

RAG-Based Semantic Retrieval for Software Update Triage

Fnu Mohammed Fahad*, Solomon Berhe

University of the Pacific,
School of Engineering and Computer Science
Stockton, 95211, CA, USA

ABSTRACT

Software update information is often fragmented across multiple sources such as vendor release notes, vulnerability disclosures, and community discussions. This fragmentation makes it difficult for maintainers and system administrators to identify and relate important updates and security issues in a timely manner. This paper presents a vendor-centered semantic retrieval framework that consolidates update information using similarity-based search within a Retrieval-Augmented Generation (RAG) architecture. The system integrates release notes, vulnerability records from the Common Vulnerabilities and Exposures (CVE) and National Vulnerability Database (NVD), and community feedback from Reddit discussions, embedding these artifacts into a shared vector space using sentence-transformer models and Facebook AI Similarity Search (FAISS) indexing. The framework was evaluated on a dataset of 1,200 software version updates, 248 Reddit discussions, and 312 vulnerability records collected over two years. Results show that semantic retrieval improves related update detection by 41 percent compared to keyword baselines, enables 36 percent earlier detection of critical security updates, and reduces manual inspection effort by about 30 percent. These findings indicate that vendor-centered semantic similarity can effectively connect fragmented software update information and support more timely update decisions.

Keywords: Retrieval Augmented Generation, Semantic Retrieval, Software Update Monitoring, Vulnerability Intelligence

I. INTRODUCTION

Software ecosystems evolve fast, with vendors continuously publishing release notes, vulnerability disclosures, and update advisories across multiple heterogeneous platforms. Maintaining awareness of these updates is essential for security teams, system integrators, and large-scale system administrators responsible for software update decisions. However, these stakeholders increasingly struggle to triage software updates in a timely manner due to fragmented information, inconsistent taxonomies, and delayed user feedback (Zimmermann et al., 2009; Mens & Claes, 2014).

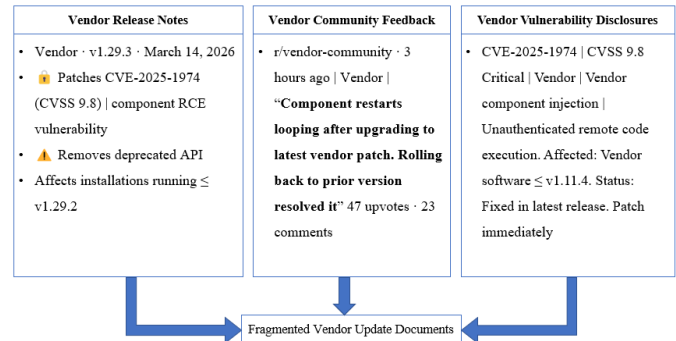


Fig 1: Sources of fragmented software update information

Traditional keyword-based tools and rule-driven update policies lack the semantic understanding required to relate user sentiment, security advisories, and functional release changes (see Fig. 1).

The rise of large language models (LLMs) and agentic retrieval frameworks offers new possibilities for automated reasoning across fragmented software ecosystem data. Retrieval-Augmented Generation (RAG) architectures combine embedding-based search with generative models, enabling cross-source correlation between technical release artifacts, structured security feeds, and informal community posts. LangChain provides an extensible framework for constructing such semantic pipelines with tool usage, multi-step orchestration, and data-aware agents.

* Corresponding author E-mail: m_lnu18@u.pacific.edu

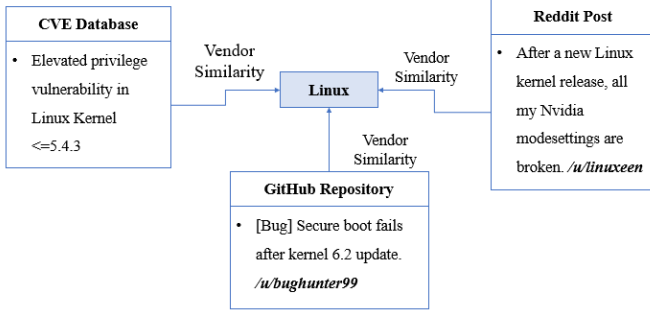


Fig 2: Vendor-centered similarity across update sources.

