A Review of the Effectiveness of AI in Automated Software Testing

[1]Abdurrahman Abdullahi Dangongola, [2]Dr Senthil Kumar

[1] 2year M.sc., SE., Department of Software Engineering, School of Post Graduate Studies, Skyline University Nigeria.

[2]Dean, School of Science & Information Technology, Skyline University Nigeria.

[1]Dangongola11@yahoo.com

[2]Senthil.kumar@sun.edu.ng

Abstract

Software testing plays a vital role in ensuring application reliability and quality. With the increasing complexity of software systems, traditional testing methods face challenges related to efficiency, accuracy, and adaptability. Artificial Intelligence (AI) has emerged as a promising approach to enhance automated software testing by leveraging techniques such as machine learning and natural language processing. This paper presents a comprehensive review of AI-powered automated testing methods, examining their applications, benefits, and challenges. The review aims to provide a deeper understanding of how AI contributes to improving defect detection, reducing testing time, and expanding test coverage. By synthesizing recent research and industry practices, this study offers insights to guide future developments and the adoption of AI in software quality assurance.

Keywords – Artificial Intelligence (AI), Automated Software Testing, Defect Detection, Test Coverage, Test Automation, AI-powered Testing Tools

INTRODUCTION

Software testing is a cornerstone of modern software development, essential for ensuring that applications function as intended and consistently deliver value to users and organizations. As software systems grow increasingly complex and development cycles accelerate, the demands on testing processes have expanded exponentially. Traditional automated testing methods have long been the industry standard, providing significant efficiency gains over manual testing. However, these conventional approaches face notable limitations, including extensive script maintenance, difficulties adapting to dynamic user interfaces, and challenges in detecting subtle yet critical defects (Bertolino et al., 2021).

The integration of Artificial Intelligence (AI) and Machine Learning (ML) is fundamentally reshaping the landscape of software quality assurance. AI-powered techniques—ranging from machine learning algorithms to natural language processing and computer vision—are increasingly deployed to enhance test coverage, accuracy, and efficiency. These advanced tools not only automate repetitive tasks but also intelligently adapt to evolving software, autonomously generate test cases, and predict defect-prone areas

(Shoaib et al., 2020). Consequently, AI-driven testing offers the potential for more efficient, effective, and resilient testing processes, aligning with the rapid and reliable release cycles demanded by contemporary software development practices.

Despite these promising advancements, the practical adoption and real-world effectiveness of AI in software testing remain subjects of ongoing investigation. Many organizations continue to explore optimal strategies for integrating AI into their testing workflows, while researchers debate issues related to scalability, reliability, interpretability, and ethical considerations (Karhu et al., 2025). This literature review critically examines the effectiveness of AI in automated software testing by synthesizing recent empirical studies, theoretical frameworks, and industry practices. It highlights key advancements, persistent challenges, and unresolved questions, providing actionable insights for both researchers and practitioners seeking to leverage AI for improved software quality assurance.

Research Objectives

The primary objectives of this study are:

To explore how AI and ML are being integrated into automated software testing.

To assess the effectiveness and efficiency of AI-powered testing techniques in real-world scenarios.

To identify the main challenges and barriers organizations face when adopting AI in software testing.

Research Gap

Despite the increasing use of AI-powered automated testing, significant gaps remain in fully understanding its effectiveness in enhancing defect detection, reducing testing time, and improving test coverage in diverse and dynamic software development environments. Most existing research emphasizes technical implementations or isolated case studies, lacking broad empirical evidence on how these AI tools adapt to evolving software, scale across different project types, and sustain efficiency gains over time. Important challenges such as managing false positives, ensuring model interpretability, and addressing long-term maintenance overhead are often overlooked. Additionally, there is insufficient distinction in the literature between traditional automated testing and AI-enhanced approaches, resulting in unclear insights into their relative advantages and limitations. A comprehensive and systematic review addressing these gaps is crucial to provide clearer guidance for researchers and practitioners on effectively applying AI to improve software quality assurance.

Scope

This study focuses on recent advancements in AI-powered automated software testing, drawing on peer-reviewed research and industry case studies from 2017 to 2025. The review encompasses the use of machine learning for test case generation, defect prediction, test suite optimization, and self-healing test automation. The scope is limited to secondary research, including systematic reviews, empirical studies, and authoritative industry analyses.

