iJRASET
International Journal For Research in
Applied Science and Engineering Technology

INTERNATIONAL JOURNAL
FOR RESEARCH
IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 13 Issue: V Month of publication: May 2025

DOI: https://doi.org/10.22214/ijraset.2025.71261

Security Report Generation via LLM-RAG Assisted Directory Scanning: An Integrated Framework for Enhanced Software Documentation and Vulnerability Detection

Utkarsh Rajendra Pingale[1], Irfan Ajmer Pasha Shaikh[2]
Artificial Intelligence & Data Science AISSMS Institute of Information Pune, India

Abstract: Thorough and recent documentation is essential for software maintenance, security audits, and knowledge transfer. Yet most software projects lack complete or recent documentation. Current tools mostly produce low-level code summaries without incorporating external knowledge, resulting in inefficiencies and enhanced security risk. This paper introduces an LLM-RAG-augmented automated documentation system utilizing Large Language Models (LLMs), Retrieval-Augmented Generation (RAG), and Information Retrieval (IR) methods. The system reads project directories, extracts metadata, and creates preliminary documentation using LLMs. A RAG module enhances this documentation by pulling external information pertinent to the task, like security advisories and bug reports, to provide a complete and actionable documentation framework. Evaluation is based on qualitative user studies and quantitative measurements. This work seeks to enhance documentation quality, increase software maintainability, and streamline security auditing through an AI-powered, explainable, and transparent report generation framework.

Keywords: Automated Documentation, Security Reports, LLM-RAG, Software Maintenance, Vulnerability Detection, Explainable AI, Software Security

I. INTRODUCTION

A. Background of study

Latest software development is supported by precise and com- prehensive documentation to warrant efficient processes, strong security auditing, as well as uncomplicated knowledge transferral. There are, nevertheless, numerous projects that are encountered with constraints owing to uncompleted or dep- recated documentation, leaving developers unsure of software structure and how security vulnerabilities can be solved. Suc- cessive updates as well as repetitions only tend to worsen this situation further, making static docu- mentation inadequate. [1] Conventional tools emphasize the summarization of code but never integrate with resources outside in the form of security databases, bug trackers and compliance models. This gap cre- ates incomplete docu- mentation that does not correct funda- mental vulnera- bilities or account for recent patches, causing ineffi- ciencies and heightened security risks. Integrated, real- time documentation also makes knowledge transfer difficult within teams. [2] New team members find it difficult to learn complicated systems without complete resources, re- forcing onboarding and the likelihood of introducing errors or vulnerabilities. To correct these issues, emergent solutions such as LLM-RAG-assisted directory scanning are required. These technologies can automate documentation procedures, merge outside re-sources, and make real-time updates, filling code summarization, security auditing, and knowledge-sharing gaps. This method improves efficiency, enhances security, and promotes collaboration in software development projects. [1]

B. Problem Statement

Existing documentation automation solutions lack in some key areas, causing serious challenges in maintaining secure and efficient software development processes. Precisely, these solutions lack:

1) In-Depth Insights into Security Vulnerabilities: Current tools tend to overlook the in-depth analysis and docu- mentation of vulnerabilities, resulting in insufficient risk assessments. This hinders developers from spotting and fixing threats efficiently, leading to a greater chance of skipped patches and exploitation of systems. [3]

2) Contextualized Knowledge from External Sources: Most automation tools are not integrated with external re- sources such as CVE databases, CWE frameworks, and GitHub discussions. This denies developers rich in-sights into new vulnerabilities and best practices, slowing down responses to security threats. [4]
3) Dynamic Updates Based on Project Evolution: Existing solutions fail to keep documentation up to date with evolving projects. As software evolves, stale documenta- tion causes inefficiencies, confusion, and manual effort to accurately reflect updates. Automated real-time up- dates are necessary to close this gap. [5]
4) The gap in these areas leads to several negative conse- quences: Outdated or incomplete documentation signif- icantly hampers the efficiency of maintenance practices. Developers are often forced to spend excessive time searching for relevant information, resulting in slower maintenance cycles and increased operational costs. Additionally, the absence of comprehensive vulnerability insights and the lack of integration with external knowledge sources heighten security risks. Teams may remain unaware of potential threats or fail to imple- ment best practices for mitigation, thereby increasing the likelihood of security breaches. Furthermore, when documentation is not dynamically updated or enriched with external knowledge, teams tend to rely heavily on individual expertise. This fosters the development of knowledge silos, which impede collaboration and hinder the effective transfer of information across the organization.