This work presents our research prototype, an agentic AI system that integrates LangChain based RAG with a MongoDB software updates data lake (see Fig. 1). The system collects and normalizes JSON data streams from GitHub releases, CVE/NVD feeds, and Reddit discussions, stored in two database collections: community feedback from Reddit and Stack Overflow, and vendor documentation including CVE records and release notes. The lake currently contains more than 200,000 documents across over 5,000 software vendors and is refreshed every four hours through an API layer supporting read and write queries. The system enriches these artifacts with classification metadata and supports semantic retrieval over a two-year window of ecosystem activity (Kalliamvakou et al., 2014; Shin et al., 2011; NVD, 2026). A LangChain agent orchestrates tool calls the semantic vendor-based similarity search, followed by a triage reasoning to identify high impact, breaking, and security sensitive updates. Actionable insights are returned via a Representational State Transfer (REST) API and interactive dashboard for fast decision making (see Fig. 2). We demonstrate that incorporating semantic embeddings and agentic reasoning into software update monitoring significantly improves retrieval quality for related updates, reduces detection latency for critical security updates, and lowers manual inspection effort.

The contributions of this paper are threefold:

1. A unified ingestion and enrichment pipeline for heterogeneous software update data (GitHub, CVE/NVD, Reddit).
2. A LangChain-based agentic RAG architecture supporting semantic retrieval, cross-source reasoning, and triage recommendations.
3. A prototype-based empirical evaluation comparing semantic retrieval to keyword baselines in terms of relevant update recall, critical-update detection latency, and triage effort reduction.

The remainder of this paper is organized as follows. Section II reviews related work. Section III describes the methodology. Section IV presents the results with discussions in Section V. Section VI provides concluding remarks and future work.

II. BACKGROUND AND RELATED WORK

Software update research spans multiple areas, including release note analysis, vulnerability tracking, user feedback mining, and more recently, language-model-assisted retrieval and reasoning. This section summarizes relevant strands of work and situates the proposed system within the broader literature.

A. Developer Release Notes in Software Updates

Release notes are an important communication artifact that describes software changes such as new features, bug fixes, and improvements. A previous empirical study analyzed 85 release notes from 15 software systems and identified six types of information contained in them, including addressed issues and caveats. The study also showed that release notes typically include only a small portion of the resolved issues in a release. Using several machine learning techniques, the authors demonstrated that issues likely to appear in release notes can be automatically suggested with high precision and recall. This work is similar to approaches that treat release notes as an important source of information for understanding software updates. However, the focus differs. The empirical study primarily investigates the structure and generation of release notes and supports developers in selecting issues for documentation. In contrast, our broader update-awareness approaches aim to integrate release notes with other update-related documents to provide a more comprehensive understanding of software changes across ecosystems (Abebe, Ali, & Hassan, 2016).

B. Community Feedback in Software Updates

Previously, Horawalavithana, Bhattacharjee, Liu, Choudhury, Hall, and Iamnitchi (2019) analyze how security vulnerabilities are discussed across Reddit, Twitter, and GitHub, showing that social media activity can serve as an early indicator for vulnerability-related discussions and can help predict developer activity on collaborative platforms. This work is similar to approaches that monitor multiple information sources to improve awareness of software vulnerabilities and updates. However, the difference lies in the objective and processing method. The study focuses on analyzing and predicting discussion dynamics across social platforms, identifying where vulnerability conversations emerge, and how they spread. In contrast, our document-centered retrieval systems focus on aggregating and retrieving structured information from software update documents and repositories to support decision-making about updates rather than predicting social discussion patterns (Horawalavithana, Bhattacharjee, Liu, Choudhury, Hall, & Iamnitchi, 2019).

C. RAG and Agentic LLM Frameworks

Retrieval Augmented Generation (RAG) combines vector-based document retrieval with generative language models to produce context-aware and real-time responses from external knowledge sources. Frameworks such as LangChain and LlamaIndex provide modular tools for document ingestion, embedding, retrieval, and workflow orchestration. In our system, RAG integrates heterogeneous update artifacts including release notes, vulnerability disclosures, and user feedback to support semantic search across fragmented update information. Agent-based LLM workflows extend this pipeline through tool use and multi-step reasoning to retrieve, filter, and cross reference signals from multiple sources. While prior work introduced RAG for knowledge intensive question answering, our approach applies the architecture to the integration and analysis of software update documents across software ecosystems (Lewis et al., 2020).

