

RAG Against the Machine: Zero-Shot Software Vulnerabilities Classification using LLMs

Edvin Nordqvist
KTH Royal Institute of Technology
Stockholm, Sweden

Changjie Wang
KTH Royal Institute of Technology
Stockholm, Sweden

Simone Ferlin
Red Hat
Stockholm, Sweden

Mariano Scazzariello
RISE Research Institutes of Sweden
Stockholm, Sweden

Marco Chiesa
KTH Royal Institute of Technology
Stockholm, Sweden

Abstract

Timely detection and mitigation of software vulnerabilities is essential for maintaining secure systems. While recent research has applied Transformer-based models to automatically identify vulnerabilities from sources such as GitHub issues, these approaches do not classify the detected vulnerabilities. Without systematic classification, it becomes difficult to prioritize issues, link them to remediation strategies, or integrate them into broader security workflows. This paper introduces a novel approach for automated vulnerability categorization using Large Language Models (LLMs) and Retrieval-Augmented Generation (RAG) within the Common Weakness Enumeration (CWE) framework. We present a RAG-based pipeline for CWE labeling and systematically evaluate multiple retrieval strategies to assess their effectiveness and efficiency. The results demonstrate that our approach achieves competitive performance compared to prior state-of-the-art methods, despite operating entirely in a zero-shot setting without any task-specific training. Moreover, the proposed method enables accurate and cost-effective vulnerability classification, helping to reduce the gap between discovery and remediation in the vulnerability lifecycle. This work highlights the potential of retrieval-driven approaches to enhance the automation, scalability, and practicality of vulnerability management across the software ecosystem.

CCS Concepts

• **Computing methodologies** → *Machine learning approaches*; • **Security and privacy** → **Vulnerability management**.

Keywords

Vulnerability classification, Retrieval-Augmented Generation (RAG), Large Language Models (LLMs), Common Weakness Enumeration (CWE)

ACM Reference Format:

Edvin Nordqvist, Changjie Wang, Simone Ferlin, Mariano Scazzariello, and Marco Chiesa. 2026. RAG Against the Machine: Zero-Shot Software Vulnerabilities Classification using LLMs. In *3rd International Workshop on Large Language Models For Code (LLM4Code '26)*, April 12–18, 2026, Rio de

Janeiro, Brazil. ACM, New York, NY, USA, 8 pages. <https://doi.org/10.1145/3786181.3788722>

1 Introduction

Early vulnerability detection is essential in modern systems.

In the digital age, the world is run on software, computer systems, and network infrastructures. The increasing dependency on digital systems in everything, from power grids to the world economy, makes vulnerabilities in such systems possibly catastrophic on a global scale. In addition, the accumulated pressure to accelerate software development increases the risk and frequency of such vulnerabilities being introduced. In fact, the Google Threat Intelligence Group has observed a rising trend in zero-day exploitations since 2021 [14]. Zero-day vulnerabilities are of particular concern, as they constitute flaws in software, hardware, or firmware that are unknown or not recognized by the manufacturer [8]. Closing the gap between the emergence, discovery, and disclosure stages of the vulnerability lifecycle is therefore crucial.

Vulnerability detection and weakness identification remain predominantly manual tasks.

Over the years, several databases have been developed to systematically track and report software vulnerabilities. One of the most widely recognized is the Common Vulnerabilities and Exposures (CVE) system, maintained by the MITRE Corporation [1]. The CVE system plays a pivotal role in the coordinated identification, classification, and public disclosure of vulnerabilities, offering a standardized framework that ensures consistency across the cybersecurity community. Each CVE entry is a machine-readable record that includes, at a minimum, a brief description of the vulnerability, the affected products and versions, and at least one relevant reference such as a GitHub issue [2]. Closely related to the CVE system is the Common Weakness Enumeration (CWE), a community-maintained taxonomy comprising more than 900 weaknesses also managed by the MITRE [3]. While CVE entries document specific, individual vulnerabilities, CWE provides a standardized taxonomy of common software and hardware weakness types that can lead to such vulnerabilities. Despite the availability of these standardized systems, the process of vulnerability detection and weakness identification remains heavily dependent on manual effort and expert knowledge. As a result, detection continues to be labor-intensive and error-prone [29], creating a significant bottleneck and highlighting the urgent need for automation and intelligence in this domain.



This work is licensed under a Creative Commons Attribution 4.0 International License. *LLM4Code '26, Rio de Janeiro, Brazil*

© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2412-1/2026/04
<https://doi.org/10.1145/3786181.3788722>

LLMs can detect early vulnerabilities, but without categorization their usefulness remains limited. Recently, the rise of Transformer-based architectures and Large Language Models (LLMs) has led to applications across diverse software engineering tasks, including code completion/refactoring, revisioning, and vulnerability detection [19]. Prior work with LLMs has primarily focused on identifying bug risks within code snippets [10, 20, 25, 31], often relying on static code analysis to detect vulnerabilities in well-defined contexts. In contrast, Cipollone *et al.*, investigated the use of Transformer-based models for the automated detection of yet-to-be disclosed vulnerabilities by leveraging textual knowledge sources such as GitHub issues and social media discussions [15]. The study demonstrates that Transformer-based models can achieve competitive accuracy in vulnerability detection. Nonetheless, the proposed pipeline is limited in that it produces only brief textual descriptions of vulnerabilities without providing any structured classification or integration into the CWE. Without categorization, prioritizing vulnerabilities, linking them to remediation strategies, or integrating them into security workflows becomes difficult, reducing their practical value for companies and organizations.

CWE’s complexity makes reliable vulnerability categorization challenging. Automatically categorizing vulnerabilities may seem straightforward, but in practice it is a highly complex task. The CWE database is organized as a forest-like hierarchy in which categories become increasingly abstract at higher levels [4]. Each CWE entry contains examples, detailed descriptions, related attack patterns, and sometimes mitigation strategies. Several attempts tried to automate CWE classification [9, 11, 12, 17, 22, 26, 27, 32], often by training task-specific models. However, the scarcity of labeled data for many CWE categories restricts the size of usable training datasets and results in low classification accuracy. LLMs offer a potential alternative for this task, yet their direct application faces significant challenges. Feeding the entire CWE database to an LLM would require processing *more than 3 million tokens*, which is computationally impractical. Moreover, as input size grows, LLMs are increasingly prone to hallucinations [24], reducing the reliability of classification. In this setting, misclassification is far from a minor issue: assigning a vulnerability to the wrong category can mislead remediation efforts and leave critical flaws unaddressed. Reliability is therefore crucial, highlighting the need for robust approaches that go beyond naive reliance on “raw” LLM outputs.

