# AI-Augmented DevSecOps Pipeline for Intelligent Vulnerability Triage: Integrating SonarQube with LLM

Steven Wu[1][0009-0001-5847-1417] and Tan-Hsu Tan[2][0000-0003-2956-6115]

and Shing-Hong Liu [3][0000-0002-3923-4387] and Chao-Rong Chen [4][0000-0003-1426-9513]

[1,2] Innovation Frontier Institute of
Research for Science and Technology
National Taipei University of Technology, Taipei, Taiwan
[3] Department of Computer Science and Information Engineering
Chaoyang University of Technology, Taichung, Taiwan
[4] Department of Electrical Engineering
National Taipei University of Technology, Taipei, Taiwan

[1]t114C74003@ntut.edu.tw

**Abstract.** This paper presents an AI-augmented DevSecOps pipeline that integrates SonarQube static analysis with GPT-4.1 to improve vulnerability triage for .NET and Angular applications. The system uses GitHub Actions to orchestrate CI/CD workflows, runs a self-hosted SonarQube instance on Azure, stores scan artifacts in Azure Blob Storage, and triggers Azure Functions to generate prioritized, context-aware vulnerability summaries. Beyond validation on Microsoft's eShopOnWeb, a real-world case study on a Taiwanese enterprise repository demonstrates increased developer responsiveness, reduced cognitive load compared to raw SonarQube reports, and the feasibility of a reproducible, privacy-preserving, end-to-end security workflow, demonstrating the practical value of systematically embedding generative AI into secure software delivery pipelines.

**Keywords:** Vulnerability Triage, Artificial Intelligence, LLM, SonarQube, DevSecOps, Azure.

## 1 Introduction

The rapid advancement of web technologies—particularly in artificial intelligence (AI), cloud computing, and microservices—has substantially expanded the attack surface of modern software systems. According to Verizon's 2025 Data Breach Investigations Report (DBIR), external actors continue to be the dominant threat group in cloud breaches, with stolen credentials remaining the most common method of unauthorized access. MisconFigurations and inherent vulnerabilities in cloud applications further contribute to these incidents, revealing ongoing weaknesses in cloud security postures [1]. At the same time, the widespread adoption of DevOps methodologies and CI/CD pipelines has

accelerated software delivery, often introducing security gaps when protective mechanisms are applied too late in the development lifecycle.

To address this challenge, this paper proposes an AI-augmented DevSecOps framework that integrates security controls from the earliest stages of software development, thereby embodying shift-left security principles. In the proposed pipeline, commit events trigger automated static analysis through open-source tools, with the resulting findings enriched by the contextual reasoning capabilities of large language models (LLMs). As an open-source and community-driven platform, SonarQube is freely accessible, extensively peer-reviewed, and continuously improved by a global ecosystem of developers. Its transparency and widespread adoption provide strong assurance of reliability and consistent performance across diverse codebases, making it an ideal foundation for automated static analysis in modern DevSecOps workflows. By combining the systematic detection provided by static analysis with the semantic understanding and prioritization enabled by LLMs, the framework aims to deliver scalable, cost-effective, and high-quality vulnerability triage throughout the continuous integration and continuous deployment (CI/CD) process.

Recent advancements in LLMs have demonstrated significant promise in supporting software engineering tasks, particularly in vulnerability detection, classification, and automated repair. Although LLMs currently lag behind specialized static analysis engines such as CodeQL in precision, they excel at identifying data-flow characteristics—including sources, sinks, and sanitization logic—providing richer contextual insights into code behavior [2]. Prior research suggests that coupling LLM-based reasoning with traditional static analysis techniques can substantially enhance software security within modern development pipelines, improving both accuracy and developer comprehension [3].

Building upon these findings, this research focuses on the design, implementation, and validation of a secure software development workflow that integrates SonarQube with LLM capabilities in a cloud-native architecture. The experimental system is developed around a .NET Core API backend and later extended to include an Angular frontend, all orchestrated through GitHub Actions for CI/CD automation and deployed on Microsoft Azure cloud infrastructure. The addition of the Angular frontend enables consistent static analysis and vulnerability triage across full-stack components.

To ensure practical relevance and applicability, the proposed workflow is applied to real-world development settings in collaboration with a leading technology company in Taiwan, providing empirical insights into the effectiveness of LLM-augmented static application security testing (SAST) within contemporary enterprise environments.

