AssertsQ: A Quantum Assertion Tool for Quantum Software Debugging

Javier Ibarra Veganzones[1][a], Danel Arias Alamo[1][b], Alfredo Cuzzocrea[2][c] and Pablo García Bringas[1][d]

[1]*University of Deusto, Avda. Universidades 24, Bilbao, Spain*

[2]*University of Calabria, Rende, Cosenza, Italy*

Keywords: Quantum Computing, Quantum Software Testing, Debugging, Swap Test, Circuit Assertions, Quantum Verification.

Abstract: Quantum software debugging faces significant challenges due to measurement-induced state collapse, decoherence, and gate noise. We introduce AssertsQ, a framework that integrates assertion-based debugging into Qiskit, enabling automated verification of quantum circuits. AssertsQ provides three key validation methods: Swap Test-based state fidelity checks, measurement-based Total Variation Distance (TVD) comparisons, and noise-aware assertions adaptable to real hardware constraints. Our evaluation on benchmark circuits, including GHZ, QFT, and Grover's algorithm, demonstrates the framework's ability to detect errors and quantify fidelity degradation under noise. By simplifying quantum circuit verification, AssertsQ enhances the reliability of quantum software in the NISQ era.

1 INTRODUCTION

Quantum computing has gained significant momentum in recent years due to its potential to outperform classical computers in certain computational tasks, including cryptographic analysis, optimization, and simulation of quantum mechanical systems (Piattini et al., 2021). However, the process of verifying the correctness of quantum circuits remains a formidable challenge (Arias et al., 2023). Unlike classical debugging techniques, where the state of a program can be inspected at any point without altering the outcome, quantum states collapse upon measurement, and gate errors or qubit decoherence can irreversibly disrupt an entire computation (Metwalli, 2024). This intrinsic fragility necessitates robust verification tools tailored to the probabilistic and non-local nature of quantum phenomena.

In this paper, we introduce AssertsQ, a framework that provides easy-to-use assertion methods integrated into Qiskit (the IBM developed open-source quantum programming language). By leveraging the Swap Test for state overlap checks and a measurement-based approach for output distribution verification, AssertsQ aims to streamline the debugging and validation pipeline for quantum software developers. The sections below detail the motivation behind this work, the specific problems it addresses, the novel contributions we present, and the layout of the paper.

1.1 Motivation

As the field of quantum computing transitions from proof-of-concept experiments to the exploration of real-world applications, reliable circuit validation grows ever more critical (Arias et al., 2023). In classical computing, developers can inspect intermediate variables and step through code execution. In quantum computing, direct state inspections remain elusive unless one resorts to full state tomography, which becomes exponentially demanding with the number of qubits (Hoag et al., 2019). Consequently, many developers rely on partial tests or repeated sampling that provide incomplete insights into circuit correctness (Li et al., 2019). Furthermore, quantum hardware remains noisy, and the complex interplay of gate infidelities and cross-talk can produce misleading results if not properly accounted for in verification (Mundada et al., 2019). These concerns highlight the pressing need for a user-friendly framework that can automate

[a] https://orcid.org/0009-0007-7382-1300
[b] https://orcid.org/0000-0002-6586-346X
[c] https://orcid.org/0000-0002-7104-6415
[d] https://orcid.org/0000-0003-3594-9534

Veganzones, J. I., Alamo, D. A., Cuzzocrea, A. and Bringas, P. G.
AssertsQ: A Quantum Assertion Tool for Quantum Software Debugging.
DOI: 10.5220/0013553300004525
In *Proceedings of the 1st International Conference on Quantum Software (IQSOFT 2025)*, pages 49-60
ISBN: 978-989-758-761-0

routine checks of circuit fidelity and identify discrepancies early in the development cycle.

Verifying whether a quantum circuit performs as intended typically involves comparing its final state or measurement distribution against an expected result(Kang et al., 2024). As circuits scale in complexity, conventional debugging techniques do not readily extend to quantum contexts. The problem is exacerbated by limited qubit counts and high error rates in today's quantum processors, making repeated sampling or complicated protocols an inefficient endeavor (Arias et al., 2022). This exacerbates the necessity of a supporting tool that seamlessly integrates with existing frameworks and provides accessible implementations of established techniques, such as the Swap Test or statistical distance measures, to facilitate debugging and validation.

1.2 Contribution

This paper addresses the challenges above by presenting AssertsQ, a modular toolkit that embeds quantum circuit assertions directly into Qiskit-based workflows. A key contribution is the implementation of methods that allow developers to confirm circuit equivalence, validate fidelity against a known target state, and compare measurement output distributions. By unifying these functionalities into a single interface, we reduce the overhead of creating custom verification schemes. Another important feature is the seamless interoperability with IBM Quantum's cloud backends and local simulators, providing flexible noise modeling and pass manager (transpiler) configuration. These capabilities allow researchers and developers to iterate rapidly on quantum algorithms, verifying correctness under both ideal and realistic hardware conditions. Moreover, the introduction of adjustable tolerances ensures that the tool can accommodate noisy intermediate-scale quantum (NISQ) devices, making it a practical resource in the current quantum computing landscape. In this way, the tool presented in this work emerges as a tool that directly addresses the need for an integrated and efficient quantum circuit verification framework, bridging the gap between theoretical validation and practical implementation.

The remainder of the paper is organized as follows. Section 2 discusses the state of the art in quantum circuit verification, highlighting why traditional classical debugging methods might fall short for quantum software development, and introduces the theoretical underpinnings of the techniques used in this work. Section 3 details the methodology behind AssertsQ, explaining how the toolkit composes and executes specialized verification circuits. Section 4 delves into the implementation aspects, including class design, key methods, and workflow integration within Qiskit. In Section 5, we describe experimental results showcasing circuit verification in both simulated environments with noise simulation and perfect simulation, followed by a discussion of the practical advantages and limitations of the approach. Finally, in Section 6 we conclude with an overview of future directions, potential optimizations, and the broader significance of automated circuit assertions in quantum computing.

2 STATE OF THE ART

In recent years, quantum computing has advanced rapidly, promising computational advantages in areas such as optimization, cryptographic analysis, and quantum simulation (Piattini et al., 2021). However, verifying the correctness of quantum computations poses new challenges that classical methods cannot readily address (Arias et al., 2023). Gate noise, qubit decoherence, and the collapse of quantum states upon measurement all contribute to the difficulty of debugging quantum programs (Metwalli, 2024). Even straightforward techniques like inspecting intermediate states become prohibitive, since measurements irrevocably alter quantum superpositions (Hoag et al., 2019).

