Auditing Technical Security Debt through Retrieval-Augmented Generation: A CWE, CAPEC, and STRIDE-Based Approach

Kleiton Ewerton de Oliveira[1], **Gleiph Ghiotto Lima de Menezes**[1], **André Luiz de Oliveira**[1]

[1]Departamento de Ciência da Computação,
Instituto de Ciências Exatas - Universidade Federal de Juiz de Fora (UFJF)
Juiz de Fora – MG – Brazil

kleiton.ewerton@estudante.ufjf.br, {andre.oliveira, gleiph.ghiotto}@ufjf.br

Abstract. *Technical Security Debt (TSD) refers to latent vulnerabilities that may not immediately compromise software functionality but progressively weaken its defenses. Conventional Static Application Security Testing (SAST) tools provide the basis for vulnerability detection and remain essential in secure development workflows. However, complementary semantic layers can support contextualizing the findings in terms of data flows, attack patterns, and architectural impact. In this paper, we propose a neuro-symbolic approach for auditing TSD by integrating Large Language Models (LLMs), Retrieval-Augmented Generation (RAG), and security ontologies. Our approach uses the CWE $\rightarrow$ CAPEC $\rightarrow$ STRIDE chain as the supporting structure to link implementation-level weaknesses to attack patterns, and preliminary architectural threat hypotheses. We use the Open Worldwide Application Security Project (OWASP) Benchmark v1.2, we built a pipeline that enriches code fragments with ontological metadata and retrieves semantically similar examples during inference. The RAG-assisted configuration achieved 90.88% of accuracy, and a weighted F1-score of 0.9142 in CWE classification, outperforming an LLM-only baseline by 20.44% points in accuracy. A paired McNemar test confirmed that this improvement was statistically significant ($p = 3.85 \times 10^{-34}$). The results indicate that retrieval and ontological grounding can reduce semantic drift in vulnerability auditing, while STRIDE-based threat mapping should be interpreted as a preliminary contextualization layer rather than as a definitive threat-modeling oracle.*

1. Introduction

Modern software development is increasingly driven by rapid delivery cycles, continuous integration, and pressure for frequent releases. In this context, functionality is often prioritized at the expense of structural robustness, favoring the accumulation of Technical Debt [Lenarduzzi et al. 2021, Khan and Uddin 2022]. Among its most critical manifestations it is Technical Security Debt (TSD), which refers to latent security weaknesses embedded in the logic or architecture of software systems. Although such weaknesses may not immediately compromise execution or functional correctness, they progressively increase the risk of exploitation as the system evolves [Izurieta et al. 2018, Martinez et al. 2021].

The automated detection of TSD remains challenging because security risks often emerge from relationships among implementation choices, data flows, and potential exploitation paths. Traditional Static Application Security Testing (SAST) tools,

commonly based on *constraint analysis*, rule matching, and control or data-flow abstractions, are effective at identifying known vulnerability patterns. However, their findings often require additional semantic interpretation to support risk prioritization, especially when exploitability depends on relationships among code behavior, assets, and architectural assumptions. This limitation contributes to false positives, hinders risk prioritization, and intensifies alert fatigue among developers and security analysts [Fu et al. 2024, Paul et al. 2025a, Lenarduzzi et al. 2021].

Large Language Models (LLMs) provide advanced semantic analysis capabilities and have shown potential for reasoning about source code, vulnerability descriptions, and security-related artifacts [Nam et al. 2024, Xue et al. 2025]. Nevertheless, their isolated application to security auditing is constrained by hallucinations, outdated parametric knowledge, and limited knowledge in authoritative security taxonomies [Yao et al. 2024]. Vulnerability auditing requires more than pattern recognition: it demands the ability to relate implementation-level weaknesses to attack patterns and architectural threats. Therefore, an effective approach to TSD auditing should combine semantic reasoning with structured security knowledge.

This exploratory study is guided by the following research question: *to what extent can a Retrieval-Augmented Generation (RAG) architecture, enriched with security ontologies, support CWE-based TSD detection and provide preliminary architectural threat contextualization from source code?* To address this question, we propose an approach based on RAG [Lewis et al. 2020], integrating LLMs with security ontologies to support contextual and traceable vulnerability analysis. We formalize TSD detection as a causal quadruple $D = \langle c, v, a, t \rangle$, where c represents the analyzed code, v the vulnerability or weakness expressed as a CWE, a the corresponding attack pattern represented by CAPEC, and t the architectural threat derived from STRIDE. This modeling strategy enables the inference of relationships between implementation-level defects and their potential impact on the attack surface.

