World Journal of Future Technologies in Computer Science and Engineering (WJFTCSE)
ISSN (Online): 3070-6203
Volume-2 Issue-1 || Jan – Mar 2026 || PP. 62-70 https://wjftcse.org/

Intelligent Vulnerability Prioritization and Remediation in CI/CD Pipelines Using Large Language Models and Knowledge Graphs

DOI: https://doi.org/10.63345/wjftcse.v2.i1.2

Theodor Ion Burcea
Cloud, Cybersecurity and AI Expert
Romania
theodorib3588@gmail.com

www.wjftcse.org || Vol. 2 No. 1 (2026): March Issue

Date of Submission: 04-02-2026	**Date of Acceptance:** 20-02-2026	**Date of Publication:** 13-03-2026

***Abstract*— The exponential growth of software dependencies and the accelerating pace of continuous integration and delivery have created a critical bottleneck in vulnerability management: security teams and developers are overwhelmed by the sheer volume of reported vulnerabilities, most of which are not exploitable in their specific deployment contexts. Traditional vulnerability scanners produce flat lists of Common Vulnerabilities and Exposures (CVEs) without contextual prioritization, leading to alert fatigue, delayed remediation of critical issues, and unnecessary deployment friction. This paper presents an intelligent vulnerability prioritization and remediation framework that combines Large Language Models (LLMs) with knowledge graph reasoning to transform how organizations manage security vulnerabilities in CI/CD pipelines. The proposed architecture integrates: (1) an LLM-based vulnerability intelligence layer that extracts and synthesizes information from CVEs, security advisories, and exploit databases to assess true exploitability; (2) a knowledge graph representation of the application environment that models dependencies, deployment contexts, and security controls; (3) a context-aware risk scoring engine that combines exploitability intelligence with environmental context to produce actionable prioritization; and (4) an automated remediation recommendation system that generates specific, verifiable fixes tailored to the application stack. Experimental validation on production CI/CD pipelines processing over 50,000 vulnerability reports demonstrates that the framework reduces actionable vulnerability volume by 78%, improves critical vulnerability remediation time by 65%, and decreases false-positive driven developer interruptions by 71%. The system generates remediation recommendations with 89% accuracy, with 94% of recommended fixes verified as correct in automated testing. This research demonstrates that combining the reasoning capabilities of LLMs with the structural knowledge of graph-based representations can fundamentally transform vulnerability management from an overwhelming flood of alerts to a targeted, actionable security intelligence system that enables organizations to focus on what truly matters.**

***Keywords*— Vulnerability Management, Large Language Models, Knowledge Graphs, DevSecOps, CI/CD Security, Software Supply Chain Security, SBOM, Remediation Automation**

I. INTRODUCTION

A. The Vulnerability Management Crisis

The software supply chain has become increasingly complex, with modern applications depending on hundreds or thousands of open-source and third-party components. According to industry reports, the average application contains over 500 dependencies, and the number of disclosed vulnerabilities reached record levels in recent years, with over 20,000 new CVEs reported annually [6]. This proliferation of components and vulnerabilities has created a crisis in vulnerability management: security teams

and developers are drowning in alerts while critical vulnerabilities remain unpatched for extended periods.

Traditional vulnerability scanners have exacerbated rather than solved this problem. These tools typically produce flat lists of CVEs with generic severity scores (CVSS), providing little context about whether a vulnerability is actually exploitable in the specific environment where the component is deployed. The result is a massive volume of alerts, most of which are either false positives or vulnerabilities that do not pose actual risk to the organization. Security teams spend excessive time triaging and dismissing low-risk issues, while critical vulnerabilities may be overlooked in the noise.

The challenge is particularly acute in CI/CD environments, where the velocity of development requires rapid security decisions. When a vulnerability scanner flags a component, developers face a choice: halt the deployment to investigate (slowing delivery) or proceed despite the alert (accepting risk). Without context about actual exploitability, this decision is often made with incomplete information, leading either to unnecessary delays or unaddressed risk.

B. The Need for Intelligent Prioritization

Effective vulnerability management requires answering three fundamental questions:

1. **Is this vulnerability actually exploitable?** Many CVEs have no known exploits, require specific conditions that may not exist in your environment, or have been patched by mitigating controls that are already in place.
2. **What is the business impact if exploited?** A vulnerability in a low-risk component of a non-critical service poses very different risk than one in a component handling sensitive customer data.
3. **What is the most efficient remediation path?** Different remediation options (upgrade, patch, configuration change, or acceptance) have different effort levels and timeframes.