Much of the existing literature highlights two main strategies for quantum circuit verification: measurement-intensive approaches and direct fidelity or overlap checks. Measurement-intensive methods can involve state tomography or repeated sampling, but they tend to be resource-heavy and scale poorly with increasing qubit numbers (Guo and Yang, 2024). Partial tests have been introduced to mitigate complexity, yet these may fail to capture subtle deviations. By contrast, overlap checks through protocols like the Swap Test estimate how closely two states (or two circuits' outputs) coincide without requiring a full reconstruction of the state (Foulds et al., 2021).

While the Swap Test is a standard method for estimating state overlap, recent works have explored alternative approaches to reduce circuit depth and improve efficiency on near-term quantum devices. For instance, (Cincio et al., 2018) used machine learning to discover algorithms with shorter depths than the Swap Test for computing state overlap, including methods that use CNOT gates and achieve constant depth. Similarly, (García de la Barrera Amo et al., 2022)presented a method for automatic generation of test circuits that verify quantum deterministic algo-

rithms using a CNOT-based test for state comparison. Our framework, AssertsQ, complements these efforts by providing a user-friendly interface for integrating various assertion methods, including the Swap Test and measurement-based validations, directly into Qiskit workflows

Despite these efforts, no consensus exists on a universal debugging protocol for quantum software. Some investigators explore specialized frameworks that unify entanglement checks, circuit optimization passes, and classical post-processing, but many remain narrowly focused on either final-state fidelity (Ramalho et al., 2024) or distribution comparison in the computational basis (Hashim et al., 2023). The latter often involves metrics such as Total Variation Distance (TVD) (Dahlhauser and Humble, 2024), which can reveal discrepancies in measurement outcomes but generally overlook phase-sensitive information. Meanwhile, advanced noise modeling is increasingly recognized as crucial for capturing the performance of near-term quantum devices (Mundada et al., 2019), yet many verification proposals treat hardware imperfections only superficially or not at all.

Recent approaches have begun to incorporate quantum assertions, that is, specialized checkpoints within algorithms, into standard development workflows (Memon et al., 2024). These assertions aim to detect anomalies early by targeting either the amplitude overlap with an expected state or the probability distribution of measurement outcomes. Nevertheless, the difficulties posed by state collapse and limited qubit availability mean that robust frameworks for automated verification are still emerging. Tools must balance direct fidelity checks with measurement-based methods and accommodate the realistic noise models of modern quantum processors. Overcoming these constraints is pivotal to ensuring that quantum algorithms remain correct and resilient to hardware imperfections.

Besides assertion-based methods, other testing techniques have been adapted for quantum software. Metamorphic testing, which leverages relations between multiple program executions to verify correctness without an oracle, has been applied to quantum programs. For instance, MorphQ (Paltenghi and Pradel, 2023) uses metamorphic relationships to test the Qiskit platform, uncovering numerous bugs. More recently, (Pontolillo and Mousavi, 2024) introduced an automated debugging technique based on delta debugging and property-based testing for quantum programs, which identifies changes causing regression test failures. Similarly, (Abreu et al., 2022) propose metamorphic testing for oracle quantum programs to overcome the measurement problem. On the other hand, mutation testing, which assesses test suite quality by introducing artificial faults, has also been explored.(Fortunato et al., 2022) introduce QMutPy, a mutation testing tool for Qiskit, demonstrating its effectiveness on real quantum programs. Additionally, (Wang et al., 2022) present a multi-objective search approach for generating mutation-based tests for quantum programs. A recent empirical evaluation by (Usandizaga et al., 2023) on quantum circuit mutations provides insights into which mutants are most effective for testing quantum software. A comprehensive survey by (Paltenghi and Pradel, 2024) provides an overview of various testing and analysis techniques for quantum software, highlighting the growing interest and diversity in this field.

3 METHODOLOGY

Quantum circuits exhibit unique challenges absent in classical computing: measurement collapses superpositions, gate errors compound as circuits deepen, and interference can mask intermediate states. To address these concerns in a principled manner, we propose a methodology leveraging three main components: the Swap Test for circuit or state overlap, amplitude-based state initialization to compare against a known reference, and a measurement-based approach using the TVD metric. Collectively, these strategies tackle typical NISQ-era constraints such as limited qubits, high noise levels, and restricted opportunities for measurement, offering a flexible verification suite applicable to various quantum algorithms.

3.1 Circuit Assertion Using Swap Tests

The Swap Test has emerged as a powerful means to evaluate how closely two quantum states resemble each other without performing a full, resource-intensive state tomography. Conceptually, one appends an ancilla qubit to the two states under comparison and performs a conditional swap followed by a final measurement. The probability of measuring the ancilla in the $|0\rangle$ state corresponds directly to the squared overlap (or similarity) of the two states(see Appendix):

$$\text{Similarity} = 2 \times P(|0\rangle) - 1.$$

For circuit assertions, one can think of each quantum circuit as producing an output state. By treating these outputs as the two states in question, the Swap Test reveals whether the circuits behave equivalently

within a chosen fidelity threshold. This approach circumvents the need to measure and compare each qubit individually, thus minimizing both the measurement overhead and the risk of disturbing fragile superpositions. As a result, the Swap Test serves as a concise, powerful gauge for circuit-to-circuit or circuit-to-state equivalences.

3.1.1 Reconstructing and Comparing States

Beyond comparing two circuits, AssertsQ also evaluates a circuit's output against a known reference state vector. In such cases, one must prepare the target (reference) state vector as a second quantum system. Internally, Qiskit's amplitude-based approach as introduced in its `initialize` method(Iten et al., 2016) can create a circuit that produces the target state vector from $|0\rangle$. This target system is then brought into the Swap Test framework. Measuring how often the ancilla remains in $|0\rangle$ indicates whether the final circuit output matches the reference state's amplitudes. Crucially, this method does not necessitate direct tomography, making it more scalable to multi-qubit states.

3.2 Measurement-Based Validation: Total Variation Distance (TVD)

While the Swap Test reveals quantum overlaps, many practical algorithms are ultimately concerned with measurement outcomes. Moreover, in large multi-qubit circuits, duplicating circuit size might not be able due to the hardware capabilities.For these scenarios, AssertsQ provides a measurement-based validation that compares classical probability distributions through the TVD. Each circuit is executed to produce a bitstring distribution in the computational basis. Given two distributions P and Q, the TVD is defined as

$$\mathrm{TVD}(P,Q) \;=\; \tfrac{1}{2}\sum_{\omega}\big|P(\omega)-Q(\omega)\big|,$$

where ω ranges over all possible bitstrings. A TVD near zero signifies that both circuits produce near-identical distributions, whereas a larger TVD indicates significant discrepancies in measurement outcomes (Hashim et al., 2023). This approach excels in scenarios where the classical output is the main figure of merit, such as sampling-based or combinational quantum algorithms (Madsen et al., 2022).