Scope. In this paper, we explore the use of Retrieval-Augmented Generation (RAG) [23] as a way to improve automated vulnerability categorization. Specifically, we extend the pipeline proposed by Cipollone *et al.* by incorporating the CWE database as an external knowledge base and leveraging the vulnerability description generated by the model as the query. In doing so, we aim to close a critical gap in prior work by moving beyond solely vulnerability detection and toward structured categorization. Our results show that this approach is both viable and practically useful, demonstrating the potential of RAG to facilitate efficient and low-cost vulnerability identification. Ultimately, such methods can help accelerate the vulnerability life cycle, support more reliable remediation, and reduce the risk of exploitation.

Contributions. In this paper, we make four major contributions:

- To the best of our knowledge, this is the first study to explore the use of LLMs and RAG for automated CWE labeling and vulnerability categorization.
- We design a task-aware RAG framework and conduct a comprehensive ablation study to assess the effect of different retrieval and reasoning strategies on classification performance.
- We extend prior work on automated vulnerability detection from GitHub issues [15] by introducing a framework that also performs categorization of vulnerabilities.
- Our results demonstrate that RAG can effectively support vulnerability labeling, achieving performance comparable to previous state-of-the-art methods without any pre-training or fine-tuning. This highlights its strong potential for integration into future automated vulnerability management pipelines.

2 Background & Related Work

In this section, we begin with an overview of CWE, followed by a discussion of prior research on automatic CWE classification.

Common Weakness Enumeration (CWE). As noted in Sec. 1, the CWE is a community-driven catalog of over 900 weaknesses [3] that describe conditions in firmware, hardware, or software which may lead to vulnerabilities [4]. A weakness is not itself a vulnerability, but rather an underlying cause *e.g.*, “missing authentication”, “improper bounds check”, or “stack-based buffer overflow” [3]. The CWE is organized in a forest-like hierarchy, where categories become increasingly abstract toward the top. The abstraction levels are “Pillar”, “Class”, “Base”, and “Variant”, ranging from the most general to the most specific [4]. For practical purposes, “Base” and “Variant” weaknesses are typically used for classification. To improve consistency, CWE provides vulnerability mappings that define whether a weakness is *allowed*, *allowed-with-review*, *discouraged*, or *prohibited* for labeling. In addition, subsets of weaknesses are grouped into *Views* to support specific use cases [3]. In this work, we consider *CWE-1000* and *CWE-1003*. *CWE-1000* represents the full conceptual hierarchy of weaknesses, while *CWE-1003* offers a simplified view tailored for vulnerability categorization [6].

Automatic CWE classification. Previous work has explored automated CWE categorization using methods such as Naive Bayes [26], Boruta feature selection [12], BiGRU and TextCNN [27], BERT-based models [17, 30], and TF-IDF retrieval [11]. However, these approaches are limited by data sparsity: more than 800 out of the 881 valid CWEs remain rarely used following a logarithmic distribution. This results in an intense focus only on frequent CWEs [11, 12, 26, 32], and high overhead, low precision, and reduced accuracy on less common categories or larger subsets of the CWE list [9, 17, 22, 27].

Notably, ThreatZoom [9] employed a tree-like hierarchical pipeline of neural networks that leveraged the CWE structure. While it reported 92% accuracy on view *CWE-1003*, performance dropped to 75% on the full dataset with 364 CWEs. The approach also incurred high overhead and imprecision, since predictions were defined as paths along the tree rather than single CWEs. To reduce overhead and improve precision, Kota *et al.*, [22] divided view *CWE-1003* into top and bottom layers and applied BERT-based cross-encoder classifiers per layer. However, this design further reduced accuracy to 72.1% across 130 CWEs.

Table 1: Statistics of the dataset splits.

Sample Size	# Positive	# Negative	# CWEs	Samples in View 1003 (%)	CWEs in View 1003 (%)
Small	306	1457	45	98.04	86.67
Medium	1364	5055	114	98.31	88.60
Large	3410	12620	152	98.12	78.95

In contrast, our work does not emphasize additional training but instead expands the capabilities of pre-trained LLMs through RAG. By leveraging advanced retrieval and minimizing dependence on training data, we aim to address the imbalance between the abundance of CWE labels and the scarcity of real-world examples.

3 Methodology

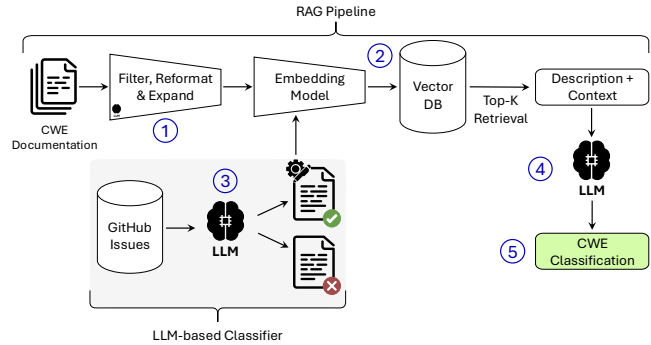
In this section, we present our novel RAG-based pipeline for automatic CWE classification. To the best of our knowledge, no prior work combines the semantic understanding of LLMs with RAG to accurately map detected vulnerabilities to CWE classes.

We first describe the construction of our CWE dataset, tailored for efficient retrieval in RAG. Next, we provide a detailed overview of the RAG pipeline. We show how this pipeline was integrated into the Cipollone *et al.* vulnerability detection framework, enabling both detection and classification of vulnerabilities and offering a fully automated solution with potential for production use.

Dataset creation. Due to the complexity of CWE assignments, rigorous data curation and filtering are necessary. To construct the dataset, we follow the methodology of Cipollone *et al.* [15]. Specifically, we collect CVEs from January 2019 to March 2025, retaining only those that reference GitHub issues and excluding those under examination or rejected. We focus on GitHub issues because, as noted in [15], it represents one of the most critical sources for defining and interpreting software vulnerabilities; nonetheless, our approach is general and can be applied to other textual sources as well. The remaining issues within the same repositories are used as negative examples. In addition, we discard issues exceeding 8,191 tokens, as this surpasses the context limit of the embedding model employed for the XGBoost classifier [15]. For each retained GitHub issue, we collect the textual description, relevant metadata (*e.g.*, creation date), and any associated code snippets.