C. Research Objectives

This research explores how Retrieval-Augmented Genera- tion (RAG) techniques can enhance automated documentation, improve security reporting, and support software maintenance and audits by leveraging external data and structured AI- generated outputs.

This research addresses the following key questions:

1) How can LLM-RAG techniques enhance automated documentation by integrating real-time, domain-specific knowledge?
2) How does incorporating external vulnerability databases (e.g., CVE, CWE) and bug-tracking systems improve the accuracy and relevance of security reporting?
3) How can structured, AI-generated reports aid in software maintenance and security audits by providing actionable insights and reducing manual effort?

The primary objectives of this research are:

- Develop an LLM-RAG-based documentation framework: Design a system that automates documentation by retriev- ing relevant external data and integrating it into structured outputs.
- Enhance documentation comprehensiveness: Leverage RAG to retrieve real-time information from external sources like vulnerability databases and bug trackers, ensuring up-to-date insights.
- Evaluate documentation quality: Use qualitative (e.g., user feedback) and quantitative (e.g., accuracy metrics) methods to assess the effectiveness of AI-generated doc- umentation in improving efficiency and security.

D. Significance of study

1) Enhanced Security Awareness and Mitigation Strate- gies: By incorporating external vulnerability databases (e.g., CVE, CWE) and bug-tracking systems, the sug- gested system raises awareness of possible threats. This allows developers to react more efficiently to vulnerabil- ities, minimizing risks and maintaining compliance with industry standards.

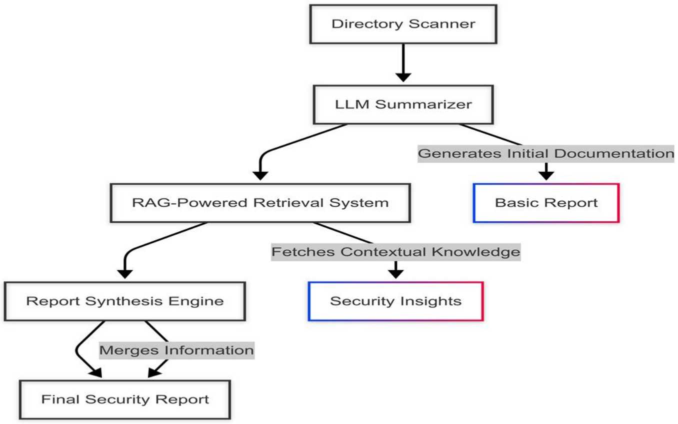

Fig. 1. LegacyGuard System Architecture

2) Less Manual Work in Documentation Mainte- nance: Manual tools reduce effort by removing repeated tasks like manual updates and vulnera- ability tracking. The dynamic update mechanism prevents documentation from becoming outdated without extensive human intervention, thereby saving time and resources for developers.

II. LITERATURE REVIEW

The research on automated documentation creation and vulnerability discovery emphasizes the increasing capability of Large Language Models (LLMs) and Retrieval-Augmented Generation (RAG) methods to solve software maintenance and security problems. Nevertheless, current research is typically deficient in integration of LLMs with external knowledge bases, which results in wide gaps in comprehensiveness, precision, and responsiveness. This section elaborates on major studies and their applicability to this research.

