Journal of Artificial Intelligence and Semiconductor Integrated Hardware (JAISIH)
Volume 1 | Issue 2 | April-June 2026 | Pages 14-20
DOI: 10.5281/zenodo.21342922
e-ISSN: 3139-5775
Review Article OPEN ACCESS https://jaisih.org/index.php/jaisih/issue/archive

A Review of Machine Learning Approaches for Software Testing in Secure Modern Systems

Mr. Ram Pratap Singh

Department of Computer Science and Engineering, Lakshmi Narnia College of Technology, Bhopal

rampratap@lnct.ac.in

Corresponding Author: *Ram Pratap Singh (rampratap@lnct.ac.in)

Received: 15 April, 2026 | **Revised:** 5 May 2026 | **Accepted:** 14 June 2026 |

***Abstract*— In today's world software testing is very important to ensure quality, reliability and security of software systems. Testing methods are typically challenged by scalability, efficiency and cost, especially as applications become more complex, fueled by cloud computing, the Internet of Things (IoT), mobile technologies and distributed architectures. Machine Learning (ML) is a revolutionary approach that has shown potential in improving software testing by automating testing processes and making better decisions. This review paper provides an insightful analysis of ML techniques applied in software testing of secure modern systems. It addresses critical uses like vulnerability assessment, automated test case development, test optimisation, and defect prediction. The article delves further into the topic, investigating potential applications of AI and LLMs in software testing and QA. Also included are the latest developments in the field, such as DevSecOps and continuous security testing, as well as security testing approaches. The review points out the advantages, obstacles, and restrictions of existing methods and suggests future research opportunities to create intelligent, scalable, and security-conscious software testing frameworks that enhance software quality and testing effectiveness.**

Keywords—***Machine Learning, Software Testing, Defect Prediction, Security Testing, Large Language Models (LLMs)***

I. INTRODUCTION

The software systems involved in cloud computing, the IoT, mobile apps, and distributed architectures have become increasingly sophisticated [1][2]. As modern secure software applications handle sensitive data and critical operations, ensuring their reliability, quality, and security has become a major concern. The software development life cycle (SDLC) would not be complete without software testing, which ensures that programs meet both functional and non-functional requirements, detects errors, and verifies the system's functionality [3].

Software testing, which seeks to confirm a software product's functionality, performance, and dependability, is an essential component of the SDLC. It entails running the program in a controlled environment to identify flaws, errors, and security holes [4]. Faults, failures, security breaches, and bad user experiences can occur from these issues if they are not corrected. Here, software testing serves as a quality control tool that may avoid expensive failures, keep a company's reputation intact, and, above all else, guarantee client happiness.

Software testing is crucial, but it's expensive since it requires running a lot of test cases at different phases of the SDLC. Both throughout development and after any modifications, additions, or updates, these test cases must be conducted continuously [5]. Thus, software testing is a resource consuming process, imposing financial and time constraints on software development projects. In this context, software development teams continually seek low-cost testing strategies that can provide them with defect detection and quality assurance.

Machine learning (ML) is the process by which computers are able to take in data, process it using algorithms, and then use that data to generate predictions or judgements [6]. ML has gained traction and strength in several domains, software included, all because of the data-centric learning strategy. The benefits of using ML in software testing include faster testing cycles and improved automation, which allows for faster generation of test cases and prediction of defects. This streamlines teams to work more efficiently on critical issues, ensuring faster releases and enhanced product quality. Another benefit of ML is that it allows to test more inputs and scenarios and hence a higher percentage of bugs to be caught. It makes techniques such as mutation testing more viable, thus more comprehensive testing. In addition to enhancing quality, ML also contributes to cost savings by eliminating labor-intensive tasks and reducing the need for post-release fixes, making it a cost-effective and efficient solution for software testing.

A. Structure of the Paper

The paper is organized as follows: Section II focuses on software testing, Section III provides software testing approaches based on machine learning, Section IV looks at security testing and recent trends, Section V reviews the literature, and Section VI concludes the paper with directions for future research.