The remainder of this paper is organized as follows. Section 2 reviews related work on static application security testing (SAST), SonarQube-based analysis, and recent advances in large language models for software security. Section 3 presents the design and implementation of the proposed AI-augmented DevSecOps architecture, detailing

system components, data flow, and cloud deployment. Section 4 reports experimental results and real-world evaluation outcomes, including comparative analysis between SonarQube and GPT-4.1–assisted vulnerability triage. Finally, Section 5 concludes the paper and discusses limitations as well as directions for future research.

## 2 Literature Review

Static Application Security Testing (SAST) is a foundational technique in the field of software security assurance. By examining source code, bytecode, or binaries without executing the application, static analysis enables early identification of software vulnerabilities, often before the code is integrated or deployed [4]. SAST tools utilize a combination of pattern matching, data flow analysis, control flow analysis, and abstract syntax tree (AST) parsing to detect vulnerabilities such as buffer overflows, SQL injections, and cross-site scripting (XSS) [5]. Despite its strengths, SAST remains limited in its ability to capture complex application logic and contextual dependencies that emerge only during runtime [6]. Because SAST analyses code statically, it can miss runtime-only flaws such as weak session handling, misconFigured APIs, or broken authentication/authorization flows—issues uncovered by SAST during execution [7]. By integrating artificial intelligence techniques, such as machine learning and natural language processing, into static analysis workflows may address some of these limitations by enabling more context-aware and intelligent code analysis [8].

SonarQube, as a SAST platform, is a modular, extensible platform widely used in web-based CI/CD pipelines for ensuring code quality and identifying security vulnerabilities. SonarQube offers interactive dashboards for browsing issues, security hotspots, code smells, and historical trends via a browser interface [9]. The SonarQube compute engine executes rule sets and performs security analysis, including taint analysis and advanced Deep SAST capabilities. Based on plugin-based architecture, SonarQube integrates a wide range of analytical capabilities. For instance, its language analysis plugins support major programming languages such as Java, C#, Python, JavaScript, Kotlin, and Go, thereby enabling consistent static analysis across heterogeneous codebases. In addition, Software Composition Analysis (SCA) plugins—such as OWASP dependency-check and tidelift—facilitate the tracking of known vulnerabilities (CVEs), license compliance verification, and the generation of software bill of materials (SBOM) [10]. SonarQube's extensibility is further demonstrated by its infrastructure-as-code (IaC) and policy-enforcement plugins, which enable scanning and auditing of terraform, ansible, and kubernetes conFigurations for security violations and compliance with organizational standards [11].

Recent comparative studies, such as Large Language Models Versus Static Code Analysis Tools, have systematically evaluated traditional SAST tools (SonarQube, CodeQL, SnykCode) against state-of-the-art large language models (GPT-4.1, Mistral Large, DeepSeek V3) for real-world vulnerability detection. Results consistently show that while LLMs achieve substantially higher recall and F1 scores—detecting subtle,

context-dependent vulnerabilities that rule-based tools often miss—they also introduce high false-positive rates, hallucinated findings, and unreliable location reporting. Conversely, conventional scanners provide precise, deterministic, and audit-ready analysis but suffer from limited coverage due to their dependence on predefined rules. These complementary strengths and weaknesses suggest that neither approach is sufficient alone; instead, the literature points toward a hybrid model in which robust static analysis (e.g., SonarQube) is augmented by LLM-driven reasoning to enhance triage, context interpretation, and remediation support [12].

# 3 System Architecture

This research proposes an AI-augmented DevSecOps CI/CD pipeline that integrates SonarQube with GPT-4.1 to enhance vulnerability triage and provide automated, context-aware remediation suggestions. The system leverages standard DevOps tools and practices, enriched with Generative AI capabilities, to bridge the gap in existing SAST workflows.

In addition to the .NET Core backend, the pipeline also integrates the Angular frontend within the same DevSecOps framework. Both stacks adhere to a unified build–test–scan sequence, ensuring that frontend and backend components are evaluated under consistent quality gates and vulnerability triage procedures. This design maintains uniform security assurance and code-quality standards across the entire full-stack application.

## 3.1 Overview of the Proposed Architecture

The proposed system architecture unifies traditional DevSecOps components with LLM-based security intelligence to enhance automated vulnerability triage. It con-sists of five tightly integrated elements, designed to operate seamlessly within a cloud-native environment. While the current implementation is realized on GitHub Actions, future work will extend the orchestration layer to Azure DevOps Services to evaluate cross-platform portability and scalability.

First, the source code repository (GitHub) serves as the authoritative version-control platform. Developers contribute changes through standard commit-and-push workflows, ensuring a single source of truth and enabling collaborative development. Each push event automatically triggers the CI/CD pipeline, ensuring that all revisions undergo compilation, testing, and security verification before integration or deployment.