3.3 Noise Handling and Tolerance

Near-term quantum devices—commonly referred to as Noisy Intermediate-Scale Quantum (NISQ) hardware—exhibit gate, readout, and decoherence errors that can obscure correct circuit behavior(Khan et al., 2024). Consequently, AssertsQ introduces tolerance thresholds in both the Swap Test and TVD calculations, providing a buffer against minor fluctuations. If the computed overlap (or distribution similarity) remains within the specified tolerance, the circuits or states are considered "equivalent" for practical purposes. At the same time, developers can opt for stricter thresholds when using higher-fidelity simulators or hardware calibrations.

Additionally, the tool accounts for three primary execution environments:

- **Noiseless Simulation:** Provides an ideal baseline, ignoring all hardware imperfections.
- **Noisy Simulation:** Incorporates realistic error models, thus approximating a real device's performance.
- **Real Hardware:** Executes the assertion on an actual quantum processor for definitive, real-world verification.

In all cases, careful transpilation ensures that the logical gates are mapped effectively to the device's physical qubits, factoring in connectivity constraints and gate availability. This interplay of error modeling, tolerance tuning, and backend selection yields a comprehensive strategy that mitigates noise while remaining flexible enough to handle evolving hardware conditions.

3.4 Experimental Setup for Tool Validation

To validate AssertsQ, we designed a set of experiments targeting canonical quantum algorithms of varying complexity: Greenberger-Horne-Zeilinger state (GHZ), quantum Fourier transform (QFT) and Grover search. Each algorithm was tested at two scales (5 and 10 qubits) to observe how circuit depth influences verification outcomes (see Section 5).

In both perfect and noisy simulation modes, GHZ circuits generate a maximally entangled state through a single Hadamard followed by a chain of CNOTs, while QFT circuits apply phase shifts and Hadamard gates in a pattern crucial for phase-estimation tasks. Grover's algorithm, meanwhile, demonstrates amplitude amplification and oracle-based state marking. For the noisy simulations, we employed a noise model generated from the calibrations of *ibm_sherbrooke*[1], mimicking realistic two-qubit gate error rates (e.g., $\approx 3.51 \times 10^{-3}$) as well as single-qubit and readout error parameters. Such a model allows us to emulate the

[1]This calibrations where retrieved from 3 Feb 2025 to 6 Feb 2025.

performance constraints of this 127-qubit device, capturing the influences of gate infidelity and crosstalk without requiring direct hardware access.

By comparing outcomes from noiseless vs. noisy simulations, and evaluating metrics such as fidelity (from the Swap Test) and TVD (from measurement-based checks), we benchmark both the robustness and granularity of our assertion methods. This layered experimental design reveals how AssertsQ behaves under increasing levels of circuit complexity and noise, highlighting its capacity to catch meaningful errors and degrade gracefully in the face of hardware imperfections. Lastly, these experiments are transpiled using the qiskit pass manager with a default optimization level of 3 and launched with 1024 shots.

4 IMPLEMENTATION

This section provides details on how AssertsQ is structured at the code level, illustrates typical usage scenarios, and demonstrates how developers can incorporate its capabilities into their existing Qiskit workflows. The implementation comprises a set of classes and functions that collectively offer circuit assertion functionalities based on both the Swap Test and measurement-based comparisons.

4.1 Code Overview

The core functionality of AssertsQ resides in the `Assertions` class. This class includes three primary methods for circuit validation: `assert_equals`, `assert_equals_state`, and `assert_equals_measure`. Each method addresses a different aspect of quantum circuit verification. The first two employ the Swap Test to measure circuit or state similarity, while the third applies a measurement-based approach, computing the TVD to determine equivalence of output distributions. All methods share a similar structure: they accept the circuits (or a circuit and target state), a tolerance parameter, and an `AssertionOptions` object that encapsulates backend selection, noise configuration, and custom transpilation pass managers. By default, the code sets up specialized quantum registers (including an ancilla qubit for the Swap Test) and composes a dedicated circuit to execute the test logic. Once the circuit is transpiled and submitted to a chosen backend via `SamplerV2`, the returned measurement data is interpreted to produce a fidelity metric or a distribution-based comparison result.

An auxiliary class, `AssertionOptions` which validates user-specified settings such as whether to run on simulated or real quantum hardware, whether noise modeling should be enabled, and which pass manager to invoke. If the user supplies invalid credentials or an incompatible configuration, the class raises appropriate errors or warnings. When the user does not specify a backend, it can attempt to automatically select an available device. This design ensures a clear separation between the high-level assertion logic and lower-level infrastructure concerns, including authentication and hardware interaction.

4.2 Simulation, Noise Modeling, and Hardware Execution

AssertsQ manages both ideal and noisy simulations through Qiskit Aer, providing a seamless way to test quantum circuits under realistic error conditions. In the default noiseless mode, it employs the standard Aer simulator to generate perfect results, establishing a baseline for fidelity and distribution checks. When noisy simulation is requested, AssertsQ loads an error model derived from calibration data (for example, gate and readout infidelities) using Qiskit Aer's noise modeling features. This allows developers to gauge algorithm performance under conditions that approximate actual hardware runs.

If real device execution is required, AssertsQ makes use of QiskitRuntimeService to authenticate with IBM Quantum backends. By switching its configuration to a non-simulator setting, AssertsQ retrieves the specified hardware or automatically selects one based on availability. The same transpilation routines and assertion methods apply: the code compiles the circuit for the chosen quantum device, submits a job via Qiskit's runtime, and interprets the measurement outcomes to yield fidelity metrics or total variation distance comparisons. This unified approach enables developers to validate circuits consistently across simulation and real-device runs, directly from the same assertion framework.

4.3 Sample Usage

In this subsection, we illustrate how AssertsQ can be seamlessly integrated into a typical Qiskit workflow to validate quantum circuit correctness using a Bell state as example see Figure 1.

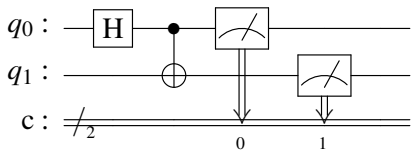

Figure 1: Bell state circuit.

