Quantum Information Processing (2026) 25:96
https://doi.org/10.1007/s11128-026-05114-3

Schrödinger's bug: a survey on quantum software debugging

Evandro Rosa[1] · **Rafael Santiago**[1]

Received: 1 August 2024 / Accepted: 16 February 2026
© The Author(s) 2026

Abstract
Quantum computing offers the potential for exponential speed-ups for classically intractable problems, yet quantum programming is still susceptible to bugs. Classical debugging methods are often inadequate, as quantum mechanical principles make state inspection disruptive and classical simulation has exponential time complexity. This survey explores the landscape of quantum assertions as a key technique for identifying and locating bugs in quantum programs. We classify these techniques into two primary categories based on their evaluation stage: classical runtime and quantum runtime assertions. For each category, we analyze the strengths, limitations, time complexity, and applicability of current methods. Our findings show that scalable quantum debugging remains an open problem—a challenge that will persist even with the advent of fault-tolerant hardware. Finally, this work highlights key challenges and proposes future directions for the development of novel quantum debugging techniques.

Keywords Quantum Computing · Quantum Programming · Debug · Assertion

1 Introduction

Quantum computing is an emerging technology that holds the promise of exponential acceleration in solving problems that are inherently difficult for classical computers. The advantage of quantum computers was initially theorized by Feynman [1] in 1981 and later substantiated by Shor [2] in 1994, notably with the quantum integer factorization algorithm. However, it was not until 2019 that the first demonstration of a quantum computer solving problems significantly faster than the current fastest supercomputer was published [3]. Subsequent experiments have further demonstrated the quantum advantage [4–7]. Despite significant advancements by key players in the

✉ Evandro Rosa
evandro.crr@posgrad.ufsc.br

Rafael Santiago
r.santiago@ufsc.br

[1] Departamento de Informatica e Estatistica, Universidade Federal de Santa Catarina, R. Eng. Agronômico Andrei Cristian Ferreira, Florianópolis 88040-900, Santa Catarina, Brazil

Published online: 05 March 2026

quantum computing space, such as IBM [8] and Google [9], current demonstrations have yet to solve problems with practical applications. Nevertheless, they mark important milestones toward the real-world engineering use of quantum computers [10, 11].

Analogous to the existence of quantum algorithms predating the full implementation of quantum computers, the development of quantum-accelerated software can be initiated even before the readiness of quantum computers for production scale. Currently, several quantum programming platforms facilitate the development of quantum software [12–14], allowing for the execution of proof of concepts on Noisy Intermediate-Scale Quantum (NISQ) computers [15] and quantum simulators [16–19]. Comparable to classical programming, quantum programming is susceptible to coding errors or flaws, commonly known as bugs. Nevertheless, the identification and rectification of bugs in quantum applications may present greater challenges compared to those encountered in classical applications.

In a study on human–computer interaction [20], researchers identified two quantum debugging approaches commonly utilized by the quantum programmers interviewed. The first method involves an interchange between different quantum representations to identify errors and code flaws. For example, this may include transitioning between a text-based description of the quantum application and a quantum circuit diagram. This process can be compared to debugging an application written in a high-level language by analyzing the assembly produced by the compiler in classical programming.

The second approach entails the use of quantum simulators to analyze the step-by-step execution of quantum algorithms. While this approach appears promising in theory, its practical utilization is constrained by the time complexity of quantum simulation. Simulating a quantum computer demands exponential execution time in relation to the number of quantum bits and generates an extensive amount of data, rendering it impractical to follow step by step. In a classical analogy, it is akin to debugging a classical application by examining the computer's memory without a clear demarcation of where a variable starts and ends. Additionally, due to entanglement, operations on one variable can induce side effects on others, further complicating the debugging process.

Although these quantum debugging approaches aid in identifying flaws within quantum programs, their scalability remains limited. This necessity drives the development of quantum debugging methods aimed at expediting the identification and correction of quantum bugs. A fundamental element in classical debugging is the use of assertions, which validate whether the value of a variable aligns with a given assumption. When evaluated successfully, assertions do not produce side effects in the classical application, although they may trigger repercussions upon failure.

In this survey, we delve into techniques for asserting the characteristics of quantum states, with the primary goal of ensuring their compliance with predefined criteria or anticipated behavior. This pursuit aims to uncover any potential anomalies or bugs within quantum systems. The methods for assertion encompass a diverse range of strategies, primarily classified into two groups: *classical runtime assertion* and *quantum runtime assertion*. The key differentiation between these lies in the timing of when the quantum assertions are assessed—either before the quantum execution (classical runtime) or during the quantum execution itself (quantum runtime).

Our survey focuses on providing a comprehensive overview of these approaches, particularly within the quantum gate model. While other computational paradigms exist, such as adiabatic [21] and photonic quantum computing [22, 23], they fall outside the primary scope of this paper. Through this exploration, we aim to shed light on their various strengths, limitations, and scalability. The ultimate goal is to establish a thorough understanding of the current state-of-the-art methods available for asserting about the quantum state. This knowledge empowers researchers and developers within the quantum computing field to navigate and apply these techniques effectively in their endeavors.

One key finding of this work is that the definition of scalable quantum debugging remains an open problem, and efficient techniques lack the expressivity required for widespread adoption in the quantum programming workflow. Additionally, the study reveals the following insights:

- Assertion-based quantum computing debugging at classical runtime can streamline and reduce the cost of quantum development. However, only two quantum assertion techniques were identified in this study, with only one showing the potential for efficient execution on classical computers.
- There are several techniques for asserting about the quantum state within a quantum computer (quantum runtime). Although these techniques are equivalent in terms of expressivity, they involve trade-offs during quantum execution. In the general case, implementing a quantum assertion takes an exponential amount of processing time on a classical computer. Nevertheless, the techniques presented in this study can provide a solid foundation for implementing efficient quantum assertions.

Related Works

To the best of our knowledge, this survey represents the first comprehensive exploration of quantum assertion techniques for quantum debugging, encompassing evaluations at both classical and quantum runtimes, alongside an in-depth analysis of their time complexities. In a related context, the work by Li et al. [24] provides a summary of several quantum runtime assertion implementations, many of which are discussed in Section 5 of this survey. Moreover, they offer a case study illustrating the application of assertions in quantum debugging and error mitigation. This survey complements their work by providing a detailed analysis of quantum runtime assertion techniques, particularly regarding their time complexities, while also introducing and evaluating classical runtime assertion techniques.

Additionally, the research conducted by Zhao [25] offers valuable insights into the broader field of quantum software engineering, including a brief overview of assertion-based debugging within quantum systems. However, their work does not provide an in-depth analysis of quantum assertion techniques, as presented in this survey. Instead, their focus extends to quantum software testing and other aspects of quality assurance in quantum software development, reflecting a broader perspective that complements the more focused scope of this work.

Document Structure

We initiate this survey with an introduction to quantum computing, encompassing foundational concepts necessary to comprehend the implementation and utilization of the presented quantum assertions in Section 2. Following this, Section 3 provides

a review of two empirical studies in quantum debugging that identified the existence of quantum-specific bugs in quantum programming platforms and applications. Our categorization of assertion techniques is bifurcated based on the stage at which the assertion is evaluated. Section 4 elucidates quantum program assertions that undergo evaluation during the classical runtime, conducted before executing the application on the quantum computer. In contrast, Section 5 reports upon assertion techniques evaluated during the quantum runtime, performed while the quantum program is executing on the quantum computer or simulator. Finally, in Section 6 we end with the conclusions and final remarks of this survey.

2 Quantum computing

In this section, we provide a summary of the basics of quantum computing. For a comprehensive introduction to quantum computing, we refer Nielsen and Chuang [26]. Readers already familiar with these concepts may wish to proceed to Section 2.7, where we address the limitations of applying classical debugging techniques to quantum computers.

The foundation of quantum computing lies in the quantum bit, referred to as a qubit. The qubit serves as the smallest unit of information that a quantum computer can store. This information is expressed as a linear combination of two distinct states, usually denoted as *zero* and *one*. In Dirac notation [27], widely used in quantum mechanics, these states are represented as $|0\rangle$ and $|1\rangle$, respectively. Any arbitrary qubit can be presented as $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$, where α and β are complex numbers referred to as probability amplitudes. When both α and β are nonzero, the qubit exists in a state of superposition. This implies that the qubit is simultaneously in both $|0\rangle$ and $|1\rangle$ states.

When considering all the potential states that n qubits can occupy simultaneously, we recognize that a quantum computer possesses an exponential memory capacity, with each additional qubit effectively doubling this capacity. Using a common analogy, we can liken an n-qubit system to having the ability to store information equivalent to 2^n classical bits.

In addition to superposition, entanglement plays a crucial role in structuring the memory of a quantum computer and enabling meaningful computations. Entanglement represents a relationship between a group of qubits, and it can manifest in various forms. An entangled set of qubits behaves as a unified quantum entity that cannot be described as the simple sum of its individual parts. In essence, the state of an entangled qubit cannot be fully characterized on its own; rather, its state is contingent upon the states of the other entangled qubits.

For quantum computations to be useful, it is essential to eventually extract results from the quantum computer. However, there is no efficient method for fully reconstructing quantum information classically. Instead, we rely on measurements to obtain solutions from quantum computations. Measurement, despite being a destructive process, allows us to extract a piece of information from a quantum superposition. For instance, consider the measurement of a qubit $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$. This measurement obtains a 0 with a probability of $|\alpha|^2$ and a 1 with a probability of $|\beta|^2$, effectively

collapsing the qubit to either $|0\rangle$ or $|1\rangle$ based on the measurement outcome. Notably, consecutive measurements of the same qubit will consistently yield the same result.

The relationship between some entanglements is apparent in their measurement probabilities. Consider the Bell state $|\text{bell}\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$, comprising two entangled qubits. When each qubit is measured, there is an equal probability of obtaining either 0 or 1. However, due to their entanglement, the two qubits consistently measure the same result. For instance, if the first returns 1, the second qubit will also measure 1.

To delve deeper into our exploration of the basics of quantum computing, we will examine it through the lens of the four postulates of quantum mechanics [26]. These postulates establish a mathematical framework for quantum mechanics and, by extension, quantum computation. This section will further elucidate each postulate, providing a concise discussion of their implications for quantum computing. It will end with an examination of density matrices and projections, topics crucial for implementing many quantum assertions presented in this paper.

2.1 State space

The first postulate states that the state of a quantum computer (or the state of its qubits) can be completely described by a unit vector, referred to as the *state vector*. This vector exists within a complex vector space that possesses a defined inner product, constituting a *Hilbert space* known as the *state space* [[26],§2.2.1]. To elaborate, the state of an n-qubit system can be entirely characterized by a unit vector residing in the Hilbert space $\mathbb{C}^{2^n}$.

In quantum computing, we refer to the standard basis as the *computational basis*, and it is common to express qubits as linear combinations of this basis. For instance, the computational basis for a single qubit can be represented as follows:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \text{ and } |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \tag{1}$$

We read the states $|0\rangle$ and $|1\rangle$ as *ket 0* and *ket 1*. Here, any arbitrary qubit $|\psi\rangle$ ("ket ψ") is defined as $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$, such that $|\alpha|^2 + |\beta|^2 = 1$. In addition to the column vector represented as a ket, there exists its dual vector $\langle\cdot|$, called a *bra*, where $\langle\psi| = |\psi\rangle^\dagger$. In simpler terms, a bra is the conjugate transpose of a ket.

For the computational basis of multiple qubits, we can represent the numbers within the ket using either decimal or binary notation. For instance, the computational basis for three qubits can be expressed as $\{|000\rangle, |001\rangle, |010\rangle, |011\rangle, \ldots, |111\rangle\}$ or equivalently $\{|0\rangle, |1\rangle, |2\rangle, |3\rangle, \ldots, |7\rangle\}$. The interchange between these representations can be done seamlessly without any risk of confusion.

2.2 Quantum gates

The second postulate states that a quantum computation can be executed through discrete steps governed by unitary operators [[26],§2.2.2]. In quantum computing,

Table 1 A compilation of frequently utilized single-qubit quantum gates, showcasing their matrix representation and effect on the computational basis

Gate Name	Matrix	Effect
Pauli X	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$	$X\|0\rangle = \|1\rangle$ $X\|1\rangle = \|0\rangle$
Pauli Y	$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$	$Y\|0\rangle = i\|1\rangle$ $Y\|1\rangle = -i\|0\rangle$
Pauli Z	$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$	$Z\|0\rangle = \|0\rangle$ $Z\|1\rangle = -\|1\rangle$
Hadamard	$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$	$H\|0\rangle = \frac{\|0\rangle+\|1\rangle}{\sqrt{2}}$ $H\|1\rangle = \frac{\|0\rangle-\|1\rangle}{\sqrt{2}}$
Phase	$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\lambda} \end{bmatrix}$	$P\|0\rangle = \|0\rangle$ $P\|1\rangle = e^{i\lambda}\|1\rangle$

these unitary operators are often referred to as quantum gates, or simply gates. Notably, a unitary operator denoted as U always possesses an inverse, which is achieved by transposing and then conjugating U, denoted as $U^\dagger$. Given that $U^\dagger$ also is a unitary operator, it becomes evident that a quantum computation reversible.

There are numerous quantum gates. In Table 1, we have compiled a list of the most commonly employed single-qubit gates. Note that some gates are the inverse of itself. For example, since the Hadamard gate is its inverse, $H\left[\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\right] = |0\rangle$.

Since the gates presented in Table 1 operate on individual qubits, they do not generate entanglement. To introduce entanglement, we can, for example, incorporate the Controlled-NOT (CNOT) gate, represented by the matrix:

$$\text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{2}$$

This gate acts on two qubits, flipping the second qubit (target qubit) if the first qubit (control qubits) is in the state $|1\rangle$. Notably, the CNOT gate establishes a conditional relationship between the states of the two qubits. Therefore, if the control qubit is in a superposition, this gate has the potential to induce entanglement between the two qubits.

