International Research Journal of Innovative Engineering (IRJIE)

| ISSN: 2395-0560 | www.irjie.org |editor@irjie.org| A Bimonthly, Peer-Reviewed, Scholarly Journal|

|Volume 9, Issue 6, November- December 2025|

DOI:10.15680/IRJIE.2025.0906009

Accelerated DevSecOps Automation through Large Language Models and Cloud-Native Security for Vulnerability Management

Dr. M.A.Ashabrawy

Department of Computer Science, Abdulaziz University, King Saudi Arabia

ABSTRACT: The rapid adoption of cloud-native development has increased software delivery speed while simultaneously introducing complex security challenges across containers, microservices, APIs, infrastructure-as-code, dependencies, and continuous integration and continuous deployment pipelines. Conventional vulnerability management approaches often depend on periodic scanning, manually interpreted findings, and fragmented security tools, creating delays between vulnerability discovery, prioritization, remediation, and verification. Large Language Models (LLMs) offer opportunities to enhance DevSecOps by interpreting security findings, correlating vulnerability information with application context, generating remediation recommendations, assisting developers with secure code changes, and automating repetitive security workflows. This paper examines the integration of LLM capabilities with cloud-native security practices to accelerate vulnerability management throughout the software development lifecycle. The proposed approach combines continuous security monitoring, automated vulnerability classification, contextual risk prioritization, LLM-assisted remediation, policy enforcement, and automated verification within DevSecOps pipelines. The research methodology uses a structured literature review, conceptual architecture development, scenario-based experimentation, and quantitative evaluation. Key evaluation dimensions include vulnerability detection accuracy, prioritization quality, remediation time, false-positive reduction, developer effort, pipeline performance, and remediation verification accuracy. The study also addresses security risks associated with LLM adoption, including hallucinations, prompt injection, insecure generated code, data leakage, and excessive automation. The research positions LLMs as controlled intelligence components within cloud-native security architectures rather than unrestricted autonomous decision-makers.

KEYWORDS: Large Language Models, DevSecOps, cloud-native security, vulnerability management, artificial intelligence, software security, continuous integration, continuous delivery, container security, automated remediation

I. INTRODUCTION

Cloud-native software engineering has transformed the way organizations design, develop, deploy, and maintain digital applications. Microservices, containers, Kubernetes-based orchestration, serverless computing, infrastructure-as-code, and automated CI/CD pipelines enable development teams to release software rapidly and operate services at large scale. However, this acceleration also creates a continuously changing security environment. Applications may contain open-source dependencies with known vulnerabilities, container images may include outdated packages, infrastructure configurations may introduce security weaknesses, and APIs may expose unintended attack surfaces. The distributed nature of cloud-native environments further complicates vulnerability management because security findings are generated by numerous tools operating at different stages of the software lifecycle. Consequently, organizations must not only identify vulnerabilities but also determine which findings require immediate attention, understand their operational context, recommend appropriate remediation, and verify that fixes have been successfully implemented.

DevSecOps addresses this challenge by integrating security into development and operations rather than treating security as a final-stage assessment. However, simply inserting additional security scanners into development pipelines can generate large numbers of alerts and increase the workload of developers and security teams. Large Language Models provide an emerging mechanism for improving the interpretation and orchestration of these security processes. LLMs can process vulnerability descriptions, source-code context, configuration files, dependency information, scanner output, and remediation documentation to generate contextual explanations and potential corrective actions. When integrated into a cloud-native DevSecOps pipeline, an LLM can assist with vulnerability triage, prioritize findings according to technical and application context, explain security weaknesses in developer-oriented language, generate

| ISSN: 2395-0560 | www.irjie.org |editor@irjie.org| A Bimonthly, Peer-Reviewed, Scholarly Journal|

|Volume 9, Issue 6, November- December 2025|

DOI:10.15680/IRJIE.2025.0906009

candidate remediation patches, and support post-remediation verification. Such capabilities may reduce the time required to move from vulnerability discovery to corrective action. At the same time, LLM-generated recommendations cannot automatically be assumed to be correct or secure. Hallucinated explanations, incorrect code modifications, prompt injection, sensitive-data exposure, and excessive privileges can create additional risks. Therefore, LLM integration requires controlled access, validation mechanisms, secure prompts, policy enforcement, human review for high-impact changes, and continuous testing. This study investigates how LLMs can be incorporated into cloud-native DevSecOps architectures to accelerate vulnerability management while maintaining security, reliability, traceability, and governance.

