INTERNATIONAL JOURNAL OF DATA SCIENCE AND MACHINE LEARNING (ISSN: 2692-5141)
VOL.06 ISSUE 02 (2026)

Research Article

Intelligent Test Automation Using Machine Learning for Modern Software Quality Assurance

Rizky Pratama [1]

[1]Department of Artificial Intelligence, Institute of Technology and Computing, Jakarta, Indonesia

Received: 28 May 2026
Revised: 22 June 2026
Accepted: 20 July 2026
Published: 09 August 2026

Page No: 57-65

Abstract

Modern software systems are characterized by rapid release cycles, continuously changing requirements, heterogeneous platforms, and increasingly complex user interactions. These conditions create substantial challenges for conventional test automation, particularly in test-case maintenance, regression-test selection, defect prediction, and adaptation to changing software interfaces. This research examines the role of machine learning (ML) in intelligent test automation and develops a conceptual framework for integrating ML techniques into modern software quality assurance (SQA). The study adopts a research-and-review approach based exclusively on the supplied literature and uses comparative synthesis to connect existing evidence on intelligent digital systems, mobile applications, augmented-reality environments, software platforms, and AI-driven test automation. The proposed framework incorporates test-case generation, test prioritization, defect-risk prediction, failure classification, regression optimization, and continuous learning into a unified quality-assurance workflow. The analysis indicates that ML can potentially transform automation from a rule-based execution mechanism into an adaptive decision-support system capable of learning from historical test outcomes and application behavior. The study further identifies limitations involving data quality, model interpretability, changing application behavior, false predictions, and maintenance requirements. The findings position intelligent automation as a complementary evolution of conventional automation rather than a complete replacement for human quality engineers. The research contributes a conceptual architecture for applying ML systematically across the software testing lifecycle and identifies directions for empirical validation.

Keywords: Machine Learning, Intelligent Test Automation, Software Quality Assurance, Automated Testing, Defect Prediction, Test Prioritization, Regression Testing, Artificial Intelligence, Continuous Testing.

INTRODUCTION

Software quality assurance has progressively moved from predominantly manual verification toward automated testing as software organizations have adopted shorter development cycles and continuous delivery practices. Conventional automation provides substantial benefits by executing predefined test cases repeatedly and consistently; however, its effectiveness depends heavily on the stability of application interfaces, test data, locators, scripts, and expected outcomes. When an application

changes frequently, automated test suites may require extensive maintenance. This creates a fundamental limitation: automation can execute tests efficiently, but conventional automation generally does not possess sufficient adaptive capability to determine which tests are most valuable, why failures occur, or how testing strategies should change according to historical evidence.

The increasing complexity of software applications further strengthens this challenge. Research concerning mobile and augmented-reality applications demonstrates that contemporary digital environments can involve multiple interaction mechanisms, device constraints, compatibility considerations, and user-experience variables. Evaluation of augmented-reality mobile applications, for example, emphasizes the importance of user experience within technologically dynamic application environments (Davidavičienė et al., 2020). Similarly, research on marker-based and markerless augmented reality illustrates that different technical implementations can produce distinct system behaviors and testing considerations (Cheng et al., 2017). These observations are relevant to intelligent SQA because testing strategies must increasingly account for application context rather than simply execute static scripts.

Machine learning offers a potential mechanism for addressing this limitation. Instead of treating every execution as an isolated event, an ML-enabled testing system can learn from historical test results, code changes, defect patterns, execution duration, failure frequency, and application behavior. AI-driven test automation has consequently been positioned as an important direction for modern software quality engineering, particularly because intelligent techniques can support automated decision-making throughout the testing lifecycle (Ramamurthy, 2023).

The primary objective of this research is to develop a conceptual model for intelligent test automation using ML and to examine how such a model can improve the efficiency, adaptability, and analytical capability of software quality assurance. The study specifically considers automated test generation, test prioritization, defect-risk prediction, failure classification, regression-test optimization, and continuous learning. Its significance lies in integrating these functions into a coherent quality-engineering architecture rather than treating ML as an isolated testing utility.

The scope of the study is conceptual and analytical. It does not claim results from a newly executed industrial experiment. Instead, the proposed framework synthesizes the supplied literature and translates its implications into an intelligent SQA model. This distinction is important because ML-based testing requires empirical datasets and controlled evaluation before performance improvements can be generalized.