To begin, a developer imports the required classes—namely, `Assertions` and `AssertionOptions`—and constructs one or more quantum circuits using Qiskit's standard API. For example, two Bell state circuits can be generated, and subsequently compared via AssertsQ's built-in methods. When invoking the `assert_equals` method , the framework automatically composes an ancillary circuit that embeds the swap test. This process involves appending an ancilla qubit, applying Hadamard gates, and executing controlled-swap operations on the qubits of the circuits under evaluation. The resulting measurement of the ancilla, combined with a user-defined tolerance parameter, produces a similarity score that confirms whether the circuits are functionally equivalent,as seen in Figure 2.

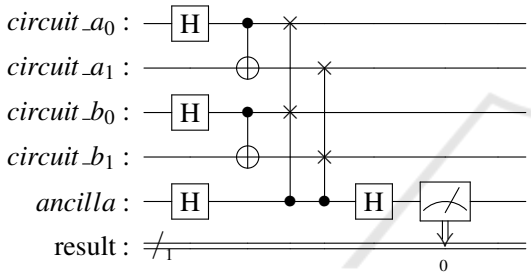

Figure 2: Bell state Assert Equals.

In another scenario , a circuit's final state is compared against a known target state vector through the `assert_equals_state` method. In this case, AssertsQ augments the original circuit by initializing a separate register to represent the reference state prior to performing the swap test. This approach enables a precise quantification of the overlap between the circuit's output and the intended state, yielding a fidelity measure that asserts correctness, as seen in Figure 3.

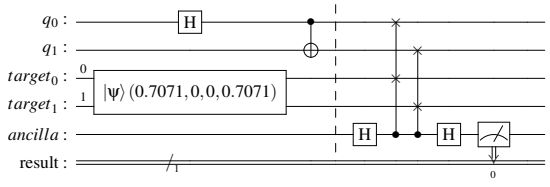

Figure 3: Bell state Assert Equals State.

For instances where the primary focus is on the final measurement outcomes, the `assert_equals_measure` method is employed. Here, see figure 4 , AssertsQ appends measurement operations to both distinct input circuits, collects the resultant bitstring counts, and computes the TVD between the corresponding probability distributions.

Throughout these examples, the user benefits from the option to enable verbose flag output, which prints

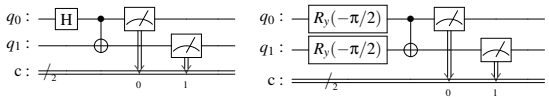

Figure 4: Comparing two distinct circuits with Assert Equals Measure.

intermediate metrics and final similarity scores. This feature proves invaluable for debugging purposes, as it provides immediate insight into the performance of the circuit assertions and facilitates iterative refinement.

4.4 Workflow Integration

Developers can seamlessly integrate AssertsQ into their standard Qiskit-based workflows. After building or modifying a quantum circuit, they invoke the desired assertion method, passing in the relevant `AssertionOptions` configuration. Under the hood, the code composes and transpiles an augmented circuit that embeds the Swap Test or measurement instructions. By leveraging Qiskit's `SamplerV2`, it supports both local simulators and remote quantum devices. The execution results are returned in a structured form, which AssertsQ analyzes to produce a definitive pass/fail statement along with a numerical similarity or distance metric. If an assertion fails due to exceeding the specified tolerance, developers immediately receive an exception with a detailed message describing the discrepancy, enabling rapid feedback loops for circuit debugging.

This integrated process reduces the burden of writing custom verification scripts and leverages established quantum programming paradigms. By automating tasks such as circuit compilation, backend orchestration, and result interpretation, AssertsQ ensures that development teams can focus on algorithm design and performance optimization, confident that any functional regressions or unintended circuit transformations will be detected promptly.

5 TOOL VALIDATION

5.1 Results

This section presents the main findings from running AssertsQ on three representative quantum algorithms (GHZ, QFT, and Grover) at different qubit scales (5 and 10) as presented in Section 3 As explained in the methodology, each circuit was tested in both noiseless and noisy simulation modes (*ibm_sherbrooke*-inspired model).

Table 1: Results for `assert_equals` with GHZ, QFT, and Grover Circuits.

Circuit	Qubits	Noiseless Fidelity	Noisy Fidelity
GHZ	5	1.0	0.4785
GHZ	10	1.0	0.1406
QFT	5	1.0	0.3633
QFT	10	1.0	0.04297
Grover	5	1.0	0.1269
Grover	10	1.0	0.0039

Table 1 reports the fidelity of GHZ, QFT, and Grover circuits when tested with `assert_equals`. The ideal (noiseless) scenario yields fidelity 1.0 across all circuits, confirming perfect overlap. Under simulated noise however, fidelity degrades significantly, especially for circuits with deeper gate sequences such as Grover of 10 qubits and QFT of 10 qubits.

Table 2: Results for `assert_equals_measure` with GHZ, QFT, and Grover Circuits.

Circuit	Qubits	Noiseless TVD	Noisy TVD
GHZ	5	0.0185	0.0361
GHZ	10	0.0263	0.0830
QFT	5	0.0996	0.1035
QFT	10	0.5332	0.5313
Grover	5	0.1201	0.0664
Grover	10	0.5292	0.5244

A measurement-based comparison via TVD (AssertsQ) was performed with the `assert_equals_measure` method. Table 2 shows how GHZ, QFT, and Grover circuits produce output distributions that remain close under ideal conditions but diverge in the presence of noise. For higher qubit counts, deeper gate depths lead to larger TVD values.

Table 3: Results for `assert_equals_state` with GHZ, QFT, and Grover Circuits.

Circuit	Qubits	Noiseless Fidelity	Noisy Fidelity
GHZ	5	1.0	0.3184
GHZ	10	1.0	*Incomplete*
QFT	5	1.0	0.2461
QFT	10	1.0	0.0039
Grover	5	1.0	0.2051
Grover	10	1.0	0.0332

Lastly, `assert_equals_state` was used to check whether a circuit's final state matched a known reference state. Table 3 reveals that noiseless fidelity remains at 1.0, whereas, as expected, noisy fidelity deteriorates sharply as the number of qubits and circuit depth increase (for instance, QFT with 10 qubits drops near 0.0039). The `assert_equals_state` noisy simulated experiment for GHZ with 10 qubits was not fully executed due to our local hardware limitations due to the large depth of the proper experiment, more info about the hardware conducted for the experiments see 6.

Table 4: Circuit depths before and after transpilation for all the techniques presented in this paper. (a) `assert_equals`, (b) `assert_equals_state`, (c) `assert_equals_measure`.