Limitations

While this research provides a comprehensive overview of current trends, it is inherently limited by its reliance on available secondary data. The rapid pace of technological innovation means that some findings may quickly become outdated as new tools and methods emerge. Additionally, access to proprietary datasets and detailed empirical studies is limited, which may affect the depth and generalizability of the analysis.

RELATED WORKS

This section reviews reported work on the application of artificial intelligence (AI) in automated software testing, focusing on recent advancements, techniques, and their practical implications. Numerous studies have explored how AI—particularly machine learning (ML), deep learning, and predictive analytics—can enhance various aspects of software testing, such as test case generation, defect prediction, test prioritization, and self-healing test automation. While these studies demonstrate significant improvements in efficiency, accuracy, and coverage, they also expose important limitations, including data dependency, algorithmic bias, and integration challenges.

A systematic mapping study by Karhu, Kasurinen, and Smolander (2025) analyzed recent industry research on AI adoption in software testing, revealing that while AI is increasingly discussed in academic literature, its practical adoption in industry remains limited. The study found that most reported research focuses on theoretical benefits, with few real-world evaluations or case studies that validate the effectiveness of AI-driven approaches. The authors also highlighted a gap in research addressing long-term maintenance, adaptability to complex systems, and the need for high-quality training data.

Building on this, Bertolino et al. (2021) conducted a comprehensive survey of machine learning applications in software testing, cataloging the main techniques and their effectiveness across different testing stages. Their review identified that ML-based test case generation and defect prediction can significantly reduce manual effort and improve test coverage. However, the authors noted that these methods often require large, labeled datasets and may struggle with interpretability and generalization to new or complex domains.

Recent work by Baqar and Khanda (2025) explored AI-powered test case generation and validation, emphasizing the transformative potential of AI in addressing long-standing challenges such as incomplete test coverage and high maintenance costs . The authors reviewed real-world examples where AI-driven techniques—including natural language processing and predictive modeling—enhanced test efficiency and adaptability. However, they also acknowledged limitations such as potential biases in AI algorithms, the need for substantial training data, and integration challenges with existing workflows.

Other studies have focused on the practical implementation of AI in software testing. Mariani et al. (2017) investigated automated test generation for software that learns, demonstrating how AI can adapt testing strategies to evolving systems . Their approach improved test coverage for dynamic applications but was

constrained by computational costs and the complexity of input data . Similarly, Shoaib et al. (2020) reviewed ML techniques for test effort estimation, finding that while AI can streamline testing workflows, its effectiveness depends on the quality and relevance of available data .

A systematic review by Islam et al. (2023) examined various AI-driven techniques and tools, concluding that AI can successfully automate multiple testing tasks—such as test case generation, defect prediction, and test prioritization—while simplifying overall testing activities . However, the review also noted that most studies are limited in scope, often focusing on specific domains or small datasets, which may affect generalizability .

In summary, while AI-powered software testing offers considerable promise, current research highlights several recurring challenges: the need for large, high-quality datasets; difficulties in integrating AI with existing workflows; concerns about algorithmic bias and interpretability; and limited real-world validation. These limitations suggest that future work should focus on validating AI-based approaches in diverse, large-scale environments and developing strategies to reduce data dependency and improve practical adoption.

Advantages And Disadvantages Of Ai-Powered Testing Techniques

Advantages of AI-Powered Testing Techniques:

1. Increased Efficiency

   AI automates repetitive and time-consuming testing tasks, enabling faster test execution and allowing human testers to focus on more strategic or exploratory testing.
2. Improved Accuracy and Consistency

   Machine learning algorithms reduce human error and deliver consistent results across test cycles, improving the reliability of test outcomes.
3. Enhanced Test Coverage
4. AI can analyze large datasets and system behaviors to generate comprehensive test cases, increasing coverage and reducing the risk of undetected defects.
5. Predictive Defect Detection

   AI models trained on historical bug data can predict likely points of failure, enabling proactive remediation and reducing post-release issues.
6. Real-Time Feedback and Self-Healing

   Tools integrated with AI can adjust to UI or code changes automatically and provide immediate feedback during CI/CD processes.
7. Reduced Testing Costs (Over Time)

   Though initial setup may be expensive, long-term automation reduces manual effort, minimizing cost related to testing resources and rework.
8. Adaptability to Rapid Releases

In agile and DevOps environments, AI testing scales with the speed of delivery pipelines, supporting rapid iterations and continuous integration.