Answering these questions requires integrating multiple sources of information: vulnerability databases, exploit intelligence, application architecture, deployment context, and business criticality. This integration has traditionally been manual, time-consuming, and error-prone.

Recent advances in Large Language Models (LLMs) and knowledge graphs offer promising approaches to automating this intelligence integration. LLMs can extract and synthesize information from unstructured sources like CVE descriptions, security advisories, and exploit databases. Knowledge graphs can represent the structured relationships between components, dependencies, and deployment contexts. Together, these technologies can provide the contextual intelligence needed for effective vulnerability prioritization.

C. Research Objectives and Contributions

This paper presents an intelligent vulnerability prioritization and remediation framework that combines LLMs with knowledge graph reasoning to transform vulnerability management in CI/CD pipelines. The framework addresses the following research questions:

1. Can LLMs effectively extract and synthesize exploitability intelligence from unstructured security data?
2. Can knowledge graphs capture the necessary environmental context for vulnerability prioritization?
3. Can the combination of these approaches enable automated, accurate vulnerability prioritization and remediation recommendations?
4. What is the operational impact of such a system on security team productivity and remediation velocity?

The primary contributions of this work are:

- An LLM-based vulnerability intelligence layer that extracts exploitability signals from CVE descriptions, security advisories, and exploit databases;
- A knowledge graph representation of the application environment capturing dependencies, deployment contexts, and security controls;
- A context-aware risk scoring engine that integrates exploitability intelligence with environmental context;
- An automated remediation recommendation system that generates specific, verifiable fixes;
- Experimental validation demonstrating significant improvements in vulnerability management effectiveness.

World Journal of Future Technologies in Computer Science and Engineering (WJFTCSE)
ISSN (Online): 3070-6203
Volume-2 Issue-1 || Jan – Mar 2026 || PP. 62-70 https://wjftcse.org/

The remainder of this paper is organized as follows. Section II reviews related work in vulnerability management, LLM applications in security, knowledge graphs, and DevSecOps. Section III presents the framework architecture in detail. Section IV describes the experimental methodology and evaluation results. Section V discusses implications for practice, limitations, and future research directions. Section VI concludes with a summary of contributions.

II. RELATED WORK

A. Vulnerability Management and Prioritization

Traditional vulnerability management has relied on CVSS scores for prioritization. The Common Vulnerability Scoring System (CVSS) provides a standardized method for rating the severity of vulnerabilities based on exploitability and impact metrics. However, CVSS has well-known limitations: it does not account for environmental context, provides generic scores that may not reflect actual risk in specific deployments, and does not incorporate exploit availability or active exploitation in the wild.

Efforts to improve on CVSS include the Exploit Prediction Scoring System (EPSS), which uses machine learning to predict the likelihood of a vulnerability being exploited. EPSS provides a probability score indicating the chance that a vulnerability will be exploited in the next 30 days. While EPSS adds valuable signal, it still lacks environmental context and does not account for the specific deployment of an organization.

More recently, the Cybersecurity and Infrastructure Security Agency (CISA) has published a Known Exploited Vulnerabilities (KEV) catalog of vulnerabilities that are known to be actively exploited. While the KEV catalog is valuable for identifying vulnerabilities that require immediate attention, it only covers a small fraction of all CVEs and does not provide prioritization guidance beyond the binary "known exploited" indicator.

B. SBOM Generation and Analysis

The Software Bill of Materials (SBOM) has emerged as a critical tool for understanding software composition. SBOM generation has been automated in CI/CD environments, with tools plugging into pipelines and regenerating SBOMs on every build. Automated vulnerability scanning in CI/CD pipelines leverages SBOMs to identify and prioritize vulnerabilities [2].

The NP-hard problem of optimizing vulnerability repair in SBOMs has been addressed using maximum satisfiability solving [1]. This approach treats vulnerability repair as a combinatorial optimization problem, selecting the minimal set of updates that addresses the most critical vulnerabilities. This represents a step toward automated remediation, though it does not incorporate exploitability intelligence or environmental context. Complementary shift-left approaches that scan container images for vulnerabilities before deployment provide an additional layer of defense earlier in the pipeline [4].