The main contribution of this study is a neuro-symbolic auditing approach that uses the CWE-CAPEC-STRIDE chain as the basis for inference. The proposed pipeline enriches code analysis with ontological metadata, retrieves semantically similar examples to guide the model, and it maps detected weaknesses to architectural threat categories. By addressing this issue, we complement syntactic and rule-based detection with a semantic layer for risk-oriented threat inference. Empirically, the RAG-assisted configuration achieved 90.88% accuracy and a weighted F1-score of 0.9142 in CWE classification over 548 test cases from the *Open Worldwide Application Security Project* [1]*(OWASP) Benchmark v1.2*, improving the accuracy of an LLM-only baseline from 70.44% to 90.88%. A paired McNemar test further confirmed the statistical significance of this improvement ($p = 3.85 \times 10^{-34}$).

This paper is organized as follows. In Section 2, we present Technical Security Debt and other concepts necessary for the reader to understand the contributions of this research. Related work is highlighted in Section 3. In Section 4, we present the research methodology and the proposed RAG-based approach to identify security technical debt in the source code. In Section 5, we present the evaluation. We highlight the threats to

[1] https://github.com/OWASP-Benchmark/BenchmarkJava

validity in Section 6. Finally, in Section 7, we present the conclusion and sketch future research directions.

2. Background

In this section, we introduce the concepts of Technical and Security Debt, the causal security modeling using CWE, CAPEC, and the STRIDE method, and describe the role of RAG in grounding LLM-based security inference to support the reader in understanding the contributions of this study.

2.1. Technical Security Debt

TSD refers to latent vulnerabilities in software logic or architecture that increase the exposure to attacks over time [Izurieta et al. 2018]. Unlike Self-Admitted Technical Debt (SATD), which is explicitly documented in source-code comments, TSD is typically implicit and it must be inferred from implementation behavior, data flows, control flows, and architectural assumptions [Martinez et al. 2021, Fu et al. 2024]. This makes TSD harder to detect than comment-based debt, since the relevant risk may not appear as an explicit textual marker, but as a relationship between code, assets, and possible exploitation paths.

2.2. CWE, CAPEC, and STRIDE as a Causal Chain

To relate implementation-level weaknesses to security impact, we adopt three complementary security knowledge structures: the *Common Weakness Enumeration* (CWE), the *Common Attack Pattern Enumeration and Classification* (CAPEC), and the STRIDE threat model. CWE provides a structured catalog of software and hardware weaknesses that may lead to vulnerabilities [MITRE 2026c], whereas CAPEC organizes known attack patterns used by adversaries to exploit such weaknesses [MITRE 2026a]. Moreover, Common Vulnerabilities and Exposures (CVE) provides a standardized reference system for publicly disclosed vulnerabilities, while NVD enriches CVE records with additional information such as severity scores, impact data, and search capabilities [MITRE 2026b, MITRE Corporation 2026].

We used this chain in this study as a structured reasoning layer for TSD auditing. The goal is not only to classify a vulnerable code fragment, but also to contextualize it as part of a broader security risk that may affect confidentiality, integrity, authentication, authorization, or system behavior.

2.3. LLMs and Retrieval-Augmented Generation

LLMs provide strong semantic capabilities for reasoning over source code and security descriptions, but their direct application to vulnerability auditing is limited by hallucinations, outdated parametric knowledge, and weak grounding in authoritative taxonomies [Yao et al. 2024]. RAG mitigates these limitations by incorporating external knowledge during inference [Lewis et al. 2020]. In this study, RAG enables the model to retrieve semantically similar examples and ontology-enriched metadata, reducing exclusive dependence on parametric knowledge and supporting more traceable CWE and STRIDE inference.

3. Related Work

Recent research on software security analysis has moved from rule-based and syntactic techniques towards approaches based on semantic representation, machine learning, and contextual reasoning. This section positions the present study in relation to three research directions: technical debt detection, LLM-based software security, and threat modeling supported by structured security knowledge.

3.1. Technical Debt and Security-Oriented Code Analysis

The detection of SATD has evolved from keyword-based techniques to deep learning models capable of capturing semantic nuances in developer comments [Sharma et al. 2022, Khan and Uddin 2022]. Hybrid neural architectures have also been explored to address class imbalance and textual heterogeneity in SATD datasets [Njeru et al. 2025]. However, these approaches remain dependent on explicit textual evidence. As a result, they are less suitable for identifying implicit security debt, where the risk is embedded in code behavior rather than declared in comments [Russo et al. 2025].

This limitation motivates the focus of the present work. Instead of detecting declared debt, we address TSD as a latent security condition that requires semantic code analysis and security knowledge. Prior studies also show that code quality affects model reliability: code smells and noisy implementation patterns may compromise generalization and lead models to learn unstable correlations [Xue et al. 2025]. This issue is particularly relevant in vulnerability auditing, where small implementation details may determine whether a fragment is actually exploitable or only syntactically suspicious.