LITERATURE REVIEW

The supplied literature primarily concerns augmented reality, mobile applications, software platforms, digital systems, and user interaction rather than direct empirical evaluations of ML-based software testing. Nevertheless, these studies provide useful theoretical foundations for understanding why intelligent testing mechanisms are increasingly necessary in heterogeneous software environments.

AlNajdi (2022) investigates augmented reality supported through QR codes in educational contexts. The study illustrates how digital functionality can be integrated with conventional content and how application behavior may depend on interaction mechanisms. From an SQA perspective, such systems create multiple testing dimensions, including QR recognition, application response, device compatibility, and user interaction. A static test suite may verify predefined scenarios, but an intelligent system could potentially use historical execution information to prioritize scenarios that exhibit greater failure risk.

Arioputra and Lin (2015) examine mobile augmented reality as a Chinese menu translation mechanism. The research is significant for intelligent automation because mobile applications can involve image recognition, contextual processing, translation behavior, and device-specific execution. Such characteristics make exhaustive manual testing difficult. ML-assisted automation can theoretically classify recurring failures and identify test scenarios that are most likely to expose defects in recognition or translation functionality.

Cheng et al. (2017) compare marker-based and markerless augmented reality, demonstrating that different technical approaches produce different application characteristics. This distinction reinforces an important testing principle: test automation should be sensitive to architectural and behavioral differences rather than relying exclusively on identical scripts across application configurations. An intelligent framework can potentially use historical outcomes and contextual information to adapt test selection according to application characteristics.

User experience constitutes another important dimension. Davidavičienė et al. (2020) evaluate user experience in augmented-reality mobile applications, demonstrating that software quality cannot be reduced exclusively to functional correctness. This has implications for intelligent SQA because quality models may need to incorporate behavioral and experience-oriented signals alongside traditional pass/fail outcomes.

The supplied work by Baruah (2022) discusses the relationship between augmented reality and QR codes, while Chaurasiya et al. (2019) and Gupta et al. (2020) describe digital and augmented-reality-oriented restaurant systems. These studies collectively demonstrate the growing diversity of interactive software systems. Their common implication is that software testing must address multiple interaction pathways, device conditions, and functional states.

Hartmann (2020) provides an overview of Blender, a software platform used for three-dimensional content creation. Although the source does not directly address testing, it illustrates the broader ecosystem of software tools in which complex interfaces and visual components can influence system behavior. Kaur and Sharma (2014), meanwhile, examine Android as a mobile platform, reinforcing the importance of platform-specific considerations in application development and evaluation.

The principal research gap is therefore not simply the absence of automated testing but the absence of a unified mechanism capable of learning from testing activity. Conventional automation typically follows predetermined rules, whereas intelligent automation aims to infer patterns and continuously improve testing decisions. Ramamurthy (2023) directly establishes the relevance of AI-driven test automation to modern software quality engineering and provides a conceptual foundation for considering AI as an integrated component of contemporary SQA.

The literature consequently supports a theoretical positioning in which intelligent testing is viewed as an adaptive layer above conventional automation. The existing studies demonstrate the complexity and variability of modern applications, while AI-driven test automation provides the technological direction for managing that complexity (Ramamurthy, 2023). The proposed research extends this positioning by organizing ML capabilities across the software testing lifecycle.

METHODOLOGY

3.1 Research Design

This study uses a conceptual research-and-review methodology. The approach consists

of three stages: literature synthesis, functional decomposition of intelligent testing, and development of a conceptual ML-enabled SQA framework. Because the supplied references do not provide a common experimental dataset, statistical meta-analysis is inappropriate. Instead, the research identifies recurring technical characteristics from the literature and maps them to software testing functions.

The methodological objective is not to claim that ML automatically improves every testing metric. Rather, the objective is to establish a technically plausible architecture in which ML supports testing decisions while conventional automation remains responsible for deterministic execution.

3.2 Proposed Intelligent Test Automation Architecture

The proposed architecture consists of five interconnected layers: application observation, test intelligence, test execution, result analysis, and continuous learning.

The application-observation layer collects information from source-code changes, requirements, user-interface structures, previous test executions, defect reports, execution logs, and environmental configurations. This information constitutes the feature space from which ML models can identify patterns.

