Key Aspect Prediction for Silent Vulnerability Fixes via Semantic Augmentation

Dongshun He[1], Linyi Han[1] and Xiaowang Zhang[1,*]

[1]*College of Intelligence and Computing, Tianjin University, Tianjin, 300350, China*

Abstract

Silent vulnerability fixes pose significant risks to downstream open-source software (OSS) users, as the lack of vulnerability details in fix patches leaves users unaware of potential threats. Previous work predicts the key aspects of vulnerability fixes using an encoder-decoder model to aid users in understanding these fixes. However, their approach overlooks the limited expressiveness of commit messages and the varied intents underlying code changes. In this poster, we propose a semantic-augmented method for key aspect prediction in silent vulnerability fixes. Firstly, we enrich commit semantics by incorporating information from multiple external sources. Then, we design a Chain-of-Thought (CoT) prompt to analyze code semantics at the hunk level and identify security-relevant changes. Finally, we design a task-specific embedding method to represent code diffs and retrieve semantically similar commits, guiding large language models (LLMs) to predict the vulnerability type, root cause, impact, and attack vector. Experiments on our constructed dataset demonstrate that our method outperforms baselines in key aspect prediction across ROUGE-L and METEOR.

Keywords

Silent Vulnerability Fixes, Key Aspect Prediction, Semantic Augmentation, Large Language Model

1. Introduction

The widespread adoption of open-source software (OSS) has heightened security concerns, with disclosed Common Vulnerabilities and Exposures (CVE) records rising from 7937 in 2014 to 39974 in 2024, according to the National Vulnerability Database (NVD) [1]. To mitigate these vulnerabilities, developers submit security patches to repositories. Coordinated Vulnerability Disclosure (CVD) is a widely used vulnerability disclosure model that requires vulnerabilities to be silently fixed [2]. This means that the details of the vulnerability must not be revealed in the patches before the vulnerability is publicly disclosed. In this context, downstream OSS users face a high risk due to their lack of awareness about silently fixed vulnerabilities.

Prior work has focused on identifying silent vulnerability fixes [3] [4], but such methods suffer from high false positive rates, placing the burden on OSS users to manually interpret AI-generated predictions. To address this limitation, Sun et al. [5] propose an encoder-decoder model that generates the key aspects of vulnerability fixes—namely, the vulnerability type, root cause, impact, and attack vector—from commit messages and code diffs, enabling users to better understand vulnerability information. However, their method overlooks the limitations of directly using commit messages and code diffs as input for key aspect prediction. Specifically, commit messages in silent vulnerability fixes are often brief and lack meaningful semantic information. For example, the message in commit:01c61f8 is simply "Proper fix for #248," which offers no insight into the vulnerability's nature, cause, or impact. Such sparse textual context makes it difficult for models to infer the underlying security issues. In addition, code diffs in a commit may contain multiple changes serving diverse purposes—such as feature updates, code refactoring, or style adjustments—many of which are unrelated to vulnerability fixing. Directly using all code diffs introduces semantic noise, which undermines the accuracy of key aspect prediction.

To address these challenges, we propose a semantic-augmented method for key aspect prediction in silent vulnerability fixes. Firstly, our approach enriches the limited commit message semantics by

*Corresponding author.

hds@tju.edu.cn (D. He); hanly2@tju.edu.cn (L. Han); xiaowangzhang@tju.edu.cn (X. Zhang)