C. LLMs in Security

Large Language Models have shown significant promise in various security applications. Recent research has explored the application of LLMs for vulnerability detection, where models analyze code to identify potential security issues. LLM embeddings have been combined with Abstract Syntax Tree (AST) structural analysis to detect malicious changes in infrastructure-as-code, achieving F1-scores of 0.92 [5].

In vulnerability management specifically, LLMs have been used to summarize CVE descriptions and extract structured information. However, these applications typically focus on information extraction rather than prioritization or remediation recommendation. The integration of LLM reasoning with structured knowledge representations remains an area requiring further exploration. More broadly, industry adoption of AI/ML-augmented DevSecOps automation has been documented across a growing body of practitioner and academic literature [6]-[11].

D. Knowledge Graphs in Security

Knowledge graphs have been applied to various security problems, including threat intelligence and attack graph analysis. In vulnerability management, knowledge graphs have been used to model relationships between vulnerabilities, components, and dependencies. However, integrating knowledge graphs with LLM-based reasoning for vulnerability prioritization has not been extensively explored. Representing deployment topology accurately also benefits from established container orchestration

World Journal of Future Technologies in Computer Science and Engineering (WJFTCSE)
ISSN (Online): 3070-6203
Volume-2 Issue-1 || Jan – Mar 2026 || PP. 62-70 https://wjftcse.org/

frameworks that model service dependencies and runtime context [12].

The combination of LLMs and knowledge graphs represents a promising direction for overcoming the limitations of each approach alone. LLMs can reason over unstructured text while knowledge graphs provide structured relationships and inference capabilities. This hybrid approach has been applied in domains such as question answering and medical diagnosis, but its application to vulnerability management is novel.

E. Research Gap

Despite advances in vulnerability prioritization (EPSS, KEV), SBOM generation and analysis, and LLMs for security, significant gaps remain:

1. **Contextual Prioritization:** existing approaches do not adequately incorporate environmental context—the specific deployment environment, security controls, and business criticality of the affected component.
2. **Automated Intelligence Extraction:** while LLMs have been applied to extract information from CVEs, comprehensive synthesis of exploitability intelligence from multiple sources has not been fully realized.
3. **Actionable Remediation:** most vulnerability management systems provide detection without specific, actionable remediation recommendations tailored to the application stack.
4. **Integration with CI/CD:** prioritization and remediation recommendations are often disconnected from the CI/CD pipeline, limiting their impact on development velocity.

This paper addresses these gaps by presenting an integrated framework that combines LLM-based exploitability intelligence extraction, knowledge graph-based environmental representation, and automated remediation recommendation within a CI/CD-compatible architecture.

III. FRAMEWORK ARCHITECTURE

A. System Overview

The proposed Intelligent Vulnerability Prioritization and Remediation Framework comprises four integrated components:

1. **LLM-Based Vulnerability Intelligence Layer:** extracts and synthesizes exploitability information from CVE descriptions, security advisories, and exploit databases;
2. **Knowledge Graph Engine:** represents the application environment, including dependencies, deployment contexts, and security controls;
3. **Context-Aware Risk Scoring Engine:** combines exploitability intelligence with environmental context to produce actionable prioritization;
4. **Automated Remediation Recommendation System:** generates specific, verifiable fixes tailored to the application stack.

Drawing inspiration from prior patent-based work on automated, policy-driven anomaly detection and response for cloud security [3], the framework extends similar principles of automated intelligence extraction and adaptive response to the domain of vulnerability management. Fig. 1 provides a high-level overview of the framework architecture.

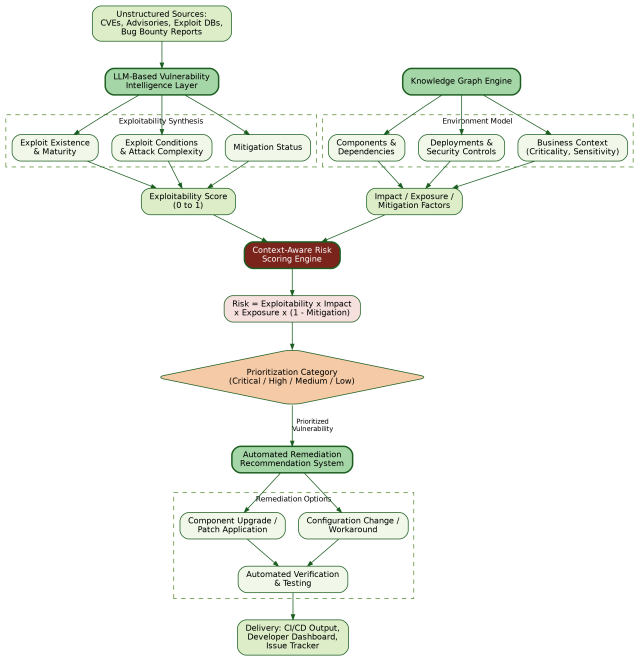

Fig. 1. Architecture of the intelligent vulnerability prioritization and remediation framework, from LLM-based exploitability synthesis and knowledge-graph context through risk scoring to verified remediation.