Figure 1.0

Disadvantages of AI-Powered Testing Techniques

1. Complexity of Implementation

   Deploying AI testing tools requires advanced knowledge in machine learning and data science, which may not be readily available in all teams.

2. Data Dependency and Quality Issues

   The effectiveness of AI relies heavily on the quality and quantity of training data. Poor data can lead to inaccurate models and test failures.

3. Lack of Interpretability (Black Box Models)

   Many AI systems provide little transparency in their decision-making processes, making it difficult to validate or trust their outputs.

4. Maintenance and Model Drift

   As software changes, AI models may become outdated and require regular retraining and maintenance, introducing ongoing operational effort.

5. Limited Human-Like Intuition

   AI lacks the domain-specific reasoning and creativity of human testers, particularly in exploratory or usability testing scenarios.

6. Security and Ethical Concerns

   AI tools can introduce new vulnerabilities, including risks of data breaches or misuse of sensitive training data.

7. False Positives/Negatives

   Without fine-tuning, AI models can generate incorrect results, either flagging non-issues or missing real bugs, which undermines trust in automation. Figure 1.0 below shows the advantages of AI-powered testing techniques.

Mitigating The Disadvantages

To mitigate the disadvantages of AI-powered testing techniques, organizations can:

1. Invest in Training and Development: Invest in training and development programs to ensure that testing teams have the necessary skills and expertise to implement and maintain AI-powered testing techniques.
2. Ensure Data Quality: Ensure that data used to train AI-powered testing techniques is of high quality and relevant to the testing context.
3. Monitor and Evaluate: Regularly monitor and evaluate the effectiveness of AI-powered testing techniques and make adjustments as needed.
4. Use Hybrid Approach: Use a hybrid approach that combines AI-powered testing techniques with human testing to leverage the strengths of both approaches.
5. Address Security Risks: Address security risks associated with AI-powered testing techniques by implementing robust security measures and protocols.

Comparative Analysis of AI and Traditional Testing

This chapter discussed the evolution of software testing, the role of AI in enhancing testing processes, and its effectiveness in improving defect detection and test case generation. While AI-powered testing provides several advantages, challenges such as cost, complexity, and lack of interpretability remain. Future advancements in AI-driven automation are expected to overcome these challenges and drive more efficient testing methodologies.

Comparison of Traditional and Automated Testing

Traditional and automated testing each play critical but distinct roles in ensuring software quality. Traditional testing, carried out manually by human testers, excels in areas where human judgment, intuition, and real-world contextual understanding are essential. Testers can identify usability issues, perform exploratory testing without predefined scripts, and respond flexibly to unexpected application behavior. These qualities are especially valuable in the early stages of development, during user acceptance testing, or in situations where user experience is a primary concern.

However, the major drawbacks of traditional testing lie in its time-consuming and resource-intensive nature. Manual execution of test cases requires considerable human effort, leading to longer testing cycles and increased costs. Additionally, human testers are susceptible to fatigue and oversight, which may result in inconsistent outcomes or missed defects—especially when handling repetitive or large-scale testing tasks.

On the other hand, automated testing leverages software tools to execute predefined test cases with speed and consistency. It is highly effective in regression testing, performance testing, and repetitive scenarios where precision and repeatability are crucial. Automated testing enables faster release cycles and supports continuous integration and delivery pipelines, making it well-suited for agile and DevOps environments. Tests can be executed across multiple environments with minimal manual intervention, significantly improving productivity and coverage.

Despite these advantages, automated testing has its limitations. It requires a significant upfront investment in tools, training, and infrastructure. The development and maintenance of automated test scripts can also be challenging, especially when the application under test undergoes frequent changes. Furthermore, automated systems may lack the contextual awareness needed to detect visual bugs, complex user interactions, or non-functional issues such as usability problems. They can also produce false positives or negatives if not properly configured or maintained.

In essence, both approaches have complementary strengths. Traditional testing provides depth and insight where human perception is critical, while automated testing offers speed and scalability for routine and repetitive tasks. As such, many organizations adopt a hybrid testing strategy, using automation for stable, high-volume testing areas and manual testing for exploratory, edge-case, and user-focused scenarios.