Second, CI/CD orchestration (GitHub Actions) manages the end-to-end workflow. Carefully designed YAML conFigurations ensure that compilation, testing, static analysis, and artifact handling execute in a deterministic and secure sequence. GitHub Actions also manages secrets and sensitive credentials through encrypted storage, ensuring the correctness and security for reliable pipeline automation.

Third, static analysis (SonarQube) is conducted on a self-hosted SonarQube instance deployed on an Azure Virtual Machine. SonarQube performs Static Application Security Testing (SAST) for both the .NET Core backend and the Angular frontend, which have been fully tested under this unified pipeline. It detects vulnerabilities, code smells, and maintainability issues, exporting the results as structured JSON artifacts for downstream processing. The findings remain accessible through the SonarQube dashboard for manual review and traceability.

Fourth, AI-assisted code review (GPT-4.1 integration) augments static analysis by applying LLM-driven reasoning to SonarQube outputs. Scan artifacts are uploaded to Azure Blob Storage, which triggers an Azure Function hosting the GPT-4.1 model via the Azure OpenAI Service. To address token-length constraints, SonarQube results are pre-processed before being fed into the model. All analysis is executed entirely within the user's Azure environment to preserve data privacy and prevent external exposure. GPT-4.1 provides context-aware insights, prioritizes critical issues, and generates actionable remediation guidance, effectively transforming raw SAST output into high-value developer intelligence.

Finally, human-audited deployment (Azure) forms the security gate before production release. After GPT-4.1 generates summarized recommendations—stored back in Azure Blob Storage—a designated reviewer validates the content for accuracy, contextual relevance, and risk alignment. Only after explicit human approval does the pipeline proceed to deploy the application to Azure infrastructure. This human-in-the-loop safeguard ensures accountability and mitigates the risk of acting on automated vulnerability assessments without expert oversight.

### 3.2 System Components and Data Flow

The data flow of the proposed system follows a linear pipeline triggered by code changes:

Source code → Build & Scan (CI) → SonarQube report → upload Azure Blob → Blob trigger Azure Function → GPT-4.1 summary uploaded → human decision (CD) → Deployment.

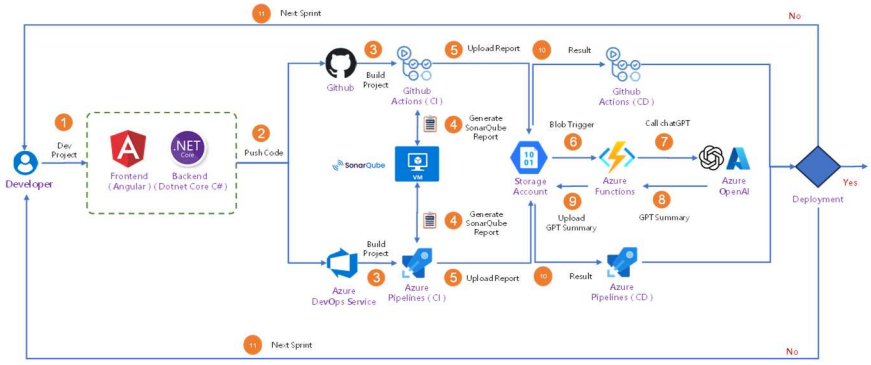


**Fig.1.** Data Flow of the proposed AI-enhanced DevSecOps pipeline for both Angular frontend and .NET Core backend, demonstrating equivalent workflows implemented on GitHub Actions and Azure DevOps Pipelines.

To provide a holistic view of the pipeline, Figure 1 depicts the end-to-end workflow of the integrated system, including static analysis, AI-based summarization, and human review checkpoints within the automated CI/CD process. This modular and extensible design allows for further enhancements, such as integrating Software Composition Analysis (SCA) tools or dynamic analysis tools in future iterations.

## 4 Experimental Results

This chapter presents the evaluation result of the proposed AI-augmented DevSecOps pipe-line integrating SonarQube with GPT-4.1. The objective is to determine whether LLM-based summarization improves triage efficiency, interpretability, and remediation guidance compared with SonarQube alone. In addition, a real-world deployment at a leading Taiwanese technology company was conducted to validate the pipeline's performance in an enterprise environment.