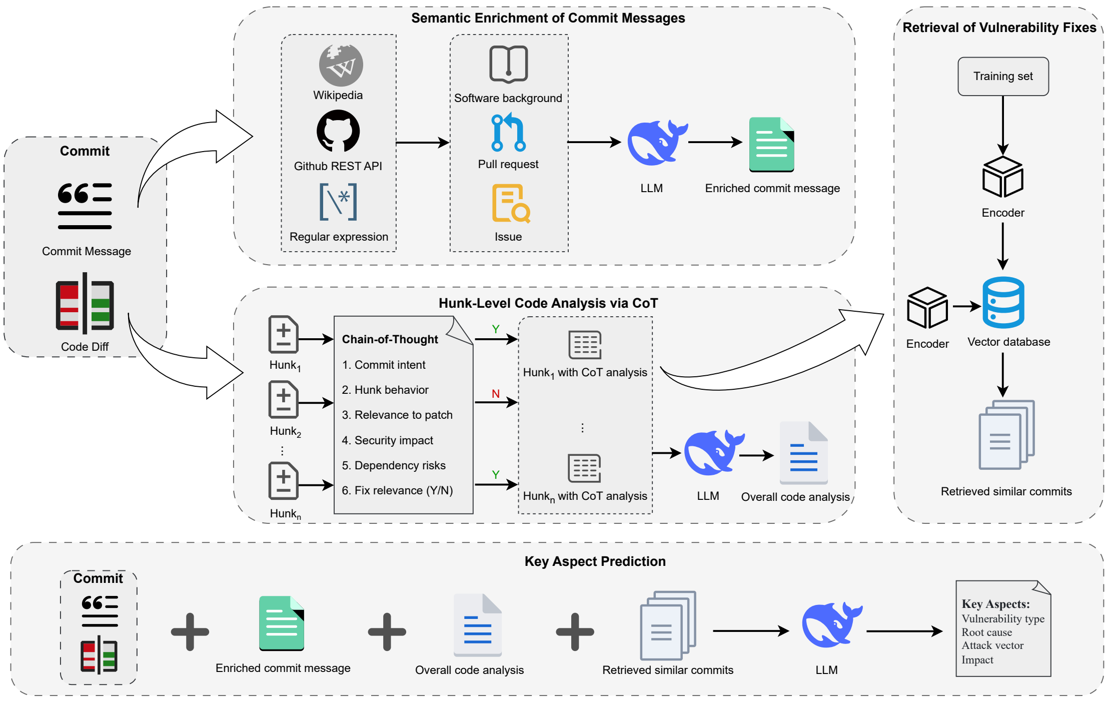

Figure 1: Overview of the proposed semantic-augmented framework for key aspect prediction.

incorporating information from multiple external sources. Then, we design a CoT prompt to analyze code semantics at the hunk level, enabling the identification of security-relevant changes within mixed-purpose commits. Finally, we design a task-specific embedding method to represent complex code diffs and retrieve semantically similar commits. These retrieved examples are combined with enriched commit messages and code diff analysis to guide LLMs in predicting the vulnerability type, root cause, impact, and attack vector.

We construct a large-scale dataset comprising 10,912 vulnerability-fixing commits and their corresponding key aspects, covering 3,575 open-source projects and 9,586 CVE records. This dataset is available at https://doi.org/10.6084/m9.figshare.29693216.v1.

Experimental results on our constructed dataset show that the proposed method outperforms baselines in key aspect prediction, achieving higher scores on ROUGE-L and METEOR.

2. Approach

Our task is to predict four key aspects of a vulnerability—its type, root cause, impact, and attack vector—given a vulnerability-fixing commit that includes a commit message and code diff. In this section, we present our semantic-augmented prediction framework, which consists of three components: (1) enriching the semantics of commit messages using external sources, (2) analyzing code diffs at the hunk level via CoT prompting, and (3) retrieving semantically similar commits with task-specific embedding method. The results from all three components are integrated to guide LLMs in predicting the vulnerability aspects.The overall framework is illustrated in Figure 1.

2.1. Semantic Enrichment of Commit Messages

Commit messages in silent vulnerability fixes are often terse and uninformative, making it difficult to infer the underlying security context. However, we observe that many such messages include references to external resources, such as issue identifiers (e.g., "fix #248"). These references can be leveraged to retrieve richer contextual information. We use regular expressions to extract issue numbers from commit messages and query the corresponding GitHub issues using the GitHub REST API. In addition, we obtain the content of the pull request (PR) associated with the commit, which often contains more detailed descriptions of the code changes. To further enrich the semantic context, we retrieve the software project's Wikipedia page as a source of background knowledge. This is because key aspects often contain software-specific features [6]. Since not all commits are associated with PRs, issues, or relevant Wikipedia pages, our method simply leverages the remaining available sources when part of this external context is missing. As the issue, PR, and Wikipedia content can be lengthy, we employ an LLM to summarize each source into a vulnerability-relevant abstract. These summaries are then concatenated with the original commit message to form an enriched commit message.