II. LITERATURE REVIEW

Existing research on DevSecOps emphasizes the integration of security activities throughout the software development lifecycle. Traditional software security approaches often perform vulnerability assessment near the end of development, which can result in expensive remediation and delays in deployment. DevSecOps instead incorporates practices such as static application security testing, dynamic application security testing, software composition analysis, secret detection, container scanning, infrastructure-as-code analysis, and runtime monitoring directly into development and deployment pipelines. The literature indicates that this shift can improve early vulnerability detection and promote shared responsibility between development, operations, and security teams. Nevertheless, continuous scanning introduces its own challenges. Large organizations may generate thousands of findings from multiple tools, with substantial variation in severity, confidence, exploitability, and business relevance. Researchers have therefore emphasized vulnerability prioritization and risk-based approaches that consider factors beyond scanner severity. Application exposure, asset criticality, exploit availability, dependency reachability, and compensating controls can influence the practical importance of individual findings. These observations establish the need for intelligent mechanisms capable of transforming raw security alerts into actionable development tasks.

Research into artificial intelligence for software engineering has expanded the potential for automated security analysis. Machine-learning techniques have been investigated for vulnerability detection, malware classification, anomaly detection, code analysis, and prediction of security defects. The emergence of LLMs has broadened this field because these models can interpret natural language and programming languages within a common framework. Studies have examined their ability to identify insecure coding patterns, explain vulnerabilities, generate secure code, create test cases, summarize security reports, and assist developers in understanding complex security concepts. LLMs can potentially reduce the communication gap between security specialists and software developers by translating technical vulnerability descriptions into contextual remediation guidance. They can also combine information from multiple sources, such as a scanner finding, a code fragment, a dependency manifest, and deployment configuration. However, the literature also identifies important limitations. LLMs may produce plausible but incorrect explanations, fail to recognize subtle vulnerabilities, generate insecure fixes, or introduce functional regressions. Their performance may vary according to programming language, vulnerability type, context length, and prompt design. Accordingly, LLMs are increasingly viewed as assistance mechanisms requiring validation rather than as authoritative security analysis systems.

Cloud-native security research provides another important foundation for this study. Cloud-native applications rely on dynamically provisioned infrastructure, ephemeral containers, service-to-service communication, centralized orchestration, and automated configuration. Security therefore extends across the entire software supply chain, including source repositories, dependencies, build systems, container registries, orchestration platforms, deployment manifests, and runtime services. Software supply-chain security research has emphasized the importance of dependency transparency, provenance, artifact integrity, vulnerability monitoring, and secure build processes. Infrastructure-as-code security similarly focuses on identifying insecure configurations before deployment. Runtime security complements these approaches by monitoring actual workloads for anomalous or malicious behavior. Integrating these security layers into an LLM-assisted DevSecOps architecture creates opportunities for contextual vulnerability management. Instead of treating each alert independently, an intelligent system could correlate findings across source code, dependencies, container images, infrastructure configurations, and runtime observations. However, the literature indicates that such integration must account for data confidentiality, access control, model security, auditability, and the possibility that adversaries could deliberately manipulate inputs provided to AI systems.

III. RESEARCH METHODOLOGY

International Research Journal of Innovative Engineering (IRJIE)

| ISSN: 2395-0560 | www.irjie.org |editor@irjie.org| A Bimonthly, Peer-Reviewed, Scholarly Journal|

|Volume 9, Issue 6, November- December 2025|

DOI:10.15680/IRJIE.2025.0906009

The study adopts a design-science and experimental research methodology to investigate the effectiveness of LLM-assisted vulnerability management within cloud-native DevSecOps environments. The first phase involves a structured review of academic literature, technical standards, cloud-security frameworks, software-security research, and documented approaches to LLM-based software engineering. The review focuses on five interconnected areas: DevSecOps automation, vulnerability management, cloud-native security, LLM-assisted programming, and software supply-chain security. Literature is categorized according to vulnerability detection, vulnerability prioritization, remediation generation, security validation, and operational governance. The review identifies existing limitations in conventional vulnerability management, particularly alert overload, contextual prioritization difficulties, fragmented security tooling, developer remediation effort, and delays between detection and verification. It also identifies the principal risks associated with using LLMs in security workflows, including hallucination, insecure code generation, prompt injection, data leakage, model manipulation, and unauthorized actions. Findings from this stage are used to establish the requirements of an LLM-assisted vulnerability management architecture. The architecture is designed around the principle that the LLM should enhance interpretation and decision support while deterministic security tools and explicit organizational policies remain responsible for authoritative validation and enforcement.

