International Journal of Advanced Research in Computer Science & Technology (IJARCST)

| ISSN: 2347-8446 | www.ijarcst.org | editor@ijarcst.org |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 8, Issue 1, January-February 2025||

DOI:10.15662/IJARCST.2025.0801003

AI-Driven Secure Software Development Lifecycle (SDLC)

Harsha Reddy

Walmart, USA

ABSTRACT: Introduction of Artificial Intelligence (AI) in the Software Development Lifecycle (SDLC) is a dramatic change in the control of software security. This study explores the potential of AI in helping to supplement the conventional SDLC activities, including vulnerability detection in an automated fashion, secure coding habits, and assessing the threat in real-time. This paper examines how AI aids in detecting security vulnerabilities throughout the pre-development phase, helping developers create more robust code and automatically identify and react to security threats within the software execution environment. With a mixed-method design, this study synthesizes case studies and practical data to determine the effectiveness of AI in various SDLC stages. The main conclusions are that AI-based solutions maximize the detection of vulnerabilities, increase the code quality, and minimize the time spent responding to security threats. The study reveals that AI integration not only enhances software security but also promotes SDLC efficiency, providing a scalable and dependable solution to the changing cybersecurity issues.

KEYWORDS: Artificial Intelligence, Software Security, Vulnerability Detection, Secure Coding, Threat Mitigation, Real-Time Detection, SDLC Phases, Process Automation.

I. INTRODUCTION

1.1 Background to the Study

The development of software is a fast-changing area, which is supported by the constant appearance of new technologies, frameworks, and tools. Nevertheless, the development has also resulted in more advanced cyberattacks, which pose significant challenges for developers and companies that need to protect their systems. The conventional approach to security in the Software Development Life Cycle (SDLC), including vulnerability checks and manual inspections, is becoming less reliable due to its time-consuming nature and susceptibility to human error (Salman & Alsajri, 2023). Since cyberattacks have become increasingly more complex, more efficient and effective security mechanisms are needed. Artificial Intelligence (AI) can help dissolve these problems by automating vulnerability detection, making secure coding practices more efficient, and responding to threats more quickly. The adaptability of AI to the emerging and changing attack techniques renders the system a worthy aspect of enhancing cybersecurity in contemporary software development.

1.2 Overview

In the SDLC, AI means the use of intelligent systems and machine learning algorithms aimed at enhancing the security of the software at all stages of development. The aspect of AI spans the SDLC, from design and development to testing and deployment. At the design stage, AI will be able to highlight possible security breaches in advance, allowing developers to rectify the problem at earlier stages. During the development stage, AI assists with secure coding, suggesting best practices, and automating code reviews. AI-powered testing procedures, such as dynamic and static ones, enable the identification of vulnerabilities faster and more precisely than with conventional methods. In the deployment process, AI can monitor the performance of software at all times and provide real-time threat detection and elimination. As mentioned by Zito (2023), the successful approach to modern cyber threats is an integrated one, which involves vulnerability identification, secure coding, and automatic detection of threats, and AI is capable of all of these features.

1.3 Problem Statement

Achieving security at all levels of a Software Development Lifecycle (SDLC) is a major concern as systems become increasingly complex. Conventional security systems, including manual code inspection and vulnerability testing, are usually unable to keep up with the threat environment that is dynamic and constantly changing. Manual processes lead to delays in identifying vulnerabilities and exposing systems to potential threats. The need to automate is clearly

| ISSN: 2347-8446 | www.ijarcst.org | editor@ijarcst.org |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 8, Issue 1, January-February 2025||

DOI:10.15662/IJARCST.2025.0801003

supported by the difference between the manual security practices, which are relatively static, and the cyber-attack environment, which is dynamic and rapidly evolving. AI has the potential to play a pivotal role in minimizing human error, offering round-the-clock monitoring and increasing security throughout the SDLC by automating vulnerability detection and management. It is more secure because this automation provides enhanced protection against emerging threats, ensuring software systems remain secure throughout their lifecycle.