DOI: 10.5281/zenodo.21342922

II. Role Of Software Testing In Modern Software Systems

Software testing ensures the safety, functionality, and reliability of software systems and is thus an integral aspect of software engineering [7][8]. In order to identify and resolve errors or defects that may have been introduced during development or maintenance, software testing involves thoroughly investigating, assessing, and confirming a software system. The goal is to make sure the software system meets user needs and specifications [9].It is inevitable that software may have bugs or faults, yet it is critical that software function or operate according to specifications. Software maintenance, feature addition, code reworking, bug patching, and development are all potential sources of these errors. Therefore, it makes sense for the software's developers to try it in a variety of situations before giving it to the client.

A. Objectives and Importance of Software Testing

The purpose of testing is to find any lingering design issues before the client receives the product.

- **Bug:** A software bug is an unanticipated defect, fault, or flaw that results from a coding mistake. That is to say, a bug is probably to blame if a software doesn't work as expected.
- **Error:** A programming mistake occurs when the program's specifications do not line up with the program itself.
- **Defect:** A defect is any discrepancy between the actual product and its ideal state, whether due to incorrect, missing, or excess information. There are two possible forms: a product defect and a deviation from what the consumer or end user expects. The software system problem in question remain dormant until it jeopardises the user experience or the system's general functionality. The majority of defects (90%) are caused by process difficulties.
- **Failure:** A flaw that interrupts normal functioning or has a detrimental effect on the user or customer.
- **Quality Assurance:** Is focused on avoiding mistakes. All project stakeholders must follow the established processes, protocols, standards, and templates, and undergo thorough evaluations to guarantee they are test ready, according to quality assurance.
- **Quality Control:** The purpose of quality engineering and quality control is to ensure that all designs meet performance criteria and to stop the production of inferior goods and services.
- **Verification**: The purpose of verification is to ensure that the product can fulfil all of the requirements specified by the client. Meetings and evaluations of plans, codes, papers, standards, and specifications are commonplace in this process. This may be accomplished through the use of inspection meetings, walkthroughs, checklists, and problems lists.
- **Validation:** The goal of validation is to guarantee that the product performs as expected in accordance with the specifications. Validation usually involves testing and happens after verifications are finished.
- **Software Reliability Estimation:** The goal of testing is to identify residual design problems before delivery to the customer [10]. It is possible to gauge the software's dependability by analysing the data collected during test failures. With consistent input from the reliability analysis, the testing process may continue to provide designers and testers with useful information.

B. Levels of Software Testing

Each stage of testing necessitates testing of the program. There are four established tiers of testing, and each one checks a different part of the program. Software quality, dependability, and system performance are all enhanced at each stage of the SDLC thanks to these levels, which guarantee systematic problem identification.

1) Unit Testing

Unit testing involves independently evaluating the smallest testable components, or units, of an application. It zeroes in on the smallest building block of software, the module. For that reason, it is also known as testing modules. Developers handle it. First, an example Think of Calculator as an example. Each module of the calculator system may be tested independently using unit tests, such as Number sense (adding, subtracting, multiplying, and dividing) has to be assessed separately.

2) Integration Testing

The goal of evaluating the integration of units is to find problems with how they work together. The term refers to the steps taken to ensure that two software modules work well together [11]. The three possible approaches are the "big bang," "top down," and "bottom up" methods.

3) System Testing

System testing just means testing the whole system. This is the final check the developer does before releasing the program to the public for review. System undergoes a variety of tests, including usability, stress, regression, and more. Designing a test strategy is the first and most crucial stage of system testing.

4) Acceptance Testing

This is the stage of testing when the system's acceptability is checked. There are a number of variations, including alpha, beta, user, and business acceptability testing [12][13]. The final consumers are the ones that accomplish this. The result gives the buyer a good idea of the product's quality, which is crucial for making a purchase decision.

C. Challenges in Modern Software Testing

Modern software testing faces several technical, organizational, and operational challenges that can affect software quality, testing efficiency, and timely project delivery. The major challenges are discussed below:

- **Lack of communication:** Miscommunication and miscommunication between clients and development teams, can cause unclear requirements, which translates to wrong test cases and poor testing efficiency.
- **Missing documentation:** Incomplete/ missing functional and non-functional requirements may result in the developers and testers missing out on critical features, which may result in unnecessary testing and implementation errors [14].
- **Diversity in testing environments:** Testing across the diverse number of devices, browsers, and

DOI: 10.5281/zenodo.21342922

platforms can be difficult to cover all user environments.