Large Language Models (LLMs) have garnered significant attention for their ability to generate natural language descriptions of code, offering a promising avenue for automated documentation. For example, Mohammed's (2024) work on LLM-driven automation in vulnerability management explores the use of LLMs for tasks like risk identification and mitigation. However, the study lacks consideration of external data integration or dynamic documentation updates, limiting its adaptability in evolving software projects. Similarly, Li et al. (2024) demonstrate how LLMs can support static analysis for detecting security vulnerabilities, yet their approach does not incorporate essential external resources such as CVE databases or bug-tracking systems, which are vital for actionable security documentation. These studies underscore the capabilities of LLMs but also emphasize the necessity for frameworks that integrate real-time external knowledge to enhance documentation accuracy and relevance. Building on this, Retrieval-Augmented Generation (RAG) techniques offer a hybrid solution that merges generative AI with information retrieval, enabling the creation of outputs enriched with up-to-date, context-aware data. Du et al. (2024), in their work on Vul-RAG, propose a RAG-based framework that improves vulnerability detection by leveraging a dedicated knowledge base. Nonetheless, this approach does not prioritize the generation of comprehensive documentation tailored to developers. Lykousas et al. (2024) also explore the role of RAG in DevSecOps, showing its potential in delivering actionable insights but falling short in addressing external knowledge integration and responsiveness to project- specific dynamics. These findings reveal the contextual advantages of RAG techniques, while also pointing out existing shortcomings in their application to documentation systems. Moreover, the integration of external knowledge sources—such as CVE databases, CWE frameworks, and collaborative platforms like GitHub—is crucial for the production of actionable and context-rich documentation. Keltek and Li (2024), through LSAST, use LLMs for static application security testing with effective vulnerability identification, yet do not explore the use of RAG for dynamic documentation. Meanwhile, Bernardi et al. (2024) showcase how RAG-based LLMs can be used to generate safety reports enriched with real-time data, though their application is limited to domains such as construction safety. These studies collectively emphasize the necessity of incorporating external knowledge to make automated documentation more relevant and functional.

Finally, current tools for automated documentation face notable limitations. Many lack adequate security insights due to their inability to integrate with external vulnerability databases, and there is an absence of unified frameworks that combine both internal project data and external knowledge sources. This leads to incomplete documentation that fails to meet the comprehensive needs of modern software development teams.

Gaps Identified: The expanded literature review reveals several key gaps in current research on automated documentation systems. One major limitation is the lack of integration with external data sources; most existing studies fail to incorporate real-time information from platforms such as CVE and CWE databases or collaborative environments like GitHub discussions. Additionally, many tools generate static documentation that does not update dynamically in response to changes within software projects, reducing their long-term usefulness. Another shortcoming is the limited focus on producing actionable security insights—few approaches provide tailored recommendations that address the specific needs of developers. Finally, there is an evident absence of comprehensive documentation frameworks that seamlessly combine internal project data with external knowledge sources, which is essential for creating rich, context-aware documentation.

Relevance of RAG: Retrieval-Augmented Generation (RAG) effectively addresses the gaps identified in current research by enhancing documentation with real-time security insights drawn from external databases. This approach ensures that documentation remains current, dynamically updating to reflect ongoing changes in software projects. Moreover, RAG provides actionable recommendations that are grounded in contextualized knowledge, making the information more rele- vant and useful for developers. By bridging the gap between traditional static analysis tools and comprehensive reporting systems, RAG offers a more holistic and adaptive solution for automated documentation and vulnerability management.

III. RESEARCH METHODOLOGY

This study employs a hybrid methodology, combining LLM- powered text generation with Retrieval-Augmented Generation (RAG) to develop an enhanced automated documentation and security reporting system. The goal is to create a framework that integrates real-time external knowledge with generative capabilities, ensuring accurate, contextual, and up-to-date out- puts. This design allows the system to dynamically adapt to evolving project data and provide comprehensive, actionable security insights.

A. Research Design

The research design is structured around three key compo- nents. First, Large Language Models (LLMs) are utilized to generate initial documentation drafts by analyzing elements such as source code, metadata, and overall project struc- ture (Mohammed, 2024) [6] Second, a Retrieval-Augmented Generation (RAG) framework is employed to enhance the generative process by integrating external knowledge retrieval, resulting in more contextualized and comprehensive outputs (Li et al., 2024) [7] Finally, a hybrid methodology is adopted, combining both qualitative and quantitative evaluation tech- niques. Qualitative assessments involve user studies to evaluate the relevance, readability, and usability of the documentation, while quantitative assessments utilize metrics to measure doc- umentation quality and the accuracy of security vulnerability detection (Du et al., 2024) [8]

B. Data Collection

To train, test, and validate the system, we use the following datasets:

1) Primary Data: Includes source code, project directories, and metadata extracted directly from software reposito- ries. The source code will be selected from a diverse set of open-source projects, including projects written in Python, Java, and JavaScript, to ensure that the system is robust across different programming languages and architectural styles.