Quantum circuits are graphical representations that depict quantum computations through the application of quantum gates. The execution of the circuit progresses from left to right, with each horizontal line symbolizing a qubit. Table 2 showcases commonly used two-qubit quantum gates along with their corresponding circuit representations. In Figure 1, we present an illustrative quantum circuit example. The graph element —⟦⟧= refers to a measurement operation on computation basis, a concept we will delve into further in sequence.

Table 2 A compilation of frequently utilized two-qubit quantum gates, showcasing their matrix and circuit representation. U is a generic single-qubit gate, I is a 2×2 identity matrix, and $\mathbf{0}$ is a 2×2 null matrix

Gate Name	Matrix	Circuit
Controlled-U	$\begin{bmatrix} I & \mathbf{0} \\ \mathbf{0} & U \end{bmatrix}$	
0-Controlled-U	$\begin{bmatrix} U & \mathbf{0} \\ \mathbf{0} & I \end{bmatrix}$	
CNOT	$\begin{bmatrix} I & \mathbf{0} \\ \mathbf{0} & X \end{bmatrix}$	
CZ	$\begin{bmatrix} I & \mathbf{0} \\ \mathbf{0} & Z \end{bmatrix}$	
SWAP	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	

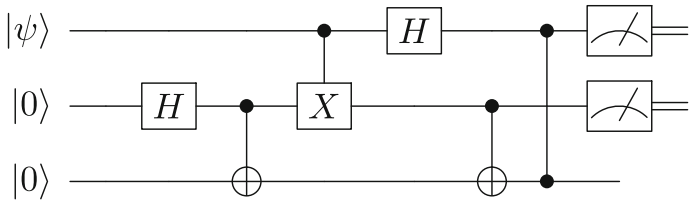


Fig. 1 Quantum circuit for the quantum teleportation protocol

2.3 Measurement

The third postulate outlines the outcomes and probabilities of a measurement. It asserts that a measurement is characterized by a set of measurement operators $\{M_m\}$, where the probability of obtaining the result m from the state $|\psi\rangle$ is given by

$$p(m) = \langle\psi| {M_m}^\dagger M_m |\psi\rangle . \tag{3}$$

After measuring and obtaining the result m, the quantum state transforms to

$$\frac{M_m |\psi\rangle}{\sqrt{p(m)}}. \tag{4}$$

The set of measurement operators must fulfill the completeness equation

$$\sum_m {M_m}^\dagger M_m = I, \tag{5}$$

which signifies that the sum of probabilities equals one, ensuring that all potential outcomes are counted [[26],§2.2.3].

Although this postulate is presented generically, quantum computers typically implement measurements on the computational basis. This basis is defined by the measurement operators:

$$\{M_0 = |0\rangle\langle 0|,\ M_1 = |1\rangle\langle 1|\}. \tag{6}$$

By utilizing these measurement operators, we can ascertain that the probability of measuring 0 or 1 from a state $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$ is $|\alpha|^2$ and $|\beta|^2$, respectively. For instance, expanding Equation 3 allows us to calculate the probability of measuring 0:

$$\begin{aligned} p(0) &= \langle\psi|{M_0}^\dagger M_0\,|\psi\rangle \\ &= \overbrace{(\alpha^\dagger\langle 0| + \beta^\dagger\langle 1|)}^{\langle\psi|}\overbrace{|0\rangle\langle 0|}^{M_0^\dagger}\overbrace{|0\rangle\langle 0|}^{M_0}\overbrace{(\alpha\,|0\rangle + \beta\,|1\rangle)}^{|\psi\rangle} \\ &= (\alpha^\dagger\overbrace{\langle 0|0\rangle}^{=1} + \beta^\dagger\overbrace{\langle 1|0\rangle}^{=0})(\alpha\overbrace{\langle 0|0\rangle}^{=1} + \beta\overbrace{\langle 0|1\rangle}^{=0}) \\ &= \alpha^\dagger\alpha = |\alpha|^2 \end{aligned} \tag{7}$$

The operation $\langle\psi|\varphi\rangle$ represents the inner product between the vectors $|\psi\rangle$ and $|\phi\rangle$. It is important to note that orthogonal vectors yield an inner product of zero, and when $|\psi\rangle = |\varphi\rangle$, the result is one. Similarly, the operation $|\psi\rangle\langle\varphi|$ signifies an outer product, which leads to the creation of a matrix. The outer product $|\psi\rangle\langle\psi|$ results in a Hermitian matrix, specifically a projector.

We can further expand Equation 4 to observe the collapse of the qubit after measurement 0:

$$\begin{aligned} \frac{M_0\,|\psi\rangle}{\sqrt{p(m)}} &= \frac{\overbrace{|0\rangle\langle 0|}^{M_0}\overbrace{(\alpha\,|0\rangle + \beta\,|1\rangle)}^{|\psi\rangle}}{\sqrt{|\alpha|^2}} \\ &= \frac{|0\rangle\,(\alpha\overbrace{\langle 0|0\rangle}^{=1} + \beta\overbrace{\langle 0|1\rangle}^{=0})}{|\alpha|} \\ &= \frac{\alpha\,|0\rangle}{|\alpha|} \end{aligned} \tag{8}$$

It is important to note that after measuring 0 on the state $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$, the state no longer remains in a superposition. The probability of measuring 0 in the collapsed state $\frac{\alpha}{|\alpha|}\,|0\rangle$ is $|\frac{\alpha}{|\alpha|}|^2 = 1$, thereby rendering the probability of measuring 1 to zero. The same behavior is observed when we expand Equations 3 and 4 using $M_1 = |1\rangle\langle 1|$, resulting in the state $\frac{\beta}{|\beta|}\,|1\rangle$ after measuring 1.

2.4 Composite systems

The fourth postulate offers a framework for extending the other postulates to systems with multiple qubits [citeNielsen:2010,S 2.2.8]. It establishes that for an n-qubit system in states $|\psi\rangle$, where the i-th qubit is represented by $|\psi_i\rangle$, we can depict $|\psi\rangle$ as the joint state of the qubits as

$$|\psi\rangle = |\psi_0\rangle \otimes |\psi_1\rangle \otimes |\psi_2\rangle \otimes \cdots \otimes |\psi_{n-1}\rangle\,. \tag{9}$$

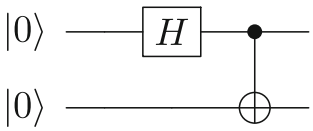


Fig. 2 Quantum circuit designed to create the state $|\text{bell}\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$

It is important to note that $|\psi_i\rangle$ is not entangled; otherwise, Equation 9 would not hold. For instance, we can describe the state

$$\sum_{k=0}^{2^n-1} \frac{1}{\sqrt{2^n}} |k\rangle = \bigotimes_{0}^{n-1} \frac{1}{\sqrt{2}} (|0\rangle + |1\rangle), \tag{10}$$

which features n qubits in superposition with a uniform distribution. However, there are no values for α, β, γ, and δ that satisfy the equation

$$\frac{1}{\sqrt{2}} (|00\rangle + |11\rangle) = (\alpha |0\rangle + \beta |1\rangle) \otimes (\gamma |0\rangle + \delta |1\rangle), \tag{11}$$

since the two qubits are entangled.

The tensor product operation, symbolized by "⊗", combines elements from distinct vector spaces into a larger joint vector space. For instance, if it operates with elements from the vector space of 1-qubit and 2-qubits, the result is an element in the vector space of 1+2-qubits. This operation can be depicted generically, as

$$A \otimes B = \begin{bmatrix} A_{11}B & A_{12}B & \cdots & A_{1n}B \\ A_{21}B & A_{22}B & \cdots & A_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1}B & A_{m2}B & \cdots & A_{mn}B \end{bmatrix}. \tag{12}$$

This operation effectively combines elements from matrix A and matrix B in such a way that each element in A is multiplied by the matrix B.

With the tensor product operation now clarified, we illustrate an instance of quantum execution using the quantum circuit depicted in Figure 2:

$$\xrightarrow{\text{Start}} |0\rangle \otimes |0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{13}$$

$$\xrightarrow{H_0} \left(\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right) \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} \tag{14}$$

$$\xrightarrow{\text{CNOT}_{0,1}} \begin{bmatrix} 1&0&0&0\\ 0&1&0&0\\ 0&0&0&1\\ 0&0&1&0 \end{bmatrix} \frac{1}{\sqrt{2}} \begin{bmatrix} 1\\0\\1\\0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1\\0\\0\\1 \end{bmatrix} \tag{15}$$

$$\xrightarrow{\text{Result}} \frac{1}{\sqrt{2}} \begin{bmatrix} 1\\0\\0\\1 \end{bmatrix} = \frac{1}{\sqrt{2}} (|00\rangle + |11\rangle) \tag{16}$$

2.5 Density matrix

A *density matrix* characterizes classical uncertainties related to a quantum state. It encapsulates the classical probabilities associated with being in a specific state [cite-Nielsen:2010,S 2.4.1]. For instance, a density matrix

$$\begin{aligned} \rho = \frac{1}{2}(|0\rangle\langle 0| + |1\rangle\langle 1|) &= \frac{1}{2}\overbrace{\begin{bmatrix}1\\0\end{bmatrix}}^{|0\rangle}\overbrace{\begin{bmatrix}1&0\end{bmatrix}}^{\langle 0|} + \frac{1}{2}\overbrace{\begin{bmatrix}0\\1\end{bmatrix}}^{|1\rangle}\overbrace{\begin{bmatrix}0&1\end{bmatrix}}^{\langle 1|} \\ &= \frac{1}{2}\overbrace{\begin{bmatrix}1&0\\0&0\end{bmatrix}}^{|0\rangle\langle 0|} + \frac{1}{2}\overbrace{\begin{bmatrix}0&0\\0&1\end{bmatrix}}^{|1\rangle\langle 1|} = \frac{1}{2}\begin{bmatrix}1&0\\0&1\end{bmatrix} \end{aligned} \tag{17}$$

assigns a fifty–fifty probability to being in either state $|0\rangle$ or $|1\rangle$. However, this probability distribution does not arise from a superposition like the state $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ does. Instead, it arises from our lack of knowledge about the system's actual state. In general, a density matrix can be defined as

$$\rho = \sum_k p_k |k\rangle\langle k|, \tag{18}$$

where p_k is a real number and $\sum_k p_k \leq 1$.

Quantum states that can be expressed as $|\psi\rangle\langle\psi|$ are called *pure states*, in contrast to *mixed states* like $\frac{1}{2}(|0\rangle\langle 0| + |1\rangle\langle 1|)$. The trace operation, denoted as $\text{tr}(\cdot)$, is commonly applied to density matrices to find the sum of their diagonal. A density matrix ρ is considered to represent a mixed state if $\text{tr}(\rho^2) < 1$.

Additionally, we can define the concept of the partial trace. Given a quantum state ρ^{AB} involving subsystems A and B, the partial trace of B is represented as $\rho^A \equiv \text{tr}_B(\rho^{AB})$. It is important to note that if the subsystems A and B are entangled, the result of $\text{tr}_B(\rho^{AB})$ will be a mixed state.

Given the definition from Equation 18, we can describe the application of a unitary operation U as follows:

$$\rho = \sum_k p_k |k\rangle\langle k| \xrightarrow{U} \sum_k p_k U\,|k\rangle\langle k|U^\dagger = U\rho U^\dagger. \tag{19}$$

Moreover, equations 3 and 4 can be expressed as:

$$p(m) = \mathrm{tr}({M_m}^\dagger M_m \rho), \text{ and} \tag{20}$$

$$\frac{M_m \rho {M_m}^\dagger}{\mathrm{tr}({M_m}^\dagger M_m \rho)}. \tag{21}$$

2.6 Projectors

Projectors play a crucial role in the implementation of some assertions discussed in this study. We will delve into specific properties of projectors that contribute to our understanding of how these assertions are evaluated.

With an orthonormal basis denoted as $\{|k\rangle\}$, we can define a *projector* as

$$P = \sum_k |k\rangle\langle k|, \tag{22}$$

and its *rank* corresponds to the dimension of the associated subspace spanned $\{|k\rangle\}$. Importantly, for any arbitrary $|\psi\rangle$, it holds that $|\psi\rangle\langle\psi| = (|\psi\rangle\langle\psi|)^\dagger$, which establishes a projector as a Hermitian matrix (self-adjoint matrix), where $P = P^\dagger$. Additionally, the matrix $Q = I - P$ also constitutes a projector, known as the complementary projector.

Using a projector P, we can establish a collection of measurement operators denoted as $M_P = \{M_0 = P,\ M_1 = I - P\}$. This collection comprises two measurement operators, where M_0 correspond to the projector P, and M_1 is crafted from the complementary projection. Given that $P = P^2$ and $Q = I - P$ is also a projector ($Q = Q^2$), we can observe that the measurement operators M_P adhere to Equation 5,

$$\sum_m M_m^\dagger M_m = P^\dagger P + Q^\dagger Q = P + (I - P) = I. \tag{23}$$

This confirms the completeness of the measurement operators.

We can establish a partial ordering relationship among projectors. Given projectors A and B, along with their associated subspaces $\mathcal{A}$ and $\mathcal{B}$, we state that $A \sqsubseteq B$ if and only if the subspace $\mathcal{A}$ is a subset of or equal to $\mathcal{B}$, $\mathcal{A} \subseteq \mathcal{B}$.

2.7 Limitations of classical debugging paradigms

The foundational principles of quantum mechanics render many classical debugging techniques ineffective or impractical when applied to quantum systems. While classical debugging relies on the ability to freely inspect and manipulate a program's internal state without affecting its future behavior, quantum mechanics imposes constraints that fundamentally invalidate this assumption.

The primary challenge arises from the destructive nature of quantum measurement. In classical computing, setting a breakpoint to inspect the value of a variable is a

non-invasive operation. In contrast, the quantum equivalent—measuring a qubit—irreversibly collapses its superposition ($|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$) into a definite classical state ($|0\rangle$ or $|1\rangle$). This collapse destroys the quantum information essential for the algorithm's continuation, making direct state inspection unsuitable for debugging quantum computations in progress.