- **Inadequate testing:** There is not enough testing and limited regression testing may leave defects undetected, negatively affecting software reliability and the risk of failure after deployment.
- **Identifying the right automation tool or framework:** Choosing the wrong automation tool or framework can harm the testing efficiency, result in higher testing costs, and lead to compatibility problems with the project requirements.
- **Employing skilled testers or training existing teams:** Effective test automation requires skilled professionals. A lack of expertise or inadequate training can hinder automation implementation and maintenance.

III. Machine Learning Approaches In Software Testing

Software testing procedures may be made more efficient with the use of ML techniques. A deeper comprehension of clients is possible with the use of this technology. There are a lot of potential for consumers when ML techniques are used for software testing [15]. The usage of ML techniques has a direct influence on software testing and helps maintain invariant testing procedures. Software testing models often use ML technologies. The steps involved in testing software are not that complex. In software testing and maintenance, predictive analytical methodologies are frequently utilised. ML technology outperforms predictive analytics, according to a few of sources. Keeping ML technology efficient requires an effective layered algorithm, which has been successfully achieved [16]. A method called ML is used to look at how the functions work at the output level. ML is put into action using a scenario that is provided by software testing. Based on the process's architecture, software testing approaches have various properties. Various analyses make use of the various activities to practise appropriate parts of their functions.

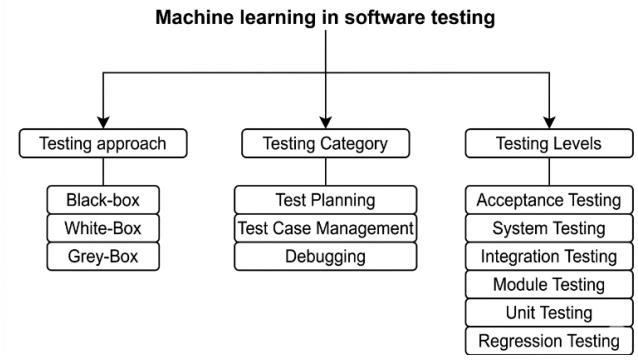

Fig. 1. Machine Learning in Software Testing: Approaches, Categories, and Levels

Figure 1 illustrates the role of ML in software testing by categorizing testing approaches, testing categories, and testing levels, providing a structured overview of its applications in modern software testing.

A. Machine Learning Techniques

ML is often considered a foundational area of AI. A simple definition would be that it is a subfield of AI that studies how to program computers to do cognitive tasks. This enables robots that have been intentionally built to mimic the mental processes of humans. ML allows for the programming and development of databases that can store massive volumes of data according to specific requirements [17]. Supervised learning and unsupervised learning are the two primary forms of ML.

1) Supervised Learning

ML algorithms are trained through supervised learning with the use of input samples and labels. "Supervised learning" refers to training an algorithm with a dataset that already has labels. An algorithm that uses supervised learning is categorised:

- **(a) Regression:** is defined by the output-representing variable. The current endeavour is referred to as a regression problem in the case where the output variable is continuous. Making predictions about the worth of a home or stock is an example of a regression issue.
- **(b)Classification:** Categorical variables are used in classification tasks, such as shape and colour. Supervised learning is utilised by the great majority of ML applications. Supervised learning techniques include things like support vector machines, logistic regression, linear regression, and random forests.

2) Unsupervised Learning

ML algorithms are trained in unsupervised learning using just input samples and no labels for output. Input examples are analysed by the algorithm in an effort to understand their structure and identify trends [18]. Clustering and Association are two other ways unsupervised learning algorithms may be classified. Algorithms like k-means attempt to find preexisting groupings or clusters in data during the clustering process.

B. ML-Based Test Case Generation and Optimization

The automated creation and optimisation of test cases made possible by ML has greatly enhanced software testing. ML algorithms analyze software requirements, source code, historical testing data, and user behaviour patterns to create relevant test cases with minimal human intervention. In addition, ML approaches may be employed to select, optimise, and prioritise test cases based on their likelihood of discovering problems. This practice can help reduce testing time and resource needs [19]. This approach can achieve higher test coverage, high efficiency in detecting failures, and shorten software delivery cycles in modern software development environments [20].