We then enrich the dataset by assigning CWE labels to each CVE. The task of assigning CWE labels is handled by CVE Numbering Authorities (CNAs) [5], though disagreements often arise. To resolve such conflicts, we rely on the CNA’s acceptance level, which reflects its credibility. Acceptance levels are divided into four categories: “Reference”, “Contributor”, “Provider”, and “NVD” (*i.e.*, National Vulnerability Database). Among these, “Reference” represents the lowest level of credibility, while “Provider” and “NVD” are considered equally authoritative [7].

With this in mind, we apply a structured CWE filtering process. First, we discard all CVEs that contain no CWE information (*i.e.*, those labeled as “NVD-CWE-Other” or “NVD-CWE-noinfo” across all CNAs), since they are not assigned any valid CWE. For the remaining CVEs, we prioritize the labels assigned by CNAs whose acceptance level is “Provider” or “NVD”. If such labels are present, all others are discarded. When labels come from the source

**Figure 1: RAG pipeline.**

nvd@nist.gov, we prioritize those as well, even if their acceptance level is listed below “NVD”. This is because such entries are issued by NVD analysts and should carry NVD credibility, and we treat any discrepancy as human error. After this step, if all remaining CNAs agree on a single CWE label, we assign that label to the CVE; otherwise, we remove the CVE as we cannot reliably determine the correct classification. Since our study focuses on single-label categorization, we only keep CVEs with exactly one assigned CWE. However, this restriction is not overly limiting, as the vast majority of CVEs (around 97%) require only one CWE label.

At the beginning of the process, an initial inspection of the data indicated approximately 133,151 valid CVEs before any filtering was applied. After applying our structured CWE filtering process 112,759 CVEs remain. A large portion of these were discarded because they were not linked to GitHub issues. After this step, 6,443 CVEs remained, roughly half of which were further removed because they originated from repositories that were not sufficiently represented in the negative dataset. This left 3,675 CVEs, and after discarding CVEs associated with issues exceeding 8,191 tokens, 3,410 were retained.

From this dataset, we further construct three splits: (i) *Small*: contains GitHub issues from the top 50 repositories since 2023; (ii) *Medium*: includes 40% of all collected issues, with the remaining 60% reserved for XGBoost training; (iii) *Large*: comprises the entire dataset and is employed for retrieval and RAG evaluation, as no additional training is required. The detailed statistics of the three dataset splits are presented in Table 1.

RAG-based pipeline for CWE classification. We now present our novel pipeline for CWE classification, shown in Fig. 1.

We begin with ① the preparation of CWE documentation. For each CWE entry, we retain its key fields, *i.e.*, description, extended description, and examples. When any of these fields are missing, we use an LLM to complete them via few-shot prompting with up to three in-context examples. These examples are constructed by artificially masking the corresponding fields from other complete CWE entries. The resulting documentation is then split into coherent segments of up to 800 tokens, with segmentation aligned to sentence or field boundaries to preserve semantic integrity. Each segment is tagged with metadata linking it to its original CWE entry. This process ensures that the documentation is both comprehensive and retrievable with sufficient granularity. The processed

CWE documentation is ② stored in a FAISS (*i.e.*, Facebook AI Similarity Search) knowledge base [18].

With the knowledge base in place, GitHub issues are then introduced into the pipeline. An initial LLM ③ filters out issues that are not associated with vulnerabilities. For the remaining issues, the model generates a concise description of the related vulnerability, along with additional information that includes an extended description and a demonstrative example. This additional information is tailored for retrieval against CWE documentation rather than tied to the specific instance in the issue. This step uses few-shot prompting with three in-context examples: two positive examples derived from CWE entries, containing correct vulnerability descriptions and demonstrative examples used as ground truth, and one negative example corresponding to a non-vulnerability issue. The resulting vulnerability information is then embedded using BAAI/bge-large-en-v1.5 [13] and compared against the knowledge base to retrieve the top- K candidate CWE entries. We selected BAAI/bge-large-en-v1.5 because it is open source and achieved the strongest performance in our preliminary evaluation.

Finally, the retrieved CWE candidates and their documentation are combined with the previously generated vulnerability description and ④ passed to an LLM using zero-shot prompting. The model reasons over this information to ⑤ determine the most suitable CWE label, without the need for task-specific examples. We observed that performance improves when additional context is provided; therefore, we supplement the retrieved chunks with the full original CWE documentation, beyond what is stored in the FAISS knowledge base. However, in the interest of fairness, we exclude documentation that may directly link back to any specific CVE.

The full combined pipeline. By integrating our RAG pipeline with the Cipollone *et al.* Transformer-based vulnerability detection solution, we obtain the complete pipeline shown in Fig. 2. In this configuration, each GitHub issue is first embedded and classified using the XGBoost model. Specifically, issues are mapped into a continuous vector space, enabling the capture of semantic patterns that may not be evident from the raw text alone. These embeddings serve as input features for the classifier trained to output a binary

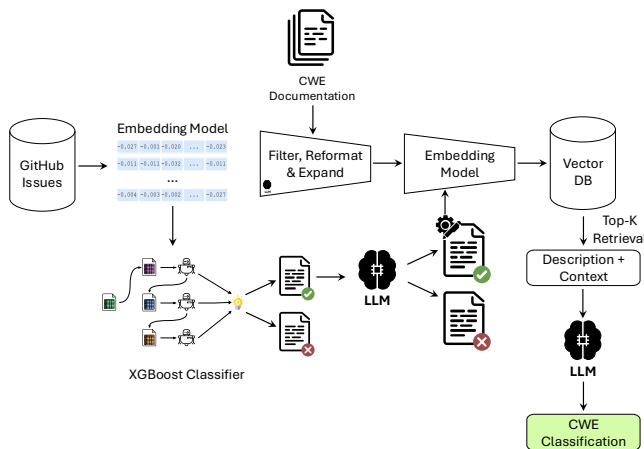


Figure 2: Full combined pipeline.

Table 2: Confusion matrix of the LLM vulnerability classifier.

Ground Truth	Prediction	
	Negative	Positive
Negative	11463 (TN)	1157 (FP)
Positive	263 (FN)	3147 (TP)

decision indicating whether a GitHub issue is related to a vulnerability. For issues predicted as vulnerabilities, an LLM is prompted to further filter potential false positives. If the issue is confirmed as a vulnerability, the LLM generates a description along with the additional information optimized for retrieval. This enriched textual representation is then passed to the RAG pipeline, which performs CWE categorization as described earlier.

4 Evaluation

In this section, we provide a detailed evaluation of the proposed solution described in Sec. 3. Specifically, we aim to answer the following questions:

- Q1 Can prompting strategies or documentation-generation techniques improve retrieval effectiveness for vulnerability categorization?
- Q2 Does the proposed pipeline outperform pure retrieval-based or pure LLM-based approaches for CWE classification?
- Q3 How well does the pipeline adapt to different retrieval systems?
- Q4 How does the proposed approach compare with current state-of-the-art solutions?