Figure 2.0 shows the comparisons between Traditional and Automated Testing.

	Manual Testing	Automated Testing
Pros	• Anyone can test • Easiest way to improve quality • Great for people who may not have formal testing experience • Focused on the customer's work flow primarily	• Faster test cycles • Able to identify more defects in shorter time frame • Saves companies money after the third time the test cases are run • Is an excellent way to meet the testing needs of agile development • Easy to focus on all possible work flows • Higher product quality over manual testing
Cons	• May not identify all test cases • May not identify all defects • Lower product quality because of higher defect count	• Test scripts typically written by automation testing scripter • Requires manual test cases to be written first that are then automated • Requires automation testing platform • Not cost effective for less than 3 test cycles

Figure 2.0

When To Use Traditional Testing

Traditional testing is suitable for:

1. Exploratory Testing: Traditional testing is ideal for exploratory testing, where human testers can explore the software application without preconceived notions or test scripts.

2. Usability Testing: Traditional testing is suitable for usability testing, where human testers can provide feedback on the software application's usability and user experience.

3. One-Time Tests: Traditional testing is suitable for one-time tests, where the test cases are not repetitive or complex.

When To Use Automated Testing

Automated testing is suitable for:

1. Repetitive Tests: Automated testing is ideal for repetitive tests, where test cases are executed multiple times, and consistency is crucial.

2. Regression Testing: Automated testing is suitable for regression testing, where changes to the software application need to be verified against existing functionality.

3. High-Risk Applications: Automated testing is suitable for high-risk applications, where defects can have significant consequences, and thorough testing is essential.

Types Of Ai-Powered Testing Tools

AI-powered testing tools can be categorized into several types based on their functionality and application:

1. Test Automation Tools: These tools use AI to automate testing tasks, such as test script generation, test execution, and defect detection.

2. Test Data Generation Tools: These tools use AI to generate test data, including input data, user interactions, and environmental conditions.

3. Defect Prediction Tools: These tools use AI to predict defects in software applications based on historical data, code complexity, and other factors.

4. Test Optimization Tools: These tools use AI to optimize testing processes, including test case prioritization, test suite reduction, and test execution scheduling.

5. AI-Powered Testing Frameworks: These frameworks provide a set of tools and APIs that enable developers to build and integrate AI-powered testing capabilities into their applications.

Examples of AI-Powered Testing Tools

Some examples of AI-powered testing tools include:

1. Applitools: A visual testing tool that uses AI to identify visual defects in software applications.

2. Functionize: A test automation tool that uses AI to generate test scripts and predict defects.

3. Sealights: A test automation tool that uses AI to optimize testing processes and predict defects.

4. Mabl: A test automation tool that uses AI to generate and maintain test scripts, and provide real-time feedback.

AI Techniques in Automated Software Testing

Artificial Intelligence (AI) is revolutionizing the field of automated software testing by introducing intelligent mechanisms that go far beyond traditional scripting. The integration of AI allows systems to learn from data, adapt to software changes, and intelligently drive test processes, resulting in greater efficiency and accuracy.

Machine Learning (ML) plays a foundational role in AI testing by analyzing historical defect data and application behavior to predict future issues. This enables automated test case generation and defect localization, helping testers focus on high-risk areas while reducing manual effort.

Deep Learning, a subset of ML, is particularly effective in recognizing complex patterns in large datasets. It is used in adaptive anomaly detection, enabling systems to identify subtle defects that rule-based systems may overlook—especially in applications with dynamic UIs and extensive data flows.

Reinforcement Learning (RL) adds a feedback loop to testing. AI agents learn from the outcomes of previous tests to improve future test strategies. This is especially beneficial in agile and DevOps environments, where rapid and continuous updates require adaptive test execution.

Natural Language Processing (NLP) transforms requirements, test documentation, and user stories into executable test cases. This bridges the gap between technical and non-technical stakeholders, ensuring that testing aligns with business goals and improving traceability.

Self-Healing Test Automation is another key innovation. These systems automatically update test scripts and UI element locators when the application changes, reducing test maintenance efforts and minimizing downtime during test cycles.

Predictive Analytics, driven by AI, proactively forecasts defect-prone areas, enabling teams to prevent bugs before they impact production. This shifts the paradigm from reactive to proactive quality assurance.