The efficacy of a test suite and the automation of test case generation are totally dependent on ML and optimisation algorithms. Make use of optimisation techniques like Genetic Algorithms, Particle Swarm Optimisation, and Firefly Algorithms to provide high-quality test cases while increasing coverage and decreasing testing energy. This procedure takes a number of UML diagrams as input. Figure.2 shows the main methods used for optimising and generating test cases using ML.

DOI: 10.5281/zenodo.21342922

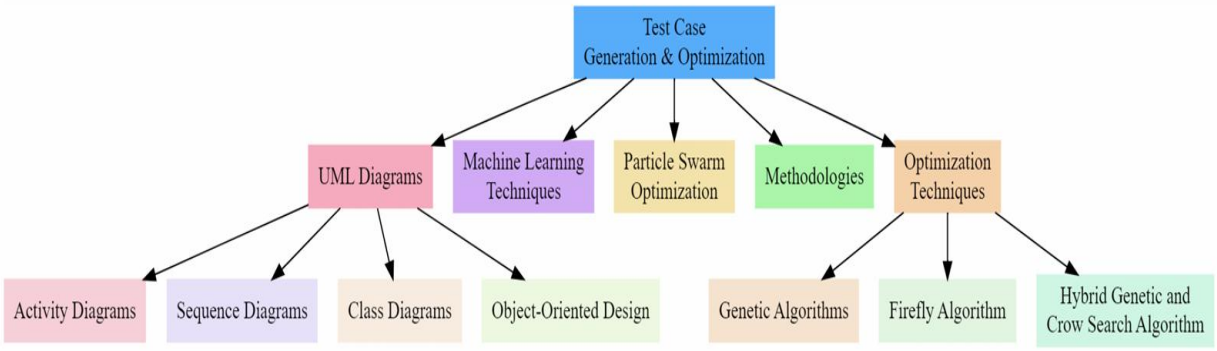

Fig. 2. The process of Test Case Generation and Optimization

C. *Defect Prediction And Fault Detection Using Ml*

Using ML models to forecast software faults from past data, SDP increases software quality while decreasing testing expenses [21]. It supports efficient resource utilization by identifying fault-prone modules during the SDLC. ML techniques analyze software metrics, source code, execution logs, and defect history to detect potential faults at an early stage[22]. Common algorithms, including Random Forest, SVM, NNs, and KNN, help prioritize high-risk modules for testing. These approaches enhance testing efficiency, improve software reliability, reduce maintenance effort, and support the development of high-quality software systems[23]

IV. Software Testing In Secure Modern Systems

Software security testing is conducted with the aim of verifying that a software system meets all security requirements. A couple of the most common approaches are functional security testing and security vulnerability testing. In contrast to security functional testing, which verifies that the security system and its designed and developed functionalities work as expected, security vulnerability testing seeks out and finds new vulnerabilities in the system that could be caused by software defects or security design flaws. It is common practice in security vulnerability testing to mimic actual assaults and other forms of intrusion [24][25][26]. So, to find bugs and make sure systems work well, many people use testing methodologies including regression testing, security testing, and penetration testing. This process aids better reliability, risk mitigation and trust in security-critical applications.

A. *Security Testing Methodologies:*

Software development teams can maximise their testing efforts to deliver high-quality software that meets users' expectations and business goals by utilising various methodologies in security testing. This practice is multi-faceted and requires informed decision-making about testing strategy design, trade-off analysis, and optimisation. This article talks about some well-known ways to test for security holes. Each one is meant to find a certain kind of weakness and teach abouts application security:

- **Penetration Testing:** An organization's security flaws can be located through penetration testing, which involves mimicking an actual assault in order to find and exploit those vulnerabilities.
- **Vulnerability Scanning and Static Code Analysis:** The first employs automated techniques to find software components with known security defects, while the latter analyses source code to find SQL injection, cross-site scripting (XSS), and insecure authentication issues before execution.
- **Dynamic and Interactive Application Security Testing (DAST/IAST):** Dynamic testing evaluates application security during execution by analyzing runtime behavior and user interactions. IAST extends this approach by combining runtime monitoring with dynamic analysis to provide more accurate vulnerability detection.
- **Threat Modeling and Security Requirements Analysis:** Security requirements analysis makes ensuring that development incorporates security goals and regulatory standards, while threat modelling finds possible attack routes throughout system design.
- **White Box Security Review and Regression Testing:** White box testing involves thoroughly reviewing the source code to find flaws in the architecture and implementation. Security regression testing verifies that previously resolved vulnerabilities are not reintroduced after software updates.
- **Fuzz Testing:** Fuzz testing supplies unexpected or malformed inputs to an application to identify input validation errors, buffer overflows, and other vulnerabilities that may affect system reliability and security.