A particularly illustrative example of this limitation is the classical assertion. In traditional software development, assertions are used to validate assumptions about the program's state at specific execution points, typically by evaluating boolean expressions over variable values. This practice is central to defensive programming and catching logic errors early. However, in quantum computing, there is no straightforward analog. Asserting the state of a qubit would require a measurement, which inherently disturbs the system and invalidates the subsequent computation. Furthermore, due to the probabilistic nature of quantum mechanics, a single measurement provides only limited statistical information. Therefore, the concept of a deterministic assertion is incompatible with quantum behavior, and any attempt to implement such checks must be carefully designed to avoid corrupting the computation.

Although quantum simulators avoid the problem of state collapse, they face severe limitations due to the exponential time and memory required to simulate quantum systems using classical resources. These challenges are further exacerbated by quantum entanglement, wherein the state of one qubit is intrinsically correlated with others, regardless of their physical separation. Such non-local correlations imply that analyzing a single qubit in isolation provides an incomplete and potentially misleading picture of the system's global state.

Moreover, techniques such as quantum state tomography [28], which aim to reconstruct the full quantum state, are computationally expensive and scale poorly, becoming infeasible even for moderately sized systems. This situation is analogous to debugging a classical program by examining a raw memory dump without any information about variable boundaries or context.

3 Empirical studies

In this section, we review two empirical studies that investigate quantum computing bugs. Our objective is to motivate the relevance of quantum debugging techniques by demonstrating the existence of quantum-specific bugs in real-world scenarios. The empirical studies discussed are conducted by Paltenghi and Pradel [29] and Luo et al. [30]. To the best of our knowledge, these are the first and only empirical studies available at the time of writing. Both studies address the same overarching issue but adopt different approaches.

The study by Paltenghi and Pradel [29] analyzes git commits from 18 quantum computing platforms and identifies a total of 223 bugs. In contrast, Luo et al. [30] investigate 96 bugs found across four quantum programming platforms by examining data from GitHub issues, Stack Overflow, and Stack Exchange. The former adopts a platform-developer perspective, focusing on bugs arising during the construction of well-known quantum computing platforms, many of which are supported by prominent

companies and startups. The latter study focuses on the perspective of end-users, analyzing bugs introduced while using quantum programming platforms.

We classify these studies into two categories: those examining bugs in the development of quantum computing platforms [29] and those investigating bugs arising from the use of quantum programming platforms [30]. While the latter aligns more closely with the scope of this survey, we argue that the findings of Paltenghi and Pradel [29] also reveal errors that could benefit from assertion-based quantum debugging techniques.

It is important to note that while this section highlights the existence of quantum-specific bugs, not all bugs related to quantum computing fall into this category, nor can all quantum-specific bugs be addressed through assertion-based debugging. In the following sections, we examine the findings of both studies and establish connections with the central theme of our work: quantum assertions.

3.1 Data collection

The research conducted by Paltenghi and Pradel [29] involved an investigation of 18 open-source quantum computation platforms, including well-known names such as Qiskit, Amazon Braket, D-Wave System, the Microsoft Quantum Development Kit [13], and StrawberryFields [31]. A complete list can be found in [29]. This study encompasses a significant portion of the most widely used quantum computation platforms. Notably, it does not only cover gate-based computation, which is the primary focus of this survey, but also extends to adiabatic [32, 33] and continuous-variable quantum computation [31]. Furthermore, this work addresses various aspects, including quantum simulation, error mitigation, and domain-specific platforms, in addition to quantum programming platforms.

To gather information about bugs, Paltenghi and Pradel [29] sourced data from the respective GitHub repositories of these platforms. They initiated the process by searching for commits containing the keywords "fix" and "#" (usually denoting an issue or a pull request), and refining their selection by filtering the keywords "refactor", "typo", "requirement", "import", and "style". This led to the identification of 2140 potential bugs. Subsequently, they adopted a random sampling approach to extract bug candidates, capping the selection at a maximum of 20 bugs identified per platform. A total of 223 bugs with fixed code were identified.

The study conducted by Luo et al. [30] focused on collecting bugs associated with the usage of quantum programming platforms, namely Qiskit [12], Cirq [34], Q#[13], and ProjectQ [35]. It is important to note that this study's primary interest lies not in the bugs originating from these platforms themselves, rather in the bugs that developers might inadvertently introduce using these platforms. This unique perspective offers a valuable complementary viewpoint to the work done by Paltenghi and Pradel [29].

For data collection, Luo et al. [30] manually explored various sources, including GitHub issues of the mentioned platforms, as well as discussions on Stack Overflow and Stack Exchange. Their objective was to identify quantum programming-related bugs that are accompanied by both valid pre- and post-fixed code, ensuring a comprehensive understanding of the context in which the bugs occurred.

3.2 Quantum-specific bugs

Both studies identified the existence of quantum-specific bugs that are intricately tied to the realm of quantum computing. Rectifying these bugs often requires specialized quantum-related knowledge, rendering traditional debugging techniques inadequate. In the study conducted by Paltenghi and Pradel [29], approximately 39.9% of the identified bugs were deemed quantum-specific. Conversely, in the study by Luo et al. [30], this figure rose to 82.3%. This variation in percentages is influenced by the datasets used in each study. The quantum computing platform [29] study encompassed numerous files and thousands of lines of code, while the quantum programming platform [30] study typically evaluated smaller projects, often confined to a single file.

An insightful interpretation of these results, given their respective datasets, is that the 82.3% figure from Luo et al. [30] might be more reflective of the reality experienced by novice quantum programmers working on modest projects. Conversely, the 39.9% figure from Paltenghi and Pradel [29] could align more closely with the situation faced by experienced quantum programmers dealing with larger, more intricate projects that involve both classical and quantum components. Regardless, the presence of 39.9% quantum-specific bugs remains substantial, compelling the advancement of dedicated quantum-specific debugging techniques.

A crucial revelation from the work of Paltenghi and Pradel [29] is that the most common indicator of a quantum-specific bug is incorrect output. In contrast, for classical bugs, crashes were the dominant symptom in their study. This disparity implies that identifying quantum bugs is notably more challenging compared to classical bugs. While a classical bug might lead to an application crash, which is immediately recognized as an error, quantum bugs might only result in inaccurate output, which could be harder to perceive as errors. This finding underscores the importance of developing specialized software debug and test techniques tailored for quantum-specific issues.

With the intricate dynamics of quantum programming, quantum assertion techniques can aid in identifying and localizing quantum-specific bugs within quantum programs. By employing these techniques, developers acquire a systematic method to interrogate quantum states, detecting the presence and specific manifestations of quantum bugs. This may facilitate precise diagnosis and targeted resolution of errors.

3.3 Bug patterns

Bug patterns are recurring code practices that lead to erroneous programs. Both studies identified similar bug patterns, which are illustrated in Figure 3. Before these studies, Huang and Martonosi [36] and Zhao et al. [37] also delved into certain bug patterns discussed in the respective papers. The significance of identifying bug patterns within quantum applications lies in the guidance they provide for the development of quantum debugging and testing techniques. Recognizing these patterns allows developers to concentrate their efforts on addressing specific dangers in quantum programs.

While both studies identified similar bug patterns, the occurrence of each pattern differs between the two studies. Notably, both studies found a significant prevalence of the *Incorrect numerical Computation* bug pattern, which underlines that the math-

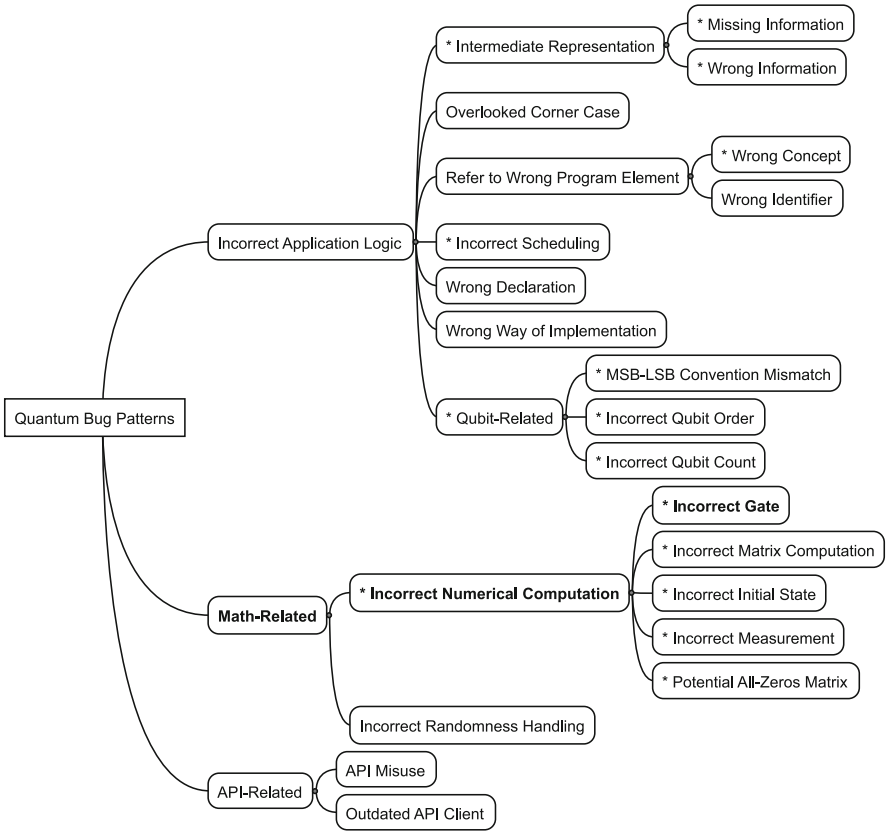


Fig. 3 Common Bug Patterns Identified in Quantum Computing Applications. The patterns marked with "*" are quantum-specific bug patterns

ematical intricacies of quantum programming are a source of concern for introducing bugs. In the context of bugs related to quantum computing platforms, the majority of bugs were cases of overlooked corner cases and incorrect information in quantum intermediary representation. On the other hand, the majority of bugs in quantum programming platforms were related to APIs, which suggests that the APIs provided by quantum programming platforms might not be intuitive for quantum programmers.

A potential underlying reason for the prevalence of API-related bugs could be the evolving nature of the programming landscape for quantum development. The absence of a standardized programming approach and the instability of many APIs might contribute to the occurrence of these bugs. This further emphasizes the need for improved API design and stabilization to enhance the overall user experience and mitigate the likelihood of such bugs in quantum programming platforms.

3.4 Bug complexity

The work of Paltenghi and Pradel [29] and Luo et al. [30] also evaluated the complexity of each bug fix using different metrics. In terms of the number of lines of code, both studies identified that bug fixes for quantum-specific issues require more changes than classical bugs. The study on quantum programming platforms found that 70% of quantum-specific bugs only require modifying a single line of code. On the other hand, the study on quantum computing platforms indicates that, on average, fixing a quantum-specific bug requires modifying approximately 8 lines of code. The study on quantum computing platforms also analyzed the number of code hunks (consecutive sequence of lines) changed. Similar to the number of lines of code, fixing quantum-specific bugs necessitates modifications in more code hunks.

Some quantum-specific bugs in quantum computing platforms require modifying more than 20 lines of code, which poses challenges to the capacity of today's bug-fix automation approaches for classical programming. In terms of the number of files modified, both studies found that fixing quantum-specific bugs usually involves changes to a single file. The study on quantum programming platforms also classified bugs into different levels of fault complexity, as proposed by Zhong and Su [38]. Bugs were classified as (C1) requiring a single repair action, (C2) needing non-data-dependent repair actions, (C3) demanding data-dependent repair actions, and (C4) involving a mixture of repair actions. In this study, most bugs were classified as C1, although it is worth noting that there were a non-negligible number of C3 and C4 bugs. For these more complex cases, programmers often lack the knowledge to implement the desired functionality, which is currently beyond the scope of today's bug-fix automation approaches.

3.5 Quantum bugs and assertion

The papers Paltenghi and Pradel [29] and Luo et al. [30] offer valuable insights into the landscape of quantum programming bugs, shedding light on quantum-specific issues in existing quantum platforms and providing motivation for the study of quantum debugging. The papers reveal that *incorrect numerical computations*, which encompass *incorrect gate* usage, represent a significant source of bugs in quantum programming. An example of this bug pattern is the misplacement of a quantum gate or quantum subroutine, precisely the type of error that quantum assertions aims to identify.

3.6 Quantum bugs and assertions

The studies by Paltenghi and Pradel [29] and Luo et al. [30] provide valuable insights into the types of bugs that arise in quantum programming and highlight the unique challenges posed by quantum-specific issues. These findings emphasize the importance of developing robust debugging techniques tailored to quantum systems. In particular, both studies identify *incorrect numerical computations*, including issues such as *incorrect gate usage*, as a prominent source of bugs in quantum programming.

For example, Paltenghi and Pradel [29] documents cases where quantum gates were applied to the wrong qubits or in an unintended order, resulting in deviations from the expected quantum state. Similarly, Luo et al. [30] describes instances where misaligned qubit indices during subroutine calls led to logical errors in quantum circuits. These issues, which are closely tied to the mathematical precision required in quantum programming, represent precisely the types of errors that quantum assertions are designed to address.

Quantum assertions play a critical role in identifying bugs by validating the correctness of quantum states at key points during program execution. While the studies focus on different contexts—Paltenghi and Pradel on bugs in platform development and Luo et al. on user-generated bugs—their findings collectively highlight the need for debugging methods that extend beyond classical approaches. Quantum assertions, particularly those discussed in Sections 4 and 5, can provide a direction for the development of a systematic framework for detecting and localizing such bugs. However, there remain significant gaps between the theoretical development of debugging methods and their practical application in quantum programming.

As we will discuss in the following sections, the time complexity and expressiveness of quantum assertions present challenges that make their widespread adoption in quantum programming an open problem.