2.2. Hunk-Level Code Analysis via CoT

Commits often include multiple code changes that serve different purposes, such as feature updates, refactoring, or formatting, which may not be related to vulnerability fixes. To isolate the security-relevant portions of a commit, we analyze the code diff at the hunk level, treating each hunk as a standalone semantic unit. We design a CoT prompting strategy to evaluate each hunk. For each hunk, the CoT prompt guides the LLM through six sequential steps: identifying the commit's overall intent, summarizing the hunk's behavior, evaluating its alignment with the commit's purpose, assessing its security implications, analyzing potential dependency risks, and determining whether the hunk is related to a vulnerability fix. We apply this process to all hunks within a code diff. For those identified as vulnerability-related, we collect their corresponding CoT analysis results and use an LLM to generate an overall analysis of the security-relevant code changes. This enriched, hunk-filtered code diff analysis is then provided to the final prediction stage alongside the enriched commit message.

2.3. Task-Specific Retrieval of Vulnerability Fixes

To better leverage the in-context learning capabilities of LLMs, we adopt a retrieval-augmented few-shot prompting approach. For each commit, we retrieve top-k similar vulnerability-fixing commits from the training set as reference examples. We construct a vector database using the commits in the training set, where each commit is represented by an embedding that captures its security-relevant code semantics. Specifically, we use only the code hunks identified as vulnerability-related by the method described in Section 2.2. Within each hunk, we exclude unchanged lines and retain only the added and deleted lines, as these are the primary carriers of vulnerability semantics. To encode the added and deleted lines, we employ the CodeT5+ 110m embedding model [7]. Since the two types of lines originate from different versions of the code, we embed them separately into two 256-dimensional vectors. These vectors are then concatenated to form a 512-dimensional representation for each commit. The resulting embeddings are stored in a vector database. During inference, we compute cosine similarity between the test commit and all training samples to retrieve the top-k most relevant examples.

Finally, we combine the original commit with the enriched commit message, the overall code analysis, and the retrieved similar commits. These components are concatenated to form the input prompt, which is then fed into a LLM to generate the four key aspects.

3. Experiments

We construct a large-scale dataset of vulnerability-fixing commits to evaluate the effectiveness of our method. We first collect all CVE entries from the NVD up to April 9, 2025. Each entry includes a CVE ID, a textual vulnerability description (TVD), and a set of external references. For OSS projects hosted on GitHub, the URL of a vulnerability-fixing commit typically follows the format: *https://github.com/owner/repo/commit/commit_hash*. Since only a small subset of CVE entries includes such URLs, we use regular expressions to extract entries containing GitHub commit links and and then obtain the corresponding patch files, from which we extract commit messages and code diffs. We filter out commits with code diffs exceeding 2,000 tokens to keep the dataset balanced and representative of common vulnerability fixes. Each CVE's TVD contains information related to the key aspects of the vulnerability. We use DeepSeek V3 [8] with prompts to extract four aspects from the TVD: vulnerability type, root cause, attack vector, and impact. As TVDs may lack certain aspects, we additionally retrieve corresponding TVDs from the IBM X-Force vulnerability database and apply the same extraction process. Commits are discarded if any key aspect remains unavailable after both stages. As a result, we obtain a dataset of 10,912 vulnerability-fixing commits and their corresponding key aspects, covering 3,575 open-source projects and 9,586 CVE entries. We randomly divide the dataset into 80% for training and 20% for testing.

We compare our method against two baselines: the CodeBERT-based encoder-decoder model proposed by Sun et al. [5], and DeepSeek V3 in both zero-shot and 8-shot settings. Our method also uses DeepSeek V3 as the backbone LLM, but differs in how the input is structured and enhanced through semantic augmentation and retrieval. We evaluate all methods using ROUGE-L and METEOR, which are widely used metrics for text generation tasks. For few-shot settings, we retrieve 8 examples using the method described in Section 2.3 and include them in the prompt.

Table 1
Comparison with baselines and ablation variants on key aspect prediction.