3.2. LLMs and RAG in Software Security

LLMs have been applied to vulnerability detection, configuration analysis, vulnerability reproduction, and automated repair. Previous work shows that prompt formulation can substantially affect detection performance, with prompt tuning improving the identification of security-relevant patterns [Ren et al. 2024]. LLMs have also been used to interpret complex configurations and reproduce state-dependent failures, indicating their potential to reason beyond isolated code fragments [Fu et al. 2024]. In industrial settings, generative models have been investigated for vulnerability fixing, although they may produce patches without fully capturing the broader security implications of a defect [Costa et al. 2025].

RAG has emerged as a way to reduce the limitations of parametric knowledge by incorporating external sources during inference [Lewis et al. 2020]. Recent security studies have used RAG to integrate knowledge bases such as CWE and CVE, improving the foundation of vulnerability analysis [Paul et al. 2025b]. However, most uses of RAG remain oriented toward retrieval support rather than structured reasoning. In contrast, this work uses RAG to operationalize the CWE $\rightarrow$ CAPEC $\rightarrow$ STRIDE chain, supporting the relation between implementation weaknesses, attack patterns, and preliminary architectural threat hypotheses.

3.3. Threat Modeling and Traceability

LLM-based approaches have also been applied to STRIDE threat modeling at the system level [AbdulGhaffar and Matrawy 2025]. However, these analyses often operate at

a high level of abstraction and remain disconnected from concrete implementation artifacts. This limits traceability between the code where a weakness is introduced and the architectural property that may be compromised. The present study addresses this gap by combining semantic code analysis, ontology-enriched retrieval, and STRIDE inference over benchmarked code instances. By doing so, it positions TSD auditing as a bridge between vulnerability detection and preliminary architectural threat contextualization.

4. Research Methodology and the Proposed Approach

The proposed approach is characterized as neuro-symbolic in a pragmatic sense: it combines neural components, namely vector-based code representation and LLM inference, with symbolic security artifacts, namely CWE, CAPEC, STRIDE mappings, and rule-based output constraints. We do not employ a formal logical prover or symbolic execution engine; instead, we use ontological metadata and decision rules to constrain, contextualize, and audit neural inference. In this setting, RAG acts as a semantic anchoring mechanism that supports the causal chain CWE $\rightarrow$ CAPEC $\rightarrow$ STRIDE.

4.1. Dataset and Data Engineering

We use the *OWASP Benchmark v1.2* as the experimental oracle, a Java-based benchmark suite designed to evaluate application security testing tools, including SAST, DAST, and IAST tools [OWASP Foundation 2026, OWASP Benchmark Project 2026]. Version 1.2 comprises 2,740 executable test cases, each associated with a vulnerability category, a CWE identifier, and an expected true-positive or false-positive outcome [OWASP Foundation 2026]. To make these instances suitable for semantic auditing, we implemented an ETL pipeline that removes irrelevant textual noise and enriches each sample with security-oriented metadata. The enrichment stage integrates CWE and CAPEC knowledge bases and maps weaknesses to STRIDE categories using CAPEC metadata and predefined decision rules.

This process transforms isolated code instances into enriched semantic units, allowing each sample to be analyzed in terms of its syntactic structure, weakness class, attack pattern, and potential architectural security impact.

4.2. RAG Architecture and Inference Pipeline

To mitigate the limitations of LLM's parametric knowledge, we adopted a RAG architecture that combines vector memory with generative inference [Lewis et al. 2020]. The dataset was divided into two disjoint partitions to avoid *data leakage*: 80% of the samples were used as the retrieval base for vector indexing, totaling 2,192 instances, while 548 cases were reserved for blind evaluation. This separation ensures that the target instance itself is not retrieved during evaluation.

For vector representation, we use **Nomic Embed v1.5**, which supports sequences of up to 8k tokens and allows complete Java classes to be represented with limited contextual loss. The resulting embeddings are indexed using *ChromaDB*, and cosine similarity is applied to retrieve the $k = 3$ closest examples for each target fragment.

During inference, the target code is embedded and used to retrieve semantically similar examples with their associated CWE, CAPEC, and STRIDE metadata. This retrieved context is incorporated into the prompt and provided to **Llama-3.3-70B**, executed

through the Groq API with the temperature set to 0. The model returns a structured JSON object containing the predicted CWE, STRIDE category, and evidence-based justification.

4.3. Prompting Strategy and Experimental Protocol

The prompting strategy was designed to reduce ambiguity between semantically close CWE classes and to prevent decisions based solely on isolated keywords. For CWE classification, the prompt enforces rules grounded in APIs, methods, and flow patterns. For instance, primitives such as `MessageDigest` and algorithms such as MD5 or SHA1 are prioritized as CWE-328, whereas ciphers such as DES and RC4, or improper uses of `Cipher.getInstance`, are prioritized as CWE-327. Similarly, CWE-614 is assigned only when there is evidence of cookie creation and addition without the corresponding security flags. The complete system prompts, along with the JSON schema definitions, are provided in the supplementary artifacts repository. To support reproducibility, the source code, scripts, configuration files, and aggregated results are available through a [2]repository.