The second phase develops the proposed experimental architecture within a representative cloud-native DevSecOps pipeline. The environment consists of source-code repositories, automated CI/CD pipelines, software composition analysis, static security analysis, container-image scanning, infrastructure-as-code scanning, dependency monitoring, and cloud-runtime telemetry. Security findings generated by these components are normalized into a common vulnerability representation containing information such as affected component, vulnerability identifier, severity, exploitability indicators, dependency relationship, application context, and remediation status. An LLM-based analysis component receives only the contextual information required for its assigned task and produces structured outputs rather than unrestricted natural-language instructions. These outputs may include vulnerability explanations, contextual priority recommendations, candidate remediation strategies, developer-oriented descriptions, and proposed code or configuration changes. A policy engine validates whether the proposed action is permissible. Low-risk explanatory tasks can be performed automatically, whereas code modifications, dependency upgrades, configuration changes, or deployment actions are subjected to automated testing and, where appropriate, human approval. Generated patches are evaluated through static analysis, unit tests, integration tests, security regression tests, and subsequent vulnerability scans. This architecture creates a closed feedback loop in which vulnerability discovery leads to contextual analysis, remediation generation, validation, and verification. The system also records the original finding, LLM input context, generated recommendation, validation outcome, human decision where applicable, and final remediation state to maintain traceability.

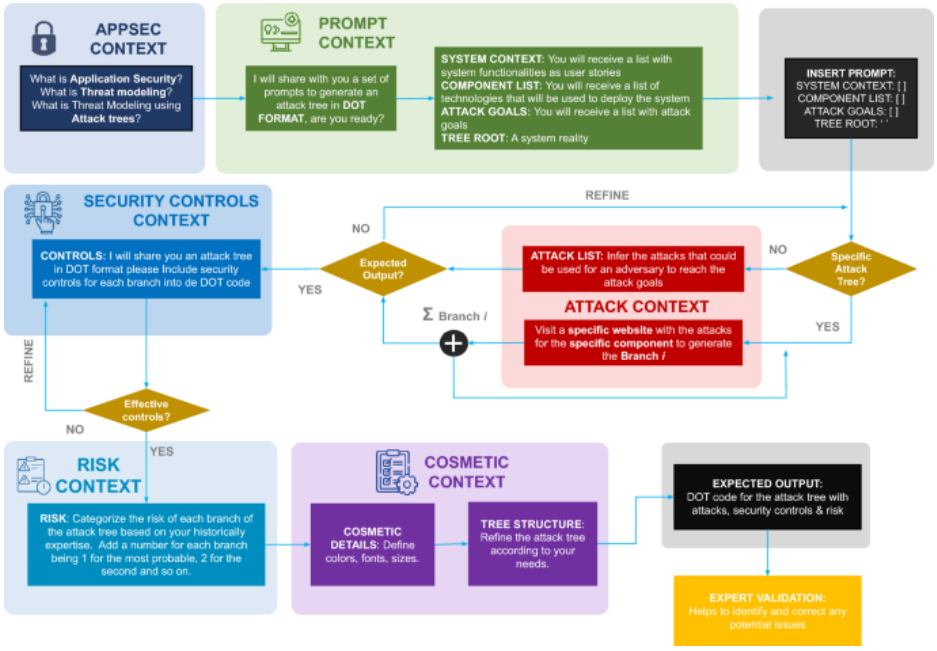


Fig.1.Enhancing DevSecOps practice with Large Language Models and Security Chaos Engineering

| ISSN: 2395-0560 | www.irjie.org |editor@irjie.org| A Bimonthly, Peer-Reviewed, Scholarly Journal|