The test-intelligence layer transforms collected information into testing decisions. Its principal functions are test-case generation, test prioritization, defect-risk prediction, and regression-test selection. A classification model, for example, may estimate whether a changed component has a relatively high or low probability of producing a regression defect. A ranking mechanism can then order test cases according to predicted risk and business relevance.

The test-execution layer integrates the ML decisions with existing automation tools. The system executes selected test cases across appropriate environments rather than attempting to replace deterministic automation. This hybrid structure is essential because ML predictions are probabilistic whereas test execution requires explicit actions and verifiable assertions.

The result-analysis layer interprets test outcomes. Instead of merely reporting that a test failed, the system can classify failures according to historical patterns, such as application defects, environmental instability, data problems, or automation-related failures. Such classification can reduce the analytical workload of quality engineers.

Finally, the continuous-learning layer feeds validated test results back into the ML system. Incorrect predictions are identified, model inputs are updated, and subsequent testing decisions can be recalibrated. This feedback mechanism represents the principal distinction between static automation and intelligent automation.

3.3 ML-Based Test-Case Generation

Test-case generation can be enhanced by identifying combinations of application states, input values, interaction sequences, and previously observed failure conditions. For example, a mobile application containing QR recognition functionality may have multiple combinations of device type, camera condition, QR format, network state, and user interaction. Instead of generating tests uniformly, an intelligent system could learn which combinations historically produce failures and increase their testing priority.

The advantage is increased exploration of high-value scenarios. However, generated cases must remain traceable to requirements and expected behavior. Excessive automated generation can create large numbers of low-value tests, increasing execution and maintenance costs. Consequently, ML-generated tests should be filtered according to

coverage, novelty, historical defect relevance, and execution cost.

3.4 Intelligent Test Prioritization

Regression testing represents one of the most suitable applications of ML because the number of available tests may substantially exceed the time available for execution. A prioritization model can estimate the relative value of each test based on variables such as recent code changes, historical failure frequency, component criticality, execution cost, and defect history.

A conceptual prioritization score can be represented as:

$$\text{Priority}(T_i) = w_1R_i + w_2F_i + w_3C_i + w_4H_i - w_5E_i$$

where R_i represents predicted regression risk, F_i historical failure likelihood, C_i component criticality, H_i historical defect relevance, and E_i execution cost. The weights can be learned or calibrated according to project objectives.

Such prioritization does not eliminate comprehensive regression testing. Rather, it enables high-risk tests to execute earlier, allowing potentially serious defects to be detected before lower-value scenarios consume available resources.

3.5 Defect Prediction and Failure Classification

Defect prediction uses historical information to estimate which software components or test scenarios are more likely to contain defects. Features may include code-change frequency, previous failures, component complexity, and defect history. The output can be used to allocate testing resources dynamically.

Failure classification complements defect prediction. A failed automated test does not necessarily mean that the software contains a defect. Infrastructure problems, unavailable services, invalid test data, timing issues, and unstable automation can also cause failures. ML-based classification can learn from previously labeled failures and distinguish likely application defects from non-product failures.

This distinction has significant practical implications because unreliable automation can reduce confidence in testing. Intelligent failure analysis therefore seeks not only to increase test execution but also to improve the informational quality of test results.

3.6 Continuous Learning and Human Oversight

Continuous learning should not be interpreted as unrestricted autonomous modification. A quality engineer should validate significant model decisions, particularly when incorrect predictions could cause important tests to be omitted. The framework therefore adopts a human-in-the-loop model in which ML provides predictions and recommendations while quality engineers maintain governance over test strategy.

This approach is consistent with the broader direction of AI-driven test automation, in which intelligent techniques augment software quality engineering rather than necessarily eliminating human involvement (Ramamurthy, 2023).

RESULTS

The conceptual analysis identifies five principal findings. First, ML is most valuable when it operates as a decision layer rather than merely as another mechanism for executing automated scripts. Conventional automation already provides deterministic execution; the distinctive contribution of ML is its ability to analyze historical evidence and support

decisions concerning what should be tested, when it should be tested, and how failures should be interpreted.