Table 1 summarizes the experimental protocol adopted in this study, detailing the dataset configuration, embedding and generative models, and evaluation metrics used.

Table 1. Summary of the experimental protocol

Aspect	Adopted configuration
Dataset	*OWASP Benchmark v1.2*, with Java/Servlet cases.
Data split	80% for retrieval and 20% for blind evaluation.
Indexed samples	2,192 enriched instances.
Evaluated samples	548 cases, all with valid CWE predictions.
Embedding model	Nomic Embed v1.5, with up to 8k tokens.
Vector database	ChromaDB with cosine similarity.
Retrieved examples	$k = 3$.
Generative model	Llama-3.3-70B via Groq API.
Temperature	0.
Output format	Structured JSON with CWE, STRIDE, and justification.
Metrics	Accuracy, precision, recall, F1-score, STRIDE agreement, and McNemar's test.

5. Evaluation and Discussion

We conducted an experimental evaluation on the reserved test set of the *OWASP Benchmark v1.2* access the feasibility of the proposed approach on identifying security debt in the source code. A total of 548 cases were evaluated, and all of them produced valid structured CWE predictions after response sanitization and parsing. The retrieval base and the evaluation set were kept separate to avoid *data leakage*. Embeddings were generated locally using **Nomic Embed v1.5**, whereas generative inference was performed with **Llama-3.3-70B** through the Groq API, with temperature set to zero to reduce response variability.

To assess the contribution of retrieval, we also executed an LLM-only baseline over the same 548 test cases. This baseline used the same generative model, temperature,

[2]https://github.com/KleitonEwerton/Auditing-Technical-Security-Debt-through-Retrieval-Augmented-Generation

output format, and CWE label space, but did not receive retrieved examples or ontological metadata from the CWE-CAPEC-STRIDE chain. This design allows us to compare the proposed RAG-assisted configuration against a controlled LLM-only setting while preserving the same evaluation partition and output constraints.

5.1. Vulnerability Classification (CWE)

The proposed RAG-assisted approach achieved **90.88% overall accuracy** and a **weighted F1-score of 0.9142** in CWE classification. Table 2 presents the per-class metrics. The results indicate particularly strong performance on data-flow vulnerabilities, such as SQL Injection, XSS, Command Injection, Path Traversal, XPath Injection, and LDAP Injection. These classes achieved perfect classification in the evaluated partition. The model also achieved perfect results for CWE-327 and CWE-328, suggesting that the retrieved examples and ontological metadata helped distinguish cryptographic weaknesses that were substantially more ambiguous in the LLM-only setting.

Table 2. Performance metrics by CWE class for the RAG-assisted configuration

Class (CWE)	Prec.	Rec.	F1	Supp.
CWE-22 (Path Traversal)	1.00	1.00	1.00	42
CWE-327 (Broken Crypto)	1.00	1.00	1.00	43
CWE-328 (Weak Hash)	1.00	1.00	1.00	47
CWE-330 (Weak Randomness)	1.00	0.49	0.66	96
CWE-501 (Trust Boundary)	0.64	0.97	0.77	29
CWE-614 (Secure Cookie)	0.28	1.00	0.43	13
CWE-643 (XPath Injection)	1.00	1.00	1.00	4
CWE-78 (Command Injection)	1.00	1.00	1.00	40
CWE-79 (XSS)	1.00	1.00	1.00	111
CWE-89 (SQL Injection)	1.00	1.00	1.00	111
CWE-90 (LDAP Injection)	1.00	1.00	1.00	12
Macro Avg.	**0.90**	**0.95**	**0.90**	**548**
Weighted Avg.	**0.96**	**0.91**	**0.91**	**548**

Table 3 presents the confusion matrix for CWE classification. The errors are concentrated in only three transitions: CWE-330 $\rightarrow$ CWE-614, CWE-330 $\rightarrow$ CWE-501, and CWE-501 $\rightarrow$ CWE-614. This concentration suggests that the model did not fail in a distributed manner across classes, but rather within a semantically specific region associated with weak randomness, cookies, sessions, and trust-boundary-related constructs.