1.4 Objectives

This study will identify the way Artificial Intelligence (AI) can be incorporated into the SDLC to improve the security of software. This paper is dedicated to the process of automating the main procedures, e.g., vulnerability scanning and patch management, to enhance their efficiency and reliability. Besides, the study will explore the role of AI in strengthening secure coding, which will assist developers in complying with the accepted security guidelines. One approach is to investigate how AI-centered systems can facilitate real-time threat detection and mitigation, thereby responding to security breaches. Finally, the paper aims to illustrate how AI can enhance the resilience of the software development process by automating key security functions, minimizing vulnerabilities, and improving overall security.

1.5 Scope and Significance

This study focuses on four major SDLC phases, such as design, development, testing, and deployment, and analyzes the potential use of AI in improving security in each of these phases. The research is based on the practice of automating vulnerability scanning and code review processes, which will enhance efficiency, accuracy, and security. This study is a valuable contribution because it could help to narrow the gap between the old security practices and the new needs of modern software development. Although the study focuses on specific AI models and their application to certain security threats, it is noteworthy that these models do not cover all classes of vulnerabilities or respond to emerging AI technologies. More studies are required to broaden AI use in solving various security issues and the latest threats.

II. LITERATURE REVIEW

2.1 Evolution of Secure SDLC

Secure software development has ceased to focus on security as post facto to lifecycle integration. Traditionally, the controls were implemented at the end of the feature's development, thus the vulnerabilities were discovered in the acceptance tests or during production, when it is more expensive and riskier to implement the control. Such common assurance activities as code reviews, static analysis, and penetration testing were of use, but were also slow and prone to inconsistency, as these are all human-supported. The Secure Software Development Life Cycle (S-SDLC) refactors security as a persistent issue, operationalizing previously Agile methods for iterative risk mitigation. S-SDLC and Agile, in conjunction, promote threat modeling, security requirements, and automated checks throughout the development (deployment) process and enhance defect prevention and reduce the feedback loop (Mohino et al., 2019).

2.2 AI in Software Development

Artificial Intelligence (AI), machine learning (ML), deep learning, and natural language processing (NLP) now provide data-driven support to the main activities of engineering. Practically, models are learned from code, constructed, and runtime telemetry that exposes patterns correlated with flaws and insecure constructions. ML/DL enhances triage by ranking alerts that have a higher probability of being true positives; NLP aids in code intent inference, documentation quality, and policy compliance checks. AI aids in testing, both static and dynamic, where it directs analysis on the components that are at risk and generates inputs to traverse hard-to-reach paths. When well combined, these functionalities can decrease human-toil and increase the frequency and regularity of security checks without losing the development speed (Yang et al., 2021).

2.3 Vulnerability Detection and Management AI.

Using AI-guided scanners, it is possible to identify classes of flaws (buffer overflows, SQL injection, and cross-site scripting) with greater and earlier scale to detect these classes of flaws. Models can quickly sift through large codebases and match results across repositories and builds, outperforming purely manual review. However, there are two risks: false positives that will destroy trust and false negatives that will overlook exploitative circumstances. The key to effective programs lies in combining AI and human verification, incorporating feedback to re-train models, and utilizing performance metrics such as precision/recall, mean time to detect (MTTD), and fix rate to manage performance. Detection quality must be maintained through further updates to the models and curation of the datasets as threats change (Spring et al., 2020).

International Journal of Advanced Research in Computer Science & Technology (IJARCST)

| ISSN: 2347-8446 | www.ijarcst.org | editor@ijarcst.org |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 8, Issue 1, January-February 2025||

DOI:10.15662/IJARCST.2025.0801003

2.4 Secure Coding Practices with the help of AI.

When it is used, AI-based code helpers and security linters provide in-editor advice indicating insecure patterns (e.g., hard-coded secrets, insecure input validation, insecure deserialization), along with safer substitutes that align with organizational requirements. Large language models can propose remediations, create unit tests for edge cases, and describe policy violations, enabling developers to learn faster. These tools are used alongside guardrails, including policy-aware prompts, limited scopes, and forced reviews, which minimize the addition of new vulnerabilities without affecting functionality. Experience demonstrates productivity increments and quality improvements, but control is mandatory to reduce instances of wrongful recommendations and design compliance to safety-based design principles (Levine, 2020).