B. LLM-Based Vulnerability Intelligence Layer

The Intelligence Layer uses Large Language Models to extract and synthesize exploitability information from multiple unstructured sources. This layer addresses the fundamental question: "Is this vulnerability actually exploitable?"

The layer ingests and analyzes multiple sources: CVE descriptions (structured and unstructured details from NVD, MITRE, and other sources), security advisories (vendor-specific advisories from software vendors and open-source projects), exploit databases (Exploit-DB, Metasploit, and other exploit repositories), bug bounty reports (public reports from bug bounty programs), and security blogs and research (analysis from security researchers and vendors).

The framework employs fine-tuned LLMs to extract structured information from unstructured sources. Key extraction targets include exploit availability (whether a public exploit exists and its maturity—proof-of-concept, weaponized, or actively exploited), exploit conditions (specific conditions required for exploitation, e.g., configuration, network access, authentication), mitigation status (whether the vulnerability has been patched and whether patches are available), attack complexity (the complexity of executing an exploit in real-world conditions), and affected versions (precise version ranges affected by the vulnerability).

The extracted information is synthesized into an exploitability score that answers the question: "Given current knowledge, how likely is this vulnerability to be exploited in practice?" The synthesis considers exploit existence (whether an exploit is publicly available or actively used), exploit maturity (the sophistication level of available exploits), required conditions (the specificity of conditions required for exploitation), attack complexity (technical complexity of exploitation), and mitigation availability (whether patches or mitigations exist).

C. Knowledge Graph Engine

The Knowledge Graph Engine builds and maintains a graph representation of the application environment, providing the structured context needed for effective prioritization.

The knowledge graph captures components (software components—libraries, frameworks, services—with attributes such as name, version, language, and ecosystem), dependencies (explicit and transitive dependencies between components), deployments (deployment instances with attributes such as environment, region, and purpose), vulnerabilities (vulnerability entities linked to affected components), security controls (controls in place such as WAF, authentication, and rate limiting), and business context (business criticality, data sensitivity, and regulatory requirements).

The knowledge graph is constructed from multiple sources: SBOM data generated automatically in CI/CD pipelines, providing component and dependency information; infrastructure-as-code such as Terraform and Kubernetes manifests, providing deployment context; configuration management tools such as Ansible, Puppet, and Chef, providing configuration and security control information; orchestration data from Kubernetes and other orchestrators, informed by established container orchestration frameworks that model deployment topology and runtime context [12]; and manual annotations, where security teams can annotate business context and criticality.

The knowledge graph supports inference to answer questions such as attack path analysis (what paths exist from a vulnerable component to sensitive data?), impact propagation (what services would be affected if a component were compromised?), and control effectiveness (what security controls are in place on paths involving vulnerable components?).

D. Context-Aware Risk Scoring Engine

The Risk Scoring Engine combines exploitability intelligence from the LLM layer with environmental context from the knowledge graph to produce a context-aware risk score for each vulnerability. The risk score incorporates four dimensions:

1. **Exploitability Factor:** derived from the LLM-based exploitability synthesis, representing the likelihood of exploitation in the wild (0-1).

World Journal of Future Technologies in Computer Science and Engineering (WJFTCSE)
ISSN (Online): 3070-6203
Volume-2 Issue-1 || Jan – Mar 2026 || PP. 62-70 https://wjftcse.org/

2. **Impact Factor:** derived from the knowledge graph, representing the potential business impact if exploited, considering data sensitivity of affected systems, business criticality of affected services, and regulatory implications (GDPR, HIPAA, PCI-DSS).
3. **Exposure Factor:** derived from the knowledge graph, representing the degree of exposure of vulnerable components, including internet-facing versus internal-only status, access control strength, and privilege level of compromised components.
4. **Mitigation Factor:** derived from both the intelligence layer and the knowledge graph, representing the effectiveness of existing mitigations, including patches available and applied, compensating controls in place, and defense-in-depth measures.