Table 3. CWE classification confusion matrix for the RAG-assisted configuration

Real \Pred	CWE-22	CWE-327	CWE-328	CWE-330	CWE-501	CWE-614	CWE-643	CWE-78	CWE-79	CWE-89	CWE-90
CWE-22	42	0	0	0	0	0	0	0	0	0	0
CWE-327	0	43	0	0	0	0	0	0	0	0	0
CWE-328	0	0	47	0	0	0	0	0	0	0	0
CWE-330	0	0	0	47	16	33	0	0	0	0	0
CWE-501	0	0	0	0	28	1	0	0	0	0	0
CWE-614	0	0	0	0	0	13	0	0	0	0	0
CWE-643	0	0	0	0	0	0	4	0	0	0	0
CWE-78	0	0	0	0	0	0	0	40	0	0	0
CWE-79	0	0	0	0	0	0	0	0	111	0	0
CWE-89	0	0	0	0	0	0	0	0	0	111	0
CWE-90	0	0	0	0	0	0	0	0	0	0	12

The main source of error was CWE-330 (*Weak Randomness*), with 49 false negatives: 33 classified as CWE-614 and 16 as CWE-501. This behavior indicates that tokens such as `Cookie`, `Session`, and `setAttribute` still exerted a strong influence on the inference process, shifting the model's focus from entropy and predictability properties to cookie configuration or trust-boundary issues. In contrast, the remaining classes showed stable performance, with perfect classification for CWE-22, CWE-78, CWE-79, CWE-89, CWE-90, CWE-327, CWE-328, and CWE-643. Thus, the observed limitation does not generally compromise the detection capability of the approach, but reveals a localized weakness in vulnerabilities whose risk depends on algorithmic properties that are less explicit in the code.

5.2. Comparison with an LLM-only Baseline

Table 4 compares the proposed RAG-assisted configuration with the LLM-only baseline. Both configurations produced valid CWE predictions for all 548 cases. The LLM-only baseline achieved 70.44% accuracy and a weighted F1-score of 0.6672, whereas the RAG-assisted configuration achieved 90.88% accuracy and a weighted F1-score of 0.9142. This corresponds to an improvement of 20.44 percentage points in accuracy and a reduction from 162 to 50 classification errors.

Table 4. Comparison between LLM-only and RAG-assisted configurations for CWE classification

Configuration	Valid outputs	Accuracy	Macro F1	Weighted F1	Errors
LLM-only	548/548	0.7044	0.6955	0.6672	162
RAG-assisted	548/548	0.9088	0.8962	0.9142	50

The comparison indicates that retrieval substantially improves CWE classification stability. The LLM-only baseline overpredicted CWE-79 and CWE-614, while underdetecting CWE-327 and CWE-330. In particular, CWE-327 was predicted only once by the LLM-only configuration despite having 43 true instances, and CWE-330 was predicted

only five times despite having 96 true instances. Conversely, the RAG-assisted configuration produced a more stable prediction distribution, correctly matching the expected support for most CWE classes and concentrating its remaining errors in the boundary between CWE-330, CWE-501, and CWE-614.

Table 5 summarizes the main error transitions in both configurations. The LLM-only setting produced broad confusions involving CWE-79 and CWE-614, including 40 cases of CWE-327 classified as CWE-79 and 16 cases of CWE-328 classified as CWE-79. These confusions disappear in the RAG-assisted configuration. The remaining errors are concentrated in weak-randomness cases misclassified as cookie or trust-boundary weaknesses.

Table 5. Main CWE confusions in the LLM-only and RAG-assisted configurations

Expected CWE	Predicted CWE	LLM-only	RAG-assisted
CWE-330	CWE-614	81	33
CWE-330	CWE-501	0	16
CWE-327	CWE-79	40	0
CWE-328	CWE-79	16	0
CWE-501	CWE-79	6	0
CWE-501	CWE-614	4	1
CWE-614	CWE-79	3	0

Since both configurations were evaluated on the same 548 test cases, we applied McNemar's exact test to assess whether the observed difference was statistically significant. The paired comparison showed that both configurations correctly classified 386 cases, while the RAG-assisted configuration corrected 112 cases misclassified by the LLM-only baseline. Conversely, there were no cases correctly classified only by the LLM-only configuration, and 50 cases were misclassified by both. The exact two-sided test indicated a statistically significant difference between the two configurations ($p = 3.85 \times 10^{-34}$), confirming that the improvement obtained by the RAG-assisted configuration is not attributable to random variation.

These results support the hypothesis that retrieval and ontological metadata reduce semantic drift in CWE classification. However, they do not isolate the individual effects of retrieval, ontological enrichment, and prompt constraints. Therefore, while the comparison strengthens the empirical evidence for the proposed configuration, a complete ablation study remains necessary to separate the contribution of each component.

5.3. Architectural Threat Inference (STRIDE)

In addition to the CWE classification, we evaluated the coherence of architectural inference based on STRIDE. The adopted metric was agreement between the STRIDE class inferred by the model and the expected STRIDE set derived from the enriched oracle. The RAG-assisted configuration achieved **75.00% agreement**, as shown in Table 6.