Evaluation setup. Since retrieval is the main bottleneck in RAG performance, we first identify the most effective retrieval strategy before evaluating the complete RAG pipeline and the full combined system. For retrieval evaluation, we measure top- K accuracy with $K \in \{1, 5, 10, 15, 20\}$. A sample is considered correctly categorized if the correct CWE appears among the top- K retrieved candidates, regardless of its position. In the evaluation of the RAG and combined pipelines, K represents the number of CWE candidates retrieved prior to the reasoning step, where the model selects a single final category. Consequently, all RAG and combined pipeline results correspond to top-1 accuracy, evaluated over different top- K retrieval settings. For completeness, all experiments are conducted on both views *CWE-1000* and *CWE-1003*.

In our experiments, we focus exclusively on true positives, *i.e.*, GitHub issues identified as actual vulnerabilities. The remaining samples are excluded since they lack meaningful vulnerability descriptions required for retrieval. As a result, a small portion of positive samples and consequently some CWEs were lost due to false negatives produced by the LLM filter. However, as long as the dataset remains sufficiently large, the reduced number of CWEs

Table 3: True positive sample statistics.

Sample Size	# Samples	# CWEs	Samples in View 1003 (%)	CWEs in View 1003 (%)
Small	296	44	97.97	86.36
Medium	1264	109	98.42	89.91
Large	3147	148	98.16	79.73

Table 4: Results of the retrieval-only experiments.

(a) Basic retrieval.							(b) Enhanced description.							(c) Enhanced description and documentation.						
K	CWE View	Class	Accuracy	Precision	Recall	F1-score	K	CWE View	Class	Accuracy	Precision	Recall	F1-score	K	CWE View	Sample Size	Accuracy	Precision	Recall	F1-score
1	1000	Light	0.1926	0.4640	0.1926	0.2303	1	1000	Light	0.4324	0.6469	0.4324	0.4619	1	1000	Small	0.4122	0.5885	0.4122	0.4266
		Normal	0.1520	0.5038	0.1520	0.1733			Normal	0.3007	0.4904	0.3007	0.3111			Large	0.4480	0.5836	0.4480	0.4624
		Heavy	0.1993	0.5566	0.1993	0.2320			Heavy	0.3209	0.6797	0.3209	0.3334			Small	0.5507	0.6703	0.5507	0.5523
	1003	Light	0.3547	0.5740	0.3547	0.3817		1003	Light	0.5000	0.6244	0.5000	0.4884		1003	Large	0.5558	0.6372	0.5558	0.5556
		Normal	0.2601	0.6297	0.2601	0.2889			Normal	0.5000	0.6868	0.5000	0.5062			Small	0.6453	0.8431	0.6453	0.7052
		Heavy	0.3209	0.6200	0.3209	0.3551			Heavy	0.5101	0.6639	0.5101	0.5018			Large	0.6079	0.8431	0.6079	0.6677
5	1000	Light	0.3986	0.8293	0.3986	0.4674	5	1000	Light	0.5709	0.8369	0.5709	0.5854	5	1000	Small	0.8074	0.8867	0.8074	0.8312
		Normal	0.4527	0.8464	0.4527	0.5077			Normal	0.5439	0.8627	0.5439	0.5866			Large	0.7718	0.8472	0.7718	0.7857
		Heavy	0.4595	0.8017	0.4595	0.4965			Heavy	0.5439	0.8438	0.5439	0.5766			Small	0.7500	0.9058	0.7500	0.7933
	1003	Light	0.5676	0.8770	0.5676	0.6308		1003	Light	0.6419	0.8963	0.6419	0.6787		1003	Large	0.6870	0.8732	0.6870	0.7368
		Normal	0.5541	0.8376	0.5541	0.6162			Normal	0.6453	0.8998	0.6453	0.6986			Small	0.8514	0.8914	0.8514	0.8644
		Heavy	0.5709	0.8556	0.5709	0.5994			Heavy	0.6149	0.9048	0.6149	0.6372			Large	0.8411	0.9040	0.8411	0.8508
10	1000	Light	0.4730	0.8392	0.4730	0.5289	10	1000	Light	0.6047	0.9049	0.6047	0.6425	10	1000	Small	0.7872	0.9298	0.7872	0.8359
		Normal	0.5304	0.8824	0.5304	0.6011			Normal	0.5980	0.9329	0.5980	0.6478			Large	0.7318	0.9352	0.7318	0.7855
		Heavy	0.5405	0.8973	0.5405	0.5875			Heavy	0.6081	0.9386	0.6081	0.6547			Small	0.8953	0.9042	0.8953	0.8948
	1003	Light	0.7162	0.9150	0.7162	0.7865		1003	Light	0.8209	0.9074	0.8209	0.8535		1003	Large	0.8700	0.9311	0.8700	0.8827
		Normal	0.7365	0.9163	0.7365	0.7906			Normal	0.8277	0.9360	0.8277	0.8704			Small	0.8243	0.9422	0.8243	0.8714
		Heavy	0.7230	0.8827	0.7230	0.7623			Heavy	0.6858	0.9501	0.6858	0.7448			Large	0.7703	0.9357	0.7703	0.8133
15	1000	Light	0.5372	0.8423	0.5372	0.5816	15	1000	Light	0.6385	0.9213	0.6385	0.6861	15	1000	Small	0.9223	0.9411	0.9223	0.9261
		Normal	0.5946	0.9201	0.5946	0.6724			Normal	0.6318	0.9544	0.6318	0.6846			Large	0.8881	0.9528	0.8881	0.9060
		Heavy	0.5845	0.9022	0.5845	0.6393			Heavy	0.6385	0.9662	0.6385	0.6877			Small	0.8243	0.9422	0.8243	0.8714
	1003	Light	0.8243	0.9144	0.8243	0.8615		1003	Light	0.8581	0.9228	0.8581	0.8835		1003	Large	0.7703	0.9357	0.7703	0.8133
		Normal	0.7872	0.9403	0.7872	0.8337			Normal	0.8649	0.9521	0.8649	0.9017			Small	0.9223	0.9411	0.9223	0.9261
		Heavy	0.7905	0.9214	0.7905	0.8238			Heavy	0.8142	0.9552	0.8142	0.8691			Large	0.8881	0.9528	0.8881	0.9060
20	1000	Light	0.5777	0.8851	0.5777	0.6507	20	1000	Light	0.6858	0.9183	0.6858	0.7440	20	1000	Small	0.8243	0.9422	0.8243	0.8714
		Normal	0.6419	0.9182	0.6419	0.7089			Normal	0.6757	0.9544	0.6757	0.7415			Large	0.7703	0.9357	0.7703	0.8133
		Heavy	0.6284	0.9253	0.6284	0.6940			Heavy	0.6622	0.9730	0.6622	0.7105			Small	0.9223	0.9411	0.9223	0.9261
	1003	Light	0.8581	0.9396	0.8581	0.8933		1003	Light	0.8818	0.9184	0.8818	0.8938		1003	Large	0.8881	0.9528	0.8881	0.9060
		Normal	0.8243	0.9480	0.8243	0.8659			Normal	0.8851	0.9487	0.8851	0.9125			Small	0.9223	0.9411	0.9223	0.9261
		Heavy	0.8243	0.9380	0.8243	0.8580			Heavy	0.8784	0.9490	0.8784	0.9090			Large	0.8881	0.9528	0.8881	0.9060