B. *Exploring Emerging Trends in Security Testing:*

Security testing constantly evolves with technology and the sophistication of threat vectors. The waterfall method does not allows to go back to a prior phase and make modifications there. In small projects when there is limited room for changes once a stage is concluded, the waterfall model is used. The article takes a look at how security testing is evolving in response to recent trends like continuous security testing, sharing threat data, and incorporating security into the DevOps pipeline.

1) *DevSecOps Integration*

The term "development, operations, security," or "Dev SecOps," has gained popularity as it has become evident how crucial it is for businesses to integrate security into the software development lifecycle. In order to minimise risk, expedite the development of secure applications, and proactively discover vulnerabilities, Dev SecOps is structured to incorporate security in an early and consistent way. The focus of this movement is on working together across the

DOI: 10.5281/zenodo.21342922

cross-functional teams, and security teams need to get involved with the developer and operations teams. This allows security issues to be identified and addressed at a more rapid rate, reducing the effects of any security risks, and matching the pace of modern development practices.

2) Continuous Security Testing

One-time security testing is being replaced by continuous security testing, where application security is continuously assessed using automated methods. Therefore, security testing is occurring organically as part of the development pipeline, in accordance with the concepts of CI/CD. Security tools that run continuously monitor the code base for vulnerabilities and notify developers in real-time whenever they make changes. By doing so, lessen the amount of time that users might be vulnerable to attacks, increase security awareness, and fix problems more quickly.

3) Threat Intelligence Sharing

Cyber dangers are getting more complicated, so companies need to work together to effectively fight new risks. When organisations and individuals share threat intelligence, they are keeping each other apprised of emerging dangers, attack vectors, and vulnerabilities. Being aware of this trend allows organisations to be more prepared for future dangers, see trends in assaults, and take proactive security steps. In order to better adapt to the ever-shifting cyber threat landscape, the cyber security sector must work together and share information [27]

V. Literature Review

This section provides a concise overview of recent publications on the topic of ML approaches used to software testing for contemporary, secure systems. The reviewed studies are summarized in Table I, with a focus on the focus area, technique, key contributions, and the limitations of each study, as well as directions for future research.

N.Varma et al, (2026) presents a research framework based on literature that integrates ML-based risk prediction with intelligent test optimization for data-driven software quality assurance. The paper suggests a multi-tiered design for contemporary quality engineering that integrates risk scoring, operational controls, feedback learning, test selection, and feature engineering [28].

R. Madhumita and D. Chandana (2025), delves into the latest developments in AI-powered software testing, with attention on AI-backed test automation, predictive defect detection, and LLM-based test generation. It also discusses the challenges and opportunities of adopting these technologies while ensuring quality in generative AI systems[29].

H. Chen and M. A. Babar (2024), takes into account the security of software systems that rely on ML by handling both the software's inherent flaws and hostile assaults at various stages. It highlights the lack of comprehensive review work covering this emerging area in software engineering[30].

J. Wang et al, (2024) explores LLMs in software testing in depth, with an emphasis on preparing test cases and fixing programs. Commonly utilised LLMs are also evaluated, along with prompt engineering methodologies, and important difficulties and prospects for the future are highlighted [31].

S. Pargaonkar, (2023) automated security testing is discussed for vulnerability detection throughout the SDLC through Dynamic Analysis, Static Analysis and IAST. It also emphasizes Dev SecOps integration, continuous security testing and proactive Vulnerability Management [32].