Table 7 summarizes the distribution of STRIDE categories inferred by the LLM-only and RAG-assisted configurations. Both configurations produced STRIDE predictions for all 548 cases. The RAG-assisted setting produced fewer *Spoofing* predictions and more *Tampering* predictions than the LLM-only baseline, suggesting that the retrieved

Table 6. STRIDE inference performance for the RAG-assisted configuration

Metric	Value
Total analyzed cases	548
Agreement	411
Divergences	137
Overall agreement rate	**75.00%**
Divergence rate	**25.00%**

CWE–CAPEC–STRIDE context influenced the model toward integrity-related interpretations in several cases.

Table 7. Distribution of STRIDE predictions by configuration

Configuration	Info. Disc.	Tampering	Spoofing	Elev. Priv.
LLM-only	281	69	158	40
RAG-assisted	260	144	104	40

The STRIDE results should be interpreted more cautiously than CWE classification. Unlike CWE labels, which are evaluated against a single weakness category, STRIDE categories are inherently context-sensitive and may admit multiple plausible threat interpretations for the same weakness. For example, an injection vulnerability may affect *Information Disclosure* when used for unauthorized reading, but *Tampering* when used to modify state. Similarly, cryptographic weaknesses may be interpreted in terms of confidentiality, integrity, or authentication depending on the operation and asset under analysis.

Therefore, the main empirical gain of the proposed approach is observed in CWE classification, where the RAG-assisted configuration substantially outperformed the LLM-only baseline. STRIDE inference should be interpreted as a preliminary threat-mapping layer that supports architectural reasoning, rather than as a definitive threat-modeling oracle. Future evaluations should consider expert validation or multi-label threat modeling to better capture the ambiguity inherent to architectural security inference.

5.4. Qualitative Analysis of Boundary Cases

To complement the quantitative metrics, we analyzed boundary cases that illustrate both the strengths and limitations of the approach. Table 8 summarizes representative patterns. The correct cases indicate that retrieved examples support reasoning over data flows, including structures involving inner classes and conditional paths. Conversely, the divergent cases show that the approach may prioritize architectural or contextual signals that differ from those adopted by the oracle. This behavior is relevant for TSD auditing: rather than merely reproducing static classifications, the model can identify risk relationships not explicitly captured by a single-label oracle, although this also requires additional validation to avoid semantic false positives.

5.5. Computational Performance

The processing of the 548 instances used an RTX 3050 GPU for local embedding generation, whereas generative inference was executed through the Groq API. The observed

Table 8. Qualitative summary of boundary cases

Case	Result	Interpretation
BenchmarkTest01396	Correct	The model tracked data flow through an inner class and classified the SQL Injection case according to the vulnerable operation.
BenchmarkTest01346	Correct	The model identified the vulnerable branch in a conditional structure, classifying XSS as a threat associated with information exposure.
BenchmarkTest01540	Incorrect	The model confused weak randomness with cookie configuration, suggesting attention bias toward tokens such as `Cookie` and `Session`.
BenchmarkTest00183	Divergent	The prediction indicated a trust-boundary-related interpretation, revealing a plausible architectural risk not necessarily prioritized by the oracle.

average processing time was approximately 1.2 seconds per audited Java class. This result indicates preliminary feasibility for integration into automated auditing workflows, such as CI/CD pipelines. However, since inference was not executed entirely in a local environment, the evaluation demonstrates only partial feasibility in privacy-constrained scenarios. Fully local execution with open-weight models remains a direction for future work.

5.6. Practical Implications

The results suggest three practical implications for secure development teams. First, CWE classification can serve as a semantic triage layer that complements SAST tools, especially for data-flow vulnerabilities and injection-related weaknesses, for which the model showed consistent performance. Second, STRIDE inference can help transform technical alerts into preliminary hypotheses about architectural impact, supporting the prioritization of fixes according to affected security properties, such as confidentiality, integrity, authentication, or authorization. Third, the RAG structure facilitates the update of security knowledge without requiring model retraining, allowing new versions of ontologies or audited examples to be incorporated into the vector database.

Nevertheless, the approach should not be interpreted as a replacement for SAST tools or deterministic validation. Its intended role is to provide a semantic auditing layer that complements static analysis, human review, symbolic execution, and *fuzzing*. Thus, the most appropriate use of the proposed approach is as a support mechanism for auditing and prioritization, reducing the initial effort required to interpret security alerts while preserving the need for expert validation in high-risk contexts.

6. Threats to Validity

Internal validity is mainly affected by the sensitivity of LLM-based inference to prompt formulation, retrieved context, and output constraints. To mitigate this risk, we used a zero-temperature setting, structured JSON outputs, and explicit decision rules for CWE classification and STRIDE inference. Moreover, the comparison with an LLM-only baseline was conducted over the same test partition, model, temperature, output format, and

label space, reducing confounding factors in the evaluation of retrieval effects. Nevertheless, the errors involving CWE-330, CWE-501, and CWE-614 indicate that the model may still prioritize semantically salient tokens, such as `Cookie`, `Session`, and `setAttribute`, over less explicit algorithmic properties such as entropy and predictability. Moreover, since the retrieved examples were selected through vector similarity, the model may be influenced by cases that are structurally similar but differ in the security property that determines the correct classification.