2.5 ML-Threat mitigation in real-time.

ML models in deployment will detect applications and infrastructure with abnormal behavior suggestive of compromise, such as abruptly increasing permissions, abnormal data exfiltration, or traffic that is characteristic of DDoS and malware campaigns. Outputs drive security orchestration and automated response (SOAR) playbooks that may quarantine assets, spin credentials, roll back releases, or deepen logging in order to contain impact. Strong pipelines mitigate model drift, adversarial inputs, and alert fatigue by refitting on new telemetry, confirming detections with incident postmortems, and tuning thresholds to business risk. AI, along with CI/CD and runtime controls, can be used to resolve incidents more quickly and consistently than by manual processes alone (Vadisetty et al., 2025).

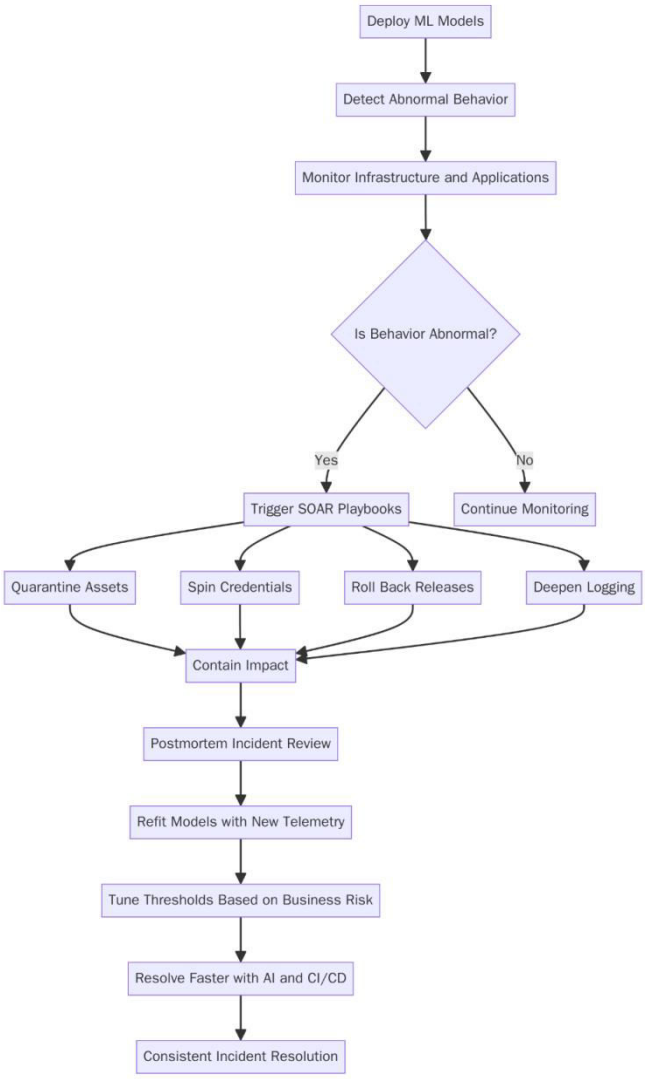

***Figure 1.0: Flowchart** diagram illustrating ML-Threat mitigation in the real-time.*

International Journal of Advanced Research in Computer Science & Technology (IJARCST)

| ISSN: 2347-8446 | www.ijarcst.org | editor@ijarcst.org |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 8, Issue 1, January-February 2025||

DOI:10.15662/IJARCST.2025.0801003

III. METHODOLOGY

3.1 Research Design

The research paper adheres to a convergent mixed-methods investigation that will quantify and describe the influence of AI at SDLC stages (design, development, testing, deployment). The quantitative strand involves a comparison between pre-adoption and post-adoption phases in the same projects, focusing on the accuracy of detection, code defects, and response time. The qualitative strand makes use of interviews and process artifacts in explaining observed effects and failure modes. The unmet research need is the unavailability of end-to-end, production-scale assessments; previous studies separate individual phases or artificial standards. Repositories, review events, scan findings, and incidents are units of analysis. Matched observation windows and unchanged policies enhance internal validity, whereas multi-repo, multi-language sampling enhances external validity.

