Current Research in Next Generation Materials Engineering

PRIME OPEN ACCESS

Volume 1, Issue 1

Research Article

Date of Submission: 12 February, 2025
Date of Acceptance: 21 April, 2025
Date of Publication: 23 April, 2025

Quantum Machine Learning Techniques Integrated in Quantum Software Testing

Bala Gangadhara Gutam* and Sunil Kumar Malchi

Computer Science and Engineering, Mohan Babu University, Sri Sainagar, Tirupati, 517102, Andhra Pradesh, India

***Corresponding Author:**
Bala Gangadhara Gutam, Computer Science and Engineering, Mohan Babu University, Sri Sainagar, Tirupati, 517102, Andhra Pradesh, India.

Citation: Gutam, B. G., Malchi, S. K. (2025). Quantum Machine Learning Techniques Integrated in Quantum Software Testing. *Curr Res Next Gen Mater Eng, 1*(1), 01-09.

Abstract
Quantum computing (QC) is the era of the mechanisms that quantum solves complex problems faster than classical computers (CCs). Today, there are more opportunities for the public and private sectors to use high-value quantum solutions. QC relies on quantum phenomena to calculate incredible speeds and set for significant expansion. Quantum computers, QC-based Internet of Things (IoT), and communication devices to create, process, and transmit quantum states and entanglement are anticipated to enhance society's quantum software. In this context, this paper introduces the Quantum Software Testing (QST) approach with Machine Learning (ML) techniques integrated into the Quantum Software Testing Life Cycle (QSTLC). The proposed Quantum Machine Learning Testing (QMLT) exploits ML techniques at critical stages of the QST to enhance testing efficiency, accuracy, and adaptability. Finally, the proposed automated test report generation, summarizing, and knowledge extraction technique facilitates comprehensive documentation and knowledge transfer.

Keywords: Quantum Computing, Software Testing Life Cycle, Software Development Phases, Machine Learning

Introduction
The Quantum Computing (QC) revolution starts in the 21st century. In the innovation technology of the computing era, Quantum Machines (QM) compute the data and perform various tasks faster than the binary digits of classical electric computing [1]. Quantum software development (QSD) involves designing quantum programs (QPs), using quantum applications, quantum software testing (QST), and maintaining quantum software (QS). QSD evolves into incremental, iterative, and agile software engineering, just as classical software engineering [2]. SDLC is required to develop QS encompassing the various QSD phases and build reliable software. QS includes communication, planning, modeling, construction, testing, deployment, and maintenance. The testing phase includes the test plan, test case development, execution, result analysis, tracking, and status report generation. The QSD team was used to design a QS with the help of the procedures, tools, and approaches required.

In our work, we initially integrate Machine learning (ML) into the QSTLC within the QC environment [3]. We introduce a novel technique applied to ML algorithms to enhance and optimize the testing for QS. Significantly enhance the ML techniques for testing in QC that address the unique challenges of error correction and detection, noise characterization, verification, and validation quantum algorithms (QA) and systems [4]. Initially, ML algorithms can analyze quantum errors (QEs) and improve error detection and correction strategies in QE correction codes [5]. ML algorithms optimize the design and implementation of QE strategies to analyze error rate, quantum system parameters, and noise characteristics. For instance, optimize the error correction iterative to minimize the QE propagation and maximize the QA effectiveness [6]. QS suggests various contributes to improve the QC design and testing aspects.

• The proposed QMLT performs faster error detection and trains on vast quantum operations and outcomes datasets.
• The QP specifications, previous test cases, and QMLT can generate new test cases that cover all QS requirements to ensure efficiency and reliability.
• The QMLT is applied to automatically optimize quantum circuits, construct them from past circuits, and improve performance.
• To compare various metrics with existing testing techniques, including CST using QMLT in a QST environment.

The remaining part of the paper is organized as follows. Section 2 describes the related works. Section 3 presents the technologies. Section 4 depicts the system model and design phases with illustrations. Performance comparisons of the QMLT are evaluated in Section 5, and concluding remarks are given in Section 6.