External validity is limited by the exclusive use of the *OWASP Benchmark v1.2*, which consists of Java/Servlet cases designed for controlled vulnerability assessment. As noted as an important limitation, restricting the evaluation to a controlled benchmark and Java cases limits the generalization of the results to other ecosystems, programming languages, and real-world software scenarios. Although CWE, CAPEC, and STRIDE are language-independent taxonomies, the effectiveness of the approach may vary across these different environments, architectural styles, and real-world systems involving complex dependencies, distributed states, authentication flows, and multi-service interactions.

Construct validity is related to the use of benchmark labels and enriched metadata as the evaluation oracle. This issue is particularly relevant for STRIDE inference, since a single weakness may plausibly affect multiple security properties, such as confidentiality, integrity, authentication, or authorization. As a result, agreement with an expected STRIDE set may still fail to capture the full ambiguity of architectural threat modeling. Although the qualitative analysis of boundary cases helps contextualize such divergences, future evaluations should include expert validation or multi-label threat modeling to better assess whether a predicted threat category is security-relevant even when it differs from the reference mapping.

Conclusion validity is mitigated, but not eliminated, by the inclusion of an LLM-only baseline and a paired McNemar test over the same 548 cases. The RAG-assisted configuration outperformed the baseline in CWE classification, correcting 112 cases that were misclassified by the LLM-only setting, with no cases corrected only by the baseline. However, this comparison does not isolate the individual effects of retrieval, ontological enrichment, and prompt constraints. Therefore, the results support the effectiveness of the proposed RAG-assisted, ontology-enriched configuration against the evaluated LLM-only baseline, but should not be interpreted as conclusive superiority over all possible LLM configurations or over traditional SAST tools. Likewise, STRIDE results should be interpreted as preliminary threat contextualization rather than as validated architectural threat modeling.

7. Conclusion

In this study, we presented a neuro-symbolic approach for auditing Technical Security Debt (TSD), combining Large Language Models, Retrieval-Augmented Generation (RAG), and security ontologies. The proposed approach uses the CWE $\rightarrow$ CAPEC $\rightarrow$ STRIDE chain to relate implementation-level vulnerabilities to attack patterns and preliminary architectural threat hypotheses, shifting the analysis from a purely syntactic perspective toward risk-oriented semantic auditing.

The evaluation on the *OWASP Benchmark v1.2* demonstratedthat that the RAG-assisted configuration achieved 90.88% accuracy and a weighted F1-score of 0.9142 for

CWE classification over 548 valid test cases. Compared with an LLM-only baseline evaluated under the same experimental protocol, the proposed configuration improved accuracy from 70.44% to 90.88% and reduced classification errors from 162 to 50. A paired McNemar test further confirmed that this difference was statistically significant, with 112 cases correctly classified only by the RAG-assisted configuration and no cases correctly classified only by the LLM-only baseline ($p = 3.85 \times 10^{-34}$).

The results were particularly consistent for data-flow and injection-related vulnerabilities, such as SQL Injection, XSS, Command Injection, Path Traversal, XPath Injection, and LDAP Injection, as well as for the evaluated cryptographic weakness classes CWE-327 and CWE-328. Conversely, the remaining errors were concentrated around weak randomness, cookie configuration, and trust-boundary-related cases, especially in the transitions CWE-330 $\rightarrow$ CWE-614 and CWE-330 $\rightarrow$ CWE-501. This suggests that, although retrieval and ontological metadata reduce broad semantic drift, generative models may still be influenced by salient tokens such as `Cookie`, `Session`, and `setAttribute` when the relevant security property depends on less explicit algorithmic characteristics, such as entropy and predictability.

For STRIDE-based threat mapping, the RAG-assisted configuration achieved 75.00% agreement with the enriched reference mapping. We interpret this result as preliminary evidence that the proposed chain can contextualize CWE predictions in terms of architectural security properties. However, STRIDE inference is inherently context-sensitive and should not be treated as a definitive threat-modeling oracle without expert validation or multi-label evaluation. Thus, the strongest empirical evidence in this study lies in CWE classification, while STRIDE serves as a complementary layer for risk-oriented interpretation.

As future work, we intend to conduct broader ablation studies to isolate the individual effects of retrieval, ontological enrichment, and prompt constraints. We also plan to compare the proposed approach with representative SAST tools and other commercial LLMs known for their effectiveness in vulnerability analysis (e.g., Claude) under the same benchmark partition. Additionally, we intend to investigate deterministic validators, such as symbolic analysis, targeted *fuzzing*, and *Code Property Graphs*, to reduce semantic false positives and improve auditing robustness. Finally, fully local execution of generative inference remains a relevant direction in scenarios with strict privacy and code-sovereignty requirements.