Circuit	Qubits	Before	After (a)	After (b)	After (c)
GHZ	5	5	223	454	25
GHZ	10	10	491	7095	33
Grover	5	13	893	770	493
Grover	10	13	7748	7689	6895
QFT	5	34	409	395	109
QFT	10	74	902	6543	423

To highlight the relationship between circuit depth and noise impact Table 4 track both the original depth (before embedding the swap test or measurement blocks) and the transpiled depth. For techniques requiring extra gates (for example, `assert_equals_state` needs reference-state initialization plus controlled swaps), the depth can multiply exponentially.

Figures 5, 6, and 7 provide a deeper look at how fidelity and total variation distance (TVD) vary with circuit depth and qubit count. In Figure 5, we observe that GHZ maintains comparatively higher fidelity than QFT and Grover when the circuit involves fewer qubits, but all three sharply decline at 10 qubits under noisy conditions. Figure 6 captures the significant jump in transpiled circuit depth (shown on a log scale) once additional gates for state assertion or measurement are introduced. GHZ, for instance, shows a notable increase in depth when asserting its state vector against a reference state vector, whereas direct measurement remains less expansive in gate count. Finally, Figure 7 illustrates how each technique's fidelity (top two plots) drops as circuit size grows, particularly for QFT and Grover at 10 qubits, and how the measurement-based TVD (bottom plot) escalates above 0.5 for larger circuits. Collectively, these plots mirror the tabulated results: circuits that are deeper or have more qubits become more susceptible to noise, and the choice of verification method (swap test, state assertion, or direct measurement) can greatly affect both circuit overhead and accuracy.

5.2 Discussion

The experimental results obtained from validating AssertsQ across the GHZ, QFT, and Grover algorithms

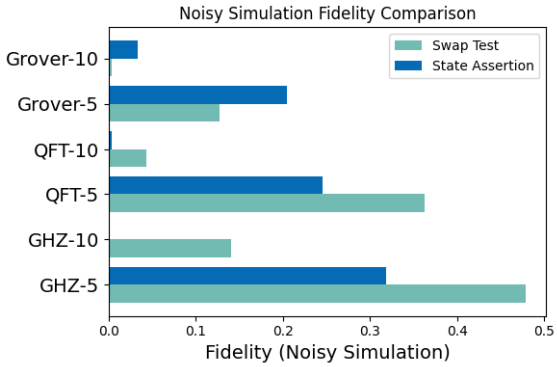

Figure 5: Fidelity Degradation with qubit count under noisy conditions for Swap Test vs. State Assertion Methods. The notation works as follows: "algorithm-number of qubits tested" (e.g. GHZ-10 means GHZ for 10 qubits).

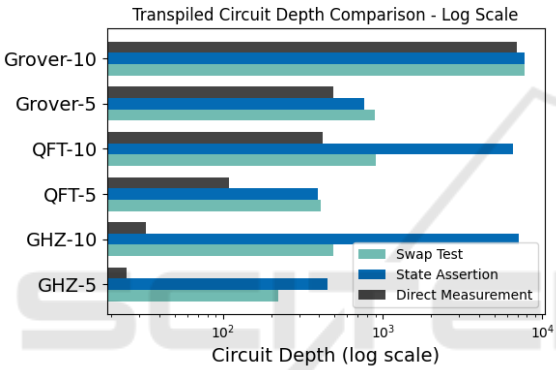

Figure 6: Transpiled Circuit Depth Comparison for the 3 studied methods: Direct Measurement, Swap Test, and State-Assertion Method. The notation works as follows: "algorithm-number of qubits tested" (e.g. GHZ-10 means GHZ for 10 qubits).

under both noiseless and noisy conditions reveal critical insights into the framework's performance and its susceptibility to key quantum circuit parameters: circuit depth and the number of qubits. These findings, as summarized in Tables1, 2, 3, and 4, alongside Figures5, 6, and 7, underscore the interplay between quantum circuit complexity, noise effects, and the choice of assertion method. Below, we provide a detailed analysis of the results and explain the underlying reasons for their observed trends.

In the noiseless simulation environment, all assertion methods consistently achieve perfect fidelity (1.0) or near-zero TVD across all tested algorithms and qubit scales. This is evident in Tables 1, 3, and 2 for the ideal cases. The perfect fidelity reflects the absence of noise, allowing the quantum states or output distributions to align precisely with their expected counterparts.

Under noisy conditions, the performace of AssertsQ degrades significantly, with fidelity decreasing

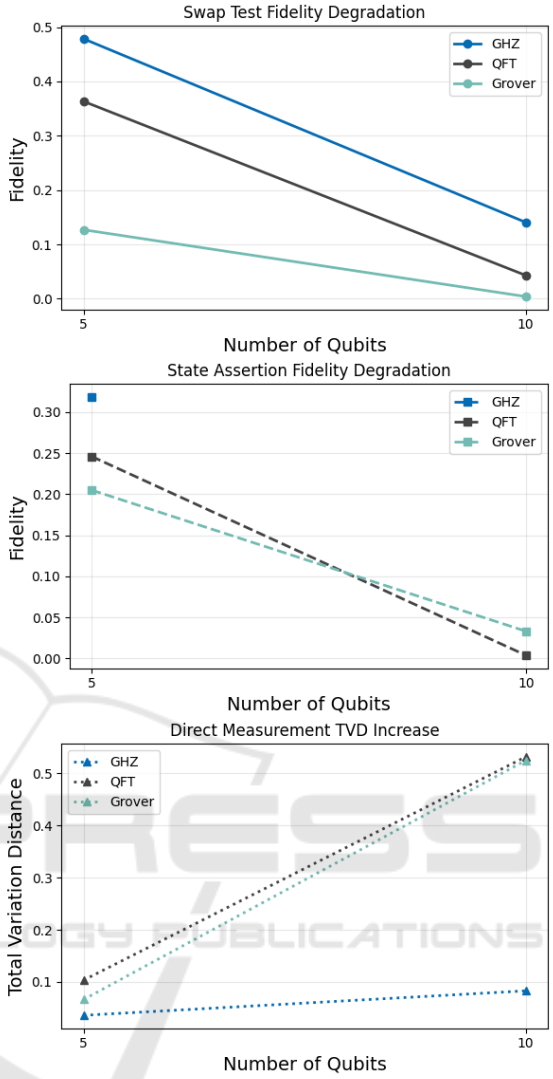

Figure 7: Fidelity and TVD Trends vs. Qubit Count.

and TVD increasing as circuit depth grows. This trend is most pronounced in algorithms with inherently deeper circuits, such as QFT and Grover, compared to the relatively shallow GHZ circuit. The primary reason for this degradation is the accumulation of errors for deeper cicuits. Each quantum gate introduces a small probability error, such as gate infidelity or decoherence in noisy environments. As circuit depth increases, the number of gates grows, amplyfing the cumulative error. For example, QFT involves a series of Hadamard and controlled-phase gates scaling with $O(n^2)$, while Grover's algorithm applies multiple iterations of the Grover operator, each adding gates proportional to the qubit count. In contrast, GHZ's depth scales linearly with n due to its single Hadamard followed by a CNOT chain, resulting in fewer opportunities for error accumulation. This depth-dependent er-

ror propagation explains why GHZ maintains higher fidelity under noise compared to QFT and Grover, as visualized in Figure 5.