2) Secondary Data: Several key external resources play a vital role in enhancing the quality and relevance of automated documentation systems. The CVE (Common Vulnerabilities and Exposures) and CWE (Common Weakness Enumeration) databases serve as essential references for identifying known security vulnerabilities and understanding best practices for mitigation. GitHub Issues offer valuable insights into real-world software problems through reported bugs, user discussions, and resolution histories within active projects. Stack Over- flow provides a rich collection of community-driven questions and answers, contributing additional context for addressing coding issues and enhancing documen- tation clarity. Furthermore, security advisories—official alerts and notices about vulnerabilities in software com- ponents, libraries, and frameworks—offer critical infor- mation that helps maintain security-aware and up-to-date documentation.

The data collection process involves both static extraction of project information and dynamic retrieval of external resources based on context. This dual approach ensures that the gener- ated documentation includes both project-specific details and relevant external knowledge.

C. System Architecture

The proposed system architecture is composed of four key components that collaboratively enable the generation of enriched and security-aware software documentation. The first component, the Directory Scanner, extracts the project's structure, metadata, and source code to form the foundational input for documentation generation. It analyzes file hierarchies, identifies dependencies, and processes configuration files to offer a complete view of the project. This component uses Abstract Syntax Tree (AST) parsing along with regular expressions and pattern-matching techniques to extract meaningful data from source files (Keltek and Li, 2024). [9] The second component is the LLM-based Summarizer, which utilizes advanced natural language processing to analyze the scanned data and generate initial drafts of the documen- tation. It offers insights into the software architecture and performs static analysis to identify potential security vulner- abilities by examining code patterns. Prompt engineering is employed to ensure the generation of accurate, relevant, and human-readable descriptions of complex code structures. The third component, the RAG-powered Retrieval System, dynamically enhances the documentation by fetching real- time, contextually relevant information from external sources such as the CVE and CWE vulnerability databases, as well as bug-tracking platforms like GitHub. It leverages semantic search techniques and uses APIs to access these external resources. Additionally, it employs vector databases for storing and retrieving embeddings of code segments and related contextual information. Finally, the Report Synthesis Engine integrates the summaries generated by the LLM and the contextual insights retrieved by the RAG system to create structured, comprehensive docu- ments tailored for software maintenance and security auditing. This component utilizes predefined templates and supports multiple output formats such as Markdown, HTML, and PDF to maximize usability and adaptability across various development and reporting environments (Bernardi et al., 2024). [10]

D. Data Analysis

The evaluation of the proposed system is conducted using both qualitative and quantitative methods to comprehensively assess its effectiveness. Qualitative evaluation involves user studies and expert reviews. In the user studies, software developers, security auditors, and project managers are engaged to assess the relevance, readability, and usability of the generated documentation. Participants perform specific tasks using the documentation and provide feedback through structured surveys and interviews. In parallel, expert reviews are conducted with security professionals and documentation specialists, who evaluate the accuracy, completeness, and structure of the reports. Their detailed feedback helps refine the quality and utility of the generated content.

On the other hand, quantitative metrics are employed to objectively measure system performance across two key dimensions: documentation quality and vulnerability detection accuracy. For documentation quality, standard NLP evaluation metrics such as BLEU (Bilingual Evaluation Understudy) and ROUGE (Recall-Oriented Understudy for Gisting Evaluation) are used to measure similarity and content overlap between the generated and reference documentation. Additionally, readability is assessed using metrics like the Flesch Reading Ease score to evaluate the clarity and complexity of the generated text.

In terms of vulnerability detection, the system's perfor- mance is evaluated using precision, recall, and the F1 score to determine its accuracy and balance in identifying actual vulnerabilities. Furthermore, time-to-detection is measured to compare the system's efficiency against manual vulnerability identification methods, thereby assessing the potential for time savings in real-world scenarios.