Second, test prioritization emerges as a particularly practical application. In large regression suites, executing every available test after every software change can produce considerable time and infrastructure costs. An ML-based ranking mechanism can concentrate early execution on scenarios associated with changed or historically unstable components. The expected benefit is not simply reduced execution time but earlier identification of high-impact failures.

Third, the literature's emphasis on heterogeneous mobile, augmented-reality, and interactive applications supports the need for context-sensitive testing. Mobile and AR systems may involve different devices, interaction mechanisms, and environmental conditions (Arioputra & Lin, 2015; Cheng et al., 2017; Davidavičienė et al., 2020). Consequently, intelligent automation should incorporate contextual information rather than assuming that one fixed test sequence adequately represents application behavior.

Fourth, intelligent failure classification can improve the reliability of automated quality reports. If failures are automatically separated into probable product defects, environmental failures, data problems, and automation instability, engineers can focus investigative effort more efficiently. This is particularly important in continuous testing environments where large quantities of execution data can otherwise obscure actionable defects.

Fifth, the analysis indicates that ML introduces its own quality risks. Poorly labeled training data can produce unreliable predictions; changes in application architecture can make historical patterns obsolete; and over-optimization can cause important but historically rare tests to receive insufficient priority. Therefore, intelligent automation should retain deterministic baseline suites and periodically evaluate model performance.

The findings collectively support the proposed hybrid architecture. ML should guide selection, prioritization, prediction, and classification, while conventional automation performs controlled execution. This division reduces the risk associated with excessive autonomy and creates a practical transition path from traditional automation to intelligent quality engineering. The conceptual findings are consistent with the broader proposition that AI-driven automation can become an important component of modern software quality engineering (Ramamurthy, 2023).

DISCUSSION

The findings suggest that the principal transformation introduced by ML is a shift from execution-oriented automation toward decision-oriented automation. Traditional automation answers the question, “Can this predefined test be executed repeatedly?” Intelligent automation additionally asks, “Which test should be executed first, what is the expected risk, and what does the observed failure most likely indicate?” This distinction has substantial implications for SQA strategy.

The proposed framework is theoretically relevant to applications characterized by diverse interfaces and interaction conditions. The supplied literature demonstrates that software environments such as mobile and augmented-reality applications can contain platform-specific and interaction-specific behavior (AlNajdi, 2022; Arioputra & Lin, 2015). Consequently, testing strategies based exclusively on fixed scripts may have limited adaptability. ML can potentially identify relationships between environmental conditions and previous failures, enabling more context-sensitive test selection.

A second implication concerns regression testing. If a system can reliably estimate defect

risk, organizations may reduce unnecessary execution without simply reducing quality coverage. However, this benefit creates an important trade-off. A model optimized solely for efficiency may incorrectly deprioritize low-frequency tests that detect severe defects. Therefore, optimization objectives should incorporate risk, coverage, severity, and diversity rather than execution time alone.

A third issue is explainability. Software quality engineers must often justify why specific tests were selected or omitted. A highly accurate but opaque ML model may therefore be less operationally useful than a slightly less accurate model whose recommendations can be explained. Human oversight remains important because test strategy involves organizational risk tolerance, regulatory requirements, and business priorities that may not be completely represented in historical data.

The literature also highlights the importance of user experience and application compatibility. Davidavičienė et al. (2020) demonstrate the relevance of user experience in AR mobile applications, while Cheng et al. (2017) emphasize differences between AR implementation approaches. These perspectives suggest that intelligent SQA should eventually expand beyond functional pass/fail testing toward multidimensional quality assessment.

Several limitations must be recognized. The present framework is conceptual and has not been validated against a controlled industrial dataset. The supplied references also do not collectively provide sufficient empirical evidence to calculate an actual improvement percentage for ML-based testing. Therefore, claims concerning efficiency, defect-detection improvement, or maintenance reduction should be treated as hypotheses for future empirical research rather than established outcomes.

A further limitation concerns model drift. Software continuously changes, meaning historical defect patterns may become less predictive. Continuous learning can address this problem, but it also introduces monitoring and governance requirements. Accordingly, intelligent automation should be evaluated as an evolving socio-technical system involving models, automation infrastructure, software developers, testers, and quality-management processes.

Overall, the discussion supports a balanced position: ML has strong potential to improve SQA decision-making, but its value depends on data quality, explainability, integration, governance, and continuous validation. AI-driven test automation should therefore complement established testing principles rather than replace them (Ramamurthy, 2023).