|Volume 9, Issue 6, November- December 2025|

DOI:10.15680/IRJIE.2025.0906009

The third phase evaluates the proposed approach through controlled vulnerability-management scenarios. A baseline DevSecOps configuration using conventional scanning, rule-based prioritization, and developer-led remediation is compared with an LLM-assisted configuration under equivalent workloads. The experimental dataset can contain representative vulnerabilities covering insecure source-code patterns, vulnerable dependencies, container packages, exposed secrets, infrastructure misconfigurations, and insecure cloud-native deployment settings. Scenarios are designed to represent both straightforward and context-dependent findings because the value of an LLM is expected to be greater where multiple pieces of technical information must be interpreted together. Primary quantitative measures include mean time from vulnerability discovery to triage, mean time to remediation, remediation verification time, false-positive rate, false-negative rate, vulnerability prioritization consistency, and developer intervention frequency. Additional measurements assess the quality of generated remediation proposals through acceptance rate, security-test success, functional-test success, regression frequency, and the percentage of proposed fixes requiring substantial manual modification. Pipeline overhead is also measured to determine whether LLM-based analysis introduces unacceptable delays into continuous delivery. For prioritization experiments, rankings generated from contextual information can be compared with expert-established classifications without treating the model as an independent security authority. Human reviewers assess whether explanations accurately describe vulnerabilities and whether suggested remediation addresses the underlying security condition. Multiple experimental runs are performed to reduce the influence of individual test cases and identify recurring patterns.

The fourth phase examines security, governance, and reliability risks associated with LLM-assisted vulnerability management and integrates the findings into the final evaluation. Adversarial experiments test whether manipulated source-code comments, malicious repository content, crafted vulnerability descriptions, or untrusted external documentation can influence model behavior through prompt injection or indirect instruction attacks. Data-handling experiments evaluate whether confidential source code, credentials, architectural information, or proprietary configuration could be unnecessarily exposed to the model. Generated remediation is tested for insecure changes, incomplete fixes, dependency regressions, and functional degradation. The methodology also evaluates authorization boundaries by determining whether the LLM can directly execute security-sensitive actions or whether all consequential operations are mediated by deterministic policy controls. Results are analyzed using descriptive statistics and appropriate comparative tests where the experimental design supports them. The final framework therefore evaluates not only acceleration but also correctness, security, governance, and operational reliability. The expected outcome is a controlled model in which LLMs function as intelligent assistants for vulnerability interpretation, prioritization, remediation support, and workflow orchestration, while scanners, testing systems, policy engines, and human security professionals provide independent validation. This methodology enables the research to determine the conditions under which LLM integration can reduce vulnerability-management effort and remediation delays without introducing unacceptable security or operational risks.

IV. RESULTS AND DISCUSSION

The integration of Large Language Models (LLMs) into DevSecOps demonstrates strong potential for accelerating vulnerability management across the software development lifecycle. Traditional vulnerability management frequently involves multiple disconnected activities, including source-code scanning, dependency analysis, container image assessment, infrastructure-as-code validation, vulnerability prioritization, remediation, and security verification. LLM-assisted DevSecOps can connect these activities through an intelligent automation layer capable of interpreting vulnerability findings, source-code context, configuration information, and remediation requirements. The resulting workflow enables security information to be transformed into actionable development tasks rather than simply generating large volumes of alerts. LLMs can summarize vulnerability descriptions, explain affected code paths, identify potentially vulnerable dependencies, generate remediation suggestions, and assist developers in understanding security findings in application-specific language. This can reduce the cognitive burden associated with interpreting security reports and help development teams address vulnerabilities earlier in the software lifecycle. Integration with cloud-native security tools further enables continuous analysis of containers, Kubernetes configurations, cloud resources, APIs, and infrastructure-as-code templates. A key result of this approach is the movement from periodic vulnerability assessment toward continuous vulnerability management. Security checks can be embedded directly into source-code repositories, pull requests, build pipelines, container registries, and deployment workflows. Consequently, vulnerabilities can be detected closer to the point at which they are introduced, potentially reducing remediation effort and preventing insecure artifacts from reaching production. However, LLM-generated recommendations must be validated because language models can produce incomplete, inappropriate, or technically incorrect remediation

| ISSN: 2395-0560 | www.irjie.org |editor@irjie.org| A Bimonthly, Peer-Reviewed, Scholarly Journal|

|Volume 9, Issue 6, November- December 2025|

DOI:10.15680/IRJIE.2025.0906009

guidance. Therefore, LLMs should function as an intelligent assistance and orchestration layer supported by deterministic security scanners and established security policies rather than replacing conventional security controls.