M. Abdul Salam et al, (2022) discusses the application of ML in software engineering including defect prediction, bug localization, code completion, and software testing. It emphasizes the benefits of ML for reducing testing time and cost, increasing efficiency of testing processes and assisting in software quality assurance processes [33]

TABLE I. Summery Of The Study On Machine Learning Approaches For Software Testing In Secure Modern Systems

Author	Focus Area	Techniques	Key Contribution	Limitations & Challenges	Future Work
N. Varma et al. (2026)	Data-driven software quality assurance	ML-based risk prediction, intelligent test optimization	Proposed a multi-layer framework integrating risk assessment and test optimization	Limited practical validation and real-world deployment	Validate the framework on large-scale industrial projects
R. Madhumita & D. Chandana (2025)	AI-driven software testing	AI-based automation, predictive analytics, LLMs	Reviewed AI-enhanced testing and generative AI quality assurance	Reliability, explainability, and adoption challenges	Develop trustworthy and explainable AI testing models
H. Chen & M. A. Babar (2024)	Security of ML-based software systems	Secure software engineering, adversarial defense	Reviewed security issues across the ML software lifecycle	Lack of comprehensive security frameworks	Design end-to-end secure ML testing methodologies
J. Wang et al. (2024)	LLM-based software testing	Large Language Models, prompt engineering	Surveyed LLM applications in test generation and program repair	Prompt dependency and evaluation challenges	Improve robustness and benchmarking of LLM-based testing
S. Pargaonkar (2023)	Automated security testing	Static analysis, dynamic analysis, IAST, DevSecOps	Presented integrated security testing across SDLC	Limited automation for emerging cyber threats	Integrate AI-driven continuous security testing
M. Abdul Salam et al. (2022)	Machine learning in software engineering	Defect prediction, bug localization, code completion	Demonstrated ML benefits for software quality assurance	Limited scalability and generalization across projects	Develop scalable and adaptive ML-based testing solutions

VI. Conclusion And Future Work

In the era of ever evolving software systems, software testing has emerged as an essential part of software engineering to guarantee software quality, reliability, and security. This article reviews a number of current ML approaches that have found use in software testing. These methods include security testing, intelligent test case development, software quality assurance, defect prediction, and fault detection. The studies reviewed show that ML has

DOI: 10.5281/zenodo.21342922

the great potential to increase the efficiency of testing by automating testing activities, improving the accuracy of defect predictions, optimizing test execution, and improving the accuracy of detecting vulnerabilities. Moreover, the rise of technologies like LLMs and Dev SecOps has broadened the scope of intelligent and secure software testing. The survey results underscore that ML is reshaping software testing towards a more adaptive, efficient, and secure approach for contemporary software systems.

Future research needs to explore how to build explainable, powerful and scalable ML models for software testing. The combination of these three technologies, LLM, RL, and secure Dev SecOps frameworks, with continuous testing can enhance vulnerability detection capabilities, improve automation processes, and boost testing reliability. For practical implementation, there is a need for standardized benchmarks for evaluation and large industrial implementation.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data supporting the findings of this study are available from the corresponding author upon reasonable request.

Conflicts of Interest: The author declares no conflict of interest.

How to cite this article

R. P. Singh, "A Review of Machine Learning Approaches for Software Testing in Secure Modern Systems," *Journal of Artificial Intelligence and Semiconductor Integrated Hardware*, vol. 1, no. 2, pp. 14–20, Apr.–Jun. 2026, DOI: 10.5281/zenodo.21342922

REFERENCES

[1] J. B. Mehta, "AI-Driven Test Engineering for Cloud-Native Systems," *Int. J. Data Sci. IoT Manag. Syst.*, vol. 5, no. 1, 2026, doi: 10.64751/ijdim.2026.v5i1.297.

[2] P. H. Desai, P. Mahalle, and P. Chandre, "Intelligent Access Control Schemes for the Internet of Everything: A Survey of Techniques, Challenges, and Future Directions," in *Information Systems for Intelligent Systems*, 2026, pp. 95–105. doi: 10.1007/978-3-032-13196-6_9.

[3] S. Ajorloo, A. Jamarani, M. Kashfi, M. Haghi Kashani, and A. Najafizadeh, "A systematic review of machine learning methods in software testing," *Appl. Soft Comput.*, vol. 162, p. 111805, Sep. 2024, doi: 10.1016/j.asoc.2024.111805.

[4] C. López-Martín, "Machine learning techniques for software testing effort prediction," *Softw. Qual. J.*, vol. 30, no. 1, pp. 65–100, February, 2022.

[5] A. Mehmood, Q. M. Ilyas, M. Ahmad, and Z. Shi, "Test Suite Optimization Using Machine Learning Techniques: A Comprehensive Study," *IEEE Access*, vol. 12, pp. 168645–168671, 2024, doi: 10.1109/ACCESS.2024.3490453.