Background Work

In this section, we overview and analyze the QC environment, which means quantum characteristics, quantum computer processes related to the qubit, its state's relevant changes, QEs, the nature of QS, quantum gates (QG), and quantum circuits [5-7]. The QC technology has particular potential for optimizing issues. The software testing technique for quality control differs from classical computing [8]. QC operates on a different mechanism and is well suited to solving computationally demanding societal and industrial issues.

QC includes drug discovery and vaccine development, finance optimization, portfolio management, and physics-related complex simulations in the medical field. The success of QC significantly impacts daily lives and replaces the majority of industries in a specific field. QS is Required to support a QS stack comprising the operating systems, compilers, and programming languages [9]. The properties of QC, such as superposition and entanglement, cannot be used in QS applications. Furthermore, the QAs require coding QPs because of the entirely different programming mindset based on quantum ideas. The software developers face several difficulties.

Unique Quantum Characteristics

A sensitive information-carrying unit in quantum computers is the qubit, which can be shown as the QM generation of a bit used in classical computing. The qubit is a 2-dimensional quantum state space. The probability of the current state is a superposition.

Superposition (ψ)

Quantum bits or qubits exist in different states simultaneously, and the superposition is identified [10]. A qubit state is mathematically expressed as:
$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$
where $|\alpha|^2 + |\beta|^2 = 1$, and $|0\rangle$ and $|1\rangle$ are the basis states.

Entanglement (Ψ)

The entangled is a qubit. One qubit's state depends on another's state, But it is spatially separated. An entangled state is mathematically represented as:

$$|\Psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

Quantum Parallelism

• QAs process with multiple inputs simultaneously and faster computations exploiting in parallel.
• QGs and unitary transformation are used to represent quantum parallelism mathematically.

Quantum Error Correction

Quantum Error Correction Code (QECC)

QECC is [7,1,3] Steane code protected quantum information from errors [11-17]. The [7,1,3] Steane code mathematically corrects up to one QE or arbitrary phase error using seven qubits.

Validation of Quantum Properties

Quantum Coherence (ρ)

The coherence defines the continuity of the phase relation between quantum states. Coherence is mathematically computed using the density parameters represented as ρ, which are evaluated based on the Schrodinger equation:

$$\frac{d\rho}{dt} = -i\frac{1}{\hbar}[H, \rho]$$

Quantum Entanglement Entropy (S)

The entropy measures the degree of entanglement between qubits in a quantum system. It mathematically computes by using the Von Neumann formula:

$S = -\mathrm{Tr}(\rho A \log \rho A)$

where ρA is the density reduced matrix of the subsystem A.

Dynamic Nature of Quantum Systems

Time-Evolution Operator (U(t))

Quantum systems develop over time, consisting of the time evolution operator $U(t)$, which describes the quantum states' unitary transformation. The operator $U(t)$ mathematically represented as the Schrodinger equation:

$$i\hbar\frac{d\,|\psi(t)\rangle}{dt} = H\,|\psi(t)\rangle$$

Quantum Gates

In a digital circuit, a logic gate can change the state of a bit; similarly, a QG can change the state of a qubit. A QG can transit a single quantum state (one input and output) or multiple states (multiple inputs and outputs). With an equal number of inputs and outputs in QG to perform a reverse transition, no information is lost in QC. QGs are classified as having one input and output or multiple inputs and outputs. First, the NoT Gate is used for one input, exchanging the coefficients of two primary vectors. The second Hadamard Gate is also used for one input; the existing quantum

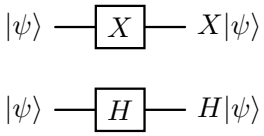

Figure 1: Single-Qubit Gates Circuit States Decompose According to Coefficients

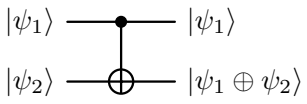

Figure 2: Two-Qubit Gate Circuit

The third, controlled-NOT Gate is used for two inputs: the control qubit and the target qubit. These are various QGs involved in changing the state of a qubit in quantum systems.