4 Classical runtime assertion

In this section, we discuss approaches for evaluating quantum assertions at classical runtime. In other words, we explore how to evaluate a quantum assertion without executing it on a quantum computer. We cover the paper by Yuan et al. [39], which introduces a new quantum programming language capable of making assertions about entanglement. Additionally, we address the paper by Yu and Palsberg [40], which presents a quantum abstract interpretation for efficiently making assertions about quantum states on a classical computer.

The objective of assertion-based debugging is to verify the correctness of a program at a given point. A related topic to Classical Runtime Assertion is formal verification methods [41–43], which aim to prove the correctness of a quantum circuit. While the work by Yu and Palsberg [40] presented in this survey falls within the broader theme of formal methods, this topic is outside the scope of our survey. For further exploration of formal methods for quantum computing, we refer to the survey by Chareton et al. [44].

4.1 Purity assertion

The quantum programming language Twist [39] introduces a novel type system tailored for entanglement analysis. In Twist, quantum types are categorized as either *pure* or *mixed* through the use of a *purity* annotation. If a variable is deemed pure, it implies that it is not entangled with other quantum variables. In contrast, mixed variables are

described by density matrices, signifying potential entanglement with other quantum variables.

Twist operates with logical qubits that remain unaffected by noise. However, when a quantum variable becomes entangled with another, it can no longer be entirely represented by a pure state. Quantum variables can take the form of single qubits or tuples of qubits. Specifically, operations performed on a variable labeled as pure do not affect other quantum variables. Conversely, operations on mixed variables may impact other variables due to entanglement.

The purity annotation does not affect quantum computation directly, but it plays a crucial role in identifying bugs within quantum programs. To uphold the integrity of the type system, Twist relies on two assertions: the *purifying-cast assertion* and the *purifying-split assertion*. The purifying-cast assertion verifies the purity of a quantum variable, ensuring it remains unentangled with other quantum variables. On the other hand, the purifying-split assertion confirms that a quantum variable can be divided into two separate pure quantum variables. Notably, the purifying-split assertion can only be evaluated during quantum runtime, as its time complexity aligns with executing the entire quantum program [45].

The purifying-cast assertion involves a conservative analysis of entangled variables. Any interaction between two variables, such as through a CNOT gate, is considered entangling. Twist implements this technique akin to fractional permissions [46, 47], where each element of a pure state shares a fraction of the entanglement, maintaining the state's purity when all elements are collectively addressed. This assertion can be efficiently evaluated at classical runtime and forms a crucial component of Twist's type system for entanglement analysis. However, to comprehensively evaluate the language's type system, the quantum runtime purifying-split assertion becomes necessary.

Evaluating the purifying-split assertion at classical runtime is generally impractical, thus expected to occur during quantum runtime. In Section 5, we will delve into some quantum runtime assertions that can be employed to implement the purifying-split assertion. As we'll discuss, achieving an efficient quantum runtime entanglement assertion remains an ongoing challenge, and consequently, an efficient implementation of the Twist type system is yet to be fully realized.

The purity assertion in Twist not only aids in identifying potential implementation errors but also ensures the safe disposal of quantum variables. Operations on an entangled qubit can inadvertently impact the entire entangled set, making deallocation of such qubits prone to unwanted side effects, leading to potential invalidation of the quantum computation. With the purity assertion in Twist, a quantum variable can be designated as pure, enabling automatic deallocation when the variable becomes inaccessible, for instance, when it goes out of scope.

Figure 4 demonstrates the implementation of the quantum teleportation protocol shown in Figure 1, implemented using the Twist language. In line 10, a purifying-cast assertion is conducted on the quantum variable `q123`, encompassing all three qubits involved in the computation. This assertion occurs at the classical runtime or statically, as outlined in the paper. Following that, in line 12, the purifying-split assertion is executed to partition the quantum variable `q123` into `q12` and `q3`, consisting of two

```
1  fun teleport (q1 : qubit<P>) : qubit<P> = (* pure type *)
2    let q23 : (qubit & qubit)<P> = bell_pair () in
3    let (q2 : qubit<M>, q3 : qubit<M>) = q23 in
4    let (q1 : qubit<M>, q2 : qubit<M>) = CNOT (q1, q2) in
5    let q1 : qubit<M> = H (q1) in
6    let (q2 : qubit<M>, q3 : qubit<M>) = CNOT (q2, q3) in
7    let (q1 : qubit<M>, q3 : qubit<M>) = CZ (q1, q3) in
8    let q123 : ((qubit & qubit) & qubit)<M> = ((q1, q2), q3) in
9    (* assert ((q1, q2), q3) is pure; check statically *)
10   let q123 : ((qubit & qubit) & qubit)<P> = cast<P>(q123) in
11   (* assert q3 is separable from (q1, q2); check dynamically *)
12   let (q12 : (qubit & qubit)<P>, q3 : qubit<P>) = split<P>(q123) in
13   let _ : bool * bool = measure (q12) in q3
```

Fig. 4 Quantum teleportation protocol implemented in Twist, featuring purity assertions

and one qubit, respectively. Subsequently, after the assertion in line 12, the first two qubits can be safely disposed, rendering the measurement in line 13 unnecessary.

In case any errors were introduced in prior steps, such as substituting a CZ gate with a CNOT gate in line 7, it would lead to the entanglement of qubits in `q123` during line 12 split assertion, causing it to fail. Although the assertion in line 10 might seem redundant in the code, it ensures that the state passed to the split assertion in line 12 is in a pure state. Without the purifying-cast assertion in line 10, the purifying-split assertion would fail. This behavior aids in reducing resource utilization by executing the assertion at the quantum runtime [39].

4.2 Quantum abstract interpretation

The *Quantum Abstract Interpretation* (QAI) method introduced by Yu and Palsberg [40] offers a means to verify whether the outcomes of a quantum computation reside within the Hilbert space defined by a projector $P = |\phi\rangle\langle\phi|$ or $P = |\phi\rangle\langle\phi| + |\varphi\rangle\langle\varphi|$ of rank 1 or 2, respectively. This innovative approach serves as a foundation for implementing quantum assertions. One intriguing aspect of this method lies in its operational efficiency—it can be executed by a classical computer in polynomial time, enabling the evaluation of quantum assertions at classical runtime.

An assertion within this method comprises two quantum states, denoted as $\{|\phi\rangle\,,\ |\varphi\rangle\}$. For a given state $|\psi\rangle$, the assertion holds true if it can be expressed as a linear combination of the eigenstates possessing nonzero eigenvalues of the projector $P = |\phi\rangle\langle\phi| + |\varphi\rangle\langle\varphi|$.

For example, consider the GHZ state $\frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$, which satisfies the assertion defined by the quantum states $\{|000\rangle\,,\ |111\rangle\}$ and the projector

$$P = |000\rangle\,\langle 000| + |111\rangle\,\langle 111|. \tag{24}$$

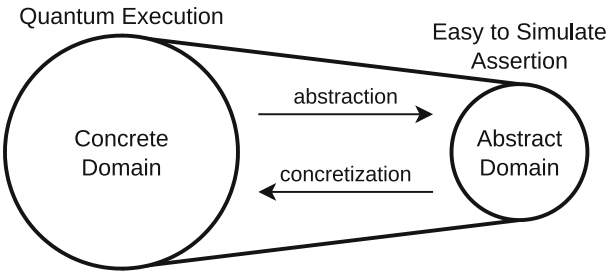
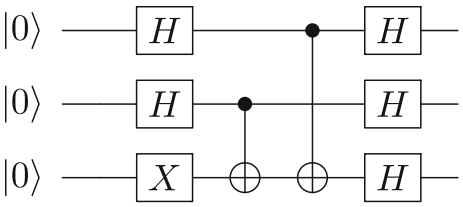


Fig. 5 Correspondence between the abstract and concrete domains in quantum abstract interpretation. Assertions verified in the abstract domain are guaranteed to hold in the concrete domain of actual quantum execution

Fig. 6 Quantum circuit that prepares the state $\frac{1}{\sqrt{2}}(|000\rangle - |111\rangle)$



Note that the states $|000\rangle$ and $|111\rangle$ individually also satisfy this assertion. Therefore, this method alone cannot verify entanglement. For instance, consider using an assertion defined by the GHZ state $\{\frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)\}$, which yields the projector

$$P = \frac{1}{2}(|000\rangle + |111\rangle)(\langle 000| + \langle 111|). \tag{25}$$

Although this operator projects specifically onto the GHZ state, the span of the Hilbert space remains the same as in the previous example. Therefore, both assertions are equivalent in terms of their support. Consequently, this approach is insufficient for verifying entanglement.

The evaluation of this assertion using a classical computer hinges on executing the quantum computation within an *abstract domain*. Once the assertion proves successful within this abstract domain, the method proposed by Yu and Palsberg [40] ensures its validity in the actual quantum computation, often referred to as the *concrete domain*. Figure 5 illustrates correspondence between the abstract and concrete domains. To enable the operation and assessment of assertions in the abstract domain, this approach relies on two pivotal mapping functions: the *abstraction function* and the *concretization function*. These functions, alongside defining the abstract domain, will be introduced in the subsequent sections.

Furthermore, we will exemplify the process by evaluating an assertion employing the projector $P = |000\rangle\langle 000| + |111\rangle\langle 111|$ for the circuit depicted in Figure 6.

4.2.1 Abstract domain tuple

Simulating an n-qubit system necessitates exponential space and time [16]. To achieve efficient simulation, the quantum abstract domain partitions the system into smaller elements comprising k-local qubits. This domain is characterized by a collection of tuples, where each tuple encapsulates the represented qubits. For instance, a three-qubit abstract domain could be denoted as $S = \{(0, 1),\ (0, 2),\ (1, 2)\}$, with each tuple embodying two local qubits. Within this framework, a quantum state is represented by projections, such as $\mathcal{S} = \{S_{0,1},\ S_{0,2},\ S_{1,2}\}$.

The overarching domain that includes all qubits is termed the concrete domain. For a three-qubit system, the concrete domain could be illustrated as $T = \{(0,\ 1,\ 2)\}$. As the initial state and the asserted projection are defined within this concrete domain, the authors introduced the abstract function to map a state from a larger domain to a smaller one.

4.2.2 Abstraction function

The authors establish a partial order between abstract domains by considering two domains, $S = \{s_0,\ s_1,\ \ldots,\ s_m\}$ and $T = \{t_0,\ t_1,\ \ldots,\ t_m\}$, denoting $S \trianglelefteq T$ (T is finer than S) if $\forall i : s_i \subseteq t_i$. For abstraction domains S and T where $S \trianglelefteq T$, and their respective quantum states $\mathcal{S} = \{S_1, S_2, \cdots, S_m\}$ and $\mathcal{T} = \{T_1, T_2, \cdots, T_m\}$, the authors proposed the abstraction function to map the projections of $\mathcal{T}$ into the abstraction domain S. The function is defined as

$$\alpha_{T \to S}(\mathcal{T}) = \{S_1, S_2, \cdots, S_m\}, \tag{26}$$

where

$$S_i = \bigcap_{t_j : s_i \subseteq t_j} \mathrm{supp}(\mathrm{tr}_{t_j / s_i} T_j). \tag{27}$$

Here, $\mathrm{supp}(P) = \sum_k |k\rangle\langle k|$ is defined as the sum over $\{|k\rangle\}$, the eigenvectors with a nonzero eigenvalue of the projector P.

For instance, the initial state of the quantum circuit in Figure 6 is defined by the projector $|000\rangle\langle 000|$. For a n-qubit system, one can use an n-choose-k approach to determine the projectors, where each local projector encompasses k qubits. For example, selecting the local projectors $S = \{(0,\ 1),\ (0,\ 2),\ (1,\ 2)\}$ for a three-qubit system with the concrete domain $T = \{(0,\ 1,\ 2)\}$, the abstraction function enables the computation of local projectors for the initial state as

$$\alpha_{T \to S}(|000\rangle\langle 000|) = \{|00\rangle_{0,1}\langle 00|,\ |00\rangle_{0,2}\langle 00|,\ |00\rangle_{1,2}\langle 00|\}. \tag{28}$$

After obtaining the projectors representing the initial state, to execute the quantum circuit in Figure 6, the concretization function is required. This function maps a state from a smaller domain into a larger one, facilitating the calculation of the quantum gate's effect on qubits that may not be present in a local projector.

4.2.3 Concretization function

The concretization function maps projectors from an abstract domain S into a finer abstract domain T. This function is defined as

$$\gamma_{S\to T}(\mathcal{S}) = \{T_1,\ T_2,\ \cdots,\ T_m\}, \tag{29}$$

where

$$T_i = \bigcap_{s_i : s_i \subseteq t_j} S_i \otimes I_{t_j/s_i}. \tag{30}$$

The intuition behind this function lies in utilizing the identity matrix I_{t_j/s_i} as an approximation for the qubit state that t_j lacks from s_i.