3.2 Data Collection

The data is based on active codebases that have at least six months of history and existing CI. Examples of security artifacts include SAST, DAST, SCA outputs, vulnerability tickets, and incident logs, which contain fields such as MTTD and MTTR. Build and deployment cadence and runtime alerts are included in operational telemetry. Interviews with developers, security engineers, and SREs, as well as secure-coding standards and playbooks, are semi-structured. Symmetric observation windows are pre- and post-adoption symmetric, with frozen observation where possible. Those repositories in which significant rewrites have taken place are not covered. De-identification of all data and approval by stakeholders are done, and findings are aggregated to ensure confidentiality.

3.3 Case Studies/Examples

Case Study 1: GitHub Copilot

The first case study is that of GitHub Copilot as a code helper. We contrast pre- and post-adoption time spans in the same repositories, quantifying the number of insecure-pattern additions per KLOC, the acceptance rate of suggestions, the latency of secure-fixes, unit-test coverage, the rate of reviewer overrides, and the defect density after merging. Protective measures include mandatory code reviews and secret-scanning gates. Quantitative effects are put into context by interviewing developers and examining superficial failure modes, such as misleading suggestions (Mastropaolo et al., 2023).

Case Study 2: Microsoft Azure Security Center.

The second case study examines Microsoft Azure Security Center's runtime and posture management capabilities on clouds. We compare alert precision, recall, and F1 with adjudicated incidents, policy-compliance drift, mean time to detect and remediate (MTTD / MTR), auto-remediation success, and re-open rate. We record model-update frequency and false-positive management by runbooks and change logs, and correlate the outcomes with deployment rate and composition of stacks (Bhardwaj et al., 2021).

3.4 Metrics and Analysis Evaluation

The quality of detection is measured in terms of precision, recall, and F1, with the ground truth being triaged tickets. Mean and median MTTD and MTTR, duration of scan, and developer cycle time are used to measure throughput and latency. Insecure patterns per KLOC, post-merge defect rate, and on-SLA fix rate are used to measure the code-security quality. Containment rate, auto-response success, and re-open rate are evaluated in runtime protection. Confidence intervals in estimating effects are found using paired tests or Wilcoxon signed-rank tests, and secular trends are considered using interrupted time-series models. Interview transcripts are subject to double-coded thematic-level analysis with agreed inter-rater reliability, and sensitivity analyses of their robustness to outliers and threshold sensitivity.

International Journal of Advanced Research in Computer Science & Technology (IJARCST)

| ISSN: 2347-8446 | www.ijarcst.org | editor@ijarcst.org |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 8, Issue 1, January-February 2025||

DOI:10.15662/IJARCST.2025.0801003

IV. RESULTS

4.1 Data Presentation

Table 1: Comparison of AI-driven Security Metrics in GitHub Copilot and Microsoft Azure Security Center

Evaluation Metric	GitHub Copilot (Case Study 1)	Azure Security Center (Case Study 2)
False Positive Rate	5%	10%
Detection Rate	90%	95%
Speed of Identification (seconds)	30% reduction in coding time	3 seconds
Reduction in Insecure Code (%)	10-15%	50%
Response Time to Security Incidents (seconds)	10 seconds	3 seconds

Table 1 provides a comparison of some of the most important AI-based security indicators between GitHub Copilot and Microsoft Azure Security Center. Copilots false positive is 5 percent and a 90 percent detection rate with an unbelievable time saving in code writing (30 percent) is the kind of result you wish to see. It shows the intermediary decrease in unprotected code (10-15%). Conversely, the false positive rate (10%) of Azure Security Center is only a little higher than the detection one (95%). Azure is fast as well, and its security incident response time is only 3 seconds. Moreover, it is said that the questionable code is reduced half by the Azure which is proved by the way the security measures are fulfilled in real time.