Recent studies and industry use cases—such as integrating AI with frameworks like Selenium, Appium, or platforms like Functionize—demonstrate significant improvements in test coverage, speed, and fault detection. These tools enhance test script optimization, error handling, and real-time reporting.

Together, these AI techniques make automated software testing more intelligent, flexible, and aligned with modern software development methodologies, supporting faster, safer, and more reliable product delivery.

METHODOLOGY

This research employs a qualitative, literature-based approach. The study is grounded entirely in secondary data, focusing on the review and analysis of five selected academic studies related to AI-powered software testing. These studies were sourced from reputable journals, conference proceedings, and scholarly articles published between 2017 and 2025.

The research did not involve any form of primary data collection such as interviews, surveys, or direct observations. Instead, the methodology centers on:

Summarizing the findings from each of the five selected studies, and
Analyzing the data comparatively to identify trends, common outcomes, and differences in their evaluations of AI testing techniques.
The selection criteria for the five studies included:
Relevance to AI-driven automated software testing
Availability of empirical or case-based results
Clear discussion of advantages, limitations, or performance metrics
Data from the studies were examined thematically and comparatively to evaluate the effectiveness of AI in improving test efficiency, accuracy, and adaptability. The final chapter presents key findings derived from the analysis and concludes with observations about the current impact and future potential of AI in software testing.

DATA ANALYSIS AND FINDINGS

Data Analysis

This section presents a detailed analysis of secondary data collected from five empirical studies on the application of AI-powered automated testing in software systems. Each study was selected based on its contribution to evaluating AI's effectiveness in improving three major testing dimensions: test execution time, defect detection, and test coverage. These metrics were analyzed across different domains,
including Salesforce environments, Oracle ERP
systems, healthcare platforms, e-commerce applications, and content management systems.

The analysis followed a comparative and thematic approach, where the reported outcomes from each study were grouped by metric and evaluated for consistency, variation, and implications. The summarized data are presented in Table 4.1 below.

Table 4.1 Analysis of different studies

Study	Domain	Time Reduction	Defect Detection	Test Coverage
Study 1 – Noone & Karne	Salesforce (QA automation)	40%	50% improvement	Not reported
Study 2 – Nagaraj et al.	Salesforce (Traditional vs AI)	45%	20% increase	Accuracy improved to 95%
Study 3 – Raman et al.	Oracle ERP (Procurement)	50%	Not specified	Performance improved by 12.7%
Study 4 – Alexander & Edward	Healthcare	35%	40% improvement	38% increase
Study 5 – Multiple domains (E-Commerce, CMS, CRM)	Various	E-commerce: 65%, CMS: 70%	CRM: 50%	CMS: 90%, E-commerce: 20%

Time Reduction

AI-powered testing proved highly effective in reducing test time across all studies. The CMS and e-commerce platforms saw the highest reductions (70% and 65%), while Salesforce and Oracle ERP systems achieved 40%–50%. These results reflect AI's ability to automate repetitive tasks, optimize scheduling, and accelerate delivery cycles.

Defect Detection

Four studies reported significant gains in defect detection. The Salesforce (Company B) and CRM systems achieved a 50% increase, while healthcare testing improved by 40%. These results demonstrate how AI enhances accuracy through predictive modeling and machine learning-based error detection.

Test Coverage

AI tools also improved test coverage, with CMS reaching 90%, healthcare 38%, and e-commerce 20%. These gains show AI's strength in generating adaptive test cases and expanding test scope without increased manual effort.

Overall Effectiveness

Across all studies, AI consistently:

Reduced test time (35%–70%)

Improved defect detection (20%–50%)

Enhanced test coverage (20%–90%)

These findings confirm that AI-powered testing is an effective solution for faster, more accurate, and broader software quality assurance.

FINDINGS

This study analyzed five empirical studies to evaluate the effectiveness of AI-powered testing compared to traditional approaches. The results show that AI significantly improves test efficiency, accuracy, and coverage. Unlike traditional testing, which is time-consuming and prone to human error, AI automates repetitive tasks and predicts defects using machine learning. Across various domains, AI reduced test time by up to 70%, increased defect detection rates by 40%–50%, and boosted test coverage—reaching 90% in CMS systems. These findings confirm that AI-powered testing not only speeds up development cycles but also delivers more reliable and scalable testing outcomes, aligning with the study's objectives.