The number of qubits further exacerbates performance degradation under noise. For swap test-based methods (`assert_equals` and `assert_equals_state`), requires $2n + 1$ qubits, where n is the number of qubits in the original circuit, plus one ancilla qubit. This doubling arises because, as stated before, the swap test compares two n-qubit states using controlled swap operations mediated by the ancilla. As n increases from 5 to 10, the total system size grows from 11 to 21 qubits, respectively.

As for `assert_equals_measure` avoids the swap test, using only n qubits and appending measurements, resulting in shallower circuits (e.g. 34 for 10-qubit GHZ), relying solely on classical post-processing. However, its TVD metric (Table2) captures only classical output distributions, missing quantum state details and yielding higher TVD values (e.g. 0.62 for 10-qubit QFT) under noise. Quantifying the precise level of improvement offered by a new tool within the evolving field of Quantum Software Engineering (QSE) is inherently complex, as noted by reviewers. However, the assertion framework presented here introduces tangible benefits to the development and verification workflow for quantum circuits.

We propose a tiered strategy for quantum circuit verification that accommodates varying hardware constraints, debugging goals, and developmental phases. Measurement-based verification (‘assert_equals_measure‘) serves as a practical starting point for rapid prototyping on NISQ devices or when usage quotas are limited, as demonstrated by the 34-gate requirement for verifying a 10-qubit QFT compared to the 1297 gates needed for swap-test-based approaches. Although this method is less sensitive to the full quantum state, it enables quick iterations for early-stage development. By contrast, the swap-test-based approach (‘assert_equals‘) strikes a middle ground, detecting subtler phase-sensitive errors such as those arising in QAOA and VQE while requiring 2n+1 qubits. This increased resource cost is justified by its ability to uncover mistakes that the measurement-based method might overlook. For applications needing the highest possible fidelity, exact state verification (‘assert_equals_state‘) delivers the strictest validation and is invaluable for final circuit confirmation and error-correcting code development. However, due to its steep computational expense, it is most appropriate for circuits with up to around ten qubits. By beginning with measurement-based checks, then adopting the swap test for phase-sensitive debugging, and ultimately employing exact state verification for final or mission-critical analyses, developers can balance resource expenditure with verification accuracy throughout the lifecycle of quantum application design.

Beyond its core validation capabilities, AssertsQ delivers critical advances to quantum software engineering by automating verification tasks—such as swap tests or custom comparison circuits—that would otherwise require ad hoc approaches, thereby substantially reducing verification time. Additionally, its standardized assertion interface fosters consistency and reusability across diverse algorithms (e.g., GHZ, QFT, Grover's), enabling a more systematic, test-driven workflow that integrates seamlessly with continuous integration pipelines. Finally, the framework's configurable tolerance thresholds align with NISQ-era hardware realities, allowing developers to incrementally adjust these thresholds based on hardware-specific noise and thus facilitate nuanced error analysis beyond binary pass/fail outcomes. This capability is especially beneficial when moving algorithms from ideal simulations to noisy devices, removing the need for separate noise analysis tools.

In addition to this, as discussed while noise does affect the fidelity of the assertions, particularly for deeper circuits, AssertsQ provides mechanisms such as adjustable tolerances and multiple verification methods to accommodate these challenges. In industrial settings, where quantum software development often involves iterative testing on simulators before hardware deployment, AssertsQ can play a crucial role in verifying circuit correctness at various stages. Furthermore, for smaller-scale circuits or those with manageable depths, the tool maintains sufficient fidelity to detect errors effectively. Thus, we contend that AssertsQ with slight improvements has indeed potential to be industrially applicable, offering a practical solution for quantum software debugging in the current NISQ era.

AssertsQ provides distinctive contributions to quantum circuit validation compared to existing frameworks AssertsQ offers a resource-efficient and integrated solution for quantum circuit validation. By using the swap test, it validates circuits with only 2n+1 qubits—far more scalable than measurement-intensive tomography (Yuen, 2023). Unlike frameworks limited to either state fidelity (Vadali et al., 2024) or distribution checks (Maciejewski et al., 2023), AssertsQ unifies both, crucial for detecting phase errors that TVD-based methods alone would miss. Its noise-aware design features configurable thresholds, balancing theoretical rigor with NISQ-era feasibility. Furthermore, direct integration with Qiskit's transpilation pipeline lowers the adoption

barrier, making AssertsQ a practical and scalable toolkit for quantum software verification.

In conclusion, AssertsQ effectively verifies quantum circuits in ideal conditions but reveals the challenges of noise, circuit depth, and qubit scaling in realistic settings. Its swap test-based methods, requiring $2n+1$ qubits,offer a compact alternative to tomography, yet their efficacy diminishes for large, deep circuits on NISQ devices whereas the measurement-based assertion offers a lighter alternative for verifying output distributions, though it sacrifices quantum state specificity.

These insights inform quantum software development by highlighting trade-offs between verification accuracy and practical feasibility, paving the way for future refinements in quantum debugging tools.

6 CONCLUSION

The development of AssertsQ demonstrates that automated, swap-test-based quantum assertions can be both practical and adaptable for a wide range of circuit verification tasks. By seamlessly integrating with Qiskit and supporting multiple verification modes—including direct circuit-to-circuit comparison, state overlap checks, and measurement-based distribution comparisons—AssertsQ simplifies the process of diagnosing and correcting circuit-level errors. The framework's reliance on either real or simulated backends, coupled with adjustable tolerance thresholds, accommodates both noisy intermediate-scale quantum devices and idealized simulation environments. Thus, aiding the circuit verification process and accelerating the development of new and accurate quantum software.

Experimental results show that each verification approach has distinct trade-offs in terms of circuit depth, susceptibility to noise, and granularity of the diagnostic information provided. Measurement-based checks via TVD are comparatively simpler but can overlook subtle state overlaps, while swap-test-based assertions offer deeper insight at the expense of additional qubits and gates. These extra resources become especially significant for multi-qubit or highly entangled circuits, where transpilation and noise accumulation pose notable challenges. Nonetheless, the ability of AssertsQ to capture discrepancies in both simulated and real-world conditions underscores its potential value in current and forthcoming quantum applications.