[6] M. Islam, F. Khan, S. Alam, and M. Hasan, "Artificial Intelligence in Software Testing: A Systematic Review," in *TENCON 2023 - 2023 IEEE Region 10 Conference (TENCON)*, IEEE, Oct. 2023, pp. 524–529. doi: 10.1109/TENCON58879.2023.10322349.

[7] J. B. Mehta, "Securing Test Automation in Zero Trust Architectures: A Framework for Continuous Verification," in *2025 International Conference on Computer and Applications (ICCA)*, IEEE, Dec. 2025, pp. 1–5. doi: 10.1109/ICCA66035.2025.11430950.

[8] V. Chaturvedi, "A Review on AI-Driven Project Management for Transforming Software Engineering Perspectives," *Int. J. Adv. Res. Sci. Commun. Technol.*, vol. 4, no. 2, pp. 895–908, Dec. 2024, doi: 10.48175/IJARSCT-22800E.

[9] X. Jia, "The Role and Importance of Software Testing in Software Quality Management," *J. Ind. Eng. Manag.*, vol. 1, no. 4, pp. 39–44, 2023, doi: 10.62517/jiem.202303406.

[10] P. Ammann and J. Offutt, "Introduction To Software Testing," *Introd. to Softw. Test.*, vol. 4, no. 3, pp. 1–345, 2016, doi: 10.1017/9781316771273.

[11] R. Palwe, "Three Layers of Trust in AI Interfaces: Interface, Behavior, and Organization," *Int. J. Sci. Res.*, vol. 15, no. 1, pp. 1152–1160, Jan. 2026, doi: 10.21275/SR26112072531.

[12] A. Anand and A. Uddin, "Importance of software testing in the process of software development," *Int. J. Sci. Res. Dev.*, vol. 12, no. 6, p. 1, 2019.

[13] S. K. Somarapu;, A. Srivastava;, A. B. Singhal;, and A. R. R. Bandaru, "Structural Dependency Analysis of IoT Adoption Drivers for Sustainable Business Operations Using ISM-MICMAC Analysis," in *2026 5th International Conference on Innovative Practices in Technology and Management (ICIPTM)*, Noida, India: IEEE, 2026, pp. 1–6, February. doi: 10.1109/ICIPTM69057.2026.11465718.

[14] E. Kesavan, "The Future of Software Testing: A Review of Trends, Challenges, and Opportunities," *Int. J. Innov. Sci. Eng. Manag.*, vol. 4, no. 2, pp. 53–57, Apr. 2025, doi: 10.69968/ijisem.2025v4i253-57.

[15] Q. Zhang *et al.*, "A survey on large language models for software engineering," *Sci. China Inf. Sci.*, vol. 69, no. 4, p. 141102, 2026.

[16] N. Mulla and N. Jayakumar, "Role of Machine Learning \& Artificial Intelligence Techniques in Software Testing," *Turkish J. Comput. Math. Educ.*, vol. 12, no. 6, pp. 2913–2921, 2021.

[17] K. K. Mohammed, "The Future is Cloud : Modernizing Big Data for the Cloud Era," *Int. J. Sci. Res. Eng. Trends*, vol. 11, no. 5, pp. 1–5, 2025, doi: 10.5281/zenodo.17339856.

[18] S. K. Sahu, A. Mokhade, and N. D. Bokde, "An Overview of Machine Learning, Deep Learning, and Reinforcement Learning-Based Techniques in Quantitative Finance: Recent Progress and Challenges," *Appl. Sci.*, vol. 13, no. 3, p. 1956, Feb. 2023, doi: 10.3390/app13031956.

[19] S. R. Kongarana, A. A. Rao, and P. R. Raju, "Review of Automated Test Case Generation, Optimization, and Prioritization using UML Diagrams: Trends, Limitations, and Future Directions," *Scalable Comput. Pract. Exp.*, vol. 25, no. 5, pp. 3651–3673, Aug. 2024, doi: 10.12694/scpe.v25i5.3030.

[20] V. Chaturvedi, "Modern Software Development with Java , Spring Boot , and Python : A Survey of Frameworks and Best Practices," *ESP J. Eng. Technol. Adv.*, vol. 3, no. 4, pp. 188–197, 2023, doi: 10.56472/25832646/JETA-V3I8P121.