Discussion

The results support the growing recognition that AI is transforming software testing from a repetitive, manual process into an intelligent, adaptive, and proactive quality assurance approach.

Effectiveness of AI in Testing

Efficiency Gains: The significant reductions in test execution time validate AI's capability to automate complex workflows, reduce bottlenecks, and align well with Agile and DevOps practices.

Accuracy & Reliability: AI's use of historical data and anomaly detection models improved defect identification, suggesting higher accuracy than traditional methods.

Coverage Expansion: NLP and test generation algorithms allowed wider and more intelligent coverage, reducing blind spots that manual or scripted testing might miss.

Practical Implications

Real-World Scalability: The studies confirmed that AI tools are effective even in large-scale and critical systems such as ERP and healthcare platforms.

Adaptability: AI frameworks such as self-healing automation demonstrated adaptability to UI and code changes, lowering maintenance costs.

Human-AI Collaboration: Despite its advantages, AI lacks human intuition—hence, a hybrid testing approach remains valuable, especially for exploratory and usability testing.

Challenges and Limitations

Model Interpretability: AI models sometimes function as black boxes, making it difficult to validate decision-making processes.

Data Dependency: The quality and quantity of training data significantly affect AI model performance.

Ongoing Maintenance: AI models must be continuously updated to adapt to software changes, which requires technical expertise and resources.

CONCLUSION

AI-powered automated testing has emerged as a transformative solution in modern software development. This study, through secondary data analysis from five empirical sources, concludes that AI significantly enhances testing by reducing time, increasing defect detection accuracy, and improving test coverage.

While AI testing tools demonstrate robust performance across diverse domains, challenges such as model transparency, data dependency, and integration complexity still need to be addressed. Nonetheless, the consistent benefits across studies indicate that AI testing is not only effective but also scalable and adaptable to evolving software environments.

Future directions/suggestions

Improving explainability and trust in AI models,
Expanding validation across more varied and larger datasets,
Developing hybrid models that balance automation with human oversight.
Ultimately, the findings suggest that AI is well-positioned to play a central role in the future of software quality assurance, enabling faster, smarter, and more reliable software releases.

FUTURE DIRECTIONS

Future research should focus on improving the reliability and transparency of AI-powered testing tools to build trust in their results. More studies are needed to create high-quality, diverse datasets that improve the accuracy of AI models used in testing. It is also important to develop testing systems that combine both AI and human input, especially for usability and exploratory testing. As software becomes more complex, future tools should be able to adapt quickly to changes and work well across different platforms like mobile, web, and enterprise systems. Integrating AI testing into fast-paced development environments, such as DevOps and CI/CD pipelines, will also be key to improving speed and efficiency. Lastly, future work should address ethical and security concerns to ensure that AI tools are safe, fair, and responsible in real-world testing.

REFERENCES

1. Ali, S., Mohan, A., Rauf, A., & Jabeen, F. (2022). Challenges and limitations of AI in software testing: A systematic review. Journal of Software Engineering and Applications, 15(3), 45–57.
2. Anand, A., Singh, M., & Kumar, R. (2019). Deep learning-based test case generation. Journal of Software: Evolution and Process, 31(10), e2218.
3. Choudhary, G., Rani, R., & Malhotra, R. (2018). A survey on machine learning techniques for software testing. Journal of Systems and Software, 146, 312–342. https://doi.org/10.1016/j.jss.2018.09.014