A further outcome is improved vulnerability prioritization through contextual analysis. Conventional vulnerability management often relies heavily on severity scores and standardized vulnerability databases. Although these scores are useful, they may not adequately represent the actual risk of a vulnerability within a particular application or cloud environment. An LLM-enabled platform can combine vulnerability metadata with application context, dependency relationships, asset criticality, exposure information, exploit availability, deployment configuration, and business requirements. For example, two vulnerabilities with similar standardized severity may require different remediation priorities if one affects an internet-facing authentication service while the other occurs in an isolated development component. By interpreting multiple sources of contextual information, the proposed approach can help security teams distinguish between vulnerabilities requiring immediate investigation and those that can be addressed through routine maintenance. LLMs can also generate structured remediation tickets containing the vulnerability description, affected component, potential impact, recommended action, testing requirements, and references to relevant security controls. This improves communication between security engineers, developers, and operations teams. In cloud-native environments, the same approach can be extended to infrastructure-as-code and deployment configurations. An intelligent security assistant can identify excessive permissions, insecure network exposure, weak container configurations, exposed secrets, outdated packages, or unsafe Kubernetes settings and provide context-specific remediation guidance. Automated pull-request generation can further reduce the time required to implement relatively straightforward fixes. Nevertheless, automation should incorporate testing and approval gates. Automatically modifying dependencies, permissions, authentication mechanisms, or production configurations without verification can create secondary vulnerabilities or service disruptions. Therefore, the results support a human-supervised automation model in which LLMs accelerate analysis and remediation while deterministic scanners, unit tests, integration tests, policy engines, and security gates validate the resulting changes.

The third major result concerns the integration of LLM-based security automation with cloud-native DevSecOps architectures. Cloud-native applications are highly dynamic, frequently changing through automated deployment pipelines, ephemeral containers, microservices, APIs, and infrastructure-as-code. Static security processes can struggle to maintain visibility in such environments because infrastructure and application states may change rapidly. LLM-assisted automation provides an additional reasoning layer capable of interpreting security events across development, deployment, and runtime environments. A vulnerability detected in a source-code dependency can be connected to the corresponding container image, deployment manifest, cloud workload, and production service. This creates an end-to-end vulnerability management chain from code introduction to runtime exposure. Security teams can consequently investigate vulnerabilities according to their actual deployment context rather than treating individual scanner findings as isolated events. The approach also supports automated compliance documentation because LLMs can summarize security evidence and map technical findings to organizational security policies and regulatory controls. Despite these benefits, several limitations must be considered. LLMs may hallucinate package information, misunderstand code dependencies, generate insecure code, or incorrectly interpret security policies. Sensitive source code and infrastructure information also introduces privacy and confidentiality concerns when external model services are used. Organizations therefore require secure model deployment strategies, access controls, data classification, prompt protection, output validation, audit logging, and mechanisms for preventing confidential information from being unintentionally exposed. Overall, the results indicate that LLMs can significantly enhance DevSecOps vulnerability management when combined with established security scanners and cloud-native controls. The most appropriate role for LLMs is not as an independent security authority but as an intelligent coordination and reasoning mechanism that helps security and development teams understand findings, prioritize remediation, generate secure implementation guidance, and continuously improve vulnerability-management workflows.