should not significantly affect the reliability of our evaluation. Table 2 reports the confusion matrix of the LLM-based vulnerability classifier, while the statistics of the true positive dataset are presented in Table 3.

Baseline. To assess the effectiveness of RAG in addressing Q2, we first establish a baseline for comparison. This baseline is a simple approach in which the entire CWE documentation is provided to an LLM (*i.e.*, GPT-4o mini) together with the vulnerability description. For view *CWE-1000*, this amounts to approximately 2.5 million tokens, which must be divided into chunks of 100,000 tokens to fit within the model’s context window. Each chunk is split between CWE entries to avoid fragmenting them, and the LLM is asked to identify which CWEs, if any, correspond to the given vulnerability description. The CWEs identified across all chunks are then aggregated and presented to the LLM in a final prompt to determine the most appropriate category.

Metrics. We utilize standard performance metrics including accuracy, precision, recall, and F1-score. Given the high data sparsity, we report the weighted average for each metric, which accounts for class imbalance by averaging metric values across classes according to their support.

Q1: Enhanced prompting and expanded CWE documentation helps the categorization. We first examine how different prompting strategies and knowledge base formats affect retrieval accuracy. To assess so, the CWE documentation was converted into structured markdown text with three levels of descriptiveness: *Light*, *Normal*, and *Heavy*, each serving as a separate experimental knowledge base. Three experiments were conducted. The first used the LLM-generated vulnerability description from the Cipollone *et al.* pipeline for retrieval. The second expanded this description with structured and generalized vulnerability information to test the effect of prompt engineering, incorporating a description, an

extended description, and an illustrative example. The third experiment used an LLM to fill in missing fields in the CWE documentation, such as the extended description and example, to better align the knowledge base with the structured queries from the second experiment and improve retrieval robustness. This last experiment was conducted using the *Light* documentation format due to token limits of the LLM used for enrichment (*i.e.*, GPT-4o mini).

The results of the three experiments are presented in Tables 4a, 4b, and 4c. Tables 4a and 4b report results on the *Small* test set, while Table 4c includes evaluations on both the *Small* and *Large* splits to enable comparison and more reliable assessment. Comparing Tables 4a and 4b, we observe that enhanced prompting, where the LLM produces generalized vulnerability descriptions optimized for retrieval, consistently improves performance, particularly for smaller values of K . Further extending the CWE documentation amplifies this effect. As shown in Table 4c, enriching the knowledge base allows retrieval accuracy to surpass 80% for higher K values. Across Tables 4a and 4b, the level of documentation descriptiveness (*i.e.*, *Light*, *Normal*, or *Heavy*) has little impact on overall performance and varies slightly with K . Given the negligible performance difference and the substantially lower token cost, we adopt the *Light* documentation for the experiments described in Q2.

Q2: RAG-based approaches outperform LLMs approaches. After evaluating different retrieval strategies from the knowledge base, we compare the performance of our RAG-based solution with the baseline approach for CWE classification. In the baseline, the LLM is directly prompted to classify a vulnerability using the description generated at the end of the Cipollone *et al.* pipeline, while the RAG approach follows the design illustrated in Fig. 1. Recall that in this setup different values of top- K are used for retrieval, and the final LLM selects one CWE class among the K retrieved candidates. We observe in Table 5 that our RAG solution consistently outperforms the baseline across all evaluated metrics. Comparing the results

of pure retrieval in Table 4c and the baseline in Table 5a, even at $K = 1$, RAG significantly outperforms the baseline on view *CWE-1003* and is within just two percentage points of the baseline on view *CWE-1000*. The efficiency gain is also substantial: the baseline required about \$55 and roughly 24 hours of inference time for view *CWE-1000*, whereas the RAG setup completed in under 15 seconds at a cost below \$1.

Our results also highlight that the final reasoning stage of the LLM in the RAG pipeline yields only limited improvements in classification accuracy. As shown in Table 5b, accuracy increases by $\sim 5\%$ for view *CWE-1000* and $\sim 7\%$ for *CWE-1003*. Comparing the best large-split performance for *CWE-1000* with the corresponding retrieval accuracy at $K = 20$ in Table 4c, the LLM (*i.e.*, GPT-4o mini) correctly isolates the CWE label about 64.10% of the time. For *CWE-1003*, the corresponding isolation accuracy is about 66.38%. These results indicate that the performance gain in *CWE-1003* is primarily driven by improved retrieval rather than reasoning, consistent with the findings from Q1. This difference between *CWE-1000* and *CWE-1003* can be explained by the class distribution. In *CWE-1003*, retrieval performance is already strong even at low K , allowing the LLM reasoning stage to perform efficiently with fewer candidates. In contrast, the broader class space of *CWE-1000* makes retrieval the main bottleneck at smaller K values, requiring more candidates to achieve comparable accuracy. As expected, our experiments consistently show higher performance on view *CWE-1003*. As explained in Sec. 3, 97% of our samples belong to CWEs included in this view, which contains 130 categories compared to the much larger set of 881 in view *CWE-1000*.

Finally, we conduct a full evaluation using the combined pipeline illustrated in Fig. 2. In this setup, the XGBoost classifier is first used for vulnerability detection. For GitHub issues identified as vulnerabilities, the LLM then generates an enhanced output that includes both a description and an illustrative example, as discussed in Q1. This detailed description is then passed to the RAG pipeline for CWE classification using the best-performing configuration (reported in Table 5b). The final results are presented in Table 5c. These results should be interpreted in the context of Tables 4c and 5b, as much of the reported accuracy is influenced by the negative samples filtered by Cipollone’s pipeline. Nonetheless, the best end-to-end accuracy remains below Cipollone’s reported 96% detection accuracy, highlighting that CWE categorization is a considerably more challenging task than vulnerability detection alone.