The concretization function becomes necessary to apply quantum gates on states within the abstract domain. Considering an abstract domain $S = \{s_1,\ s_2,\ \ldots,\ s_m\}$ and a quantum gate U acting on the qubits tuple s_U, a new abstract domain $T = \{t_1 = s_1 \cup s_U,\ t_2 = s_2 \cup s_U,\ \ldots,\ t_m = s_m \cup s_U\}$ can be defined, where $S \trianglelefteq T$. To apply U on the quantum state $\mathcal{S}$ in the abstract domain S, the following function is employed

$$U_T(\mathcal{T}) = \{UT_1U^\dagger,\ UT_2U^\dagger,\ UT_mU^\dagger\}. \tag{31}$$

Thus, the composition of functions to apply the gate U on $\mathcal{S}$ is

$$\alpha_{T\to S} \circ U_T \circ \gamma_{S\to T}. \tag{32}$$

Continuing with the example of executing the circuit from Figure 6, we can produce the following steps applying the quantum gates:

$$\begin{array}{lllll}
\text{Init} & \to\{ & |00\rangle_{0,1}\langle 00|, & |00\rangle_{0,2}\langle 00|, & |00\rangle_{1,2}\langle 00| \quad \} \\
\text{H}_0 & \to\{ & |+0\rangle_{0,1}\langle +0|, & |+0\rangle_{0,2}\langle +0|, & |00\rangle_{1,2}\langle 00| \quad \} \\
\text{H}_1 & \to\{ & |{+}{+}\rangle_{0,1}\langle {+}{+}|, & |+0\rangle_{0,2}\langle +0|, & |+0\rangle_{1,2}\langle +0| \quad \} \\
\text{X}_2 & \to\{ & |{+}{+}\rangle_{0,1}\langle {+}{+}|, & |+1\rangle_{0,2}\langle +1|, & |+1\rangle_{1,2}\langle +1| \quad \} \\
\text{CNOT}_{1,2} & \to\{ & |+0\rangle\langle +0|+|+1\rangle\langle +1|, & |+0\rangle\langle +0|+|+1\rangle\langle +1|, & \tfrac{1}{2}(|01\rangle+|10\rangle)(\langle 01|+\langle 10|) \quad \} \\
\text{CNOT}_{0,2} & \to\{ & |{+}{+}\rangle\langle {+}{+}|+|{-}{-}\rangle\langle {-}{-}|, & |{+}{+}\rangle\langle {+}{+}|+|{-}{-}\rangle\langle {-}{-}|, & |{+}{+}\rangle\langle {+}{+}|+|{-}{-}\rangle\langle {-}{-}| \quad \} \\
\text{H}_0 & \to\{ & |0+\rangle\langle 0+|+|1-\rangle\langle 1-|, & |0+\rangle\langle 0+|+|1-\rangle\langle 1-|, & |{+}{+}\rangle\langle {+}{+}|+|{-}{-}\rangle\langle {-}{-}| \quad \} \\
\text{H}_1 & \to\{ & |00\rangle\langle 00|+|11\rangle\langle 11|, & |0+\rangle\langle 0+|+|1-\rangle\langle 1-|, & |0+\rangle\langle 0+|+|1-\rangle\langle 1-| \quad \} \\
\text{H}_2 & \to\{ & |00\rangle\langle 00|+|11\rangle\langle 11|, & |00\rangle\langle 00|+|11\rangle\langle 11|, & |00\rangle\langle 00|+|11\rangle\langle 11| \quad \}
\end{array}$$

4.2.4 Assert evaluation

A quantum state $|\psi\rangle$ satisfies an assertion defined by a projector P if $|\psi\rangle$ is an eigenstate of P, or consequently, $P|\psi\rangle = |\psi\rangle$. Extending this concept to quantum abstract interpretation, the projectors $\mathcal{S}$ representing the quantum state in the abstract domain satisfy an assertion projector P if the set of eigenstates with a nonzero eigenvalue of each projector in $\mathcal{S}$ is a subset of the eigenstates with a nonzero eigenvalue of $\alpha_{T\to S}(\{P\})$.

The authors assert that if an assertion defined by the projectors $\alpha_{T\to S}(\{P\})$ holds true in the abstract domain S, then the assertion also holds true in the concrete domain.

For instance, to evaluate the assertion defined by the projector $|000\rangle\langle 000| + |111\rangle\langle 111|$ at the end of the quantum circuit in Figure 6, we first find the assertion projectors acting on the local qubits:

$$\begin{aligned}\alpha_{T\to S}(|000\rangle\langle 000| + |111\rangle\langle 111|) = \{&|00\rangle_{0,1}\langle 00| + |11\rangle_{0,1}\langle 11|,\\ &|00\rangle_{0,2}\langle 00| + |11\rangle_{0,2}\langle 11|,\\ &|00\rangle_{1,2}\langle 00| + |11\rangle_{1,2}\langle 11|\}.\end{aligned} \tag{33}$$

Next, we evaluate the assertion on the local qubits. Notably, the assertion projectors in the abstract domain are equal to the final state of the quantum circuit execution in the abstract domain. Therefore, it is evident that the quantum circuit satisfies the assertion.

4.2.5 Assertion time complexity

To evaluate an assertion within the abstract domain, all quantum computations must be executed in that domain, and the time complexity of this method relies on the time complexity of the quantum circuit being assessed. Therefore, this analysis focuses on evaluating the time complexity of performing one computational step within the abstract interpretation.

Employing an n-choose-k approach to establish the abstract domain of k-local qubits for an n-qubit concrete domain, the number of projectors becomes

$$\frac{n!}{k!(n-k)!}. \tag{34}$$

Consequently, the count of projectors escalates with $n!$ and diminishes with $k!(n-k)!$. Additionally, each projection's size is 2^k. Therefore, the time complexity for a step within an abstract domain formed via the n-choose-k approach is

$$\frac{n!}{k!(n-k)!}2^k \in O(n^k). \tag{35}$$

This time complexity makes the assertion evaluation method scalable concerning the number of qubits. The authors managed to evaluate a 300-qubit GHZ circuit in less than 3 hours and 30 minutes and a 300-qubit Grover algorithm [48] in under 2 days using a MacBook Pro equipped with an Intel Core i7 2.2 GHz. It is worth noting that the Grover's algorithm exhibits a time complexity of $O(\sqrt{2^n})$, rendering it inefficient in terms of qubits.

5 Quantum runtime assertion

In this section, we delve into methods for implementing assertions to be evaluated during quantum runtime, labeled as *quantum runtime assertions* (QRAs). We divide this section into two parts. First, we conduct an analysis of diverse design methodologies

Table 3 Comparison of Quantum Runtime Assertion Approaches

Method	Complexity(as Hard as)	AuxiliaryQubits	StateCorrection	Mid-Circuit Measurement
Statistical	Sampling	None	×	×
SWAP-Based	State Preparation	$O(n)$	✓	×
OR-Based	State Preparation	1	×	×
NDD-Based	Diagonalization	1	×	×
Projection-Based	Diagonalization	None	✓	✓

for QRA. We present quantum circuit implementations for each method, discussing their associated quantum and classical time complexities for each assertion. Subsequently, we explore practical implementations of QRA.

5.1 Design methods for QRA

There are various design methods available for implementing the same assertion evaluation, and each design has its set of advantages and disadvantages, as presented in Table 3. A *statistical assertion*, for instance, does not increase the quantum circuit size, but it necessitates repeated execution of the same quantum program. On the other hand, *SWAP-based assertions* maintain the correctness of the quantum state regardless of the assertion result, yet they demand n auxiliary qubits to assert n qubits. *Projection-based assertions* offer the benefits of SWAP-based assertions without the need for auxiliary qubits, but they do require mid-circuit measurements, which are not always available in today's quantum computers. *NDD-based assertions* and *OR-based assertions*, however, only require one auxiliary qubit, but they do not correct the quantum state in the event of a failed assertion. Each design method presents unique trade-offs that must be carefully considered based on the specific requirements and constraints of the quantum computing scenario. Also, all methods are susceptible to false positive.

The SWAP-based assertion, NDD-based assertion, and projection-based assertion share the same expressive power. Implementing an arbitrary assertion incurs exponential time complexity in both classical and quantum runtimes. Although crucial, achieving efficient implementations for specific assertion methods remains an ongoing challenge. For example, verifying if a quantum state resides within the space defined by a projector demands exponential time. In the subsequent subsection, we discuss each QRA implementation design and implementation.

5.1.1 Statistical assertion

Statistical assertion relies on measurement statistics to reason about the quantum state. The approach introduced by Huang and Martonosi [36] employs the *chi-square* test to assert whether a qubit is in a classical state or a superposition state. Furthermore, this method utilizes a *contingency table* to assess the entanglement of multiple qubits. The statistical analysis provided by this technique provides valuable insights into the

properties of quantum states, aiding in the identification and characterization of quantum bugs. Furthermore, its simple implementation and the potential for automation render this approach a suitable starting point for quantum debugging.

Since statistical assertions rely on measurements, which collapse the quantum state and can invalidate the quantum computation, Huang and Martonosi [36] proposes splitting the quantum problem up to the point of the assertion. For example, if a program contains two assertions, it will generate two separate programs—one for each assertion. These derived programs are designed to evaluate the assertions without interfering with the original quantum computation. To evaluate the assertions, each corresponding program is executed multiple times (several shots), and the obtained results are subjected to the probability test.

The drawback of this approach is that it increases the number of executions required. However, considering that modern quantum computers already execute numerous shots for a single job (program), adding assertions will only result in a linear increase in the number of jobs during debugging. Similarly to classical programs, it is common practice to disable assertions when running in a production environment. The same principle can be applied to quantum programs, effectively eliminating the overhead associated with assertions during a regular quantum execution. This approach allows researchers and developers to maintain efficient and optimized quantum computations in a production setting while utilizing assertions solely for debugging and validation purposes.

Noise can lead to assertion failures even in the presence of a correct program. As a result, this technique can be adapted for NISQ computers, and assertion failures can be reinterpreted in such cases. It is important to note that alternative assertion techniques might yield better results when reasoning about quantum applications on NISQ computers. These techniques have the potential to correct the quantum state during runtime or enable the post-selection of measurement results to mitigate the effects of noise.

A testing framework with assertions analogous to the statistical approach was implemented by Honarvar et al. [49] for the quantum programming language Q#. This framework aims to evaluate properties such as quantum superposition and entanglement within quantum programs using Q#.

5.1.2 SWAP-based assertions

The SWAP-based assertion method, proposed by Liu and Zhou [50], aims to verify whether a tested state $|\psi\rangle$ intersects with a set of desirable quantum states. Depending on the size of the desirable state set, this assertion can be categorized as a *precise assertion* if the set contains only one state or as an *approximate assertion* if it includes multiple states. Figure 7 illustrates the general schema for this assertion method. In the case of a precise assertion, the number of auxiliary qubits required is equal to the number of qubits in the tested state $|\psi\rangle$. For approximate assertions, the number of auxiliary qubits required is typically determined by the size and form of the set of desirable states. One advantage of this method is the ability to perform multiple assertions during a single quantum execution, and the quantum execution always remains correct after the assertion. Even in instances where assertions may fail, the

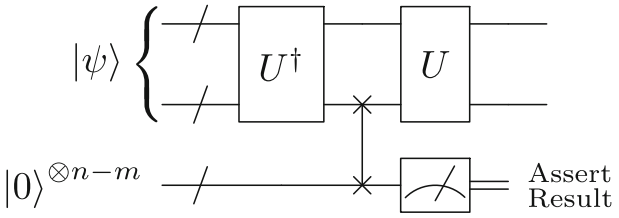


Fig. 7 The general schema for SWAP-based assertions on the n-qubit state $|\psi\rangle$ with $t = 2^m$

approach ensures that the quantum state is altered to the correct one, thereby preventing any adverse effects on subsequent computations. However, this method requires a substantial number of auxiliary qubits and, in the worst case, has exponential time complexity.

A precise assertion is considered successful when the tested quantum state $|\psi\rangle$ matches a desirable state $|\varphi\rangle$. To evaluate this assertion, a unitary transformation $U^\dagger$ is applied, where $U|0\rangle = |\varphi\rangle$. If the assertion is successful, it implies that $U^\dagger|\psi\rangle = |0\rangle$, ensuring that the auxiliary qubits will remain in the quantum state $|0\rangle$ after the SWAP gate is executed (see Figure 7). On the contrary, if the assertion fails, the auxiliary qubits will be found in a state different from $|0\rangle$. To determine the success of the assertion, a measurement is performed on the auxiliary qubits. If the measurement outcome is zero, the assertion succeeded; otherwise, it failed. When the assertion fails, the SWAP gate sets the tested qubits to $|0\rangle$, with the unitary operation U preparing them on the desired state, correcting the quantum execution.

In an approximate assertion, we need to prepare a density matrix $\rho = \sum_i |\psi_i\rangle\langle\psi_i|$ (we do not need to normalize ρ)[1]. The assertion is considered successful if a state $|\psi\rangle$ can be expressed as a linear combination $\sum_i \alpha_i |\psi_i\rangle$, where $\exists\alpha_i \neq 0$. For example, the states $|00\rangle$, $|11\rangle$, and $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ satisfy the assertion with the density matrix $\rho = |00\rangle\langle 00| + |11\rangle\langle 11|$. This type of assertion allows for a broader acceptance of states that are close to the desired states specified in the density matrix ρ. This flexibility is useful when we are interested in verifying quantum states that are not precisely equal to desirable states but still meet specific criteria within an acceptable range of similarity. Though initially formulated using a density matrix, the approximate assertion can also be described as a projector, where it succeeds if the tested state resides within the space spanned by the projector.

To construct the unitary operator U for an approximate assertion, we start by diagonalizing the density matrix ρ to obtain its orthonormal eigenstates. Since some of these eigenstates may have eigenvalues of zero, let t represent the number of nonzero eigenvalues, which corresponds to the rank of ρ. Assuming $t = 2^m$ for a positive integer m, and ensuring that $t \leq 2^{n-1}$ (where n is the total number of qubits), the unitary operator U can be defined as:

[1] This can be considered as a projector, but we adhere to the use of density matrix to match the author's description.

$$U = \sum_{i=0}^{2^n} |i\rangle \langle \varphi_i|, \tag{36}$$

where $|\varphi_i\rangle$ represents an eigenstate of ρ with a nonzero eigenvalue for $i < t$, corresponding to computational basis states $|i\rangle$ whose last $n - m$ qubits are in the state $|0\rangle$. For $i \geq t$, where $|\varphi_i\rangle$ does not correspond to eigenstates with nonzero eigenvalues, the computational basis states $|i\rangle$ have at least one of their last $n - m$ qubits in the state $|1\rangle$.

With this unitary operator U, applying the approximate assertion becomes analogous to performing a precise assertion. However, this method requires only $n - m$ auxiliary qubits, and the operation involves SWAP gates acting on the last $n-m$ qubits of the tested qubits. When assertions conform to $t \neq 2^m$ or $t > 2^{n-1}$, transformations can be employed to adjust the assertion. This process is analogous to modifying the rank of the projector, a concept discussed in Section 5.1.5 for the projector-based assertion.