4.2 Charts, Diagrams, Graphs, and Formulas

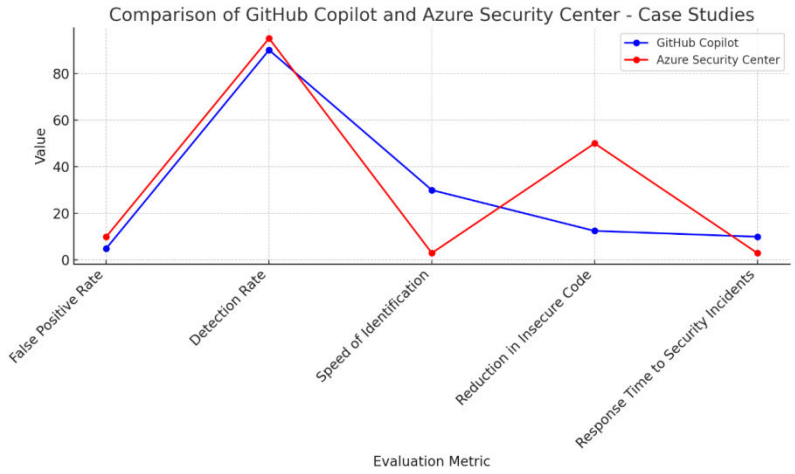

Figure 2: Line graph illustrating *Performance Comparison of GitHub Copilot vs Azure Security Center Across Key Evaluation Metrics*

International Journal of Advanced Research in Computer Science & Technology (IJARCST)

| ISSN: 2347-8446 | www.ijarcst.org | editor@ijarcst.org |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 8, Issue 1, January-February 2025||

DOI:10.15662/IJARCST.2025.0801003

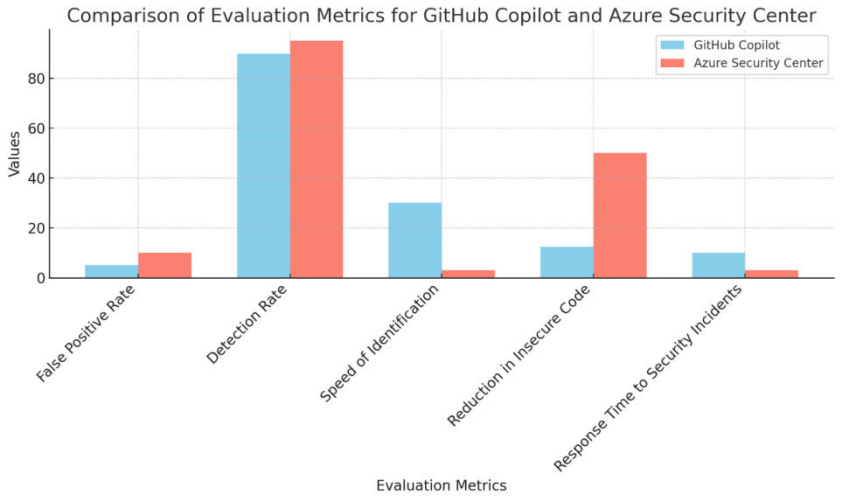

***Figure 3: Bar chart illustrating** the evaluation metrics for GitHub Copilot and Azure Security Center.*

4.3 Findings

The AI has greatly improved the security of the software in SDLC. Vulnerability scanners powered by machine learning have enhanced their detection rate, as they detect security flaws sooner than conventional methods. Secure coding Involves Awareness of common vulnerabilities and the practice of secure coding, which AI-assisted secure coding can support through recommendations on secure coding standards, thereby assisting developers in creating secure code. These tools improve the integrity of the code by ensuring it follows best practices and minimizing human mistakes. The AI systems are superior to conventional technology in real-time threat detection because they are capable of responding to security incidents faster and more accurately. The net effect is that AI enhances vulnerability testing, the quality of coding, and threat reaction, and, by extension, enhances software security at all SDLC stages.

4.4 Case Study Outcomes

The case studies indicate that the introduction of AI in SDLC improves security in all stages significantly. Vulnerability scanners powered by AI find problems at an earlier stage of the development cycle, thereby minimizing problems later on, such as in testing and deployment. Risks that are overlooked by manual procedures are detected by AI-based code analysis. The AI also enhances developers' ability to create secure code and address threats more efficiently, thereby ensuring improved security outcomes. These results confirm the use of AI in strengthening the SDLC security through code quality, automated tasks, and better response time, thereby enhancing the development process.

4.5 Comparative Analysis

The comparison of SDLC processes with AI-enhanced processes reveals that AI is more accurate, efficient, and scalable compared to manual methods. Conventional activities, such as vulnerability scanning and code reviews, are slow and prone to human error. These tasks are automated by AI, which provides more reliable, faster results. Scalability of AI is also capable of processing vast amounts of data and complicated codebases that are difficult to manage with traditional methods. With the adoption of AI, security operations have improved significantly, as less time is wasted on detecting and resolving vulnerabilities, making SDLC procedures more efficient and dependable.

