Inoue, Scott Lekuch, Zlatko K. Minev, Jerry M. Chow, and Jay M. Gambetta. 2021. Exploiting Dynamic Quantum Circuits in a Quantum Algorithm with Superconducting Qubits. *Phys. Rev. Lett.* 127 (Aug 2021), 100501. Issue 10. doi:10.1103/PhysRevLett.127.100501

[8] Lior Ella, Lorenzo Leandro, Oded Wertheim, Yoav Romach, Lukas Schlipf, Ramon Szmuk, Yoel Knol, Nissim Ofek, Itamar Sivan, and Yonatan Cohen. 2023. Quantum-classical processing and benchmarking at the pulse-level. *arXiv preprint arXiv:2303.03816* (2023).

[9] Daniel Fortunato, José Campos, and Rui Abreu. 2022. Mutation testing of quantum programs written in QISKit. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings* (Pittsburgh, Pennsylvania) *(ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 358–359. doi:10.1145/3510454.3528649

[10] Daniel Fortunato, José Campos, and Rui Abreu. 2022. QMutPy: a mutation testing tool for Quantum algorithms and applications in Qiskit. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis* (Virtual, South Korea) *(ISSTA 2022)*. Association for Computing Machinery, New York, NY, USA, 797–800. doi:10.1145/3533767.3543296

[11] Daniel Fortunato, José Campos, and Rui Abreu. 2024. Gate Branch Coverage: A Metric for Quantum Software Testing. In *Proceedings of the 1st ACM International Workshop on Quantum Software Engineering: The Next Evolution* (Porto de Galinhas, Brazil) *(QSE-NE 2024)*. Association for Computing Machinery, New York, NY, USA, 15–18. doi:10.1145/3663531.3664753

[12] J. P. Gaebler, C. H. Baldwin, S. A. Moses, J. M. Dreiling, C. Figgatt, M. Foss-Feig, D. Hayes, and J. M. Pino. 2021. Suppression of midcircuit measurement crosstalk errors with micromotion. *Phys. Rev. A* 104 (Dec 2021), 062440. Issue 6. doi:10.1103/PhysRevA.104.062440

[13] György P Gehér, Marcin Jastrzebski, Earl T Campbell, and Ophelia Crawford. 2025. To reset, or not to reset—that is the question. *npj Quantum Information* 11, 1 (2025), 39.

[14] Akel Hashim, Arnaud Carignan-Dugas, Larry Chen, Christian Jünger, Neelay Fruitwala, Yilun Xu, Gang Huang, Joel J. Wallman, and Irfan Siddiqi. 2025. Quasiprobabilistic Readout Correction of Midcircuit Measurements for Adaptive Feedback via Measurement Randomized Compiling. *PRX Quantum* 6 (Jan 2025), 010307. Issue 1. doi:10.1103/PRXQuantum.6.010307

[15] Johannes Heinsoo, Christian Kraglund Andersen, Ants Remm, Sebastian Krinner, Theodore Walter, Yves Salathé, Simone Gasparinetti, Jean-Claude Besse, Anton Potočnik, Andreas Wallraff, and Christopher Eichler. 2018. Rapid High-fidelity Multiplexed Readout of Superconducting Qubits. *Phys. Rev. Appl.* 10 (Sep 2018), 034040. Issue 3. doi:10.1103/PhysRevApplied.10.034040

[16] 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* (Seoul, Republic of Korea) *(ICSEW'20)*. Association for Computing Machinery, New York, NY, USA, 430–435. doi:10.1145/3387940.3391459

[17] Yipeng Huang and Margaret Martonosi. 2019. Statistical assertions for validating patterns and finding bugs in quantum programs. In *Proceedings of the 46th International Symposium on Computer Architecture* (Phoenix, Arizona) *(ISCA '19)*. Association for Computing Machinery, New York, NY, USA, 541–553. doi:10.1145/3307650.3322213

[18] IBM. 2025. Classical feedforward and control flow. https://quantum.cloud.ibm.com/docs/en/guides/classical-feedforward-and-control-flow.

[19] Neilson Carlos Leite Ramalho, Higor Amario de Souza, and Marcos Lordello Chaim. 2025. Testing and Debugging Quantum Programs: The Road to 2030. *ACM Trans. Softw. Eng. Methodol.* 34, 5, Article 155 (May 2025), 46 pages. doi:10.1145/3715106

[20] Gushu Li, Li Zhou, Nengkun Yu, Yufei Ding, Mingsheng Ying, and Yuan Xie. 2020. Projection-based runtime assertions for testing and debugging Quantum programs. *Proc. ACM Program. Lang.* 4, OOPSLA, Article 150 (Nov. 2020), 29 pages. doi:10.1145/3428218

[21] Ji Liu, Gregory T. Byrd, and Huiyang Zhou. 2020. Quantum Circuits for Dynamic Runtime Assertions in Quantum Computation *(ASPLOS '20)*. Association for Computing Machinery, New York, NY, USA, 1017–1030. doi:10.1145/3373376.3378488

[22] 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 Computer Society, Los Alamitos, CA, USA, 179–193. doi:10.1109/HPCA51647.2021.00025

[23] Peixun Long and Jianjun Zhao. 2024. Testing Multi-Subroutine Quantum Programs: From Unit Testing to Integration Testing. *ACM Trans. Softw. Eng. Methodol.* 33, 6, Article 147 (June 2024), 61 pages. doi:10.1145/3656339

[24] Thomas Lubinski, Cassandra Granade, Amos Anderson, Alan Geller, Martin Roetteler, Andrei Petrenko, and Bettina Heim. 2022. Advancing hybrid quantum–classical computation with real-time execution. *Frontiers in Physics* 10 (2022), 940293.

[25] Eñaut Mendiluze, Shaukat Ali, Paolo Arcaini, and Tao Yue. 2022. Muskit: a mutation analysis tool for quantum software testing. In *Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering* (Melbourne, Australia) *(ASE '21)*. IEEE Press, 1266–1270. doi:10.1109/ASE51524.2021.9678563

[26] Jiyuan Wang, Fucheng Ma, and Yu Jiang. 2021. Poster: Fuzz Testing of Quantum Program. In *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*. 466–469. doi:10.1109/ICST49551.2021.00061

[27] Xinyi Wang, Paolo Arcaini, Tao Yue, and Shaukat Ali. 2021. Application of Combinatorial Testing to Quantum Programs. In *2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS)*. 179–188. doi:10.1109/QRS54544.2021.00029

[28] 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* (Pittsburgh, Pennsylvania) *(ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 173–177. doi:10.1145/3510454.3516839

[29] Mingsheng Ying and Yuan Feng. 2011. A Flowchart Language for Quantum Programming. *IEEE Transactions on Software Engineering* 37, 4 (2011), 466–485. doi:10.1109/TSE.2010.94