Similar to the precise assertion, the approximate assertion also corrects the quantum state. When the approximate assertion succeeds, it means that the last $n - m$ qubits of the state $U^\dagger |\psi\rangle$ will be in the state $|0\rangle$ due to the nature of the unitary operator U. In the event of an assertion failure, the SWAP gate will set those last $n - m$ qubits to the state $|0\rangle$, effectively correcting the quantum state. This behavior ensures that the quantum computation can continue correctly regardless of the assertion outcome.

The time complexity of implementing a precise assertion is determined by the time taken to prepare the desired state $|\varphi\rangle$. If the initial state of the quantum computation is $|0\rangle$, it is reasonable to expect that the time required to prepare $|\psi\rangle$ would be equivalent to the time needed to prepare $|\varphi\rangle$ through the unitary operation U. However, in practical implementations, achieving optimal efficiency in executing the unitary U might not be guaranteed. In approximate assertions, the process of diagonalizing the density matrix ρ, which is a $2^n \times 2^n$ matrix, has an exponential time complexity.

As the number of qubits increases, the complexity of diagonalizing ρ or preparing the desirable state $|\varphi\rangle$ can become computationally prohibitive. This highlights the need for careful consideration of the efficiency and scalability of the assertion method for large-scale quantum computations. Researchers may need to explore alternative techniques and optimizations to handle the growing complexity and resource demands as quantum systems scale up.

Due to its requirement for a significant number of auxiliary qubits, the SWAP-based assertion method may not be well-suited for NISQ devices that have limited qubit resources. Using OR-based assertions can be a viable alternative in these situations.

5.1.3 OR-based assertions

The OR-based assertion method proposed by Liu and Zhou [50] is equivalent to the SWAP-based assertion in terms of assertion expressivity, and both methods use the same unitary transformation U. This method has the advantage of using only one auxiliary qubit. However, this approach cannot correct the quantum state in the event of a failed assertion.

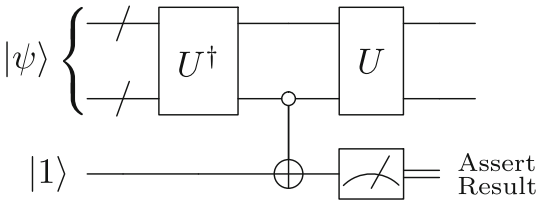


Fig. 8 The general schema for OR-based assertions on state $|\psi\rangle$

Figure 8 illustrates the implementation of an OR-based assertion. This method relies on the property that the last $n - m$ qubits of the test quantum state are $|0\rangle$ when the assertion succeeds. The OR-based assertion utilizes only one auxiliary qubit, initialized in the state $|1\rangle$, which serves as the target for a multi-controlled-NOT (MCNOT) gate. The qubits that would be part of the SWAP operation act as control qubits for this MCNOT gate. It is important to note that the MCNOT gates are applied with the control qubits in the state $|0\rangle$, ensuring that the auxiliary qubit will be in the state $|0\rangle$ in the case of a successful assertion.

Just like the SWAP-based assertion method, OR-based assertions can also suffer from false positives, where the tested quantum state may erroneously appear to satisfy the assertion when it is close to the desirable state. In such cases, the quantum state will still collapse to the correct state after measurement, but the assertion itself may have been misleading.

Since OR-based assertions do not correct the quantum state when failed, post-selection can be employed. Post-selection involves discarding measurement outcomes that do not match the desired assertion result. By doing so, only the executions that end successfully with the auxiliary qubit in the state $|0\rangle$ are retained. Moreover, post-selection can be particularly beneficial for NISQ computers that are susceptible to assertion errors due to decoherence and noise. By effectively filtering out incorrect measurement outcomes, post-selection can help mitigate the impact of noise on the assertion results, increasing the overall reliability of the quantum computation.

In essence, OR-based assertions with post-selection bear similarities to quantum error correction codes [51], which are designed to detect and correct errors in quantum computations. While the primary purpose of OR-based assertions is quantum state verification, their ability to leverage post-selection and handle noise effects can also contribute to error mitigation on the quantum computer, depending on the time complexity of implementing the assertion.

5.1.4 NDD-based assertion

In quantum information, *non-destructive discrimination* (NDD) is a technique used to distinguish between different entangled states without destroying the quantum information contained in those states [52]. Building on this concept, Liu et al. [53] proposed an assertion method, which was later specialized into the NDD-based assertion [50].

The NDD-based assertion method shares similarities with the OR-based assertion method, with both methods having similar advantages and disadvantages. However, the NDD-based assertion method offers a linear reduction in the number of quantum gates required for its implementation. This reduction in quantum gates can lead to more

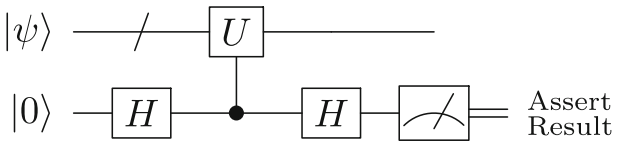


Fig. 9 The general schema for NDD-based assertions on state $|\psi\rangle$

efficient implementations and potentially lower resource requirements, making the NDD-based assertion method attractive for scenarios with limited quantum resources, such as NISQ devices.

To implement the NDD-based assertion, a unitary operator U is constructed similarly to the one used in the SWAP-based approximate assertion. Starting with the density matrix $\rho = \sum_i |\psi_i\rangle \langle\psi_i|$ representing the desirable state for the assertion, diagonalization is performed on $\rho = D\Lambda D^\dagger$ to find the orthogonal basis. The basis $\{|\varphi_i\rangle\}$ are the eigenvectors (columns) of the diagonalizing matrix D, and Λ is a diagonal matrix formed by the corresponding eigenvalues. The eigenstates $|\varphi_i\rangle$ are then sorted such that the first t eigenvalues are nonzero, where t represents the number of nonzero eigenvalues of ρ, and it is the rank of ρ. The unitary operator U for the NDD-based assertion is constructed as follows:

$$U = \sum_{i=1}^{t} |\varphi_i\rangle\langle\varphi_i| - \sum_{i=t+1}^{2^n} |\varphi_i\rangle\langle\varphi_i|, \tag{37}$$

where n is the total number of qubits in the quantum state.

Figure 9 illustrates the circuit implementation for a NDD-based assertion. In this circuit, an auxiliary qubit is prepared in the $|+\rangle$ state and is used as a control to apply the unitary operator U on the tested state. It is important to note that the unitary U is similar to the identity matrix and does not alter the tested state. However, it introduces a phase of -1 on states that do not satisfy the assertion condition. The phase introduced by U causes the auxiliary qubit to transition to the $|-\rangle$ state. The last Hadamard gate then transforms the auxiliary qubit from $|-\rangle$ to $|1\rangle$. In this configuration, a measurement outcome of 1 on the auxiliary qubit represents an assertion error. On the other hand, when the tested quantum state satisfies the assertion condition, the auxiliary qubit remains in the state $|+\rangle$, and a measurement outcome of 0 is obtained.

The same considerations regarding time complexity and impact on NISQ devices that apply to OR-based assertions also hold true for NDD-based assertions.

5.1.5 Projection-based assertion

The work presented by Li et al. [54] utilizes the formalism of *Applied Quantum Hoare Logic*[55] to implement a projection-based assertion. This builds upon the previous works of Li and Ying [56]. The assertion method proposed relies on *projective measurements*, which possess the property of not disturbing the quantum state if the assertion succeeds. This approach offers several advantages. Firstly, it can reduce the number of auxiliary qubits required for the assertion, making the implementation

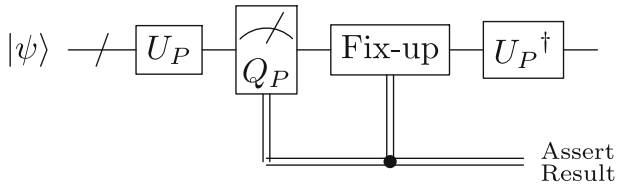


Fig. 10 The general schema for projection-based assertions with projector P on state $|\psi\rangle$

more efficient. Additionally, it enables the evaluation of multiple assertions during the same quantum execution, allowing corrective actions during execution. However, a limitation of this method is that it necessitates mid-circuit measurement, which is not widely available in existing quantum computers. Notwithstanding this drawback, the potential advantages it offers in terms of reducing auxiliary qubits render it a promising approach for future assertion implementations.

A projection-based assertion is defined by a projector P and the testing state $|\psi\rangle$. The assertion is considered successful if the application of the projector P to the testing state $|\psi\rangle$ results in the original state $|\psi\rangle$ itself. Mathematically, the condition for the assertion to succeed is given by $P|\psi\rangle = |\psi\rangle$.

We can create a set of measurement operators, denoted as $M_P = \{M_0 = P, M_1 = I - P\}$, derived from a projector P. Utilizing these measurement operators, we can effectively implement a projection-based assertion by conducting a measurement on the quantum state $|\psi\rangle$. The assertion's success or failure is determined based on the measurement outcome: obtaining the result associated with $M_0 = P$ confirms the assertion's success for the state $|\psi\rangle$. Conversely, if the measurement yields the outcome corresponding to M_1, the assertion is deemed unsuccessful for the state $|\psi\rangle$.

Although quantum computers are restricted today to measurements in the computational basis, Li et al. [54] presents a method to implement arbitrary projectors within these limitations. This technique allows for the implementation of projection-based assertions despite the constrained measurement basis and projector rank. An equivalent approach is also proposed by Liu and Zhou [50] for SWAP-based, OR-based, and NDD-based assertions.

Figure 10 illustrates a general schema for applying a projection-based assertion with a projector P on the state $|\psi\rangle$. Considering a projection P with rank 2^m (where m is a positive integer), we can find a unitary transformation U_P by diagonalizing P. The resulting transformation satisfies $U_P P U_P{}^\dagger = Q_P$, where Q_P is a projector in the computational basis. Consequently, it is possible to implement Q_P by performing measurements on the computational basis, which is available on current quantum computers.

When a projector does not fit to the form 2^m, Li et al. [54] defines two transformations to adapt any projection, which can be applied sequentially to handle specific mismatches. For a projection P on n qubits, the transformations are:

1. If the rank of P is not equal to 2^m for a non-negative integer m, and the rank of P is greater than 2^{n-1}, an auxiliary qubit is added.

2. If the rank of P is not equal to 2^m for a non-negative integer m, and the rank of P is less than 2^{n-1}, there exist projectors P_1 and P_2 where $P = P_1 \cap P_2$, allowing the assertion to be split into two.

The application of this projection-based assertion involves the following steps: first, apply the transformation U_P to the state $|\psi\rangle$, resulting in $U_P |\psi\rangle = |\psi_1\rangle$. Next, check if the assertion succeeds by measuring the state $|\psi_1\rangle$ with the measurement operators $\{M_0 = Q_P,\ M_1 = I - Q_P\}$. The assertion is successful if the measurement outcome is 0. If the assertion is indeed successful, then the quantum state after the measurement is $\left|\psi_{M_0}\right\rangle = Q_P |\psi_1\rangle = |\psi_1\rangle$ and the last step is to apply the unitary transformation ${U_P}^\dagger$ to return the state to its original form, ${U_P}^\dagger |\psi_1\rangle = |\psi\rangle$.

However, if the assertion fails, the measurement results in 1, and the quantum state after the measurement is $\left|\psi_{M_1}\right\rangle \neq |\psi_1\rangle$. Nevertheless, since $\left|\psi_{M_1}\right\rangle$ is in a classical state, we can optionally execute a fix-up code composed of Pauli X gates to rectify the state and continue with the execution.

We can calculate the probability of an assertion succeeding with the expression $\mathrm{tr}(P |\psi\rangle \langle\psi|)$. It is important to note that $\mathrm{tr}(P |\psi\rangle \langle\psi|) = 1$ only when $P |\psi\rangle = |\psi\rangle$, which means that false positives can occur. However, in the case of a false positive, the quantum state will collapse to a correct state, allowing the quantum computation to continue correctly.

Furthermore, when $1 - \mathrm{tr}(P |\psi\rangle \langle\psi|) < \epsilon$ for a small ϵ, the assertion may practically never fail. This small error in the assertion result may not be a significant problem, since it would have a minor impact on the final measurement probabilities. Therefore, in practice, a small deviation from a perfect assertion success probability may be acceptable and not significantly affect the overall results of the quantum computation.

Although most NISQ computers currently do not support mid-circuit measurement, this projection-based assertion method could become a suitable option in the future if this scenario changes. Its ability to correct the quantum state in the case of assertion errors allows for multiple assertion evaluations in a single quantum execution, which could significantly reduce the overall quantum debugging cost.

It is important to consider the time complexity associated with implementing projection-based assertions. The time complexity grows at least linearly with the number of qubits and can even become exponential in the worst-case scenario. As a result, while this assertion method's ability to correct the quantum state is valuable for debugging purposes, it may not be suitable for quantum error mitigation. Other assertion methods provide more efficient and scalable approaches for quantum error mitigation.

5.2 Quantum runtime assertion examples

While the quantum runtime techniques discussed in this survey may prove intractable in the general case, certain efficient and valuable assertions can still be evaluated. This subsection is dedicated to showcasing a range of quantum assertion implementations and exploring their utility and impact on quantum execution.

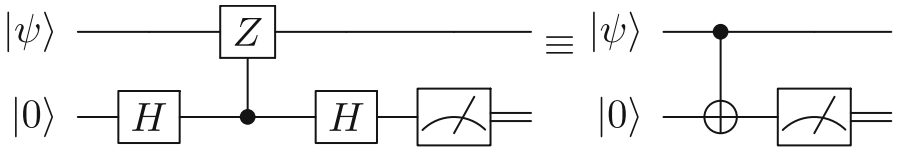


Fig. 11 Implementation of the NDD-based assertion for verifying classical states

5.2.1 Classical assertion

Asserting whether quantum bits are in a classical state, devoid of superposition equates to measuring them. Therefore, employing a statistical assertion aligns well with this scenario. Since the assertion expects the qubits to be in a classical state, performing a measurement will not disrupt the quantum computation. Preparing the anticipated classical state in the qubits incurs a linear time cost. However, in many quantum computer implementations, circuit measurements are not immediate, often performed at the circuit is end. This poses an issue for assertions that evaluate only a portion of the qubits.