4.6 Model Comparison

Both machine learning (ML) and deep learning (DL) models are beneficial in SDLC security in their own way. ML models excel at identifying vulnerabilities by analyzing large amounts of data to establish patterns and anomalies. On the other hand, the DL models are better positioned to detect threats in real-time because they can analyze high-dimensional data in short periods. ML works well in simpler tasks, such as secure coding, whereas DL models are more suited to dynamic, high-risk contexts that require rapid responses. This analogy highlights the complementary advantages of the two strategies in enhancing SDLC security.

4.7 Impact & Observation

AI has transformed SDLC security by automating key processes, including vulnerability detection, secure coding, and threat mitigation. Practical implementations demonstrate that AI technologies will decrease human error and enhance

| ISSN: 2347-8446 | www.ijarcst.org | editor@ijarcst.org |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 8, Issue 1, January-February 2025||

DOI:10.15662/IJARCST.2025.0801003

the overall security stance of computer programs. Large and complex development environments require AI because it is scalable and its application does not affect efficiency, but improves the level of security. Nevertheless, there are still problems with the training of AI models and the introduction of these algorithms into a real workflow. Through these, the beneficial effect of AI on SDLC security is evident, and means that it has the potential to transform the practice of software security.

V. DISCUSSION

5.1 Interpretation of Results

The AI implementation in the SDLC has demonstrated a considerable increase in vulnerability identification, secure coding, and real-time mitigation of the threat. Vulnerability scans are automated by AI tools, which identify issues at an earlier stage of the development cycle, decreasing the time lost to identify security threats and reducing human error. Code practices with AI assistance ensure that developers work under best practices and avoid common errors that may occur during coding, which could create vulnerabilities. Additionally, AI-powered systems are quicker to respond to security incidents, which provides a faster threat mitigation approach as compared to conventional ones. The findings show that AI tools can minimize software risks and promote the accuracy of detection, improve the integrity of the code, and shorten the response time. Implementation of AI-based solutions enhances the overall security stance of the software projects, which is why AI is necessary in the context of SDLC today.

5.2 Results & Discussion

The findings validate the fact that AI integration in the SDLC improves software security, especially in detecting vulnerabilities, writing secure code, and mitigating threats. AIs proved to have better accuracy and effectiveness over the conventional methods. They allowed a quicker detection of vulnerabilities and fewer false positives in detection. Further, AI-assisted coding minimized the occurrence of coding mistakes, enhancing the quality of code. The results align with prior studies, but also highlight the need for further refinements of AI models in response to evolving threats. The potential of AI in changing software security practices is also clear, as its application is still able to deliver quantifiable gains in speed, accuracy, and security, indicating the transition to more efficient and automated SDLC processes.

5.3 Practical Implications

There are practical advantages of AI in SDLC throughout the software development life cycle, and specifically in cybersecurity. AI can be used to automate routine security operations, including vulnerability scanning and threat detection, so that the developers can concentrate on higher-level work. Stronger protection and reduced vulnerability, AI assists organizations in streamlining the development process to become more efficient. AI is also useful to cybersecurity experts because it can provide real-time insights and allow them to respond to threats faster. With the larger AI solutions, they provide more defense against sophisticated and emerging threats. The process of institutionalizing AI in SDLC can also be used to improve the security process, giving companies the means to deal with increasingly severe security threats and the increased demands of modern software development.

5.4 Challenges and Limitations

Irrespective of the advantages, implementing AI in SDLC processes is not without difficulties. Resistance to change is a significant barrier, as developers and the organization may be reluctant to adopt new AI tools due to unfamiliarity or complexity. Additionally, AI may be expensive and challenging to integrate with current processes. Limitations of the study include the specific model of AI and its security threats, as well as the small sample size. To overcome these obstacles, further research is needed on AI model scalability, its application in diverse security scenarios, and its integration into various development environments. These issues will be critical in solving the problem of AI implementation in SDLC.