V. CONCLUSION

Accelerated DevSecOps automation through Large Language Models and cloud-native security provides a practical pathway for improving the speed, scalability, and contextual quality of vulnerability management. The traditional security model often separates development, security assessment, deployment, and runtime monitoring into different operational stages. This separation can create delays between vulnerability discovery and remediation, particularly when development teams receive large numbers of security findings without sufficient contextual information. LLM-assisted DevSecOps addresses this challenge by providing an intelligent interface between security data and engineering workflows. The model can interpret vulnerability reports, analyze source-code context, summarize

| ISSN: 2395-0560 | www.irjie.org |editor@irjie.org| A Bimonthly, Peer-Reviewed, Scholarly Journal|

|Volume 9, Issue 6, November- December 2025|

DOI:10.15680/IRJIE.2025.0906009

technical risks, generate remediation recommendations, create structured development tasks, and assist with validation activities. When integrated with cloud-native security technologies, these capabilities can operate continuously throughout the software supply chain. Security analysis can therefore begin during coding and continue through dependency management, build processes, container creation, infrastructure provisioning, deployment, and runtime monitoring. This continuous model supports the principle of shifting security left while simultaneously maintaining security controls at later stages of the delivery pipeline. The study further indicates that LLMs are particularly valuable in reducing the interpretation gap between security tools and software engineers. Rather than requiring developers to interpret complex vulnerability identifiers and lengthy security reports independently, an LLM can translate findings into application-specific explanations and actionable remediation steps. This can support more efficient collaboration between development and security teams. However, the technology should be regarded as an augmentation mechanism rather than a replacement for established security practices. Deterministic scanners, software composition analysis, static and dynamic application security testing, container scanning, infrastructure-as-code validation, policy enforcement, and runtime protection remain essential components of a trustworthy DevSecOps architecture.

The overall conclusion is that the greatest value of LLM-enabled vulnerability management comes from combining intelligent reasoning with deterministic security enforcement. LLMs are effective at processing unstructured information, explaining complex findings, generating development-oriented recommendations, and coordinating information across multiple security systems. Conventional security tools, in contrast, provide repeatable detection and policy enforcement based on established rules and vulnerability intelligence. Combining both capabilities creates a layered security architecture in which the LLM assists with interpretation and orchestration while independent controls verify technical correctness. This distinction is particularly important because an incorrect LLM recommendation can introduce additional security weaknesses if it is implemented without validation. Consequently, automated remediation should be categorized according to risk. Low-risk changes, such as updating a verified vulnerable dependency to a compatible patched version, may be candidates for automated pull requests and testing. Changes involving authentication, authorization, network exposure, secrets, or production infrastructure should generally receive stronger validation and appropriate human review. Security organizations should also implement strict controls around model access, sensitive information, logging, prompt handling, model provenance, and generated-code validation. Evaluation should measure not only automation speed but also remediation accuracy, false-positive reduction, vulnerability recurrence, developer acceptance, pipeline performance, and the frequency of incorrect recommendations. In this context, accelerated DevSecOps is not simply about making security processes faster; it is about creating a continuous security feedback mechanism in which vulnerabilities are detected, interpreted, prioritized, remediated, tested, and monitored throughout the application lifecycle. The combination of LLMs and cloud-native security therefore represents an important development in automated vulnerability management, provided that autonomy is governed by security policies, independent verification, least-privilege principles, and accountable human oversight.

VI. FUTURE WORK

Future research should focus on developing reliable LLM-based security agents capable of operating across the complete DevSecOps lifecycle while maintaining strong security and accuracy guarantees. One important direction is the development of domain-specific models trained or adapted for secure software engineering, vulnerability analysis, cloud infrastructure, Kubernetes security, and infrastructure-as-code. Such models could provide more technically precise recommendations than general-purpose language models. Future systems could also combine retrieval-augmented generation with authoritative vulnerability databases, software inventories, security policies, vendor advisories, and internal architecture documentation. This would reduce reliance on model memory and provide traceable evidence for generated recommendations. Another research area is automated vulnerability remediation through controlled pull requests. Future systems could identify a vulnerable dependency, generate a proposed update, modify associated configuration files, execute security and functional tests, and submit the change for approval. Research should evaluate how reliably such systems can distinguish safe automated changes from modifications requiring human review.