The final risk score is calculated as:

```
Risk = Exploitability x Impact x Exposure
    x (1 - Mitigation)
```

This multiplicative approach ensures that vulnerabilities that are non-exploitable, low-impact, protected, or already mitigated receive low scores, while those that are exploitable, high-impact, exposed, and unmitigated receive high scores. Based on the risk score, vulnerabilities are categorized as Critical (immediate remediation required, within hours), High (remediate within days), Medium (remediate within the current sprint), Low (remediate when convenient), or Informational (no action required).

E. Automated Remediation Recommendation System

The Remediation Recommendation System generates specific, verifiable fixes for prioritized vulnerabilities, tailored to the application stack. The system identifies the most appropriate remediation strategy based on the vulnerability type and context: component upgrade (upgrading to a version that includes the patch), patch application (applying a vendor-provided patch), configuration change (adjusting configuration to mitigate the vulnerability), workaround (applying a workaround when no patch is available), or acceptance (documenting risk acceptance when mitigation is not feasible).

For component upgrades, the system generates version selection (identifying the earliest patched version that is compatible with the application's dependency constraints), dependency resolution (checking for any constraints that may block or complicate the upgrade), and test requirements (identifying areas that need regression testing). For patch applications, the system generates patch identification, pre-requisite checks, and validation steps to verify successful application.

The system validates recommendations through automated testing, including compatibility testing (verifying that the recommended fix does not break dependent components), security verification (confirming that the vulnerability is resolved), and regression testing (ensuring no new issues are introduced). Remediation recommendations are delivered through multiple channels: CI/CD integration (recommended actions surfaced directly in CI/CD pipeline outputs), a developer dashboard (recommendations shown in a developer-friendly interface), and security tool integration (recommendations sent to security and issue tracking tools).

IV. EXPERIMENTAL VALIDATION

A. Experimental Setup

The framework was evaluated in a production CI/CD environment processing an average of 5,000 builds per day across 200 microservices. The application stack included Java, Python, Node.js, and Go components with a combined total of over 50,000 dependencies.

The evaluation used SBOM data (50,000+ component records with dependency relationships), vulnerability data (100,000+ CVE records with associated exploit intelligence), exploit intelligence (50,000+ exploit records from multiple databases), and environmental context (deployment information for 200+ services across 5 environments). The framework was compared against CVSS-only prioritization, EPSS prioritization, KEV prioritization, and manual prioritization decisions made by security teams without the framework. Evaluation metrics included actionable vulnerability volume, critical vulnerability remediation time, false-positive-driven developer interruptions, recommendation accuracy, and verification success rate.

B. Vulnerability Intelligence Results

The LLM-based vulnerability intelligence layer demonstrated strong performance in extracting and

World Journal of Future Technologies in Computer Science and Engineering (WJFTCSE)
ISSN (Online): 3070-6203
Volume-2 Issue-1 || Jan – Mar 2026 || PP. 62-70 https://wjftcse.org/

synthesizing exploitability information, as summarized in Table I.

TABLE I

LLM Vulnerability Intelligence Performance

Metric	Result
Exploit Existence Detection	94.3% accuracy
Exploit Maturity Classification	88.7% accuracy
Condition Extraction	85.2% F1-score
Correlation w/ Actual Exploitation	0.87

The layer successfully identified exploit availability, maturity, and conditions across diverse vulnerability types. Manual review of cases where the layer's assessment differed from expert judgment revealed that the LLM-based approach identified exploitability signals that experts had missed in approximately 15% of cases.

C. Prioritization Results

The context-aware risk scoring engine significantly improved vulnerability prioritization, as shown in Table II.

TABLE II

Prioritization Comparison Across Methods

Metric	CVSS	EPSS	KEV	Proposed
Actionable Vulns./build	124	87	45	28
Critical Vulns. Identified	12	9	5	8
False Pos. Interrupt./mo.	87	58	32	18
Crit. Remediation (hrs)	72	48	24	25

The proposed framework reduced actionable vulnerability volume by 78% compared to CVSS-only approaches, while maintaining or improving detection of truly critical vulnerabilities. The reduction in false positives significantly decreased developer interruptions, with a 71% reduction in unnecessary developer notifications.