Quantum Circuits

Quantum circuits are called quantum logic circuits commonly used in QC models, which are abstract representations of circuits that function on qubits. A group of Connected QGs are known as quantum circuits. The unitary transformation carried out by the quantum circuits determines the connection scheme, the quantity and kind of gates, and the physical structure of the quantum circuits. Quantum measurements can be used to read out the output of a quantum circuit.

Important Technologies

In this section, we discuss the details of QC technologies in automated smart cities and explore QC-based IoT applications in urban areas. Smart cars evolved significantly through QC technologies, optimizing road safety and fuel consumption via IoT integration. QC facilitates the development of self-driving vehicles, thus reducing accidents and congestion on intelligent roads. Competent healthcare undergoes a transformation shift with QC-based IoT devices [18]. These QCs are integral in shaping a more efficient, sustainable, and interconnected future across industries.

Automated Smart City

QC-based IoT applications based on quality control will be the fully connected and automated intelligent cities of the future. This complex system manages various aspects such as energy generation and distribution, waste management, pedestrian and vehicular traffic, electricity, and atmospheric regulation: increased urban population and the impact of climate change on ecosystems. The cities of the future could help their residents sustain a high quality of life amidst the challenges by increasing their populations.

Smart Cars

QC in Smart Cars Smart roads can eliminate road accidents, optimize consumption to improve efficiency, and integrate self-driving smart cars into QC-based IoT devices. QC-based IoTs reduce congestion and become feasible. QC proposed a cross-scenario autonomous driving solution for intelligent vehicles.

Smart Healthcare

Smart healthcare apps offer a better future for the sector. QC-based IoT devices incorporate wearable technologies, which can improve remote patient monitoring facilities with accurate and real-time health data. QC-based IoT devices can identify health issues early and take more preventive measures, as well as the potential for patient care and healthcare operations through QC and IoT. The capacity of QC can perform operations and reduce costs by optimizing healthcare logistics like supply chain management, hospital maintenance, and resource allocation.

Ref	Authors&Year	Methodology	Contribution
[12]	Khan, A.A., et al. (2023)	The proposed software architecture for quantum computing systems is in the systematic, complete review.	It mainly contributes to the architecture activities, modeling notations, and design patterns.
[7]	Scheerer M., et al. (2023)	The proposal was related to quantum software engineering activities.	It contributes to overall Quantum Algorithms and Quantum Systems.

[7]	Scheerer M., et al. (2023)	The proposal was related to quantum software engineering activities.	It contributes to overall Quantum Algorithms and Quantum Systems.
[13]	Weder B., et al. (2022)	Proposed an overview of the Quantum software development life cycle.	It contributes to the development of quantum programs, classical programs.
[14]	Coccia M. (2022)	The proposal on quantum computing is to design a quantum ecosystem for industrial activities.	It contributes to Quantum Computing that can lay the foundation for a quantum industry.
[15]	Barrera A.G., et.al (2022)	The Quantum software is testing current trends and emerging proposals.	It contributes to Quantum Software Testing based on mutation of software, including the errors.
[2]	Miranskyy A. (2022)	This proposed method is related to using quantum computers to speed up the dynamic testing of software.	It contributes to Quantum Programming for Software Engineering.
[11]	Cavaliere F.,et al. (2020)	It was proposed on the security implications of quantum computing and cryptography.	It contributes to Quantum cryptography relies instead on fundamental quantum physics laws.
[16]	Li G. et al. (2020)	It was proposed for quantum testing and debugging quantum programs.	It contributes to the testing and debugging quantum programs on a quantum computer.
[17]	Sodhi B., et al. (2021)	Using a quantum computing environment was proposed to assess the impact on quality attributes and SDLC activities.	It contributes significantly to SDLC activities.
[1]	De Wolf R. (2019)	It was proposed that complete Quantum computing-related activities be provided in this.	It contributes a chapter on quantum machine learning, and a final chapter about quantum error correction. The main quantum algorithms.
[4]	Adedoyin A. et al. (2018)	This proposal on quantum algorithm implementation is from a beginner's point of view.	It mainly contributes to different Quantum Algorithms relevant information.