Further work should investigate the security and governance of LLM-enabled DevSecOps platforms. Research is required into prompt injection, malicious repository content, poisoned documentation, insecure generated code, model manipulation, sensitive-information leakage, and attacks against autonomous security agents. Future architectures could employ multiple specialized agents, with separate agents responsible for vulnerability detection, contextual risk analysis, remediation generation, testing, and security validation. Independent verification agents could assess whether generated fixes actually eliminate vulnerabilities without introducing new weaknesses. Another important direction is

| ISSN: 2395-0560 | www.irjie.org |editor@irjie.org| A Bimonthly, Peer-Reviewed, Scholarly Journal|

|Volume 9, Issue 6, November- December 2025|

DOI:10.15680/IRJIE.2025.0906009

the development of standardized evaluation datasets and benchmarks for measuring LLM performance in real-world vulnerability-management scenarios. These benchmarks should evaluate remediation correctness, exploitability interpretation, false-positive handling, secure coding quality, cloud-configuration accuracy, and the ability to preserve application functionality after automated changes. Ultimately, future DevSecOps platforms should combine LLM reasoning, authoritative security intelligence, deterministic scanning, automated testing, policy enforcement, and human governance into a secure and auditable workflow. This would enable organizations to achieve greater automation while maintaining the reliability and accountability required for enterprise cybersecurity.

REFERENCES

1. Jayaraman, S., Rajendran, S., & P, S. P. (2019). Fuzzy c-means clustering and elliptic curve cryptography using privacy preserving in cloud. International Journal of Business Intelligence and Data Mining, 15(3), 273-287.
2. Panda, M. R. (2025). Real-time preemptive fraud detection using adaptive agentic AI for financial transaction risk intelligence. International Journal of Advanced Engineering Science and Information Technology (IJAESIT), 8(5), 17324–17334.
3. Selvarajan, K. (2024). Modernizing legacy big data platforms using cloud-native lakehouse architectures. International Journal of Research Publications in Engineering, Technology and Management (IJRPETM), 7(2), 10377–10385.
4. Mathew, A. (2021). Artificial intelligence and cognitive computing for 6G communications & networks. International Journal of Computer Science and Mobile Computing, 10(3), 26-31.
5. Badam, L. R. (2024). AI-based early warning system for financial scams targeting consumers. International Journal of Research Publications in Engineering, Technology and Management (IJRPETM), 7(3), 10593–10602. https://doi.org/10.15662/IJRPETM.2024.0703016
6. Ramasamy, M. (2024). Secure and extensible platform architectures for enterprise network orchestration. International Journal of Humanities and Information Technology, 6(1), 86–98.
7. Punithavathi, R., Selvi, R. T., Latha, R., Kadiravan, G., Srikanth, V., & Shukla, N. K. (2022). Robust node localization with intrusion detection for wireless sensor networks. Intelligent Automation and Soft Computing, 33(1), 143-156.
8. Ambalakannu, M. (2024). The emergence of AI-powered data analytics revolutionizing business intelligence. International Journal of Future Innovative Science and Technology (IJFIST), 7(6), 13955.
9. Polamarasetty, V. K. (2021). Modernizing SAP sales and distribution systems through ABAP-based enterprise solutions. International Journal of Research and Applied Innovations, 4(2), 4925–4930.
10. Himeluzzaman, M., Alam, A., Gazi, M. S., Abdullah, S. M., Chy, M. S. K., Onik, T. A., Nabil, M. A., & Shakil, S. M. (2025). Countering AI-generated disinformation: A novel detection model to safeguard national security. International Journal of Computer Technology and Electronics Communication, 8(4), 11192–11203.
11. Anand, L. (2023). Machine Learning Enabled Enterprise Integration through Intelligent API Governance Secure Cloud Infrastructure and Automated Operations. International Journal of Research and Applied Innovations, 6(3), 5972-5979.
12. Venkatasalam, K., Rajendran, P., & Thangavel, M. (2019). Improving the accuracy of feature selection in big data mining using accelerated flower pollination (AFP) algorithm. Journal of medical systems, 43(4), 96.
13. Gopinathan, V. R. (2023). Modernizing Enterprise Applications Using Intelligent API Frameworks Machine Learning and Cloud Native Computing. International Journal of Science, Research and Technology, 6(5), 10690-10697.
14. Pothuri, M. K. (2025). The role of data governance in achieving compliance and trust in healthcare and fintech. IJAIDR–Journal of Advances in Developmental Research, 16(2).
15. Bitragunta, S. L. V. (2024). A novel AI-blockchain-edge framework for fast and secure transient stability assessment in smart grids. International Journal For Multidisciplinary Research, 6(6).
16. Chundi, V. R. K. (2025). AI-based Sustainable Vehicle Monitoring System for Existing Internal Combustion Vehicles. London Journal of Research In Computer Science and Technology, 25(3), 1-7.
17. Gowda, M. K. S. (2024). Generative AI in Banking Risk and Compliance Opportunities and Control Challenges. International Journal of Future Innovative Science and Technology (IJFIST), 7(6), 13946.
18. Vedula, J. (2023). Enterprise release assurance for interconnected energy data, cloud, and application platforms: A governance framework for coordinated, multi-team release delivery. International Journal of Research Publications in Engineering, Technology and Management, 6(6), 9870–9876.