Q3: Vector retrieval outperforms more complex methods.

We built our RAG-based pipeline using a straightforward vector retrieval approach implemented with FAISS. However, we also explored whether more advanced retrieval systems could offer improved performance over this simple baseline.

To investigate this, we evaluated two alternatives: fusion retrieval [21] and a RAPTOR-inspired hierarchical traversal [28]. Fusion retrieval combines the semantic understanding of vector-based retrieval with the keyword-matching precision of BM25 [16]. For each query, we compute similarity scores from both retrieval types, normalize them, and combine them through a weighted sum to obtain the final relevance score. In contrast, RAPTOR builds hierarchical abstraction trees through iterative LLM summarization to enable structured retrieval. Instead of constructing such trees, we

Table 5: Results of the pipeline experiments.

(a) Baseline.						
CWE View	Accuracy	Precision	Recall	F1-Score		
1000	0.4628	0.6278	0.4628	0.4885		
1003	0.4358	0.5951	0.4358	0.4628		

(b) RAG pipeline experiment.						
K	CWE View	Sample size	Accuracy	Precision	Recall	F1-score
3	1000	Small	0.4764	0.6805	0.4764	0.5063
		Large	0.4779	0.6339	0.4779	0.5074
	1003	Small	0.6115	0.7011	0.6115	0.6208
		Large	0.6225	0.6688	0.6225	0.6255
5	1000	Small	0.4865	0.6355	0.4865	0.5060
		Large	0.4827	0.6681	0.4827	0.5093
	1003	Small	0.6047	0.7387	0.6047	0.6266
		Large	0.6012	0.6791	0.6012	0.6141
10	1000	Small	0.4831	0.6992	0.4831	0.5079
		Large	0.4881	0.6649	0.4881	0.5187
	1003	Small	0.5912	0.6955	0.5912	0.6058
		Large	0.5904	0.6675	0.5904	0.6047
15	1000	Small	0.4831	0.7074	0.4831	0.5100
		Large	0.4909	0.6567	0.4909	0.5169
	1003	Small	0.5709	0.6765	0.5709	0.5834
		Large	0.5917	0.6570	0.5917	0.6004
20	1000	Small	0.4932	0.6677	0.4932	0.5115
		Large	0.4938	0.6697	0.4938	0.5161
	1003	Small	0.5811	0.6517	0.5811	0.5850
		Large	0.5895	0.6583	0.5895	0.5983

(c) Full combined pipeline experiment.						
K	CWE View	Sample size	Accuracy	Precision	Recall	F1-score
3	1000	Small	0.8145	0.8638	0.8145	0.8337
		Medium	0.8450	0.8744	0.8450	0.8528
	1003	Small	0.8361	0.8694	0.8361	0.8488
		Medium	0.8696	0.8796	0.8696	0.8718
5	1000	Small	0.8151	0.8626	0.8151	0.8341
		Medium	0.8453	0.8693	0.8453	0.8521
	1003	Small	0.8395	0.8734	0.8395	0.8527
		Medium	0.8702	0.8813	0.8702	0.8730
10	1000	Small	0.8174	0.8655	0.8174	0.8369
		Medium	0.8439	0.8663	0.8439	0.8509
	1003	Small	0.8395	0.8727	0.8395	0.8514
		Medium	0.8727	0.8816	0.8727	0.8734
15	1000	Small	0.8162	0.8670	0.8162	0.8360
		Medium	0.8445	0.8769	0.8445	0.8520
	1003	Small	0.8383	0.8673	0.8383	0.8490
		Medium	0.8740	0.8819	0.8740	0.8738
20	1000	Small	0.8140	0.8587	0.8140	0.8328
		Medium	0.8448	0.8779	0.8448	0.8525
	1003	Small	0.8372	0.8664	0.8372	0.8478
		Medium	0.8704	0.8805	0.8704	0.8716

leverage the existing hierarchy in the CWE taxonomy as a natural top-down structure. This allows us to efficiently traverse CWE categories and surface increasingly specific candidates without additional clustering or summarization steps.

Across all tested configurations, both approaches consistently underperformed simple vector retrieval by 5–20 percentage points relative to the results in Table 4c. Moreover, the RAPTOR-inspired method retrieved up to twice as many documents and incurred at least double the latency, making both approaches impractical for this task. These findings align with observations from [28], which reported that the collapsed tree structure achieved better performance than hierarchical traversal. The collapsed tree considers all

nodes simultaneously by merging them into a single retrieval space and selecting the top- K results. Under our RAPTOR-inspired setup, this process effectively reduces to standard vector retrieval, since the CWE hierarchy already provides an implicit tree structure.

A possible extension would be to generate vulnerability descriptions at multiple abstraction levels consistent with the CWE hierarchy. However, this approach would significantly increase computational and token costs, while offering limited expected gains. Consequently, both fusion and RAPTOR-style retrieval are considered unsuitable for CWE categorization in our setting.

Q4: The RAG-assisted pipeline achieves competitive performance compared to prior state-of-the-art solutions. We now discuss how our RAG-based pipeline compares with current state-of-the-art approaches for CWE classification. Compared to previous work, our RAG-based approach achieves competitive results despite operating entirely in a zero-shot setting without fine-tuning or task-specific training. ThreatZoom [9] reports $\sim 75\%$ accuracy on the MITRE benchmark, which includes fewer than half the classes in view *CWE-1000*. Achieving approximately 50% accuracy in our setup across a much larger label space demonstrates strong generalization capabilities. Similarly, Kota *et al.*, [22] report 72% accuracy on view *CWE-1003*, while our best RAG configuration reaches about 62% under zero-shot conditions.

From a practical perspective, pure top- K retrieval already shows strong potential for supporting vulnerability categorization. Our approach narrows the candidate space from roughly 900 CWEs to 20 in *CWE-1000* (or from 130 to 5 in *CWE-1003*) with around 80% accuracy, significantly reducing the workload for analysts. Previous state-of-the-art methods, such as Das *et al.*, [17], achieved only 14.9%, 44.0%, and 67.5% top- K accuracy for $K = 1, 6,$ and 20, respectively, when evaluated on CWEs not included in their training data. Like ThreatZoom, their approach predicted paths rather than individual CWEs, which reduced precision. Even when restricted to CWEs seen during training, their top-1, top-6, and top-20 accuracies are 83.6%, 91.4%, and 94.4% respectively, which is comparable to our retrieval performance at higher values of K . Also, in practical use, even imperfect retrievals can still be valuable, as incorrect predictions often correspond to neighboring nodes in the CWE hierarchy, providing meaningful directional guidance for human analysts.