For partial evaluations of the quantum system with classical assertions, a NDD-based assertion using the Pauli Z projector can be employed. Figure 11 illustrates this assertion's implementation. This approach can be likened to a partial copy, transferring the basis state from the qubit of interest to a zero-initialized qubit. It is essential to note that this is not a clone, since it does not replicate the amplitude probability. This methodology shifts all assertion measurements to the circuit is end, albeit at the expense of not reusing the auxiliary qubit for multiple assertions.

If the qubit of interest is not in a classical state, performing the asserting will entangle the auxiliary qubit. Consequently, measuring it will collapse the qubits of interest. As discussed in previous sections, this collapse may rectify the qubits into the correct state during the process. Therefore, combining this technique with measurement post-selection can help mitigate quantum noise.

5.2.2 Parity assertion

Bit parity involves determining whether the sum of the bits in a bit string is odd or even. It is a simple yet useful technique for error detection in classical computation and can be applied in quantum computation as well. For instance, quantum states like GHZ $= \frac{1}{\sqrt{2}}(|0\rangle^{\otimes 2n} + |1\rangle^{\otimes 2n})$ and W $= \frac{1}{\sqrt{n}} \sum_{k=0}^{n-1} |2^k\rangle$ possess a known parity. Evaluating the qubits' parity without disturbing the quantum state involves using an NDD-based assertion approach employing a $Z^{\otimes n}$ projector. The quantum circuit for a three-qubit parity assertion is depicted in Figure 12, illustrating how we can ascertain odd or even parity by altering the initial state of the auxiliary qubits.

The quantum state W $= \frac{1}{\sqrt{n}} \sum_{k=0}^{n} |2^k\rangle$ always exhibits odd parity, regardless of the number of qubits. Consider the example of a three-qubit W state $\frac{1}{\sqrt{3}}(|001\rangle + |010\rangle + |100\rangle)$, where each element in the superposition has only one qubit set to $|1\rangle$, resulting in odd parity for each element. Executing the circuit depicted in Figure 12 on the

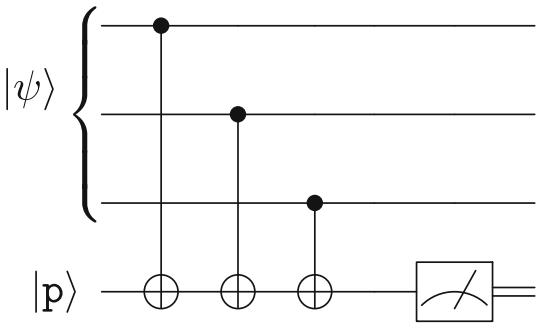


Fig. 12 NDD-based assertion implementation for three-qubit parity check, where p can be 0 or 1, representing even or odd parity, respectively

three-qubit W state with the auxiliary qubit initialized in $|1\rangle$ for odd parity will yield a final state of the auxiliary qubit as $|0\rangle$, indicating a successful assertion.

$$\frac{1}{\sqrt{3}}(|001\rangle + |010\rangle + |100\rangle)_{0,1,2} \otimes |1\rangle_{\mathrm{aux}} \xrightarrow{\mathrm{CNOT}_{0,\mathrm{aux}}} \frac{1}{\sqrt{3}}(|001\rangle\,|1\rangle + |010\rangle\,|1\rangle + |100\rangle\,|0\rangle) \tag{38}$$

$$\xrightarrow{\mathrm{CNOT}_{1,\mathrm{aux}}} \frac{1}{\sqrt{3}}(|001\rangle\,|1\rangle + |010\rangle\,|0\rangle + |100\rangle\,|0\rangle) \tag{39}$$

$$\xrightarrow{\mathrm{CNOT}_{2,\mathrm{aux}}} \frac{1}{\sqrt{3}}(|001\rangle + |010\rangle + |10\rangle) \otimes |0\rangle \tag{40}$$

However, while the GHZ and W are entangled states, the parity assertion cannot determine entanglement. For instance, quantum states like $\frac{1}{\sqrt{4}}\sum_{k=0}^{3}|2^k\rangle$, $\frac{1}{\sqrt{2}}(|1111\rangle + |0000\rangle)$, $|1111\rangle$, and $|1001\rangle$ all satisfy a four-qubit even parity assertion. Additionally, GHZ states with an odd number of qubits fail this assertion as they exhibit a state with all qubits at zero, resulting in even parity regardless of the qubit count. For instance, evaluating a three-qubit GHZ state $\frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$ using the quantum circuit in Figure 12 will leave the qubits entangled in the mixed state $\frac{1}{2}(|0\rangle\langle 0| + |1\rangle\langle 1|)$, causing interference with the intended state.

5.2.3 State preparation assertion

Asserting whether the state of a qubit set matches a desired state involves a time complexity akin to preparing the desired quantum state, which can be exponential in the worst case. Despite this complexity, such assertions retain significance and feasibility, especially within smaller qubit systems. For instance, leveraging a quantum algorithm to prepare an exact state [57] and subsequently testing it, while not the most efficient, can assist in identifying potential bugs in the implementation. Moreover, certain simple quantum state preparations have known state preparation circuits, such as the equal superposition state, as well as well-known states like the W and GHZ states.

Table 4 Comparison of Quantum Assertion Approaches

Runtime	Method	Expressivity	Feasibility on NISQ	Key Limiting Factor
Classical	Purity Assertion	Checks for the absence of entanglement	Low	Requires quantum runtime check for validation
Classical	QAI	Checks if a state is in a subspace of rank 1 or 2	N/A	Limited expressiveness
Quantum	Statistical	Infers superposition or entanglement from measurements	High	Requires many shots, susceptible to noise
Quantum	SWAP-Based	Checks for an exact state match or general subspace membership	Low	Exp. complexity, requires $O(n)$ auxiliaries
Quantum	OR/NDD-Based	Checks for an exact state match or general subspace membership	Medium	Exponential complexity
Quantum	Projection-Based	Checks for subspace membership using an arbitrary projector	Low	Exp. complexity, requires mid-circuit measurement

6 Conclusion

This survey delves into techniques for making assertions about quantum states to uncover bugs in quantum programs. We have categorized these techniques into classical and quantum runtime assertions, and a summary of our findings is presented in Table 4. It is important to note that due to significant limitations in time complexity and expressivity, the practical usage of these assertions is currently restricted.

Regarding implementation, most quantum programming platforms can support the various quantum runtime assertion methods. The primary exception is the projection-based assertion, which requires a platform to support mid-circuit measurement and conditional operations based on the measurement outcome. For classical runtime methods, support is currently limited to the specific solutions developed by the authors who proposed them.

Exploring classical runtime methods, we discuss the Twist quantum programming language and its purity assertion, a tool used to detect entanglement. Additionally, we present quantum abstract interpretation, enabling efficient execution of quantum applications on classical computers to evaluate assertions defined by projectors of rank 1 or 2. Although these avenues offer novel techniques for making assertions about quantum states and identifying bugs, this survey underscores that classical runtime assertion remains an open problem.

One potential technique for classical runtime assertions involves Clifford circuit simulators [58]. These circuits, limited in gate types, do not allow universal

quantum computation but permit efficient execution on classical computers without constraints on superposition and entanglement [59]. Another possible approach is utilizing ZX-calculus, a graph representation for quantum computation [60]. With certain restrictions, we can map quantum circuits to ZX-calculus and vice versa [61]. ZX-calculus enables quantum code optimizations and provides an alternative perspective on quantum circuits, which may aid in bug identification.

Empirical studies presented in this survey reveal that many quantum bugs arise from API misuse and incorrect numerical computations. This could be due to quantum programming platforms focusing on low-level quantum instructions like quantum gates and lacking standard data structures. Evolving these platforms to incorporate high-level quantum programming directives and standardized quantum state structures might mitigate the occurrence of quantum bugs and simplify bug identification. However, despite works addressing high-level quantum programming and quantum data structures [62, 63], this remains an open problem in need of further exploration.

While the Twist language executes the purifying-cast assertion at classical runtime, the complete implementation of this language proposal requires assertions that validate the existence of entanglement, which cannot be efficiently achieved in the general case on a classical computer. Hence, the full proposal implementation demands assertions that are executed at quantum runtime. Regarding quantum runtime assertions (QRAs), we presented five design methods and analyzed the scalability of each. Except for the statistical assertion, all these methods possess the same expressivity when asserting properties about the quantum state. However, they exhibit different trade-offs when implemented in quantum computers.

Due to their exponential time complexity in the general case, these quantum runtime assertions lack scalability. Nonetheless, specific implementations for particular aspects of quantum states could offer efficient avenues for investigation. Identifying efficient implementations tailored to specific aspects of quantum states could be a promising area of research. Such targeted implementations might mitigate the challenges posed by exponential complexities, offering manageable approaches for certain quantum state assertions.

In our survey, we identify no hybrid approaches for quantum assertion. The development of hybrid approaches that harness both classical and quantum runtimes for assertion validation is a potential area for exploration. By utilizing the classical runtime to validate a section of the assertion, we aim to streamline the process of evaluating assertions in the quantum runtime. This hybrid methodology has the potential to distribute computational tasks between classical and quantum systems, potentially optimizing the overall assertion validation process. Investigating and developing these hybrid strategies could yield efficient and scalable methods for verifying quantum states and detecting bugs in quantum programs.

In conclusion, while each assertion method presented offers unique benefits for quantum debugging, their suitability varies based on the quantum system's scale, complexity, and computational demands. As quantum computing continues to advance, future research must focus on optimizing these techniques, mitigating their limitations, and exploring novel assertion methodologies to ensure the accuracy and scalability.

In this survey, we focused on assertion methods tailored for gate-based quantum computers. These models, prevalent in leading quantum technologies such as

superconducting qubits and trapped ions, represent a universal quantum computation model. However, there exist other prominent quantum computing paradigms, such as integrated quantum photonic systems that rely on continuous-variable circuits [31], and quantum annealing computers [21, 64], specifically designed for optimization tasks. Addressing quantum debugging in these diverse quantum platforms presents a promising avenue for future works. Investigating available techniques and adapting gate-based assertions, if feasible, for these platforms will be essential for comprehensively tackling bugs and verifying quantum states across varied quantum computing architectures.

Acknowledgements This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior—Brasil (CAPES)—Finance Code 001.

Author Contributions E.R. drafted the main manuscript text under the supervision of R.S. All authors contributed to the review and editing of the manuscript.

Funding The Article Processing Charge (APC) for the publication of this research was funded by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) (ROR identifier: 00x0ma614).

Data Availability No datasets were generated or analyzed during the current study.

Declarations

Competing interests The authors declare no competing interests.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Feynman, R.P.: Simulating physics with computers. Int. J. Theor. Phys. **21**(6), 467–488 (1982). https://doi.org/10.1007/BF02650179
2. Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. In: Proceedings 35th Annual Symposium on Foundations of Computer Science, pp. 124–134 (1994). https://doi.org/10.1109/SFCS.1994.365700
3. Arute, F., Arya, K., Babbush, R., Bacon, D., Bardin, J.C., Barends, R., Biswas, R., Boixo, S., Brandao, F.G.S.L., Buell, D.A., Burkett, B., Chen, Y., Chen, Z., Chiaro, B., Collins, R., Courtney, W., Dunsworth, A., Farhi, E., Foxen, B., Fowler, A., Gidney, C., Giustina, M., Graff, R., Guerin, K., Habegger, S., Harrigan, M.P., Hartmann, M.J., Ho, A., Hoffmann, M., Huang, T., Humble, T.S., Isakov, S.V., Jeffrey, E., Jiang, Z., Kafri, D., Kechedzhi, K., Kelly, J., Klimov, P.V., Knysh, S., Korotkov, A., Kostritsa, F., Landhuis, D., Lindmark, M., Lucero, E., Lyakh, D., Mandrà, S., McClean, J.R., McEwen, M., Megrant, A., Mi, X., Michielsen, K., Mohseni, M., Mutus, J., Naaman, O., Neeley, M., Neill, C., Niu, M.Y., Ostby, E., Petukhov, A., Platt, J.C., Quintana, C., Rieffel, E.G., Roushan, P., Rubin, N.C., Sank, D., Satzinger, K.J., Smelyanskiy, V., Sung, K.J., Trevithick, M.D., Vainsencher, A., Villalonga, B., White, T., Yao, Z.J., Yeh, P., Zalcman, A., Neven, H., Martinis, J.M.: Quantum supremacy using a programmable superconducting processor. Nature **574**(7779), 505–510 (2019). https://doi.org/10.1038/s41586-019-1666-5