E. Workflow

The workflow of the proposed system is structured into a series of interconnected stages, beginning with project initial- ization, where the user either provides a project directory or a repository URL. The Configuration Manager then applies default or user-defined settings that determine the scope of analysis, desired output format, and intended audience. At this stage, the system also establishes a dedicated working directory to store intermediate results and log files.

Next, in the directory scanning phase, the Directory Scanner navigates through the project structure to extract relevant data such as file hierarchy, source code, and metadata. Dependency information is collected through configuration files like re- quirements.txt for Python or package.json for JavaScript. Ad- ditionally, metadata such as timestamps, author information, and version history is gathered to provide contextual depth. In the preliminary documentation generation stage, the LLM- based Summarizer processes the extracted data to generate initial drafts. This includes textual descriptions of functions, classes, APIs, and the overall architecture of the software. Concurrently, the system performs static analysis to identify and flag potential vulnerabilities by recognizing risky code patterns and outdated dependencies.

To enrich the documentation with real-time insights, the con- text retrieval step employs a RAG-powered Retrieval System. It dynamically queries external knowledge sources such as CVE/CWE databases and GitHub issues, focusing on the pre- viously identified vulnerabilities and dependencies. Retrieved information is ranked based on relevance and recency using semantic search algorithms, then pre-processed and aligned with specific sections of the documentation.

The documentation enhancement phase integrates this contex- tual knowledge into the LLM-generated drafts. This process not only adds security insights but also embeds actionable mitigation strategies for the flagged vulnerabilities. Supple- mentary content such as best practices and real-world exam- ples from similar projects is appended to further enrich the documentation.

Following this, the report synthesis module compiles the en- hanced content into structured documents, supporting multiple output formats including Markdown and HTML. Tailored versions of the documentation are produced for different stakeholder groups—developers, security auditors, and project managers. To improve comprehension, the reports include visual aids like dependency graphs and vulnerability heatmaps. A feedback loop is incorporated into the system to ensure con- tinuous improvement. Users review the final documentation and provide feedback regarding its accuracy, relevance, and usability. This feedback is then used to fine-tune subsequent iterations of the system, and interaction logs help improve underlying models over time.

Finally, the output delivery stage generates and delivers com- prehensive documentation in the requested format, complete with embedded links to external resources for traceability. Security reports include a prioritized list of identified vulner- abilities, each accompanied by detailed recommendations for remediation.

IV. EXPECTED OUTCOMES AND IMPACT

The proposed framework for automated documentation generation using LLM-RAG (Large Language Model with Retrieval-Augmented Generation) techniques offers a paradigm shift in how software documentation and security auditing are approached.

Traditionally, software documentation has relied heavily on manual authoring, which is both time-consuming and prone to becoming outdated as the project evolves. Moreover, security documentation, particularly re- lated to vulnerabilities and mitigation strategies, often lacks integration with real-time threat intelligence. The use of LLMs allows for natural language understanding and generation capabilities, enabling the system to produce human-readable, coherent, and technically accurate documentation from code, metadata, and other internal project artifacts. However, to address the limitations of static outputs, the framework incor- porates a RAG-based architecture, enabling it to retrieve up-to- date, contextually relevant information from external sources and dynamically integrate this into generated reports. [11] One of the central benefits of this approach is the abil- ity to produce automated, structured, and security-enriched documentation. By leveraging real-time data from reputable sources such as CVE (Common Vulnerabilities and Exposures) and CWE (Common Weakness Enumeration) databases, the system ensures that its outputs are not only comprehensive but also current. These databases are authoritative sources widely used in the cybersecurity community to classify and communicate known security flaws. Their integration into the documentation process means that developers are made aware of vulnerabilities in their dependencies and code patterns as they are discovered, rather than post-facto. This shifts security left in the software development lifecycle (SDLC), improving security posture while reducing the time spent manually reviewing advisories and incorporating them into documentation. [11]