### 4.1 Comparative Analysis of SonarQube and GPT-4.1 Summaries

We provided GPT-4.1 with the following structured prompt:

```
Identify the top 10 most severe issues and explain why they
are critical.
Group issues by file or module and summarize their overall
security and code-quality state.
Recommend specific refactoring or remediation Actions for
the top issues, following secure coding best practices.
Present the output in well-structured Markdown, including:
• prioritized issue summary Table,
• per-module security assessment, and
• actionable recommendations.
```

After executing the pipeline on Microsoft's eShopOnWeb repository, two reports were produced: the full SonarQube report—which is comprehensive but verbose—and the GPT-4.1–generated summary, which is concise and prioritized. The GPT-4.1 summary effectively distilled the scan output while preserving all critical security insights.

**Table 1.** Severity distribution in SonarQube vs. GPT-4.1 summary report

| Severity Level | Full SonarQube Report | GPT4.1 Summary |
|---|---|---|
| CRITICAL | 12 | 4 |
| MAJOR | 23 | 6 |
| MINOR | 35 | 0 |
| INFO | 8 | 0 |
| Total | 78 | 10 |

The Severity Coverage Rate (SCR) was calculated as (10 ÷ 35) × 100% = 28.6%, indicating that the summary preserved roughly one-quarter of all findings while filtering out low-value duplicates and trivial issues. For example, minor items such as "Remove unused local variable" were omitted and replaced with more context-aware insights, such as "Empty method in authentication flow—add a comment or throw NotSupportedException." GPT-4.1 transformed the exhaustive static analysis output into a concise, prioritized, and actionable summary, as reflected in Table 2.

**Table2.** Summary of Gpt-4.1 performance metrics

| Metric | Value |
|---|---|
| Reduction Rate | 87.2% |
| Coverage of High-Severity Issues | 28.6% |
| Minor/Info Issue Filtering | 100% |
| Actionability of Output | High (contextual) |

### 4.2 Real-World Deployment Study

A nine-day evaluation on an internal corporate repository was conducted to measure how GPT-4.1-assisted triage influenced developer focus and remediation speed. Table 3 presents the results across four sprints, including the number of days per sprint, critical and major issues closed, newly identified issues, and the distribution of issue severities—both resolved and newly generated—after each sprint.

**Table 3.** Results from four sprints, including sprint length, critical/major issues closed, new issues detected, and severity distribution during the nine-day evaluation.

| Interval | Days | closed (crit) | closed (major) | New (crit) | New (major) | high-severity closed | high-severity new | high-sev net delta |
|---|---|---|---|---|---|---|---|---|
| Jul30–Aug5 | 6 | 9 | 25 | 2 | 13 | 34 | 15 | **19** |
| Aug5–Aug6 | 1 | 4 | 20 | 0 | 2 | 24 | 2 | **22** |
| Aug6–Aug7 | 1 | 2 | 14 | 0 | 2 | 16 | 2 | **14** |
| Aug7–Aug8 | 1 | 5 | 14 | 1 | 6 | 19 | 7 | **12** |

Across all intervals, the High-Severity Net Delta remained positive, indicating that remediation consistently outpaced the introduction of new issues. Developers also demonstrated rapid response times, suggesting that GPT-4.1–driven prioritization helped maintain stronger focus on critical vulnerabilities and accelerated secure-code delivery.

### 4.3 Overall Comparison

**Table 4.** Comparison between SonarQube and SonarQube + GPT-4.1

| **Aspect** | **SonarQube Only** | **SonarQube + Gpt-4.1** |
|---|---|---|
| Coverage | High (full stack) | Inherits from SonarQube |
| Readability | Low (flat, exhaustive) | High (ranked, structured, grouped) |
| Actionability | Moderate (Standard messages) | High (with rationale and code suggestions) |
| Contextual Insight | Minimal | Moderate to High (Grouped by module) |
| Developer Productivity | Moderate | Significantly improved (~70% time reduction) |
| Risk Communi-cation | Weak | Strong (summaries for support team planning) |

The evaluation confirms that SonarQube provides thorough coverage but often causes cognitive overload. GPT-4.1 post-processing significantly improves triage clarity, prioritization, and remediation guidance. In real-world use, consistent positive Net Deltas demonstrate the effectiveness of AI-assisted triage in supporting faster, more informed, and security-driven software development—while preserving the necessity of expert human review for high-risk cases.