Most importantly, our RAG-based pipeline avoids any supervised training, which reduces bias toward specific CWEs and relies entirely on retrieval and the latent knowledge of the LLM. This also minimizes operational costs and allows the pipeline to be easily improved by integrating newer LLMs, while ad-hoc supervised approaches, though effective on specific datasets, typically require substantial retraining efforts to generalize.

5 Limitations & Future Work

Model limitations. The high computational cost of the baseline approach required the use of a cost-efficient model, *i.e.*, GPT-4o mini, which introduces several constraints on reasoning depth and output size. In particular, GPT-4o mini is limited to a maximum of 16,384 output tokens, which restricts the complexity and length of viable prompts and consequently limits prompt engineering flexibility. Future work should explore the use of cutting-edge models with

larger context and advanced reasoning capabilities (*e.g.*, GPT-5) to enable richer vulnerability descriptions and more comprehensive documentation generation.

Dataset inconsistencies and hierarchical evaluation. The NVD dataset contains several inconsistencies across CNAs, including conflicting annotations and occasional use of prohibited CWE labels. These issues introduce human error into the ground truth, meaning our pipeline may propose a more appropriate CWE classification than the official one. Involving domain experts in the review process could help resolve ambiguous cases and improve dataset quality. Moreover, incorporating the hierarchical nature of the CWE taxonomy into evaluation would offer a more nuanced performance metric than a flat accuracy score. For instance, classification accuracy could be weighted based on the distance or relationship between the predicted and correct CWEs within the abstraction tree. Similarly, incorporating semantic similarity between CWEs into the evaluation framework could provide a more informative and realistic measure of model performance.

Unexplored RAG enhancements. This work mainly focused on demonstrating the feasibility of automated CWE classification rather than exhaustively exploring all potential RAG enhancements. Several promising directions remain open for improvement. For example, fully implementing hierarchical RAG methods such as RAPTOR, which use clustering and recursive summarization to construct abstraction trees, could further improve performance. Although the CWE corpus already has a hierarchical structure, it is not always ideal, as higher-level entries may not accurately summarize their descendants. Future research could explore optimized hierarchical representations specifically designed for security reasoning and retrieval.

Pre-training and fine-tuning opportunities. Finally, due to the limitations of using a closed-weights model (*i.e.*, GPT-4o mini), we did not perform any pre-training. Moreover, since one of the main motivations for adopting RAG was to reduce dependence on the inherently sparse vulnerability data, we chose not to explore fine-tuning. But, we hypothesize that pre-training and fine-tuning may represent the most promising path toward achieving truly competitive performance with prior state-of-the-art methods. While data scarcity remains a challenge, the strong results obtained without any fine-tuning or extensive in-context learning suggest that this limitation can be addressed. Future work should therefore explore open-weights models that allow for end-to-end adaptation and domain-specific optimization within the RAG pipeline.

6 Conclusions

This paper assesses the feasibility and effectiveness of utilizing RAG and more broadly, knowledge retrieval, to support automated vulnerability categorization using the CWE database. We extend Cipollone *et al.* pipeline with a retrieval-driven component, creating a valuable tool for not only timely vulnerability detection but also precise categorization. This is an essential step toward understanding and addressing the underlying nature of identified vulnerabilities. Our results demonstrate that retrieval-based approaches provide a scalable and cost-efficient path to automating CWE classification, achieving competitive performance without

model training or fine-tuning. This highlights the strong potential of embedding-based retrieval in bridging the gap between vulnerability discovery, disclosure, and remediation guidance. Future research should investigate advanced RAG configurations, model fine-tuning strategies, and improved data quality pipelines to address dataset noise and scale toward fully automated security analysis.

Acknowledgments

This work has been partially supported by Vinnova (the Sweden's Innovation Agency), the Swedish Research Council (agreement No. 2021-04212), KTH Digital Futures, and the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