Although AssertsQ currently relies heavily on simulations, rigorous testing on real quantum hardware (e.g., IBM Quantum) is essential to confirm its noise resilience and empirically align simulation results with physical behavior. Extending cross-platform support to frameworks like Cirq and Amazon Braket will further broaden its applicability by accommodating different circuit representations and backends. Robust diagnostics also motivate the creation of assertion coverage metrics—akin to classical code coverage—to guide where and how often quantum assertions should be placed for maximum effect and therefore being capable of knowing how much these assertions could improve quantum computing software development. Coupling these metrics with IDE integration (e.g., VS Code) would enable live fidelity visualization, assertion breakpoints, and automated diagnostics within a familiar development workflow.

Since many assertions depend on ancillary qubits, minimizing their overhead and vulnerability to noise is critical; strategies such as probabilistic error cancellation (PEC), extrapolation, advanced transpiler optimizations, and careful qubit mapping can help preserve circuit fidelity. Finally, scaling to larger circuits (beyond 50 qubits) calls for approaches like partial tomography, circuit cutting, and adaptive protocols informed by calibration data or machine learning. These collective enhancements will strengthen AssertsQ as a robust debugging platform for near-term quantum systems and establish a foundation for enduring verification methodologies.

ACKNOWLEDGEMENTS

The authors would like to acknowledge the partial financial support by Ministry of Science (project QSERV-UD, PID2021-124054OB-C33). Additionally, Mr. Danel Arias thanks the Basque-Q strategy of Basque Government for partially funding his doctoral research at the University of Deusto, within the Deusto for Knowledge - D4K team on applied artificial intelligence and quantum computing technologies.

REFERENCES

Abreu, R., Fernandes, J. P., Llana, L., and Tavares, G. (2022). Metamorphic testing of oracle quantum programs. In *Proceedings of the 3rd International Workshop on Quantum Software Engineering*, pages 16–23.

Arias, D., García Rodríguez de Guzmán, I., Rodríguez, M., Terres, E. B., Sanz, B., Gaviria de la Puerta, J., Pastor, I., Zubillaga, A., and García Bringas, P. (2023). Let's do it right the first time: Survey on security concerns

in the way to quantum software engineering. *Neurocomputing*, 538:126199.

Arias, D., Sanz, B., de la Puerta, J. G., Pastor, I., and Bringas, P. G. (2022). A repeated mistake is a choice: Considering security issues and risks in quantum computing from scratch. In *14th International Conference on Computational Intelligence in Security for Information Systems and 12th International Conference on European Transnational Educational (CISIS 2021 and ICEUTE 2021) 14*, pages 156–166. Springer.

Cincio, L., Subaşı, Y., Sornborger, A. T., and Coles, P. J. (2018). Learning the quantum algorithm for state overlap. *New Journal of Physics*, 20(11):113022.

Dahlhauser, M. L. and Humble, T. S. (2024). Benchmarking characterization methods for noisy quantum circuits. *Physical Review A*, 109(4):042620.

Fortunato, D., Campos, J., and Abreu, R. (2022). Mutation testing of quantum programs: A case study with qiskit. *IEEE Transactions on Quantum Engineering*, 3:1–17.

Foulds, S., Kendon, V., and Spiller, T. (2021). The controlled swap test for determining quantum entanglement. *Quantum Science and Technology*, 6(3):035002.

García de la Barrera Amo, A., Serrano, M. A., García Rodríguez de Guzmán, I., Polo, M., and Piattini, M. (2022). Automatic generation of test circuits for the verification of quantum deterministic algorithms. In *Proceedings of the 1st International Workshop on Quantum Programming for Software Engineering*, pages 1–6.

Garcia-Escartin, J. C. and Chamorro-Posada, P. (2013). Swap test and hong-ou-mandel effect are equivalent. *Physical Review A—Atomic, Molecular, and Optical Physics*, 87(5):052330.

Guo, Y. and Yang, S. (2024). Scalable quantum state tomography with locally purified density operators and local measurements. *arXiv preprint arXiv:2307.16381*.

Hashim, A., Seritan, S., Proctor, T., Rudinger, K., Goss, N., Naik, R. K., Kreikebaum, J. M., Santiago, D. I., and Siddiqi, I. (2023). Benchmarking quantum logic operations relative to thresholds for fault tolerance. *npj Quantum Information*, 9(1):109.

Hoag, E., Zhu, M., and Decker, S. (2019). qdb: Inserted tomography for breakpoint debugging in.

Iten, R., Colbeck, R., Kukuljan, I., Home, J., and Christandl, M. (2016). Quantum circuits for isometries. *Physical Review A*, 93(3):032318.

Kang, C. G., Lee, J., and Oh, H. (2024). Statistical testing of quantum programs via fixed-point amplitude amplification. *Proceedings of the ACM on Programming Languages*, 8(OOPSLA2):140–164.

Khan, M. U., Kamran, M. A., Khan, W. R., Ibrahim, M. M., Ali, M. U., and Lee, S. W. (2024). Error mitigation in the nisq era: Applying measurement error mitigation techniques to enhance quantum circuit performance. *Mathematics*, 12(14):2235.

Li, G., Zhou, L., Yu, N., Ding, Y., Ying, M., and Xie, Y. (2019). Proq: Projection-based runtime assertions for debugging on a quantum computer. *arXiv preprint arXiv:1911.12855*.

Maciejewski, F. B., Puchała, Z., and Oszmaniec, M. (2023). Operational quantum average-case distances. *Quantum*, 7:1106.

Madsen, L. S., Laudenbach, F., Askarani, M. F., Rortais, F., Vincent, T., Bulmer, J. F., Miatto, F. M., Neuhaus, L., Helt, L. G., Collins, M. J., et al. (2022). Quantum computational advantage with a programmable photonic processor. *Nature*, 606(7912):75–81.

Memon, Q. A., Al Ahmad, M., and Pecht, M. (2024). Quantum computing: navigating the future of computation, challenges, and technological breakthroughs. *Quantum Reports*, 6(4):627–663.

Metwalli, S. A. F. M. (2024). A suite for testing and debugging quantum programs.

Mundada, P., Zhang, G., Hazard, T., and Houck, A. (2019). Suppression of qubit crosstalk in a tunable coupling superconducting circuit. *Physical Review Applied*, 12(5):054023.

Paltenghi, M. and Pradel, M. (2023). Morphq: Metamorphic testing of the qiskit quantum computing platform. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 2413–2424. IEEE.