4. Garg, P., & Sharma, S. (2021). AI-driven test automation for modern software systems. ACM Computing Surveys, 54(3), 1–29. https://doi.org/10.1145/3446847
5. Garousi, V., Joy, N., & Keleş, A. B. (2024). AI-powered test automation tools: A systematic review and empirical evaluation. arXiv preprint arXiv:2409.00411.
6. Gupta, A., Kumar, R., & Singh, M. (2020). Adversarial learning-based test case generation. Journal of Systems and Software, 162, 110047.
7. Islam, M., Khan, F., Alam, S., & Hasan, M. (2023). Artificial Intelligence in Software Testing: A Systematic Review. In IEEE Tencon 2023.
8. Karhu, K., Kasurinen, J., & Smolander, K. (2025). Expectations vs Reality — A Secondary Study on AI Adoption in Software Testing. arXiv preprint arXiv:2504.04921.
9. Karan, D., Edward, E., & Alexander, D. (2024). Model-based testing automation framework for critical healthcare.
10. Kim, J., Lee, S., & Kim, S. (2019). Transfer learning-based test case automation. IEEE Transactions on Software Engineering, 45(6), 631-644.
11. Kim, J., Lee, S., & Kim, S. (2020). Natural language processing-based test oracle generation. Journal of Software: Evolution and Process, 32(5), e2235.
12. Kumar, R., Singh, M., & Anand, A. (2018). Defect prediction using convolutional neural networks. Journal of Software: Evolution and Process, 30(10), e2203.
13. Lee, S., Kim, J., & Kim, S. (2019). Ensemble learning-based defect prediction. IEEE Transactions on Software Engineering, 45(8), 761-774.
14. Li, Z., Chen, J., & Wang, J. (2020). Graph neural network-based defect prediction. Journal of Systems and Software, 161, 110042.
15. Mandaloju, N., Karne, V. K., Srinivas, N., & Nadimpalli, S. V. (2024). AI-powered automation in Salesforce testing: Efficiency and accuracy.
16. Noone, S., & Karne, V. (2023). A unified approach to QA automation in Salesforce using AI, ML, and cloud computing. International Journal on Recent and Innovation Trends in Computing and Communication (IJRITCC), 11(9). https://doi.org/10.17762/ijritcc.v11i9.8342
17. Panichella, S., Zaidman, A., Di Penta, M., & Canfora, G. (2018). Learning to classify change requests to support software maintenance tasks. IEEE Transactions on Software Engineering, 44(7), 615–639.
18. Patel, S., Singh, M., & Kumar, R. (2018). Natural language processing-based test oracle generation. Journal of Software: Evolution and Process, 30(5), e2193.
19. Raman, V., Sivasankaran, V., & Kumar, S. (2025). AI-powered automation in Oracle ERP procurement systems. In Proceedings of the ICASDGAI Conference 2025. Koneru Lakshmaiah Education Foundation.
20. Ricca, F., Marchetto, A., & Stocco, A. (2024). A multi-year grey literature review on AI-assisted test automation. arXiv preprint arXiv:2408.06224.
21. Singh, M., Kumar, R., & Anand, A. (2019). Hybrid approach for test case generation. Journal of Software: Evolution and Process, 31(5), e2213.

22. Sinha, K., & Sinha Jana, D. (2025). AI-powered software testing: Transforming quality assurance through artificial intelligence. Journal of Computer Science Engineering and Software Testing, 11(1), 20–38.
23. Tan, L., Wang, J., & Li, Z. (2020). Deep reinforcement learning-based test case generation. IEEE Transactions on Software Engineering, 46(4), 434-447.
24. Wairagade, A., & Cuthrell, K. M. (2025). A systematic review of AI-powered software testing in healthcare: Methodologies, challenges, and future directions. Journal of Engineering Research and Reports, 27(4), 264–277.
25. Wang, J., Li, Z., & Chen, J. (2019). Reinforcement learning-based test case generation. Journal of Software: Evolution and Process, 31(10), e2219.
26. Wang, J., Li, Z., & Chen, J. (2020). Transfer learning-based defect prediction. IEEE Transactions on Software Engineering, 46(2), 174-187.
27. Zhang, Y., Li, Z., & Wang, J. (2018). Test case prioritization using reinforcement learning. Journal of Software: Evolution and Process, 30(8), e2206.
28. Zhang, Y., Li, Z., & Wang, J. (2020). Ensemble learning-based test case prioritization. IEEE Transactions on Software Engineering, 46(6), 654-667.

citations consecutively within brackets [1]. The sentence punctuation follows the bracket [2]. Refer simply to the reference number, as in "[3]"—do not use "Ref. [3]" or "reference [3]". Do not use reference citations as nouns of a sentence (e.g., not: "as the writer explains in [1]").

Unless there are six authors or more give all authors' names and do not use "et al.". Papers that have not been published, even if they have been submitted for publication, should be cited as "unpublished" [4]. Papers that have been accepted for publication should be cited as "in press" [5]. Capitalize only the first word in a paper title, except for proper nouns and element symbols.

For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [6].

[1] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955. *(references)*

[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[3] S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[4] K. Elissa, "Title of paper if known," unpublished.

[5] R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.

[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[7] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.