Table 1: Quantum Software Development and Testing State of the Art Work

Smart Industry

QC in industrial automation will enable future factories to be effectively enhanced, assuming control of various tasks now overseen by humans [14]. There are advantages to economic and human labor for many purposes.

Machine Learning

ML allows computers to learn and reprogram themselves to improve the accuracy or efficiency of resulting tasks. Develop more powerful QCs that enable advanced machine learning technology. MI can improve medicine, eradicate many diseases, provide therapy efficiently, and identify and cure diseases early. Moreover, machine learning is used in QCs to improve the quality of life unimaginably, as computers learn and develop to serve human goals of survival, harmony with the planet's ecosystem, and effortless and luxurious lives.

Smart Grids

QC-based IoT connectivity eliminates the inefficiencies of distribution and supply systems in QC-based grids, reducing human impact on energy consumption while maintaining current living standards. Forecasting and quality-based modeling have been used in the twentieth century to improve living standards. Dramatically improve quality control by collecting and analyzing inputs from multiple complex systems and predicting their interactions. In this section, the critical technologies in QC will be explored.

System Model and Design Phases

This section discusses QC, which is usually involved in developing Quantum Software systems (QSS), a new-age software engineering paradigm. In Quantum Software Engineering (QSE) development, QP is easy to understand but simple and efficient [19]. QSE mainly focuses on technologies to reduce complexity and programming issues. The QSE chooses process activities that include communication, planning, modeling, construction, and deployment [20]. The QSD process starts with gathering requirements and analyzing them. QSD is the new procedure for designing QS, which follows the Quantum Software Development Life Cycle (QSDLC) and provides complete information about process activities [13, 17].

Communication Phase

QSD communication involves requirements and analysis. In QC, gather requirements from customers. Here, the organization provides the QS with various requirements that help teams learn how to design these applications. After accepting the conditions from the customer, Then only all phases of work begin. The team prepares the required documents and then verifies whether everything is covered. Once every requirement was decided, an analysis was started on QS's feasibility, time, and cost. They also analyze which technology is suitable for development and the testing challenges. The outcome of this phase is the gathering of services from QS.

Planning Phase

The result of the previous phase of collecting QS requirements and essential parts is as follows: This phase provides a rough architecture for QS, which consists of the entire design. High- and low-level documents introduced with the help of QuantumUnified Modeling Languages (QUML) are depicted. Includes the complete analysis the essential phase

converts the logical model to the physical model of QS. The modeling purpose uses the algorithms for QS, which include various functions and features compared to actual software systems [21]. The development team starts work with the help of the structure approved by the architects.

Modeling Phase

Design mainly concentrates on representing those aspects of the QS visualized for end users. Exposes building models that help developers, testers, and clients. Features and functionalities meet the requirements understood by clients. Figure 1 shows the QS phases. The first phase of QSD includes gathering requirements, preparing the design structure, and implementing, deploying, and maintaining it [12]. The QS related QPs that create and perform quantum operations rely on QGs. Quantum gates map qubits to their states, such as superposition and entanglement of qubits measure.

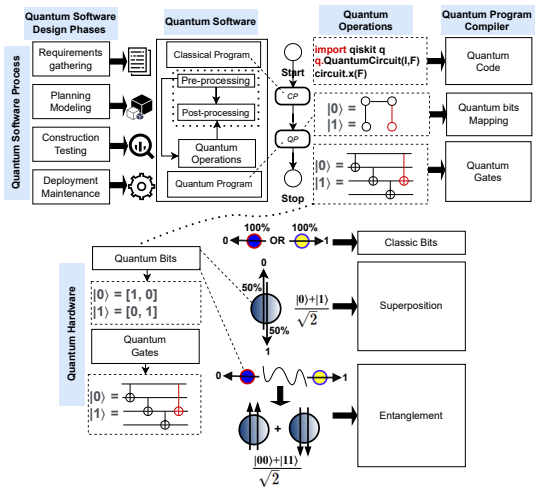

Figure 3: Quantum Software Design (QSD) Process

Construction Phase

Coding Phase

Code implementation starts when developers use high-level programming languages, i.e., C#(Q#), C++(Scafflod), C(QCL), Python(ProjectQ et al.), Scala(Chisel-Q), F#(LIQUi|⟩), and Haskell (Quiper). Because these are very familiar to end users, they easily integrate into environments and are more supportive [22]. Every QS module is tested separately to ensure good operation. The source code is the output of this phase and deployment of Quantum Software Applications.