References

- [1] [n. d.]. *CVE: Common Vulnerabilities and Exposures*. <https://www.cve.org/About/Overview>
- [2] [n. d.]. *CVE: Common Vulnerabilities and Exposures*. <https://www.cve.org/About/History>
- [3] [n. d.]. *CWE - CVE → CWE Mapping "Root Cause Mapping" Guidance*. https://cwe.mitre.org/documents/cwe_usage/guidance.html
- [4] [n. d.]. *CWE - CWE Glossary*. <https://cwe.mitre.org/documents/glossary/index.html#Weakness>
- [5] [n. d.]. *NVD - CNAs and CVE Counting*. <https://nvd.nist.gov/general/cna-counting>
- [6] [n. d.]. *NVD - How We Assess Acceptance Levels*. <https://nvd.nist.gov/vuln/cvmap/How-We-Assess-Acceptance-Levels>
- [7] [n. d.]. *NVD - Understanding Acceptance Levels*. <https://nvd.nist.gov/vuln/cvmap/Understanding-Acceptance-Levels>
- [8] 2023. *What Is a Zero-Day Exploit?* | IBM. <https://www.ibm.com/think/topics/zero-day>
- [9] Ehsan Aghaei, Waseem Shadid, and Ehab Al-Shaer. 2020. ThreatZoom: Hierarchical Neural Network for CVEs to CWEs Classification. In *Security and Privacy in Communication Networks* (Cham, 2020), Noseong Park, Kun Sun, Sara Foresti, Kevin Butler, and Nitesh Saxena (Eds.). Springer International Publishing, 23–41. doi:10.1007/978-3-030-63086-7_2
- [10] Vishwanath Akuthota, Raghunandan Kasula, Sabiha Tasnim Sumona, Masud Mohiuddin, Md Tanzim Reza, and Md Mizanur Rahman. 2023. Vulnerability Detection and Monitoring Using LLM. In *2023 IEEE 9th International Women in Engineering (WIE) Conference on Electrical and Computer Engineering (WIECON-ECE)*, 309–314. doi:10.1109/WIECON-ECE60392.2023.10456393
- [11] Massimiliano Albanese, Olutola Adebisi, and Frank Onovae. 2024. CVE2CWE: Automated Mapping of Software Vulnerabilities to Weaknesses Based on CVE Descriptions. In *Proceedings of the 21st International Conference on Security and Cryptography* (Dijon, France, 2024), SCITEPRESS - Science and Technology Publications, 500–507. doi:10.5220/0012770400003767
- [12] Masaki Aota, Hideaki Kanehara, Masaki Kubo, Noboru Murata, Bo Sun, and Takeshi Takahashi. 2020. Automation of Vulnerability Classification from Its Description Using Machine Learning. In *2020 IEEE Symposium on Computers and Communications (ISCC) (2020-07)*, 1–7. doi:10.1109/ISCC50000.2020.9219568
- [13] Beijing Academy of Artificial Intelligence (BAAI). 2023. BAAI/bge-large-en-v1.5: an English sentence-embedding model. <https://huggingface.co/BAAI/bge-large-en-v1.5>.
- [14] Casey Charrier, James Sadowski, Clement Lecigne, and Vlad Stolyarov. 2025. Hello 0-Days, My Old Friend: A 2024 Zero-Day Exploitation Analysis. <https://cloud.google.com/blog/topics/threat-intelligence/2024-zero-day-trends>
- [15] Daniele Cipollone, Changjie Wang, Mariano Scazzariello, Simone Ferlin, Maliheh Izadi, Dejan Kostić, and Marco Chiesa. 2025. Automating the Detection of Code Vulnerabilities by Analyzing GitHub Issues. In *2025 IEEE/ACM International Workshop on Large Language Models for Code (LLM4Code) (2025-05)*, 41–48. doi:10.1109/LLM4Code66737.2025.00010
- [16] Fabio Crestani, Mounia Lalmas, Cornelis J. Van Rijsbergen, and Iain Campbell. 1998. "Is this document relevant?...probably": a survey of probabilistic models in information retrieval. *ACM Comput. Surv.* 30, 4 (Dec. 1998), 528–552. doi:10.1145/299917.299920
- [17] Siddhartha Shankar Das, Edoardo Serra, Mahantesh Halappanavar, Alex Potchen, and Ehab Al-Shaer. 2021. V2W-BERT: A Framework for Effective Hierarchical Multiclass Classification of Software Vulnerabilities. In *2021 IEEE 8th International Conference on Data Science and Advanced Analytics (DSAA) (2021-10)*, 1–12. doi:10.1109/DSAA53316.2021.9564227
- [18] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2025. The Faiss library. arXiv:2401.08281 [cs.LG] <https://arxiv.org/abs/2401.08281>
- [19] Angela Fan, Beliz Gokkaya, Mark Harman, Mitya Lyubarskiy, Shubho Sen-gupta, Shin Yoo, and Jie M. Zhang. 2023. Large Language Models for Software Engineering: Survey and Open Problems. arXiv:2310.03533 [cs.SE] <https://arxiv.org/abs/2310.03533>
- [20] Yuejun Guo, Constantinos Patsakis, Qiang Hu, Qiang Tang, and Fran Casino. 2024. Outside the Comfort Zone: Analysing LLM Capabilities in Software Vulnerability Detection. In *Computer Security – ESORICS 2024*, Joaquin Garcia-Alfaro, Rafal Kozik, Michal Choraś, and Sokratis Katsikas (Eds.). Springer Nature Switzerland, Cham, 271–289.
- [21] Quentin Herreros and Thomas Veasey. 2023. Improving Information Retrieval in the Elastic Stack: Hybrid Retrieval. <https://www.elastic.co/search-labs/blog/improving-information-retrieval-elastic-stack-hybrid>. Accessed: 2025-11-01.
- [22] Kethan Kota, Manjunatha A, and Sree Vivek S. 2024. CWE Prediction Using CVE Description - The Semantic Similarity Approach. *Procedia Computer Science* 235 (2024), 1167–1178. doi:10.1016/j.procs.2024.04.111
- [23] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems* (Vancouver, Canada, 2020), H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 9459–9474. https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf
- [24] Siyi Liu, Kishalay Halder, Zheng Qi, Wei Xiao, Nikolaos Pappas, Phu Mon Htut, Neha Anna John, Yassine Benajiba, and Dan Roth. 2025. Towards Long Context Hallucination Detection. arXiv:2504.19457 [cs.CL] <https://arxiv.org/abs/2504.19457>
- [25] Guilong Lu, Xiaolin Ju, Xiang Chen, Wenlong Pei, and Zhilong Cai. 2024. GRACE: Empowering LLM-based software vulnerability detection with graph structure and in-context learning. *Journal of Systems and Software* 212 (2024), 112031. doi:10.1016/j.jss.2024.112031
- [26] Sarang Na, Tae-eun Kim, and Hwankuk Kim. 2017. A Study on the Classification of Common Vulnerabilities and Exposures Using Naive Bayes. In *Advances on Broad-Band Wireless Computing, Communication and Applications* (Cham, 2017), Leonard Barolli, Fatos Xhafa, and Kangbin Yim (Eds.). Springer International Publishing, 657–662. doi:10.1007/978-3-319-49106-6_65
- [27] Mengyuan Pan, Po Wu, Yiwei Zou, Chong Ruan, and Tao Zhang. 2023. An Automatic Vulnerability Classification Framework Based on BiGRU-TextCNN. *Procedia Computer Science* 222 (2023), 377–386. doi:10.1016/j.procs.2023.08.176
- [28] Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D. Manning. 2024. RAPTOR: Recursive Abstractive Processing for Tree-Organized Retrieval. In *The Twelfth International Conference on Learning Representations* (2024-01-31). arXiv:2401.18059 [cs] doi:10.48550/arXiv.2401.18059
- [29] Clemens Sauerwein, Christian Sillaber, and Ruth Breu. 2018. Shadow Cyber Threat Intelligence and Its Use in Information Security and Risk Management Processes.
- [30] Haifeng Wang, Jiwei Li, Hua Wu, Eduard Hovy, and Yu Sun. 2023. Pre-Trained Language Models and Their Applications. *Engineering* 25 (2023), 51–65. doi:10.1016/j.eng.2022.04.024
- [31] Jin Wang, Zishan Huang, Hengli Liu, Nianyi Yang, and Yin hao Xiao. 2023. DefectHunter: A Novel LLM-Driven Boosted-Conformer-based Code Vulnerability Detection Mechanism. arXiv:2309.15324 [cs.CR] <https://arxiv.org/abs/2309.15324>
- [32] Tianyi Wang, Shengzhi Qin, and Kam Pui Chow. 2021. Towards Vulnerability Types Classification Using Pure Self-Attention: A Common Weakness Enumeration Based Approach. In *2021 IEEE 24th International Conference on Computational Science and Engineering (CSE) (2021-10)*, 146–153. doi:10.1109/CSE53436.2021.00030