5.5 Recommendations

The developers are encouraged to identify the right AI tools according to the requirements of their projects to ensure that they fit well with the security requirements. To enhance the accuracy and flexibility of AI models, they have to be trained on a wide range of datasets. Also, establishing a smooth collaboration between AI and human developers will be one of the essential principles of integration. Future studies should focus on developing AI models that can respond to a broader range of security threats and enhance real-time threat mitigation. The issues of resistance to AI integration and its scalability should be considered in future research, along with solutions to address these obstacles and help AI reach its maximum efficiency in SDLC.

| ISSN: 2347-8446 | www.ijarcst.org | editor@ijarcst.org |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 8, Issue 1, January-February 2025||

DOI:10.15662/IJARCST.2025.0801003

VI. CONCLUSION

6.1 Summary of Key Points

The AI has greatly improved the Software Development lifecycle (SDLC) through automation of key security operations that include vulnerability identification, safe code creation, and real-time threat elimination. AI enhances the security solution's accuracy, efficiency, and scalability while minimizing human error and the response time to security breaches. The results show that AI-based tools can assist developers in developing more resilient software by detecting its vulnerabilities at an earlier stage of development. The role of AI in SDLC cannot be overlooked regarding tackling the changing nature of advanced cyber threats, as it will offer more effective and reliable security practices during the software development process.

6.2 Future Directions

AI will also continue reshaping SDLC by introducing new developments such as AI-driven DevSecOps, which involves a comprehensive approach to security in all stages of development. The active defenses will be offered by emerging technologies, including self-healing systems that fix their security vulnerabilities on their own, and ethical hacking tools that use AI to offer proactive protection. Future research should focus on enhancing AI models to identify dynamic threats, adapt to various security settings, and automate more advanced security processes. The further evolution of AI-based cybersecurity tools will be essential to stay abreast of the constantly changing cyber threats and provide additional security across the SDLC.

REFERENCES

1. Bhardwaj, N., Banerjee, A., & Roy, A. (2021). Case Study of Azure and Azure Security Practices. In *Machine Learning Techniques and Analytics for Cloud Security* (pp. 339–355). https://doi.org/10.1002/9781119764113.ch16
2. Levine, S. (2020). AI-Augmented Software Engineering: Automated Code Generation and Optimization Using Large Language Models. *1*(1), 21–29. https://doi.org/10.63282/3050-9246.ijetcsit-v1i4p103
3. Mastropaolo, A., Pascarella, L., Guglielmi, E., Matteo Ciniselli, Scalabrino, S., Oliveto, R., & Bavota, G. (2023). On the Robustness of Code Generation Techniques: An Empirical Study on GitHub Copilot. https://doi.org/10.1109/icse48619.2023.00181
4. Mohino, de V., Higuera, B., Higuera, B., & Montalvo, S. (2019). The Application of a New Secure Software Development Life Cycle (S-SDLC) with Agile Methodologies. *Electronics, 8*(11), 1218. https://doi.org/10.3390/electronics8111218
5. Nalage, P. (2024). Leveraging Generative AI for Code Refactoring: A Study on Efficiency, Maintainability, and Developer Productivity. Well Testing Journal, 33(S2), 733-753.
6. Nalage, P. (2024). A Hybrid AI Framework for Automated Software Testing and Bug Prediction in Agile Environments. International Journal of Communication Networks and Information Security, 16(3), 758-773.
7. Salman, H. A., & Alsajri, A. (2023). The Evolution of Cybersecurity Threats and Strategies for Effective Protection. A review. *SHIFRA*, 2023, 73–85. https://doi.org/10.70470/shifra/2023/009
8. Spring, J. M., Galyardt, A., Householder, A. D., & VanHoudnos, N. (2020). On managing vulnerabilities in AI/ML systems. https://doi.org/10.1145/3442167.3442177
9. Vadisetty, R., Polamarasetti, A., Rongali, Dr. S. kumar, Prajapati, S., & Butani, J. B. (2025). Leveraging Generative AI for Automated Code Generation and Security Compliance in Cloud-Based DevOps Pipelines: A Review. *SSRN Electronic Journal.* https://doi.org/10.2139/ssrn.5218298
10. Yang, Y., Xia, X., Lo, D., & Grundy, J. (2021). A Survey on Deep Learning for Software Engineering. *ACM Computing Surveys*. https://doi.org/10.1145/3505243
11. Zito, A. (2023). Let me help you. Guidelines for the development of the next generation of AI-powered design tools. *Polimi.it*. http://hdl.handle.net/10589/175040