Challenges and Future Directions for Large Language Models in Source Code Vulnerability Detection

윤수빈 [1], 김현준 [1], 백윤흥 [1]
[1] 서울대학교 전기정보공학부, 반도체공동연구소

subyun@sor.snu.ac.kr, hjkim@sor.snu.ac.kr, ypaek@snu.ac.kr

Challenges and Future Directions for Large Language Models in Source Code Vulnerability Detection

Subin Yun[1], Hyunjun Kim[1], Yunheung Paek[1]
[1]Dept. of Electrical and Computer Engineering and Inter-University Semiconductor Research Center, Seoul National University

Abstract

Detecting vulnerabilities in source code is essential for maintaining software security, but traditional methods like static and dynamic analysis often struggle with the complexity of modern software systems. Large Language Models (LLMs), such as GPT-4, have emerged as promising tools due to their ability to learn programming language patterns from extensive datasets. However, their application in vulnerability detection faces significant hurdles. This paper explores the key challenges limiting the effectiveness of LLMs in this domain, including limited understanding of code context, scarcity of high-quality training data, accuracy and reliability issues, constrained context windows, and lack of interpretability. We analyze how these factors impede the models' ability to detect complex vulnerabilities and discuss their implications for security-critical applications. To address these challenges, we propose several directions for improvement: developing specialized and diverse datasets, integrating LLMs with traditional static analysis tools, enhancing model architectures for better code comprehension, fostering collaboration between AI systems and human experts, and improving the interpretability of model outputs. By pursuing these strategies, we aim to enhance the capabilities of LLMs in vulnerability detection, contributing to the development of more secure and robust software systems.

1. Introduction

As software systems become increasingly integral to modern society, detecting and preventing vulnerabilities in source code has become critical. Software vulnerabilities can lead to malicious attacks, causing significant economic losses and posing severe security threats. Traditional vulnerability detection techniques, such as static and dynamic analysis, have been effective to some extent. However, the complexity and diversity of newly emerging vulnerabilities make it challenging for these methods to keep pace. The rise of sophisticated attack vectors necessitates advanced tools capable of deeper code understanding and analysis.

Large Language Models (LLMs), like GPT-4, have shown remarkable capabilities in natural language processing, code generation, and code analysis. Trained on vast datasets containing both natural language and programming code, these models have learned patterns and structures inherent in programming languages. Their ability to generalize across tasks makes them promising tools for code understanding and analysis. However, applying LLMs to source code vulnerability detection remains in its infancy, and several challenges limit their effectiveness.

This paper explores the challenges faced by LLMs in source code vulnerability detection, discusses their limitations, and proposes directions for improvement to enhance their performance in this critical area.

2. Related Work

LLMs have been applied to various code-related tasks, including code generation, completion, and summarization. Models like GPT-4 have demonstrated proficiency in generating syntactically correct code snippets and providing natural language explanations for code segments. These successes stem from training on extensive datasets containing both natural language and code, enabling them to learn programming language patterns.

Traditional methods for source code vulnerability detection rely on rule-based systems and classical machine learning techniques. Static analysis tools use predefined rules to identify potential vulnerabilities but often produce numerous false positives and struggle with unknown vulnerability patterns. Dynamic analysis involves executing code to find vulnerabilities but is resource-intensive and may not cover all execution paths.

Recent approaches have incorporated machine learning to improve detection capabilities. Techniques involving support

vector machines, decision trees, and neural networks are trained on labeled datasets to predict vulnerabilities. However, these methods often require extensive feature engineering and may not generalize well across different programming languages or vulnerability types.

Li et al. [1] introduced VulDeePecker, a deep learning-based system specifically designed for vulnerability detection. VulDeePecker employs bi-directional Long Short-Term Memory (LSTM) networks to learn from manually extracted code gadgets associated with vulnerabilities, focusing on domain-specific model design. While VulDeePecker demonstrated promising results in detecting vulnerabilities without extensive feature engineering, it has certain limitations. One significant limitation is that VulDeePecker primarily accommodates data flow analysis (i.e., data dependency) but does not incorporate control flow analysis (i.e., control dependency), potentially missing vulnerabilities that depend on the control structures within the code. This limitation suggests that, although deep learning approaches like VulDeePecker have advanced vulnerability detection, challenges remain in achieving comprehensive solutions. The need for models that can handle both data flow and control flow analyses underscores the importance of exploring more versatile approaches, such as LLMs, despite their own set of challenges, to enhance the effectiveness of automated vulnerability detection systems.