## 5 Conclusion

This research presents a seamless and fully integrated workflow that spans the entire DevSecOps lifecycle—from code commit to automated build, static analysis, AI-driven summarization, and human-in-the-loop review prior to deployment on Microsoft Azure. The results demonstrate that large language models (LLMs) can substantially enhance the usability and effectiveness of Static Application Security Testing (SAST)

outputs. While SonarQube alone provides comprehensive coverage, it often overwhelms developers with excessive information and limited actionable context. By incorporating GPT-4.1 as a post-processing layer, the pipeline effectively filters noise, prioritizes high-impact vulnerabilities, and delivers context-aware remediation guidance. This integration bridges the gap between raw static analysis and informed, security-driven development decisions, while maintaining the necessity of expert human oversight for risk-critical issues.

The proposed pipeline has been validated on both the .NET Core backend and the Angular frontend, confirming its applicability across full-stack applications. Despite its effectiveness, several limitations remain. SonarQube's ruleset provides limited coverage for certain frameworks—such as Blazor WebAssembly—leading to partial exclusion of specific frontend components. This limitation could be mitigated by integrating complementary analyzers or adopting hybrid SAST–DAST approaches. Moreover, GPT-4.1's performance is inherently constrained by SonarQube's detection scope, as undetected vulnerabilities remain beyond its reach. Future enhancements may involve multi-tool fusion, combining results from CodeQL, Snyk, and other analyzers to broaden AI-assisted vulnerability detection. Additionally, Azure operational costs associated with large or frequent scans can be addressed through token optimization, model tiering, and adaptive scan scheduling to balance cost efficiency and analytical quality.

Future research should extend in three strategic directions. First, develop predictive cost-optimization models that correlate LLM type, report size, and scanning frequency with Azure OpenAI expenditures, enabling organizations to identify cost-effective conFigurations before deployment. Second, perform systematic benchmarking across multiple LLMs to evaluate cost efficiency, accuracy, coverage, latency, and developer experience, thereby providing empirical guidance for model selection. Third, and most importantly, the orchestration layer should be expanded to Azure DevOps Services to support multi-platform CI/CD environments—as illustrated in Fig. 1—and to integrate SonarQube with additional security tools such as CodeQL, Snyk, and dynamic analyzers. Such extensions would enable a multi-source AI triage framework, enrich security context, and further reduce dependence on any single detection engine.

## References

1. Verizon: Data Breach Investigations Report 2025. Verizon, New York (2025). Available at: https://www.verizon.com/business/resources/Tea/reports/2025-dbir-data-breach-investigations-report.pdf
2. Khare, A., Dutta, S., Li, Z., Solko-Breslin, A., Alur, R., Naik, M.: Understanding the Effectiveness of Large Language Models in Detecting Security Vulnerabilities. In: Proceedings of the 2023 ACM Conference on Neural Information Processing Systems (NeurIPS 2023) Workshop on Large Language Models for Code Intelligence, pp. 1–12. ACM, New York (2023).

3. Pearce, H., Tan, B., Ahmad, B., Karri, R., Dolan-Gavitt, B.: Examining Zero-Shot Vulnerability Repair with Large Language Models. In: Proceedings of the 2024 IEEE Symposium on Security and Privacy Workshops (SPW), pp. 1–10. IEEE, Los Alamitos (2024).
4. Chess, B., West, J.: Secure Programming with Static Analysis. Addison-Wesley Professional, Boston (2007).
5. Ayewah, N., Hovemeyer, D., Morgenthaler, J.D., Penix, J., Pugh, W.: Using Static Analysis to Find Bugs. IEEE Software, 25(5), 22–29. IEEE, Los Alamitos (2008).
6. Egele, M., Scholte, T., Kirda, E., Kruegel, C.: A Survey on Automated Dynamic Malware-Analysis Techniques and Tools. ACM Computing Surveys, 44(2), 1–42. ACM, New York (2012).
7. Teitler-Santullo, K.: Dynamic Application Security Testing (DAST): A Practical Guide for DevSecOps Teams. OX Security, Tel Aviv (2025).
8. Li, Z., Dutta, S., Naik, M.: IRIS: LLM-Assisted Static Analysis for Detecting Security Vulnerabilities. In: Proceedings of the 13th International Conference on Learning Representations (ICLR 2025), pp. 1–15. OpenReview, (2025).
9. SonarSource. What's New in SonarQube 10.5 LTA (2025.1). SonarSource, Geneva (2025).
10. SonarSource. Advanced Security in SonarQube: SCA, Deep SAST, and SBOM. So-narSource, Geneva (2025).
11. SonarSource. Plugin Library for SonarQube (June 2025). SonarSource, Geneva (2025).
12. Gnieciak, D., Szandała, T.: Large Language Models Versus Static Code Analysis Tools: A Systematic Benchmark for Vulnerability Detection. IEEE Access (2025).