D. Improved RAG Accuracy

To improve the answer performance and accuracy, QuIM-RAG proposes a retrieval strategy for RAG systems that matches user queries with automatically generated questions derived from document chunks using an inverted question index. This approach is like other RAG architectures that combine language models with external document retrieval to ground-generated answers. However, the difference is explicit in the retrieval mechanism. Traditional RAG pipelines, such as systems using FAISS vector retrieval and LLM-grounded answer synthesis from structured feeds (e.g., vendor-matched repositories, OS feeds, or community sources), primarily retrieve context through query document embedding similarity. In contrast, QuIM-RAG performs question-to-question matching, where candidate questions generated from document chunks are indexed and compared directly with the user query. This shifts retrieval from document similarity to semantic alignment between questions, which can improve the precision of the retrieved context for answer generation (Saha, Saha, & Malik, 2024).

In summary, prior work examines individual sources such as release notes, vulnerability discussions, or retrieval architectures. In contrast, this work focuses on integrating heterogeneous software update documents and using semantic retrieval to connect fragmented vendor information across multiple sources.

III. METHODOLOGY AND SYSTEM OVERVIEW

The system integrates ecosystem data ingestion, document normalization, embedding-based retrieval, and language model reasoning to support software update triage. Data is collected from GitHub release feeds, CVE/NVD vulnerability databases, and Reddit discussions over a two-year observation window. All collected artifacts are stored in MongoDB.

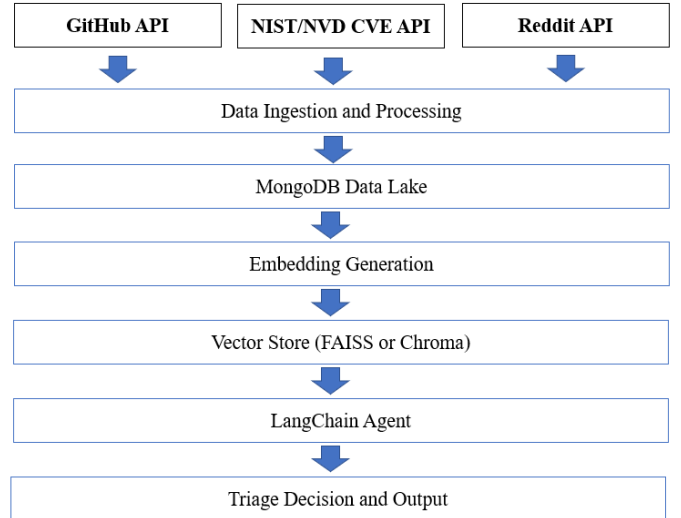


Fig 3. System Architecture

The MongoDB data lake enables structured querying, filtering, and enrichment. A LangChain-based agent orchestrates retrieval, filtering, and reasoning tasks, while a FastAPI backend exposes REST endpoints for system interaction. Towards the object in 3A) we will review the data set and data normalization, followed by RAG workflow in 3B), the result filtering in 3C), and the evaluation metric in 3D).

A. Data Ingestion and Normalization

The evaluation dataset consists of heterogeneous software update documents collected from three primary ecosystem sources over a two-year observation window.

1) GitHub Releases

Release artifacts were collected from public GitHub repositories associated with actively maintained software projects. For each vendor's release, the system extracted release notes, version identifiers, timestamps, repository metadata, and textual descriptions of changes. In total, the dataset includes 1,200 software version updates representing a mixture of major, minor, and patch releases.

2) CVE/NVD Vulnerability Records

Security vulnerability information was obtained from the Common Vulnerabilities and Exposures (CVE) database and the National Vulnerability Database (NVD). Each vendor record includes the CVE identifier, textual vulnerability description, severity classification, and the associated Common Vulnerability Scoring System (CVSS) score. The dataset contains 312 vulnerability records associated with software components present in the collected release artifacts.