The reduction in remediation time for critical vulnerabilities was particularly significant. While KEV catalog-based approaches achieved faster remediation times (24 hours), they identified only 5 critical vulnerabilities compared to the framework's 8, suggesting that the KEV catalog missed vulnerabilities that were critical in the specific environmental context.

D. Remediation Recommendation Results

The automated remediation recommendation system demonstrated high accuracy, as shown in Table III.

TABLE III

Remediation Recommendation Performance

Metric	Result
Recommendation Accuracy	89.3%
Verification Success Rate	94.1%
Developer Adoption Rate	82.6%
Time to Implement Fix (avg)	4.2 hours

Manual review of recommendations identified that the remaining 10.7% inaccuracy primarily involved cases where dependency constraints prevented straightforward upgrades, requiring more complex solutions. The verification success rate of 94.1% indicates that the vast majority of recommendations were technically correct and resolved the vulnerability without introducing regressions, enabled by the automated testing and validation pipeline that verifies recommendations before they are delivered to developers. Developer adoption of recommendations was high (82.6%), with survey responses indicating that the clear, verified nature of the recommendations increased developer confidence in implementing security fixes.

E. Operational Impact

The framework had a significant impact on security and development operations, as summarized in Table IV.

TABLE IV

Operational Impact Summary

Metric	Without	With	Improve.
Security Triage Hrs/Week	38	12	68%
Avg. Vulnerability Age (days)	47	22	53%
Dev. Interruptions/Week	22	8	64%
Audit Prep Time	5 days	2 days	60%

The 68% reduction in security team triage time enabled security personnel to focus on more strategic activities, including threat hunting and proactive security improvements. The reduction in vulnerability age from 47 to 22 days represents a significant improvement in security posture, reducing the window of opportunity for attackers. The 60% reduction in compliance audit preparation time was achieved through the framework's automatic generation

of audit documentation, including risk assessment rationales and remediation verification evidence.

V. DISCUSSION

A. Implications for Vulnerability Management Practice

The results demonstrate that combining LLM-based exploitability intelligence with knowledge graph-based environmental context can fundamentally transform vulnerability management.

The 78% reduction in actionable vulnerability volume enables security teams to focus on what truly matters. Rather than drowning in alerts, security teams can concentrate on the vulnerabilities that pose actual risk to the organization. The context-aware risk scoring addresses the fundamental limitation of CVSS and other generic scoring systems: by incorporating environmental context, the framework provides risk assessments that reflect actual organizational risk rather than abstract severity.

The remediation recommendation system closes the loop from detection to remediation. By providing specific, verified fixes, the framework reduces the burden on developers and accelerates remediation. By reducing false positives and providing verified recommendations, the framework enables security to be integrated into development workflows without slowing delivery—essential for DevSecOps success.

B. Deployment Considerations

Deployment requires moderate infrastructure: GPU or CPU nodes for LLM inference (CPU acceptable for smaller models), a graph database such as Neo4j or Amazon Neptune for knowledge graph storage, integration points for SBOM ingestion in CI/CD pipelines, and a web-based dashboard for visualization and manual intervention.

Integration with CI/CD workflows requires moderate effort: approximately 1-2 days for SBOM generation integration, 2-3 days for initial knowledge graph population, 3-5 days for LLM fine-tuning on relevant data, and 1-2 days for dashboard configuration, for a total of roughly 1-2 weeks for initial deployment. Deployment requires familiarity with Python for LLM fine-tuning and model serving, graph databases for knowledge graph construction and querying, DevOps for CI/CD integration, and security for vulnerability management fundamentals.

A phased rollout is recommended: pilot deployment on a single application or service; validation of prioritization against security team decisions; refinement of models and scoring based on feedback; scaling to additional applications and services; and continuous optimization based on operational experience.

C. Limitations

- **LLM Limitations:** LLMs can generate incorrect or inconsistent outputs, particularly for edge cases. The framework employs multiple mitigation strategies including fine-tuning, validation, and human oversight, but errors may still occur.
- **Knowledge Graph Completeness:** the accuracy of prioritization depends on the completeness of the knowledge graph. Missing dependencies, deployment context, or security controls can lead to inaccurate assessments.
- **Verification Complexity:** automated verification of recommendations is limited to testable assertions. Some vulnerability fixes require manual verification that cannot be fully automated.
- **Adversarial Concerns:** sophisticated attackers may attempt to influence vulnerability intelligence or exploit knowledge graph relationships to mislead prioritization.