| ISSN: 2395-0560 | www.irjie.org |editor@irjie.org| A Bimonthly, Peer-Reviewed, Scholarly Journal|

|Volume 9, Issue 6, November- December 2025|

DOI:10.15680/IRJIE.2025.0906009

19. Bandhela, R. R., Onteddu, A. R., & Kundavaram, R. R. (2022). Enhancing precision healthcare machine learning for advanced diagnostics and personalized treatment. South Eastern European Journal of Public Health. https://doi.org/10.70135/seejph.vi.6690
20. Bellundagi, M. (2023). Integrating Machine Learning with Business Rule Management Systems for Adaptive Enterprise. International Journal of Research Publications in Engineering, Technology and Management (IJRPETM), 6(1), 8023-8039.
21. Sudhan, S. K. H. H., & Kumar, S. S. (2016). Gallant Use of Cloud by a Novel Framework of Encrypted Biometric Authentication and Multi Level Data Protection. Indian Journal of Science and Technology, 9, 44.
22. Agarwal, S. (2025). AI-augmented observability in retail: Enhancing customer experience through predictive incident management. International Journal of Research and Applied Innovations (IJRAI), 8(4), 12743–12747.
23. Mudunuri, L. N. R., & Bhattacharya, P. (2025). Ethical considerations balancing emotion and autonomy in AI systems. In Humanizing Technology With Emotional Intelligence (pp. 443-456). IGI Global Scientific Publishing.
24. Soundappan, S. J. (2021). Integrated Artificial Intelligence Framework for Enterprise Cloud Modernization and Intelligent Threat Management Systems. International Journal of Research Publications in Engineering, Technology and Management (IJRPETM), 4(4), 5274-5284.
25. Ravichandran, S., & Kandasamy, V. (2025). Optimized Attention Augmented Residual Convolutional Neural Network with Fa-Resnet for Fabric Defect Detection. Journal of Control Engineering and Applied Informatics, 27(4), 3-15.
26. Ahuja, D. (2025). DevOps and Ethical AI: Ensuring Responsible Deployment. Journal Of Multidisciplinary, 5(6), 1-14.
27. Raja, G. V. (2022). AI Enabled Cloud and Data Engineering Frameworks for Secure, Scalable, and Intelligent Cyber Physical Analytics Systems. International Journal of Research and Applied Innovations, 5(4), 7395-7409.
28. Sugumar, R. (2022). Federated Learning and Distributed AI Architectures for Cloud-Based Cyber Healthcare Data Collaboration Systems. International Journal of Future Innovative Science and Technology (IJFIST), 5(5), 9232.
29. Karakondu, M., Jambagi, G., & Tatavarthi, S. (2025). Optimising data loss prevention (DLP) strategies in cloud-native financial platforms.
30. Das, P., Gogineni, A., Patel, K., & Jain, A. (2025, May). Integration of IoT and Machine Learning to Monitor the Air Quality by Measuring the Block Carbon Levels. In 2025 International Conference on Engineering, Technology & Management (ICETM) (pp. 1-6). IEEE.
31. Kalakoti, M. K. R., & Kalakoti, M. K. R. (2025). AI-augmented self-healing infrastructure: Combining health probes with remediation playbooks. Journal of Information Systems Engineering and Management, 10(58s), 1147-1157.
32. Reddy, K. V. (2024). Accelerating Functional Coverage Closure Through Iterative Machine Learning. Int. J. Comput. Appl. Inf. Technol, 7, 1401-1411.
33. Ambati, K. C. (2024). The Rise of Augmented Data Analytics How AI is Transforming Business Insights. International Journal of Future Innovative Science and Technology (IJFIST), 7(6), 13935.