Acknowledgments

We thank the Department of Computer Science and Computer Science Postgraduate Program from Universidade Federal de Juiz de Fora, CAPES, and CAPES PRO-DEFESA grant number V3084362P for funding this research.

Use of AI Tools

Writing-support tools were used only for linguistic revision and textual clarity, without participation in the methodology, experiments, analysis or conclusions.

References

AbdulGhaffar, A. and Matrawy, A. (2025). Llms' suitability for network security: A case study of stride threat modeling. *arXiv preprint arXiv:2505.04101*.

Costa, L. A. M., Fontão, A., Dos Santos, R. P., and Serebrenik, A. (2025). Applying generative artificial intelligence for vulnerability fixing in a proprietary software ecosystem. *Journal of Systems and Software*, page 112723.

Fu, Y., Wang, T., Li, S., Ding, J., Zhou, S., Jia, Z., Li, W., Jiang, Y., and Liao, X. (2024). Missconf: Llm-enhanced reproduction of configuration-triggered bugs. In *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings*, pages 484–495.

Izurieta, C., Rice, D., Kimball, K., and Valentien, T. (2018). A position study to investigate technical debt associated with security weaknesses. In *Proceedings of the 2018 International Conference on technical debt*, pages 138–142.

Khan, J. Y. and Uddin, G. (2022). Automatic detection and analysis of technical debts in peer-review documentation of r packages. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 765–776. IEEE.

Lenarduzzi, V., Besker, T., Taibi, D., Martini, A., and Fontana, F. (2021). A systematic literature review on technical debt prioritization: Strategies, processes, factors, and tools. *J. Syst. Softw.*, 171:110827.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., et al. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474.

Martinez, J., Quintano, N., Ruiz, A., Santamaria, I., de Soria, I. M., and Arias, J. (2021). Security debt: characteristics, product life-cycle integration and items. In *2021 IEEE/ACM International Conference on Technical Debt (TechDebt)*, pages 1–5. IEEE.

MITRE (2026a). Common attack pattern enumeration and classification (capec). `https://capec.mitre.org/`. Accessed em: 29 jan. 2026.

MITRE (2026b). Common vulnerabilities and exposures (cve). `https://cve.mitre.org/`. Accessed em: 29 jan. 2026.

MITRE (2026c). Common weakness enumeration (cwe). `https://cwe.mitre.org/`. Accessed em: 29 jan. 2026.

MITRE Corporation (2026). CVE and NVD Relationship. `https://www.cve.org/Resources/Media/Archives/OldWebsite/about/cve_and_nvd_relationship.html`. Accessed: 2026-05-22.

Nam, D., Macvean, A., Hellendoorn, V., Vasilescu, B., and Myers, B. (2024). Using an llm to help with code understanding. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pages 1–13.

Njeru, V. K., Kuria, J., and Kituku, B. (2025). Hybrid bert, cnn-bilstm model for detecting self-admitted technical debt in software development. In *2025 IEEE Conference on Computer Applications (ICCA)*, pages 1–6. IEEE.

OWASP Benchmark Project (2026). OWASP Benchmark for Java. `https://github.com/OWASP-Benchmark/BenchmarkJava`. Accessed: 2026-05-20.

OWASP Foundation (2026). OWASP Benchmark Project. `https://owasp.org/www-project-benchmark/`. Accessed: 2026-05-20.

Paul, D. G., Zhu, H., and Bayley, I. (2025a). Investigating the smells of llm generated code. *arXiv preprint arXiv:2510.03029*.

Paul, S., Alemi, F., and Macwan, R. (2025b). Llm-assisted proactive threat intelligence for automated reasoning. *arXiv preprint arXiv:2504.00428*.

Ren, Z., Ju, X., Chen, X., and Shen, H. (2024). Prorlearn: boosting prompt tuning-based vulnerability detection by reinforcement learning. *Automated Software Engineering*, 31(2):38.

Russo, B., Melegati, J., and Mock, M. (2025). Leveraging multi-task learning to improve the detection of satd and vulnerability. *arXiv preprint arXiv:2501.15934*.

Sharma, R., Shahbazi, R., Fard, F. H., Codabux, Z., and Vidoni, M. (2022). Self-admitted technical debt in r: detection and causes. *Automated Software Engineering*, 29(2):53.

Xue, Z., Zhang, X., Gao, Z., Hu, X., Gao, S., Xia, X., and Li, S. (2025). Clean code, better models: Enhancing llm performance with smell-cleaned dataset. *arXiv preprint arXiv:2508.11958*.

Yao, Y., Duan, J., Xu, K., Cai, Y., Sun, Z., and Zhang, Y. (2024). A survey on large language model (llm) security and privacy: The good, the bad, and the ugly. *High-Confidence Computing*, 4(2):100211.