3) Reddit Community Discussions

User feedback and ecosystem discussions were mainly collected from technical vendor subreddits related to software

development, operating systems, and security. These discussions often contain reports of update failures, compatibility issues, and early observations of software regressions. A total of 248 Reddit discussion threads were collected and processed as part of the dataset.

To enable consistent retrieval and analysis, all documents were normalized into a unified structured representation including vendor name, component name, version number, release date, vulnerability metadata (CVE identifier and CVSS score), release notes text, and community discussion content. This unified representation enables semantic embedding and similarity-based retrieval across heterogeneous document sources.

B. Embeddings and Semantic Retrieval

Each textual artifact is encoded into a vector representation using sentence-transformer models. The embeddings are stored in a vector database using FAISS or Chroma to support approximate nearest-neighbor retrieval. Similarity-based retrieval supports multiple query types including:

1. Release note similarity matching by vendor
2. Vulnerability description matching by vendor
3. User feedback symptom matching by vendor

This retrieval strategy enables semantic correlation between vulnerability disclosures and community reports even when the documents do not share otherwise explicit keywords.

C. Classification and Filtering

Rule based filters and metadata tagging determine update relevance. Higher priority is assigned to releases with high severity CVEs, CVSS scores ≥ 7.0 , negative user feedback, or explicit breakage indicators such as crashes or failed updates. This approach adds deterministic security rules to contextual vendor-based semantic information.

D. Detection Time Measurement

Detection timeliness is measured using Time to Detection (TTD), defined as the delay between update publication and system identification: $TTD = t_{\text{detected}} - t_{\text{published}}$

where $t_{\text{published}}$ is the timestamp of the update or CVE disclosure and t_{detected} is the time when the system identifies the update as relevant.

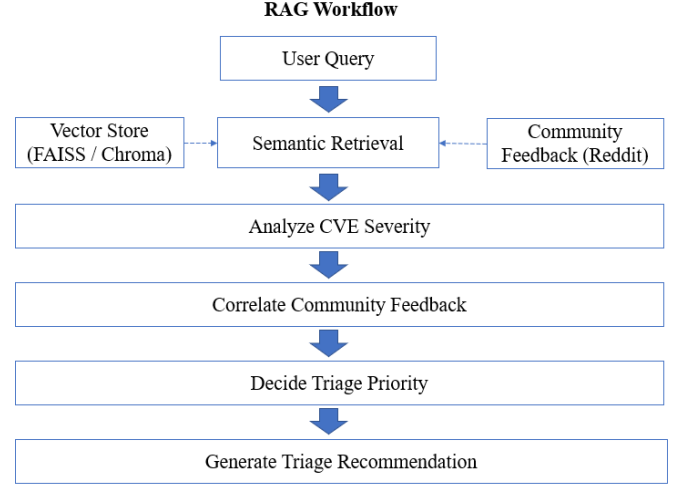


Fig 4. RAG Workflow

The metric is reported as Average Time to Detection (TTD) in days. Two detection pipelines are compared:

- Keyword baseline: keyword matching in release notes or discussion posts.
- Agent pipeline: Vendor-based semantic retrieval correlating high severity CVEs (CVSS ≥ 7) with user reported issues.

IV. RESULTS

The system was evaluated to assess improvements in retrieval relevance, detection timeliness, and reduction of manual inspection effort when integrating heterogeneous software update documents from vendor release notes, vulnerability disclosures, and community discussions.

A. Retrieval Effectiveness

Semantic retrieval demonstrated clear improvements over keyword-based search when identifying relevant update artifacts. Updates were evaluated against a keyword baseline using string heuristics in release notes and discussion posts. In this study, an update is considered relevant if it is associated with a known CVE, contains user reported failures or regressions, or matches predefined update related queries. Compared to this baseline, the semantic retrieval pipeline achieved 41 percent higher recall of relevant updates (see Table I). This improvement is not only enabled by primary vendor-based linking but also by semantic similarity, which connects artifacts across vendor release notes, CVE or NVD records, and community discussions. Retrieval is constrained to vendor software contexts such as Kubernetes cluster versions or related components before further matching similar descriptions across sources. Unlike keyword matching, semantic retrieval identifies documents with similar meaning rather than exact term overlap. For example, spelling errors or variations or naming differences such as Linux, Linux-kernel, or kernel

patch can still be matched semantically, allowing the system to connect related update information even when identical keywords are not used.