4. Madsen, L.S., Laudenbach, F., Askarani, M.F., Rortais, F., Vincent, T., Bulmer, J.F.F., Miatto, F.M., Neuhaus, L., Helt, L.G., Collins, M.J., Lita, A.E., Gerrits, T., Nam, S.W., Vaidya, V.D., Menotti, M., Dhand, I., Vernon, Z., Quesada, N., Lavoie, J.: Quantum computational advantage with a programmable photonic processor. Nature **606**(7912), 75–81 (2022). https://doi.org/10.1038/s41586-022-04725-x
5. Zhong, H.-S., Wang, H., Deng, Y.-H., Chen, M.-C., Peng, L.-C., Luo, Y.-H., Qin, J., Wu, D., Ding, X., Hu, Y., Hu, P., Yang, X.-Y., Zhang, W.-J., Li, H., Li, Y., Jiang, X., Gan, L., Yang, G., You, L., Wang, Z., Li, L., Liu, N.-L., Lu, C.-Y., Pan, J.-W.: Quantum computational advantage using photons. Science **370**(6523), 1460–1463 (2020). https://doi.org/10.1126/science.abe8770
6. Wu, Y., Bao, W.-S., Cao, S., Chen, F., Chen, M.-C., Chen, X., Chung, T.-H., Deng, H., Du, Y., Fan, D., Gong, M., Guo, C., Guo, C., Guo, S., Han, L., Hong, L., Huang, H.-L., Huo, Y.-H., Li, L., Li, N., Li, S., Li, Y., Liang, F., Lin, C., Lin, J., Qian, H., Qiao, D., Rong, H., Su, H., Sun, L., Wang, L., Wang, S., Wu, D., Xu, Y., Yan, K., Yang, W., Yang, Y., Ye, Y., Yin, J., Ying, C., Yu, J., Zha, C., Zhang, C., Zhang, H., Zhang, K., Zhang, Y., Zhao, H., Zhao, Y., Zhou, L., Zhu, Q., Lu, C.-Y., Peng, C.-Z., Zhu, X., Pan, J.-W.: Strong quantum computational advantage using a superconducting quantum processor. Phys. Rev. Lett. **127**, 180501 (2021). https://doi.org/10.1103/PhysRevLett.127.180501
7. Zhong, H.-S., Deng, Y.-H., Qin, J., Wang, H., Chen, M.-C., Peng, L.-C., Luo, Y.-H., Wu, D., Gong, S.-Q., Su, H., Hu, Y., Hu, P., Yang, X.-Y., Zhang, W.-J., Li, H., Li, Y., Jiang, X., Gan, L., Yang, G., You, L., Wang, Z., Li, L., Liu, N.-L., Renema, J.J., Lu, C.-Y., Pan, J.-W.: Phase-programmable gaussian boson sampling using stimulated squeezed light. Phys. Rev. Lett. **127**, 180502 (2021). https://doi.org/10.1103/PhysRevLett.127.180502
8. AbuGhanem, M.: IBM quantum computers: evolution, performance, and future directions. The Journal of Supercomputing 81(5) (2025) https://doi.org/10.1007/s11227-025-07047-7
9. AbuGhanem, M.: Google Quantum AI's Quest for Error-Corrected Quantum Computers (2024). https://doi.org/10.48550/arXiv.2410.00917
10. AbuGhanem, M., Eleuch, H.: Nisq computers: A path to quantum supremacy. IEEE Access **12**, 102941–102961 (2024). https://doi.org/10.1109/ACCESS.2024.3432330
11. AbuGhanem, M.: Characterizing grover search algorithm on large-scale superconducting quantum computers. Sci. Rep. **15**(1), 1281 (2025). https://doi.org/10.1038/s41598-024-80188-6
12. Qiskit contributors: Qiskit: An Open-source Framework for Quantum Computing (2023). https://doi.org/10.5281/zenodo.2573505
13. Svore, K., Geller, A., Troyer, M., Azariah, J., Granade, C., Heim, B., Kliuchnikov, V., Mykhailova, M., Paz, A., Roetteler, M.: Q#: Enabling scalable quantum computing and development with a high-level dsl. In: Proceedings of the Real World Domain Specific Languages Workshop 2018. RWDSL2018. Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3183895.3183901
14. Da Rosa, E.C.R., De Santiago, R.: Ket quantum programming. J. Emerg. Technol. Comput. Syst. 18(1) (2021) https://doi.org/10.1145/3474224
15. Preskill, J.: Quantum computing in the NISQ era and beyond. Quantum **2**, 79 (2018). https://doi.org/10.22331/q-2018-08-06-79
16. Jones, T., Brown, A., Bush, I., Benjamin, S.C.: Quest and high performance simulation of quantum computers. Sci. Rep. **9**(1), 10736 (2019). https://doi.org/10.1038/s41598-019-47174-9
17. Guerreschi, G.G., Hogaboam, J., Baruffa, F., Sawaya, N.P.D.: Intel quantum simulator: a cloud-ready high-performance simulator of quantum circuits. Quantum Science and Technology **5**(3), 034007 (2020). https://doi.org/10.1088/2058-9565/ab8505
18. Rosa, E.C.R., Taketani, B.G.: QSystem: bitwise representation for quantum circuit simulations (2020). https://doi.org/10.48550/arXiv.2004.03560
19. Gheorghiu, V.: Quantum++: A modern c++ quantum computing library. PLoS ONE **13**(12), 0208073 (2018). https://doi.org/10.1371/journal.pone.0208073
20. Ashktorab, Z., Weisz, J.D., Ashoori, M.: Thinking too classically: Research topics in human-quantum computer interaction. In: Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems. CHI '19, pp. 1–12. Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3290605.3300486
21. Rajak, A., Suzuki, S., Dutta, A., Chakrabarti, B.K.: Quantum annealing: an overview. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences 381(2241) (2022) https://doi.org/10.1098/rsta.2021.0417
22. AbuGhanem, M.: Photonic Quantum Computers (2024). https://doi.org/10.48550/arXiv.2409.08229

23. AbuGhanem, M.: Information processing at the speed of light. Frontiers of Optoelectronics **17**(1), 33 (2024). https://doi.org/10.1007/s12200-024-00133-3
24. Li, P., Liu, J., Li, Y., Zhou, H.: Exploiting quantum assertions for error mitigation and quantum program debugging. In: 2022 IEEE 40th International Conference on Computer Design (ICCD), pp. 124–131 (2022). https://doi.org/10.1109/ICCD56317.2022.00028
25. Zhao, J.: Quantum Software Engineering: Landscapes and Horizons (2021). https://doi.org/10.48550/arXiv.2007.07047
26. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information, 10th anniversary edition edn. Cambridge University Press, Cambridge (2010). https://doi.org/10.1017/CBO9780511976667
27. Dirac, P.A.M.: A new notation for quantum mechanics. Math. Proc. Cambridge Philos. Soc. **35**(3), 416–418 (1939). https://doi.org/10.1017/S0305004100021162
28. AbuGhanem, M., Eleuch, H.: Full quantum tomography study of google's sycamore gate on IBM's quantum computers. EPJ Quantum Technology **11**(1), 36 (2024). https://doi.org/10.1140/epjqt/s40507-024-00248-8
29. Paltenghi, M., Pradel, M.: Bugs in quantum computing platforms: An empirical study. Proc. ACM Program. Lang. 6(OOPSLA1) (2022) https://doi.org/10.1145/3527330
30. Luo, J., Zhao, P., Miao, Z., Lan, S., Zhao, J.: A comprehensive study of bug fixes in quantum programs. In: 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 1239–1246 (2022). https://doi.org/10.1109/SANER53432.2022.00147
31. Killoran, N., Izaac, J., Quesada, N., Bergholm, V., Amy, M., Weedbrook, C.: Strawberry fields: A software platform for photonic quantum computing. Quantum **3**, 129 (2019). https://doi.org/10.22331/q-2019-03-11-129
32. Farhi, E., Goldstone, J., Gutmann, S., Sipser, M.: Quantum Computation by Adiabatic Evolution (2000). https://doi.org/10.48550/arXiv.quant-ph/0001106
33. Farhi, E., Goldstone, J., Gutmann, S., Lapan, J., Lundgren, A., Preda, D.: A quantum adiabatic evolution algorithm applied to random instances of an np-complete problem. Science **292**(5516), 472–475 (2001). https://doi.org/10.1126/science.1057726
34. Developers, C.: Cirq. Zenodo (2023). https://doi.org/10.5281/zenodo.10247207
35. Steiger, D.S., Häner, T., Troyer, M.: Projectq: an open source software framework for quantum computing. Quantum **2**, 49 (2018). https://doi.org/10.22331/q-2018-01-31-49
36. Huang, Y., Martonosi, M.: Statistical assertions for validating patterns and finding bugs in quantum programs. In: Proceedings of the 46th International Symposium on Computer Architecture. ISCA '19, pp. 541–553. Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3307650.3322213
37. Zhao, P., Zhao, J., Ma, L.: Identifying bug patterns in quantum programs. In: 2021 IEEE/ACM 2nd International Workshop on Quantum Software Engineering (Q-SE), pp. 16–21 (2021). https://doi.org/10.1109/Q-SE52541.2021.00011
38. Zhong, H., Su, Z.: An empirical study on real bug fixes. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, pp. 913–923. IEEE, Florence, Italy (2015). https://doi.org/10.1109/ICSE.2015.101
39. Yuan, C., McNally, C., Carbin, M.: Twist: Sound reasoning for purity and entanglement in quantum programs. Proc. ACM Program. Lang. 6(POPL) (2022) https://doi.org/10.1145/3498691
40. Yu, N., Palsberg, J.: Quantum abstract interpretation. In: Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation. PLDI 2021, pp. 542–558. Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3453483.3454061
41. Chareton, C., Bardin, S., Bobot, F., Perrelle, V., Valiron, B.: An automated deductive verification framework for circuit-building quantum programs. In: Yoshida, N. (ed.) Programming Languages and Systems, pp. 148–177. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-72019-3
42. Hietala, K., Rand, R., Hung, S.-H., Li, L., Hicks, M.: Proving Quantum Programs Correct. LIPIcs, Volume 193, ITP 2021 193, 21–12119 (2021) https://doi.org/10.4230/LIPICS.ITP.2021.21
43. Zhou, L., Barthe, G., Strub, P.-Y., Liu, J., Ying, M.: Coqq: Foundational verification of quantum programs. Proc. ACM Program. Lang. 7(POPL) (2023) https://doi.org/10.1145/3571222
44. Chareton, C., Bardin, S., Lee, D., Valiron, B., Vilmart, R., Xu, Z.: Formal Methods for Quantum Programs: A Survey (2022). https://doi.org/10.48550/arXiv.2109.06493

45. Hayden, P., Milner, K., Wilde, M.M.: Two-message quantum interactive proofs and the quantum separability problem. In: 2013 IEEE Conference on Computational Complexity, pp. 156–167 (2013). https://doi.org/10.1109/CCC.2013.24
46. Boyland, J.: Checking interference with fractional permissions. In: Cousot, R. (ed.) Static Analysis, pp. 55–72. Springer, Berlin, Heidelberg (2003). https://doi.org/10.1007/3-540-44898-5
47. Heule, S., Leino, K.R.M., Müller, P., Summers, A.J.: Fractional permissions without the fractions. In: Proceedings of the 13th Workshop on Formal Techniques for Java-Like Programs. FTfJP '11. Association for Computing Machinery, New York, NY, USA (2011). https://doi.org/10.1145/2076674.2076675
48. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing. STOC '96, pp. 212–219. Association for Computing Machinery, New York, NY, USA (1996). https://doi.org/10.1145/237814.237866
49. Honarvar, S., Mousavi, M.R., Nagarajan, R.: Property-based testing of quantum programs in q#. In: Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops. ICSEW'20, pp. 430–435. Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3387940.3391459
50. Liu, J., Zhou, H.: Systematic approaches for precise and approximate quantum state runtime assertion. In: 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA), pp. 179–193 (2021). https://doi.org/10.1109/HPCA51647.2021.00025
51. Devitt, S.J., Munro, W.J., Nemoto, K.: Quantum error correction for beginners. Rep. Prog. Phys. **76**(7), 076001 (2013). https://doi.org/10.1088/0034-4885/76/7/076001
52. Jain, S., Muralidharan, S., Panigrahi, P.K.: Secure quantum conversation through non-destructive discrimination of highly entangled multipartite states. Europhys. Lett. **87**(6), 60008 (2009). https://doi.org/10.1209/0295-5075/87/60008
53. Liu, J., Byrd, G.T., Zhou, H.: 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. ASPLOS '20, pp. 1017–1030. Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3373376.3378488
54. Li, G., Zhou, L., Yu, N., Ding, Y., Ying, M., Xie, Y.: Projection-based runtime assertions for testing and debugging quantum programs. Proc. ACM Program. Lang. 4(OOPSLA) (2020) https://doi.org/10.1145/3428218
55. Zhou, L., Yu, N., Ying, M.: An applied quantum hoare logic. In: Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation. PLDI 2019, pp. 1149–1162. Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3314221.3314584
56. Li, Y., Ying, M.: Debugging quantum processes using monitoring measurements. Phys. Rev. A **89**, 042338 (2014). https://doi.org/10.1103/PhysRevA.89.042338
57. Araujo, I.F., Park, D.K., Petruccione, F., Silva, A.J.: A divide-and-conquer algorithm for quantum state preparation. Sci. Rep. **11**(1), 6329 (2021). https://doi.org/10.1038/s41598-021-85474-1
58. Aaronson, S., Gottesman, D.: Improved simulation of stabilizer circuits. Physical Review A 70(5) (2004) https://doi.org/10.1103/physreva.70.052328
59. Gottesman, D.: The Heisenberg Representation of Quantum Computers (1998). https://doi.org/10.48550/arXiv.quant-ph/9807006
60. Kissinger, A., Wetering, J.: Pyzx: Large scale automated diagrammatic reasoning. Electronic Proceedings in Theoretical Computer Science **318**, 229–241 (2020). https://doi.org/10.4204/eptcs.318.14
61. Duncan, R., Kissinger, A., Perdrix, S., Wetering, J.: Graph-theoretic simplification of quantum circuits with the zx-calculus. Quantum **4**, 279 (2020). https://doi.org/10.22331/q-2020-06-04-279
62. Yuan, C., Carbin, M.: Tower: data structures in quantum superposition. Proceedings of the ACM on Programming Languages **6**(OOPSLA2), 259–288 (2022). https://doi.org/10.1145/3563297
63. Bichsel, B., Baader, M., Gehr, T., Vechev, M.: Silq: A high-level quantum language with safe uncomputation and intuitive semantics. In: Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation. PLDI 2020, pp. 286–300. Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3385412.3386007
64. Pakin, S., Reinhardt, S.P.: A survey of programming tools for d-wave quantum-annealing processors. In: Yokota, R., Weiland, M., Keyes, D., Trinitis, C. (eds.) High Performance Computing, pp. 103–122. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-92040-5

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.