Testing Phase

In this phase, test plans are developed by gathering test requirements for QS. This includes documents, databases, and input/output specifications. Plan the test activities, which include planning, development, execution, result analysis, trashing, and report preparation.

- *Test Plan:* It specifies the overall QS working procedure. What are the input and output functionalities, as well as the non-functionalities.
- *Test Development:* Implements test scenarios and test cases for each requirement. Here are all the requirements covered.
- *Test Execution:* Perform execution for test cases after QS is received from the development team.
- *Result Analysis:* Reports generate requirements that meet or not. The whole QS is working as expected and is otherwise unmatched.
- *Trashing:* Tracking the unmatched functions re-designs the team. Findings are updated for the client, and every requirement is tested and retested.
- *Status Reports:* As a tester, I generate the reports that give the overall status of QS.

Deployment and Maintenance Phase

In this last phase, QS is ready to demonstrate to the client with a 99% error-free application. The maintenance phase includes changes, modifications, and updates to the customer-related QSE. QC is a trending area for research in QSE, focusing mainly on classical software systems integrated with quantum algorithms [23]. Quantum Computers are more cost-effective to design because they adopt existing classical software systems. Therefore, re-engineering is essential for the migration process.

Proposed Work

This section discusses the QMLT technique with the QSTLC in detail, including various testing phases, findings, verification, validation, debugging, result analysis, and status reports. Quantum Computers are more efficient than CC. Furthermore, quantum computers have features different from those of classical computers, such as superposition, entanglement, and no cloning. In the design phase, the developers make common mistakes when designing Quantum Applications

compared to classical computers. In classical software engineering, testing applications based on the knowledge of the internal design of the application meant a white box and testing [16]. Applications based on requirements and functionality and not based on the knowledge of the internal design meant black box testing.

QSTLC, including various testing phases, findings, verification, validation, debugging, result analysis, and status reports. After the compilation of development, the QA is tested to verify and validate quantum circuit behavior based on specified requirements before being demonstrated to end users. Like the design phases, QMLT includes the testing phases, i.e., plan, development, execution, result analysis, tracking, and reports [24]. The testing of these activities represents the Software Testing Life Cycle (STLC). Figure 2 above shows the flow of the QSTLC. In the first phase, test plans are

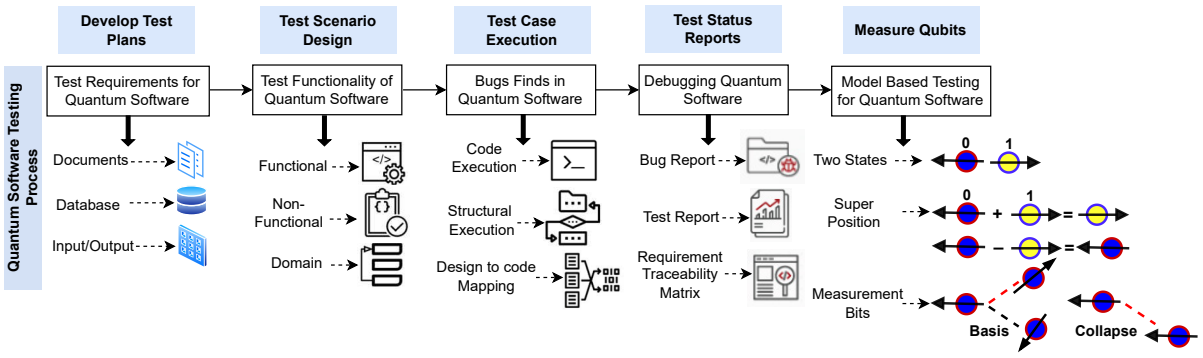

Figure 4: Quantum Software Testing Life Cycle (QSTLC)