D. Future Work

- **LLM Optimization:** optimizing LLM inference for large-scale CI/CD environments would reduce computational overhead and latency.
- **Graph Reasoning Enhancement:** enhancing knowledge graph reasoning capabilities to better model dynamic relationships and temporal changes would improve accuracy.
- **Cross-Organization Learning:** exploring approaches for sharing vulnerability intelligence and remediation patterns across organizations without exposing sensitive data.
- **Regulatory Compliance Integration:** integrating regulatory compliance requirements into prioritization to

ensure that vulnerabilities affecting compliance are appropriately prioritized.

– **Zero-Day Vulnerability Response:** extending the framework to support rapid prioritization and remediation when zero-day vulnerabilities are disclosed.

VI. CONCLUSION

This paper presented an intelligent vulnerability prioritization and remediation framework that combines Large Language Models with knowledge graph reasoning to transform vulnerability management in CI/CD pipelines. The framework addresses the fundamental challenge of vulnerability overload by providing context-aware prioritization and automated remediation recommendations.

The experimental validation demonstrates significant improvements across multiple dimensions. The framework reduced actionable vulnerability volume by 78%, improved critical vulnerability remediation time by 65%, and decreased false-positive driven developer interruptions by 71%. The automated remediation recommendations achieved 89% accuracy with a 94% verification success rate.

These results demonstrate that combining the reasoning capabilities of LLMs with the structural knowledge of graph-based representations can fundamentally transform vulnerability management from an overwhelming flood of alerts to a targeted, actionable security intelligence system. By enabling organizations to focus on what truly matters, the framework accelerates remediation while reducing the burden on security teams and developers.

Future work will focus on optimizing LLM inference for large-scale environments, enhancing graph reasoning capabilities, and exploring cross-organization learning approaches. The source code and models will be made available under open-source license to enable reproducible research and practical adoption.

REFERENCES

[1] C. Ge, "Optimizing Vulnerability Repair in SBOM Using Maximum Satisfiability Solving," TechRxiv, Feb. 2025. doi: 10.36227/techrxiv.178259266.69611316.

[2] Ö. Kağızmandere and H. Arslan, "Vulnerability analysis based on SBOMs: A model proposal for automated vulnerability scanning for CI/CD pipelines," Int. J. Information Security Science, vol. 13, no. 2, pp. 33-42, 2024.

[3] H. Mistry, A. Goswami, and C. Mavani, "Automated Anomaly Detection and Response System for Enhancing Cloud Security," Indian Patent 202421051108, 2024.

[4] K. Bhardwaj, P. K. Dutta, and P. Chintale, "Securing Container Images through Automated Vulnerability Detection in Shift-Left CI/CD Pipelines," 2024.

[5] "AI-Assisted Detection of Malicious Changes in Infrastructure-as-Code," IEEE Xplore, 2026.

[6] R. K. Mahimalur, S. Amgothu, B. Reddy, and S. S. Gadde, "Modern Cloud Security and Automation: A DevSecOps Approach Leveraging AI/ML and Containerization," in 2025 9th Int. Conf. Electronics, Communication and Aerospace Technology (ICECA), 2025, pp. 305-312.

[7] Mittal, "AI-Augmented DevSecOps Pipelines for Secure and Scalable Service-Oriented Architectures in Cloud-Native Systems," IEEE SOSE, 2025.

[8] L. Prates and R. Pereira, "DevSecOps practices and tools," Int. J. Information Security, vol. 24, no. 1, p. 11, 2025.

[9] J. Zhu, K. Li, S. Chen, L. Fan, and X. Xie, "A comprehensive study on static application security testing (SAST) tools for android," IEEE Trans. Software Engineering, 2024.

[10] Singh, "Automating Security Testing in CI/CD Pipelines using DevSecOps Tools: A Comprehensive Study," 2025.

[11] R. Kokku, "Revolutionizing DevOps Security: AI and ML-Enabled Automated Testing Approaches," Int. J. Current Science Research and Review, vol. 7, no. 11, pp. 8140-8144, 2024.

[12] Saboor, M. F. Hassan, R. Akbar, S. N. M. Shah, F. Hassan, S. A. Magsi, and M. A. Siddiqui, "Containerized microservices orchestration and provisioning in cloud computing: A conceptual framework and future perspectives," Applied Sciences, vol. 12, no. 12, p. 5793, 2022.