Model	Vulnerability Type		Root Cause		Attack Vector		Impact	
	Rouge-L	METEOR	Rouge-L	METEOR	Rouge-L	METEOR	Rouge-L	METEOR
CodeBERT-based	42.63	36.50	14.58	11.43	23.82	16.79	24.23	21.92
DeepSeek (0-shot)	35.26	29.80	11.57	10.84	22.30	16.74	18.40	17.36
DeepSeek (8-shot)	41.72	35.78	14.01	12.42	24.21	18.25	18.71	20.32
Ours	**45.31**	**38.50**	14.69	12.62	**30.90**	**24.51**	**29.41**	**30.25**
Ours w/o Message Enrichment	43.15	36.48	14.64	12.64	29.77	19.44	26.52	24.32
Ours w/o Code Analysis	43.41	34.68	**17.41**	**14.70**	30.57	20.90	28.06	25.48
Ours w/o Retrieval	38.49	31.68	12.64	10.88	25.20	19.34	18.99	19.87

The experimental results are shown in Table 1. Our method outperforms all baselines across all four key aspects under both ROUGE-L and METEOR metrics. For root cause, however, the overall scores remain relatively low across all methods. This is mainly because root cause is strongly tied to specific vulnerabilities and can be expressed in diverse ways. The current metrics emphasize textual overlap rather than semantic similarity, which makes it difficult to capture the underlying consistency in this aspect. We also observe that the variant without message enrichment achieves slightly better performance on root cause prediction. The reason may be that additional external information introduces expressions that diverge from the annotated labels, thereby reducing alignment under text-similarity metrics. Even so, our full model still yields the best overall performance across all aspects. The ablation study further demonstrates that each component—commit message enrichment, code diff analysis, and example retrieval—contributes meaningfully to the final result, with retrieval having the most significant impact.

4. Limitations

Our method requires several steps of interaction with LLMs, which increases computational cost. Future work will focus on exploring optimization strategies to reduce resource consumption and improve scalability. In addition, the evaluation is currently limited to ROUGE-L and METEOR, which measure textual similarity rather than semantic correctness. Future studies will incorporate expert annotation and domain-specific metrics to provide a more reliable assessment of practical utility.

5. Conclusion

In this poster, we propose a semantic-augmented framework for predicting key aspects of silent vulnerability fixes, including the vulnerability type, root cause, impact, and attack vector. Our approach enhances commit understanding through external knowledge, hunk-level code analysis, and retrieval-based few-shot prompting. Experiments on a large-scale dataset demonstrate that our method outperforms baselines across all key aspects.

Declaration on Generative AI

During the preparation of this work, we used ChatGPT in order to: Grammar and spelling check. After using this tool, we reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] National Vulnerability Database (NVD), National vulnerability database, 2025. URL: https://nvd.nist.gov/.

[2] Wikipedia, Coordinated vulnerability disclosure, 2025. URL: https://en.wikipedia.org/wiki/Coordinated_vulnerability_disclosure.

[3] J. Zhou, M. Pacheco, Z. Wan, X. Xia, D. Lo, Y. Wang, A. E. Hassan, Finding A needle in a haystack: Automated mining of silent vulnerability fixes, in: 36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021, Melbourne, Australia, November 15-19, 2021, IEEE, 2021, pp. 705–716. URL: https://doi.org/10.1109/ASE51524.2021.9678720. doi:10.1109/ASE51524.2021.9678720.

[4] Y. Zhou, J. K. Siow, C. Wang, S. Liu, Y. Liu, SPI: automated identification of security patches via commits, ACM Trans. Softw. Eng. Methodol. 31 (2022) 13:1–13:27. URL: https://doi.org/10.1145/3468854. doi:10.1145/3468854.

[5] J. Sun, Z. Xing, Q. Lu, X. Xu, L. Zhu, T. Hoang, D. Zhao, Silent vulnerable dependency alert prediction with vulnerability key aspect explanation, in: 45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023, IEEE, 2023, pp. 970–982. URL: https://doi.org/10.1109/ICSE48619.2023.00089. doi:10.1109/ICSE48619.2023.00089.

[6] L. Han, S. Pan, Z. Xing, J. Sun, S. Yitagesu, X. Zhang, Z. Feng, Do chase your tail! missing key aspects augmentation in textual vulnerability descriptions of long-tail software through feature inference, IEEE Trans. Software Eng. 51 (2025) 466–483. URL: https://doi.org/10.1109/TSE.2024.3523284. doi:10.1109/TSE.2024.3523284.

[7] Y. Wang, H. Le, A. D. Gotmare, N. D. Bui, J. Li, S. C. H. Hoi, Codet5+: Open code large language models for code understanding and generation, arXiv preprint (2023).

[8] DeepSeek-AI, Deepseek-v3 technical report, 2024. URL: https://arxiv.org/abs/2412.19437. arXiv:2412.19437.