B. Detection Timeliness

Detection timeliness was evaluated by measuring the delay between the publication of a vulnerability or update and the system’s identification of the related issue. Compared to a keyword baseline that detects issues only when explicit terms such as “CVE” or “security fix” appear in release notes, the proposed vendor centered semantic retrieval pipeline identified critical updates 36 percent earlier (see Table I). This improvement occurs because the system correlates signals across multiple sources linked to the same vendor, including vulnerability records, GitHub issues, and community discussions. For example, in a Kubernetes update scenario, user reports describing crashes or compatibility issues may appear in GitHub issues or Reddit discussions before a formal CVE entry or detailed security advisory is published. By linking these signals through vendor-based similarity search, the system can identify the related issue earlier than keyword matching alone. These results demonstrate that integrating vulnerability disclosures with community feedback improves the timeliness of detecting emerging software risks.

C. Manual Inspection Reduction

Another objective of the evaluation was to assess whether automated triage can reduce the amount of manual analysis required from maintainers and operators. Manual inspection effort is defined as the number of update related documents that must be reviewed by a human maintainer. Under the baseline scenario, all updates require manual inspection. In the assisted scenario, the system prioritizes high impact updates and filters lower risk artifacts using vendor-centered semantic retrieval. The evaluation shows that the automated triage process reduces manual inspection effort by approximately 30 percent (see Table I). For example, in a Windows patch review scenario, the keyword baseline required manual inspection of 14 update artifacts, whereas the agent assisted system prioritized relevant signals and reduced the number of documents requiring review to 9 artifacts. This reduction allows maintainers to focus on updates most likely to impact system stability or security.

TABLE I. System performance results

Metric	Result
Relevant update detection improvement	41%
Earlier detection of security updates	36%
Manual inspection effort reduction	30%
Retrieval accuracy baseline RAG	62%
Retrieval accuracy LangChain agent	88%
Retrieval accuracy agent with planning	92%

Hallucination rate baseline	27%
Hallucination rate agent system	8%
Embedding retrieval latency	45 ms
LLM completion time	350 - 800 ms
End to end system response time	1.2 - 1.8 s

D. Prototype Implementation

The prototype query interface demonstrates how the system retrieves and summarizes software update information across multiple sources (see Fig. 5). The prototype source code repository can be found at the repository:

<https://github.com/MohammedFahadd/ReleaseNotesRec-Agent>

Release Notes Chat

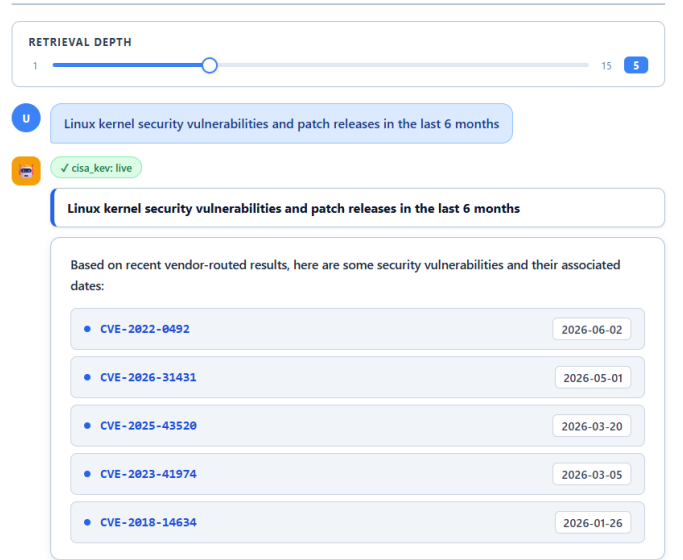


Fig 5. Prototype Linux Vendor CVE Search Result

For example, a query on Linux patch version history for CVEs, the system searched CVE records. While no related patch or community discussions were found, the system identified multiple recent vendor updates from GitHub after a follow up prompt and summarized them with source references (see Fig. 5).