Developed by gathering the test requirements for QS, such as documents, databases, and input/output. In the second phase of test scenario design, prepare the test scenarios with the help of these written test cases and check whether all requirements are covered, i.e., functional, non-functional, and domain specifications. Third-phase test case execution is here to execute all the test cases and find the functionality of the requirements matched with the output of the QS module. Four-stage test status report team survey results are sent to developers using a debugging process, then the bugs are fixed and the reports prepared. The final phase measures qubit states like superposition and measures whether the qubit requirement meets or not in the QS application. QA-team verification involves static analysis and review without executing the circuit. QA-team validation involves dynamic analysis to review the function and non-function and execute the circuit. Programmers analyze results from the execution of the circuit and make different assertions.

Test Scenario Design

Prepare test scenarios according to design test cases. To ensure the coverage of all the requirements, which includes the functional, non-functional, and QS specifications.

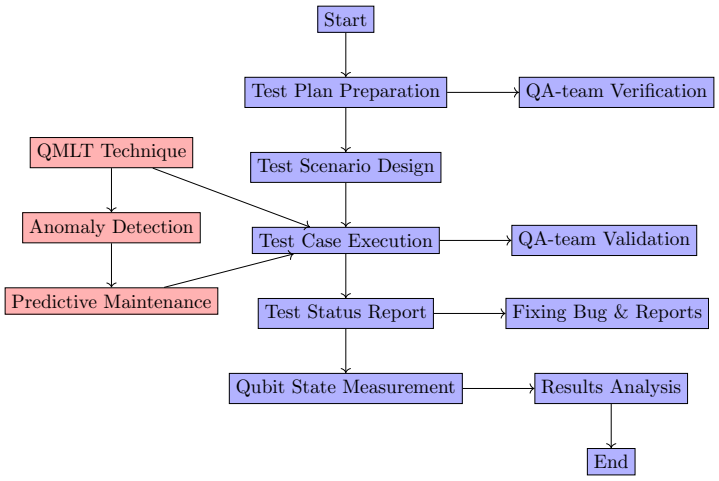

Figure 5: Quantum Software Testing Life Cycle (QSTLC) with Quantum Machine Learning

Test Case Execution

Execute all test cases, then verify the functionality of the QS module. Match the output compared with the expected results.

Test Status Report

The results of the test execution were in a test status report, and the QA-team conducted surveys and gathered feedback. Identify the bugs and send them to developers for debugging.

Verification and Validation

Finally, QA-team verification and validation include static analysis and review of the circuit without execution and dynamic analysis and review functionality with execution [25].

Algorithm 1 QMLT Algorithm with Quantum Gates

- function CorrectInitialQuantumValues(Q)
- for i ← 0 to len(Q) − 1 do if Qi ∈ {/ 0,1} then
- Qi ← 0
- 43 end if
- end for
- end function
- function QMLTAlgorithm(E)
- Q ← [0,1,0,0]
- Q ← CorrectInitialQuantumValues(Q)
- print ”Corrected Quantum Values:”, Q
- ApplyQuantumOperations
- print ”Quantum operations applied.”
- ComposeQuantumOperations
- print ”Quantum operations composed.”
- ApplyQMLT Technique
- print ”QMLT techniques applied.”
- TransformQuantumState
- print ”Quantum state transformed.”
- UseClassicalParameters(0.5)
- print ”Classical parameters used.”
- DeallocateQubits
- print ”Qubits deallocated.”
- end function
- return Q

The efficiency of testing techniques and scenarios is shown in Figure 6 in terms of Gbps. Techniques such as QST, QMLT, and CST are used to evaluate quantum applications' performance and dependency. The scenarios include error detection (ED), hardware characterization (HC), algorithm optimization (AO), quantum simulation (QS), and error testing (ET). According to each scenario, this simulation quantum systems, error detection, hardware characterization, and algorithm improvement with QST different performance.