[21] A. Khalid, G. Badshah, N. Ayub, M. Shiraz, and M. Ghouse, "Software Defect Prediction Analysis Using Machine Learning Techniques," *Sustainability*, vol. 15, no. 6, Mar. 2023, doi: 10.3390/su15065517.

[22] S. K. Malaraju and S. K. Madishetty, "AI-Augmented Compiler Optimization for Energyefficient Software Execution on Embedded Systems," in *2025 14th International Conference on System Modeling & Advancement in Research Trends (SMART)*, Moradabad, Uttar Pradesh, India: IEEE, 2025, pp. 1–6, November. doi: 10.1109/SMART66937.2025.11389313.

[23] P. Chandra Shekhar, "Driving Agile Excellence in Insurance Development through Shift-Left Testing," *Int. J. Multidiscip. Res.*, vol. 3, no. 6, pp. 1–18, 2021.

[24] O. Bin Tauqeer, S. Jan, A. Omar Khadidos, A. Omar Khadidos, F. Qudus Khan, and S. Khattak, "Analysis of Security Testing Techniques," *Intell. Autom. Soft Comput.*, vol. 29, no. 1, pp. 291–306, 2021, doi: 10.32604/iasc.2021.017260.

[25] S. A. D. Poovaiah, "EVALUATION OF LOGIC LOCKING SCHEMES AGAINST ORACLE-LESS MACHINE LEARNING ATTACKS AT THE RTL LEVEL," *Int. J. Artif. Intell. Mach. Learn.*, vol. 2, no. 1, pp. 273–296, Apr. 2023, doi: 10.34218/IJAIML_02_01_024.

[26] M. H. Hamza Afzal, "Securing AI Systems Against Adversarial Attacks: A Framework for Building Robust and Trustworthy Machine Learning Models," *Comput. Fraud Secur.*, no. 5, pp. 65–74, May 2024, doi: 10.52710/cfs.867.

[27] L. A. Yeruva, D. Singh, S. Suddala, N. Bhatt, and R. Uddin,

DOI: 10.5281/zenodo.21342922

"Augmented Data Management for Cache Performance, Cybersecurity, and Mobile Integration," *J. Comput. Mech. Manag.*, vol. 5, no. 3, pp. 280–293, May, Jun. 2026, doi: 10.57159/jcmm.5.3.26691.

[28] N. Varma, R. Chandra, S. Iyer, and P. Narayanan, "Data-Driven Software Quality Assurance: Leveraging Machine Learning for Risk Prediction and Test Optimization," *Int. J. Math. Anal. Res.*, vol. 2, no. 1, pp. 01–08, Jan. 2026, doi: 10.64137/3108-2637/IJMAR-V2I1P101.

[29] R. Madhumita and D. Chandana, "Advancing Software Testing: Integrating AI, Machine Learning, and Emerging Technologies," *Int. J. Res. Eng. Sci. Manag.*, vol. 8, no. 1, pp. 93–97, 2025.

[30] H. Chen and M. A. Babar, "Security for Machine Learning-based Software Systems: A Survey of Threats, Practices, and Challenges," *ACM Comput. Surv.*, vol. 56, no. 6, pp. 1–38, Jun. 2024, doi: 10.1145/3638531.

[31] J. Wang, Y. Huang, C. Chen, Z. Liu, S. Wang, and Q. Wang, "Software Testing With Large Language Models: Survey, Landscape, and Vision," *IEEE Trans. Softw. Eng.*, vol. 50, no. 4, pp. 911–936, 2024, doi: 10.1109/TSE.2024.3368208.

[32] S. Pargaonkar, "Advancements in Security Testing: A Comprehensive Review of Methodologies and Emerging Trends in Software Quality Engineering," *Int. J. Sci. Res.*, vol. 12, no. 9, pp. 61–66, Sep. 2023, doi: 10.21275/SR23829090815.

[33] M. Abdul Salam, M. Abdel-Fattah, and A. Moemen, "A Survey on Software Testing Automation using Machine Learning Techniques," *Int. J. Comput. Appl.*, vol. 183, no. 51, pp. 12–19, 2022, doi: 10.5120/ijca2022921919.