In addition, the framework significantly reduces manual effort in documentation maintenance. Traditional tools and work- flows require developers to frequently update documentation as codebases change or as new security advisories emerge. This manual upkeep is not only labor-intensive but often leads to inconsistencies and outdated records, which can become liabilities during audits or when onboarding new developers. The proposed system addresses this by using dynamic update mechanisms that automatically detect changes in project files, dependencies, and external knowledge, and seamlessly reflect these changes in the documentation. By automating repetitive tasks such as vulnerability tracking and configuration summary generation, the system allows developers to focus on higher- order tasks like code optimization and architectural planning Another critical advantage lies in improving vulnerability awareness and mitigation strategies. The framework retrieves and integrates insights from external sources such as CVE and CWE databases, GitHub issue trackers, and Stack Overflow discussions, allowing developers to understand the context, severity, and remediation steps for identified threats [12]. This helps bridge the gap between detection and action, making vulnerability data not just informative but actionable. In traditional setups, developers must manually search for this information, evaluate its credibility, and determine relevance to their project. With the proposed system, this process is stream- lined and embedded directly into the development workflow, thereby reducing response times and increasing the efficacy of mitigation strategies.

The system also enhances explainability and trust in AI- generated documentation. One common critique of AI- generated content is the black-box nature of model outputs. By embedding traceable references to external data sources—such as direct CVE links, GitHub issue numbers, and Stack Over- flow threads—the framework ensures that users can verify and trace the origins of the information presented. This level of transparency increases stakeholder trust in the generated outputs, which is essential for security audits, compliance assessments, and cross-functional collaboration. Furthermore, structured formatting and the use of templates improve doc- ument readability, allowing both technical and non-technical stakeholders to derive insights without steep learning curves.

A. Gantt Chart

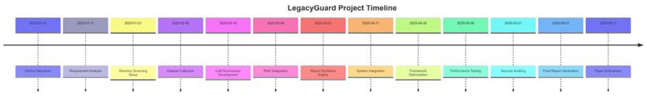

Fig. 2. Gantt Chart

B. Breakdown

Another transformative feature of the system is its support for dynamic updates in evolving software projects. Unlike static documentation tools that require manual regeneration, the proposed framework automatically detects changes in project files, version history, and external security advisories. This ensures that the documentation always mirrors the current state of the project, reducing the risks associated with outdated or incomplete documentation, which is a common issue in agile and continuous integration environments. Moreover, the system greatly streamlines security audits and compliance checks. By embedding best practices, detailed vulnerability data, and mitigation steps into the documentation, the system makes it easier for security teams to assess risk, identify gaps, and provide feedback.

For regulated indus- tries—such as finance, healthcare, or defense—where com- pliance with standards like ISO 27001, HIPAA, or OWASP is critical, this automation reduces the cost and time associated with maintaining audit-ready documentation. [12]

A further benefit is fostering collaboration across development and security teams. Documentation often becomes siloed, with different teams maintaining separate artifacts with inconsistent or conflicting information. The unified, real-time documenta- tion produced by the proposed framework serves as a single source of truth, accessible to all relevant stakeholders. This promotes a culture of shared responsibility for security and knowledge dissemination, which is especially vital in large or distributed teams.

Finally, due to its modular and API-driven architecture, the framework demonstrates high scalability across domains. While it is tailored for software development, the core princi- ples—automated report generation, contextual enrichment via RAG, and dynamic updates—are applicable to other fields such as construction safety documentation, medical record compliance, or even legal reporting. By customizing the knowledge base and summarization logic, the system can be repurposed to meet the reporting requirements of various industries, thereby demonstrating strong potential for cross- domain adaptability and reuse. [11] [12]

V. TIMELINE

The LegacyGuard research project is structured to run over a comprehensive 12-month period, divided into six major phases:

1) *Phase 1: Project Initialization and Planning (Month 1):* The initial month establishes the foundation for the entire project, focusing on refining project objectives, forming the research team, and setting up the necessary technical infras- tructure. During this phase, a research team is assembled, consisting of experts in static code analysis, machine learning for code understanding, and security vulnerability assess- ment. Concurrently, the research infrastructure is established, including development environments, code repositories, and collaborative tools that will facilitate smooth coordination and productivity throughout the project. In addition, the team finalizes initial project planning documents such as detailed work breakdown structures and communication protocols, which outline task distribution, timelines, and reporting methods. A comprehensive technical requirements specification is also completed, which includes documentation of the target legacy programming languages—namely COBOL, C/C++, Java, FORTRAN, and Visual Basic—and their specific vulnerability patterns. Finally, evaluation metrics and baseline performance targets for the framework are established to assess progress and effectiveness in later stages of the project.
2) *Phase 2: Data Collection and Preparation (Months 2-4):* During this period, the project team will focus on gathering diverse legacy codebases and establishing a comprehensive vulnerability knowledge base. The team will acquire open- source legacy codebases spanning the targeted programming languages, with a particular emphasis on systems developed prior to 2010. At the same time, partnerships will be pur- sued with organizations willing to grant access to propri- etary legacy systems under strict confidentiality agreements, enabling a broader and more representative dataset for anal- ysis. This phase will also center around the construction of the vulnerability knowledge base by integrating data from reliable external sources such as the National Vulnerability Database (NVD), Common Vulnerabilities and Exposures (CVE), OWASP Top 10, and other industry-specific security reports focused on legacy software environments. To support the Retrieval-Augmented Generation (RAG) component of the system, additional contextual information—including system documentation, maintenance histories, and architecture records—will be collected. Furthermore, stratified sampling techniques will be implemented to ensure diversity across programming languages, application domains, and vulnerability types. Labeled datasets will be prepared for both training and evaluation purposes, ensuring that the machine learning models developed later in the project are robust and reflective of real-world conditions in legacy code environments.
3) *Phase 3: Component Development (Months 5-7):* During this phase, the three core components of the LegacyGuard framework—static analysis integration, LLM adaptation, and Retrieval-Augmented Generation (RAG) system implementa- tion—will be developed in parallel. The team will begin by configuring and customizing static analysis tools tailored to each target programming language. This includes utilizing CodeSonar for C/C++, FindBugs/SpotBugs for Java, Veracode SAST for COBOL, Visual Expert for Visual Basic, and FLfortra for FORTRAN. These tools will undergo baseline testing to establish performance benchmarks and identify the limitations of traditional static analysis methods when applied to legacy systems. Simultaneously, work will proceed on adapting and fine- tuning large language models (LLMs) for vulnerability detec- tion across multiple languages. The team will experiment with architectures such as CodeBERT and CodeT5 to evaluate their effectiveness in understanding legacy code. Custom fine-tuning strategies will be developed, alongside prompt engineering techniques, to ensure the LLMs can accurately identify and explain vulnerabilities in diverse codebases. In parallel, the RAG system will be implemented to enrich the analysis with contextual knowledge. This will involve setting up a vector database

using ChromaDB, creating embedding generation pipelines for both code samples and vulnerability descriptions, and integrating context-aware retrieval mechanisms. To ensure secure operation, appropriate security controls will be put in place to mitigate potential risks associated with external data retrieval in the RAG system.