Experimental Results

In this section, the performance of the proposed testing technique is compared with existing QST techniques. QS bug is an abnormal program behavior requirement that does not match customers' needs. Programmers developed a QP using programming languages. Developers usually need to understand these features. The bad coding practices have occurred. To debug and test the QS, thoroughly understand the behavior of bugs in the QP bug types to support QS debugging in QC. Implementing the QA's identified several bug types for QS and helped them understand the different strategies

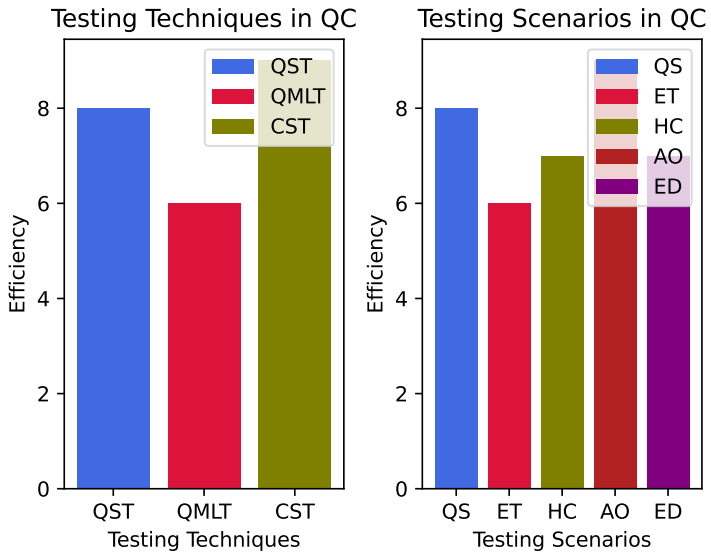

Figure 6: Quantum Software Testing Technique Efficiency

for each bug type. QS bug types include incorrect initial qubits, quantum operations, and quantum state transformations, the composition of operations using iteration, the composition of these operations using recursion and mirroring, the classical program input parameters, and the de-allocation of the qubits.

Conclusion

The proposed QMLT technique enhances software testing by integrating advanced methodologies, including QSTLC principles. QMLT automates test case selection and execution using ML for anomaly detection and predictive maintenance, analyzing data to prioritize high-risk cases. NLP generates automated status reports, offering real-time insights and facilitating team communication. QMLT evaluates bug reports to identify patterns, refining testing processes for continuous improvement. QC for speedups and thorough coverage despite current hardware and integration challenges. Combining ML, NLP, and QC, QMLT provides a robust, automated framework that enhances software quality through early issue detection and resolution, positioning itself as essential in modern QS development and testing.

Conflicts of Interest

No conflict of interest exists in the submission of this manuscript, and the manuscript is approved by all authors for publication.We would like to proclaim that the work described is an original study that has not been published previously, and is not under consideration for publication elsewhere, in whole or in part.