CONCLUSION

This research examined intelligent test automation using machine learning as an emerging approach to modern software quality assurance. The analysis identified limitations in conventional rule-based automation and proposed a conceptual framework integrating application observation, ML-based test intelligence, automated execution, failure analysis, and continuous learning.

The principal contribution is the conceptual integration of test-case generation, test prioritization, defect prediction, regression optimization, and failure classification within a single SQA architecture. The framework positions ML primarily as an intelligent decision-support layer while retaining deterministic automation for actual test execution. This hybrid strategy provides a practical pathway for organizations seeking to modernize existing automation without assuming that AI can independently manage the entire testing lifecycle.

The reviewed literature also indicates why adaptive approaches are increasingly relevant. Mobile, augmented-reality, interactive, and platform-dependent applications introduce multiple dimensions of behavior and compatibility that can challenge static testing strategies (AlNajdi, 2022; Cheng et al., 2017; Kaur & Sharma, 2014). Intelligent automation can potentially respond to this complexity by learning from historical evidence and dynamically adjusting testing priorities.

Nevertheless, the proposed model requires empirical validation. Future research should implement the architecture on industrial software repositories and evaluate test coverage, defect-detection capability, execution time, false-positive rates, maintenance effort, and model stability. Comparative experiments between conventional automation and ML-assisted automation would provide stronger evidence regarding the actual value of intelligent testing. Future systems should also investigate explainable ML, reinforcement-based test selection, multimodal application analysis, and human-in-the-loop governance.

The broader implication is that the future of software quality engineering is unlikely to be defined by automation alone. Instead, it will increasingly depend on automation that can learn, reason over historical evidence, prioritize intelligently, and continuously adapt while remaining subject to engineering controls. In this context, ML represents not merely an additional testing technology but a potential foundation for transforming software testing into a more predictive, adaptive, and evidence-driven quality-assurance discipline (Ramamurthy, 2023).

REFERENCES

1. AlNajdi, S. M. (2022). The effectiveness of using augmented reality(AR) to enhance student performance: using quick response(QR) codes in student textbooks in the Saudi educationsystem.Educational Technology Research and Development,70, 1105–1124.

2. Arioputra, D., & Lin, C. H. (2015). Mobile augmented reality as aChinese menu translator. InIEEE International conferenceon consumer electronics Taiwan.

3. Baruah, B. (2022, May 27). Augmented reality and QR code–Whatyou need to know. Beaconstac.https://blog.beaconstac.com/2020/03/augmented-reality-qr-codes/

4. Chaurasiya, A., Mhatre, S., Chaudhari, R., & Pawar, P. (2019).Smart restaurant menu card by using augmented reality.JETIR,6.http://www.jetir.org/papers/JETIR1903814.pdf

5. Cheng, J., Chen, K., & Chen, W. (2017). Comparison of marker-based AR and markerless AR: A case study on indoordecoration system.

6. Davidavičienė, V., Raudeliūnienė,J.,&Viršilaitė, R. (2020).Evaluation of user experience in augmented reality mobileapplications.Journal of Business Economics and Management,22, 467–481.

7. Gupta V., Gaddam N., Narang L., & Gite, Y. (2020). Digitalrestaurant.International Research Journal of Engineeringand Technology (IRJET),07, 5340–5344.https://www.irjet.net/archives/V7/i4/IRJET-V7I41009.pdf

8. Hartmann, T. (2020, May 21).What Is Blender (Software)?–SimplyExplained.All3DP.https://all3dp.com/2/blender-simply-explained/

9. Kaur, P., & Sharma, S. (2014). Google Android a mobile platform: Areview.Recent Advances in Engineering and ComputationalSciences (RAECS),1–5,

10. Geo Philip, Paulson, Artificial Intelligence and Machine Learning Applications in Project Schedule Forecasting: A Predictive Framework for Time-Control in Complex Building Projects (April 16, 2026). Available at SSRN: https://ssrn.com/abstract=6588119 or http://dx.doi.org/10.2139/ssrn.6588119

11. Ramamurthy, K. (2023). AI-Driven Test Automation Frameworks for the Modern Software Quality Engineering. International Journal of Emerging Trends in Computer Science and Information Technology, 4(4), 257-269.