Paltenghi, M. and Pradel, M. (2024). A survey on testing and analysis of quantum software. *arXiv preprint arXiv:2410.00650*.

Piattini, M., Peterssen, G., and Pérez-Castillo, R. (2021). Quantum computing: A new software engineering golden age. *ACM SIGSOFT Software Engineering Notes*, 45(3):12–14.

Pontolillo, G. J. and Mousavi, M. R. (2024). Delta debugging for property-based regression testing of quantum programs. In *Proceedings of the 5th ACM/IEEE International Workshop on Quantum Software Engineering*, pages 1–8.

Ramalho, N. C., Amario de Souza, H., and Lordello Chaim, M. (2024). Testing and debugging quantum programs: The road to 2030. *ACM Transactions on Software Engineering and Methodology*.

Usandizaga, E. M., Yue, T., Arcaini, P., and Ali, S. (2023). Which quantum circuit mutants shall be used? an empirical evaluation of quantum circuit mutations. *arXiv preprint arXiv:2311.16913*.

Vadali, A., Kshirsagar, R., Shyamsundar, P., and Perdue, G. N. (2024). Quantum circuit fidelity estimation using machine learning. *Quantum Machine Intelligence*, 6(1):1.

Wang, X., Yu, T., Arcaini, P., Yue, T., and Ali, S. (2022). Mutation-based test generation for quantum programs with multi-objective search. In *Proceedings of the genetic and evolutionary computation conference*, pages 1345–1353.

Yuen, H. (2023). An improved sample complexity lower bound for (fidelity) quantum state tomography. *Quantum*, 7:890.

APPENDIX

Mathematical Derivation of the Swap Test Similarity

In this subsection the derivation that underpins the functionality of methods like `assert_equals` and `assert_equals_state` in AssertsQ is presented. The measured probability P_0 (see below) is translated to a numerical *similarity* or *fidelity* metric, thereby enabling automated verification of circuit or state equivalence. The following derivation is based on the state of art (Garcia-Escartin and Chamorro-Posada, 2013) and (Foulds et al., 2021).

Definition 1 (Quantum States). Let $|\psi\rangle$ and $|\phi\rangle$ be two quantum states defined on an n-qubit Hilbert space, and let the ancilla qubit be initialized in the state $|0\rangle$.

Problem Statement. The objective of the Swap Test is to estimate the squared overlap (or *similarity*) between these two states ($|\psi\rangle$ and $|\phi\rangle$) by measuring a single ancilla qubit:

$$\left|\langle\psi \mid \phi\rangle\right|^2$$

Proposition 1 (Swap Test Fidelity). After applying a sequence of operations to the system, the probability P_0 of measuring the ancilla in state $|0\rangle$ is related to the squared overlap of $|\psi\rangle$ and $|\phi\rangle$ by:

$$\left|\langle\psi \mid \phi\rangle\right|^2 = 2P_0 - 1.$$

Proof. We apply the *Swap Test circuit* and track the system step by step.

1. *Initial state*

$$|0\rangle_{\text{ancilla}} \otimes |\psi\rangle \otimes |\phi\rangle.$$

2. *Hadamard on the Ancilla.*
The Hadamard transform acts as follows:

$$H|0\rangle_{\text{ancilla}} = \frac{1}{\sqrt{2}}\Big(|0\rangle_{\text{ancilla}} + |1\rangle_{\text{ancilla}}\Big).$$

The system state becomes:

$$\frac{1}{\sqrt{2}}\Big(|0\rangle_{\text{ancilla}} + |1\rangle_{\text{ancilla}}\Big) \otimes |\psi\rangle \otimes |\phi\rangle.$$

3. *Controlled*-SWAP *Operation.*
The controlled-SWAP swaps $|\psi\rangle$ and $|\phi\rangle$ **only if** the ancilla is in $|1\rangle$, producing:

$$\frac{1}{\sqrt{2}}\Big[|0\rangle_{\text{ancilla}}|\psi\rangle|\phi\rangle + |1\rangle_{\text{ancilla}}|\phi\rangle|\psi\rangle\Big].$$

4. *Second Hadamard on the Ancilla.*
Applying H again:

$$H|0\rangle = \frac{1}{\sqrt{2}}\Big(|0\rangle + |1\rangle\Big), \quad H|1\rangle = \frac{1}{\sqrt{2}}\Big(|0\rangle - |1\rangle\Big).$$

Expanding this transformation:

$$\frac{1}{2}\Big[|0\rangle_{\text{ancilla}}(|\psi\rangle|\phi\rangle + |\phi\rangle|\psi\rangle) + |1\rangle_{\text{ancilla}}(|\psi\rangle|\phi\rangle - |\phi\rangle|\psi\rangle)\Big].$$

5. *Measurement of the Ancilla.*
The probability of measuring $|0\rangle_{\text{ancilla}}$ is given by the squared norm of its coefficient:

$$P_0 = \frac{1}{4}\Big\| |\psi\rangle|\phi\rangle + |\phi\rangle|\psi\rangle\Big\|^2.$$

Using the identity:

$$\||x\rangle + |y\rangle\|^2 = \langle x \mid x\rangle + \langle y \mid y\rangle + 2\text{Re}\langle x \mid y\rangle.$$

Note that:

$$\langle\psi \mid \psi\rangle = \langle\phi \mid \phi\rangle = 1.$$

Thus:

$$P_0 = \frac{1}{4}\Big(2 + 2\text{Re}\langle\psi \mid \phi\rangle\Big) = \frac{1 + \text{Re}(\langle\psi \mid \phi\rangle)}{2}.$$

6. *Final Expression.*
For **pure states**, it holds that:

$$P_0 = \frac{1}{2}\Big(1 + \left|\langle\psi \mid \phi\rangle\right|^2\Big).$$

Solving for $\left|\langle\psi \mid \phi\rangle\right|^2$:

$$\left|\langle\psi \mid \phi\rangle\right|^2 = 2P_0 - 1.$$

□

Hardware Used for the Experiments

All the experiments described in this paper were carried out on a system running Microsoft Windows 10 Pro. The system was an HP ZBook Firefly 16 inch G10 Mobile Workstation PC, equipped with a 13th Gen Intel(R) Core(TM) i7-1365U processor at 1.8 GHz, featuring 10 physical cores and 12 logical processor and 32.0 GB of RAM.

Code Availability

The code used to implement the tool in this work is available at the AssertsQ GitHub repository[2]. The repository includes all relevant scripts necessary to replicate the results presented in this paper. For questions regarding the code or its application, please contact the corresponding author at javi.ibarra@deusto.es.

[2]Repository available in https://github.com/DeustoTech/AssertsQ.