References

1. De Wolf, R. (2019). Quantum computing: Lecture notes. *arXiv preprint arXiv*:1907.09415.
2. Miranskyy, A. (2022, November). Using quantum computers to speed up dynamic testing of software. In *Proceedings of the 1st International Workshop on Quantum Programming for Software Engineering* (pp. 26-31).
3. de la Barrera, A. G., de Guzmán, I. G. R., Polo, M., & Cruz-Lemus, J. A. (2022). *Quantum software testing: current trends and emerging proposals. In Quantum Software Engineering (pp. 167-191). Cham: Springer International Publishing.*
4. Adedoyin, A., Ambrosiano, J., Anisimov, P., Casper, W., Chennupati, G., Coffrin, C., ... & Lokhov, A. Y. (2018). Quantum algorithm implementations for beginners. *arXiv preprint arXiv*:1804.03719.
5. Salvakkam, D. B., Saravanan, V., Jain, P. K., & Pamula, R. (2023). Enhanced quantum-secure ensemble intrusion detection techniques for cloud based on deep learning. *Cognitive Computation, 15*(5), 1593-1612.
6. Galanis, I. P., Savvas, I. K., Chernov, A. V., & Butakova, M. A. (2021). Reliability testing, noise and error correction of real quantum computing devices. *Telfor Journal, 13*(1), 41-46.
7. Scheerer, M., Klamroth, J., Garhofer, S., Knäble, F., & Denninger, O. (2023, May). Experiences in quantum software engineering. In *2023 IEEE International Parallel and Distributed Processing Symposium Workshops* (IPDPSW) (pp. 552-559). IEEE.
8. Dupouët, O., Pitarch, Y., Ferru, M., & Bernela, B. (2024). Community dynamics and knowledge production: forty years of research in quantum computing. *Journal of Knowledge Management, 28*(3), 651-672.
9. De Stefano, M., Pecorelli, F., Di Nucci, D., Palomba, F., & De Lucia, A. (2022). Software engineering for quantum programming: How far are we?. *Journal of Systems and Software, 190*, 111326.
10. Arute, F., Arya, K., Babbush, R., Bacon, D., Bardin, J. C., Barends, R., ... & Martinis, J. M. (2019). Quantum supremacy using a programmable superconducting processor. *Nature, 574*(7779), 505-510.
11. Cavaliere, F., Mattsson, J., & Smeets, B. (2020). The security implications of quantum cryptography and quantum computing. *Network Security, 2020*(9), 9-15.
12. Khan, A. A., Ahmad, A., Waseem, M., Liang, P., Fahmideh, M., Mikkonen, T., & Abrahamsson, P. (2023). Software architecture for quantum computing systems—A systematic review. Journal of Systems and Software, 201, 111682.
13. Weder, B., Barzen, J., Leymann, F., & Vietz, D. (2022). Quantum software development lifecycle. In Quantum Software Engineering (pp. 61-83). *Cham: Springer International Publishing.*
14. Coccia, M. (2024). Technological trajectories in quantum computing to design a quantum ecosystem for industrial change. *Technology Analysis & Strategic Management, 36*(8), 1733-1748.
15. Bayerstadler, A., Becquin, G., Binder, J., Botter, T., Ehm, H., Ehmer, T., ... & Winter, F. (2021). Industry quantum computing applications. *EPJ Quantum Technology, 8*(1), 25.
16. Li, G., Zhou, L., Yu, N., Ding, Y., Ying, M., & Xie, Y. (2020). Projection-based runtime assertions for testing and debugging quantum programs. *Proceedings of the ACM on Programming Languages, 4*(OOPSLA), 1-29.
17. Sodhi, B., & Kapur, R. (2021, March). Quantum computing platforms: assessing the impact on quality attributes and sdlc activities. In 2021 IEEE 18th International Conference on Software Architecture (ICSA) (pp. 80-91). IEEE.
18. Motta, M., & Rice, J. E. (2022). Emerging quantum computing algorithms for quantum chemistry. *Wiley Interdisciplinary Reviews: Computational Molecular Science, 12*(3), e1580.
19. Gemeinhardt, F., Garmendia, A., & Wimmer, M. (2021, June). Towards model-driven quantum software engineering. In *2021 IEEE/ACM 2nd International Workshop on Quantum Software Engineering* (Q-SE) (pp. 13-15). IEEE.
20. Piattini, M., & Murillo, J. M. (2022). Quantum software engineering landscape and challenges. In Quantum Software Engineering (pp. 25-38). Cham: *Springer International Publishing.*
21. Sævarsson, B., Chatzivasileiadis, S., Jóhannsson, H., & Østergaard, J. (2022). Quantum computing for power flow algorithms: Testing on real quantum computers. *arXiv preprint arXiv*:2204.14028.
22. Zhao, J. (2020). Quantum software engineering: Landscapes and horizons. *arXiv preprint arXiv*:2007.07047.

23. Steinberg, M. A., Feld, S., Almudever, C. G., Marthaler, M., & Reiner, J. M. (2022). Topological-graph dependencies and scaling properties of a heuristic qubit-assignment algorithm. *IEEE Transactions on Quantum Engineering, 3*, 1-14.
24. Paltenghi, M. (2022, May). Cross-platform testing of quantum computing platforms. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings* (pp. 269-271).
25. Saki, A. A., Alam, M., Phalak, K., Suresh, A., Topaloglu, R. O., & Ghosh, S. (2021, May). A survey and tutorial on security and resilience of quantum computing. *In 2021 IEEE European Test Symposium* (ETS) (pp. 1-10). IEEE.