4) *Phase 4: Integration and Framework Development (Months 8-9):* This phase concentrates on integrating the individual components into a cohesive and functional Lega- cyGuard framework, while also establishing interfaces and workflow mechanisms for smooth user interaction. During this month, the development team will focus on building the integration layer, which will include the implementation of a multi-model voting system to combine outputs from different analysis engines, along with confidence scoring mechanisms to assess the reliability of identified vulnerabilities. Cross- language correlation capabilities will be integrated to support analysis across heterogeneous legacy systems. Additionally, robust APIs and user interfaces will be developed to facilitate seamless interaction with the framework. To enhance the interpretability and usability of the framework, this month will also involve the implementation of explanation generation capabilities. Leveraging LLMs, the system will produce human-readable explanations for each detected vulnerability, accompanied by actionable remediation recommendations. Furthermore, visualization tools will be designed to help users understand the relationships between vulnerabilities, dependencies, and risk factors, thereby promoting better decision-making in security audits and maintenance planning.
5) *Phase 5: Evaluation and Validation (Months 10-11):* This is a critical stage in the project, focusing on rigorous testing and comprehensive evaluation of the LegacyGuard framework against both established benchmarks and real- world legacy codebases. During this phase, systematic ablation studies will be conducted to quantify the individual contri- bution of each core component—static analysis tools, LLMs, and the RAG system—to the framework's overall performance. These studies will help identify the strengths and limitations of each module and their synergy when combined. In parallel, a comparative analysis will be undertaken to benchmark LegacyGuard against existing vulnerability detection tools across various legacy programming languages, evaluating its competitiveness and performance efficiency. Furthermore, this phase will include real-world validation using proprietary legacy codebases provided by industry partners under confidentiality agreements. The framework will be tested in realistic environments, allowing for robust performance measurement in terms of precision, recall, and F1-score. In addition to quantitative assessments, expert evaluators will perform a qualitative analysis of the human- readable explanations and remediation suggestions generated by the system, offering insights into its practical utility and trustworthiness in real-world software maintenance and security auditing contexts.
6) *Phase 6: Documentation and Dissemination (Month 12):* The final stage of the project is centered on documenting research findings and preparing for their dissemination to both academic and industry stakeholders. During this period, the team will complete all research-related documentation, including comprehensive technical details of the LegacyGuard framework, a thorough analysis of evaluation outcomes, and the drafting of academic papers aimed at submission to relevant conferences and peer-reviewed journals. In parallel, dissemination activities will be initiated, which include pack- aging open-source components for public release, developing case studies that showcase the practical applications of the framework, and creating best practices guides for potential adopters. To further extend outreach, workshops and webinars will be organized to demonstrate the capabilities and value of LegacyGuard to software developers, security professionals, and academic researchers alike, ensuring the framework's impact is both far-reaching and sustainable.

REFERENCES

[1] N. Lykousas, V. Argyropoulos, and F. Casino, "The potential of llm- generated reports in devsecops," arXiv.org, vol. abs/2410.01899, Oct. 2024. [Online]. Available: https://export.arxiv.org/pdf/2410.01899v1.pdf

[2] M. L. Bernardi, M. Cimitile, and R. Pecori, "Automatic job safety report generation using rag-based llms," vol. abs/1605.02592, p. 1–8, Jun. 2024.

[3] X. Du, G. Zheng, K. J. Wang, J. Feng, W. Deng, M. Liu, X. Peng, T. Ma, and Y. Lou, "Vul-rag: Enhancing llm-based vulnerability detection via knowledge-level rag," Jun. 2024. [Online]. Available: https://arxiv.org/pdf/2406.11147

[4] K. Mohammed, "Llm-driven automation in vulnerability management," Open access research journal of science and technology, Sep. 2024.

[5] Z. Li, S. Dutta, and M. Naik, "Llm-assisted static analysis for detecting security vulnerabilities," May 2024. [Online]. Available: https://arxiv.org/pdf/2405.17238

[6] M. Keltek and Z. Li, "Lsast – enhancing cybersecurity through llm-supported static application security testing," Sep. 2024. [Online]. Available: https://export.arxiv.org/pdf/2409.15735v2.pdf

[7] R. Gupta, G. Pandey, and S. K. Pal, "Automating government report gen- eration: A generative ai approach for efficient data extraction, analysis, and visualization," Sep. 2024.

[8] K. E. Hill, "Systems and methods for soft- ware scanning tool," Mar. 2016. [Online]. Available: https://patents.google.com/patent/US20160274903

[9] J. Chen, H. Xiang, L. X. Li, Y. Zhang, B. Ding, and Q. Li, "Utilizing precise and complete code context to guide llm in automatic false positive

mitigation," Nov. 2024. [Online]. Available: http://arxiv.org/pdf/2411.03079

[10] W. Dai, Q. Ouyang, X. Zeng, C. Zhao, L. Zhu, and Y. Chen, "Method of automatically generating report," Sep. 2018.

[11] J. L. Turner and R. E. Turner, "Method for providing customized and automated security assistance, a document marking regime, and central tracking and control for sensitive or classified documents in electronic format," Sep. 2006. [Online]. Available: https://patents.google.com/patent/US7958147B1/en

[12] S. Pranathi, T. Akshita, M. Vaishnavi, M. Ramachandra, and D. Sundaragiri, "Transforming raw data into polished reports: An llm- powered solution for customizing template-based pdfs," International Journal For Multidisciplinary Research, May 2024. [Online]. Available: https://www.ijfmr.com/papers/2024/3/18590.pdf