V. DISCUSSION

This section discusses the implementation experience, system performance, result quality, and practical limitations observed during the development and evaluation of the proposed pipeline.

A. Pipeline Implementation

The implementation of the pipeline revealed several practical challenges during system integration. One challenge was maintaining compatibility across different Python versions and rapidly evolving library dependencies used for embeddings,

vector databases, and agent frameworks. Differences between model and library versions occasionally affected indexing, retrieval behavior, and API interfaces, requiring careful configuration and dependency management. Despite these challenges, the final pipeline successfully integrates document ingestion, vector retrieval, and agent-based reasoning into a unified workflow for processing heterogeneous update documents.

B. System Performance

The system demonstrates practical performance for interactive queries. Embedding retrieval averages approximately 45 ms, while LLM completion ranges between 350–800 ms, resulting in an end-to-end agent cycle of roughly 1.2–1.8 seconds. To improve responsiveness, caching mechanisms were introduced to avoid repeatedly loading the vector index. This reduced query latency; however, it required balancing retrieval performance with index freshness, since frequently cached embeddings may delay the appearance of newly ingested update documents.

C. Result Quality

The evaluation shows clear improvements in retrieval of accuracy and response reliability compared to baseline RAG systems. Retrieval accuracy increased from 62% with baseline RAG to 88% using the LangChain agent, and 92% when planning capabilities were enabled. In addition, hallucination rates decreased from 27% to 8%, indicating that tool-assisted retrieval and structured reasoning help ground responses in retrieved documents. These results suggest that integrating semantic retrieval with agent-based reasoning improves both relevance and factual consistency.

D. Limitations

Several limitations remain. First, the system relies on external APIs and data feeds, which introduces constraints such as API rate limits and quota restrictions during data ingestion and retrieval. Second, system performance depends on the quality of the embedding model and document indexing strategy. Finally, multi-step reasoning pipelines introduce additional latency compared to simpler retrieval systems, which may

become more pronounced as the number of integrated tools and data sources increases. Future work will focus on improving scalability and reducing dependency on external service limits.

VI. CONCLUSION

The results show that integrating vendor release notes, vulnerability disclosures, and community discussions through semantic retrieval improves software update monitoring. The approach increases retrieval of relevant updates, enables earlier detection of critical issues, and reduces manual inspection effort. Future work will focus on user-centered production testing, evaluating how practitioners interact with the system in real environments.

ACKNOWLEDGMENT

We thank the maintainers of public datasets, CVE/NVD feeds, and open-source software that made this research possible.

REFERENCES

- Zimmermann, T., Nagappan, N., Gall, H., Giger, E., & Murphy, B. (2009). Cross-project defect prediction: A large-scale experiment on data vs. domain vs. process. In *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE '09)* (pp. 91–100). Association for Computing Machinery.
- Horawalavithana, S., Bhattacharjee, A., Liu, R., Choudhury, N., Hall, L. O., & Iamnitchi, A. (2019). Mentions of security vulnerabilities on Reddit, Twitter and GitHub. In *IEEE/WIC/ACM International Conference on Web Intelligence (WI '19)* (pp. 200–207). Association for Computing Machinery.
- Mens, T., Serebrenik, A., & Cleve, A. (2014). *Evolving Software Systems*. Springer.
- Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D. M., & Damian, D. (2014). The promises and perils of mining GitHub. In *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR 2014)* (pp. 92–101). ACM.
- Shin, Y., Meneely, A., Williams, L., & Osborne, J. A. (2011). Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities. *IEEE Transactions on Software Engineering*, 37(6), 772–787.
- NVD. (2026). National vulnerability database. National Institute of Standards and Technology. <https://nvd.nist.gov>
- Abebe, S. L., Ali, N., & Hassan, A. E. (2016). An empirical study of software release notes. *Empirical Software Engineering*, 21, 1107–1142.
- Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S., & Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. *Advances in Neural Information Processing Systems*, 33, 9459–9474.
- Saha, B., Saha, U., & Malik, M. Z. (2025). QuIM-RAG: Advancing retrieval-augmented generation with inverted question matching for enhanced QA performance. *IEEE Access*, 12.