Studies have attempted to apply LLMs directly to vulnerability detection. Zhang et al. [2] explored prompt-enhanced vulnerability detection using ChatGPT, showing that while LLMs can identify some vulnerabilities, they often struggle with complex or context-dependent issues. Ullah et al. [3] provided a comprehensive evaluation, revealing that LLMs cannot reliably identify and reason about security vulnerabilities yet. Zhou et al. [4] discussed emerging results and future directions for LLMs in vulnerability detection, highlighting both the potential and limitations of current models.

3. Challenges of LLMs in Vulnerability Detection

One significant challenge is the limited understanding of code context. LLMs often lack deep comprehension of intricate relationships within codebases. Vulnerabilities frequently involve complex interactions across multiple functions or modules, requiring an understanding of data flows, control structures, and execution paths. LLMs typically analyze code at a syntactic level and may miss vulnerabilities that require semantic analysis and reasoning about program behavior.

Data scarcity is another critical issue. Effective training of LLMs for vulnerability detection requires large, labeled datasets of vulnerable and secure code across various programming languages and vulnerability types. However, such datasets are scarce due to the sensitive nature of vulnerabilities and the substantial effort required for accurate labeling. Moreover, many existing vulnerability detection benchmarks suffer from data duplication problems, which can lead to overestimated model performance. Ding et al. [5] analyzed several popular datasets and found significant amounts of duplicated code between training and test sets, as shown in Table 1. This duplication allows models to memorize code snippets rather than learn to generalize, undermining the validity of evaluation results. Although datasets like PrimeVul have been introduced to address duplication issues, they are limited to C/C++ programs, leaving a gap in coverage for other programming languages. Consequently, the scarcity of high-quality, diverse, and duplication-free datasets limits the models' ability to learn diverse vulnerability patterns, hindering progress in this area.

Benchmark	De-dup	Copy (%)
BigVul [9]	✗	12.7
CVEFixes [10]	✗	18.9
CodeXGLUE [8]	✗	0.6
DiverseVul [5]	✓	3.3
PRIMEVUL (Ours)	✓	**0.0**

(Table 1) The statistics of data duplication in existing vulnerability detection benchmarks [5].

Accuracy and reliability problems hinder the adoption of LLMs in security-critical contexts. Models often capture superficial code structures without thoroughly analyzing execution behavior, leading to high false positive rates where safe code is incorrectly flagged as vulnerable, and false negatives where actual vulnerabilities are missed. Empirical studies by Ullah et al. [3] have shown that LLMs exhibit significant inconsistencies when applied to vulnerability detection. Their evaluation highlighted that LLMs not only struggle with correctly identifying vulnerabilities but also provide varying results upon repeated analyses of the same code snippets. As illustrated in Figure 1, the results demonstrate inconsistencies across different Common Weakness Enumeration (CWE) scenarios, even when using a recommended temperature setting of 0.2 for the LLM. This unreliability poses a substantial challenge for adopting LLMs in security-critical contexts where consistent and accurate detection is paramount.

(a) CWE-787

	S1		S2		S3		S4		S5		S6	
Models	2_v	2_p	2_v	2_p	2_v	2_p	2_v	2_p	2_v	2_p	2_v	2_p
chat-bison	10/10	0/10	10/10	0/10	10/10	0/10	10/10	0/10	2/10	8/10	10/10	0/10
codechat-bison	9/10	0/10	10/10	0/10	10/10	0/10	0/10	9/10	0/10	10/10	0/10	10/10
codellama34b	10/10	0/10	10/10	0/10	10/10	0/10	10/10	0/10	10/10	0/10	5/10	7/10
gpt-3.5	0/10	10/10	0/10	10/10	0/10	10/10	0/10	10/10	0/10	10/10	10/10	0/10
gpt-4	0/10	10/10	6/10	10/10	1/10	10/10	6/10	10/10	0/10	10/10	7/10	10/10

(b) CWE-89

	S1		S2		S3		S4		S5		S6	
Models	2_v	2_p	2_v	2_p	2_v	2_p	2_v	2_p	2_v	2_p	2_v	2_p
chat-bison	10/10	10/10	10/10	8/10	10/10	10/10	10/10	0/10	10/10	10/10	0/10	10/10
codechat-bison	10/10	10/10	10/10	2/10	10/10	7/10	10/10	0/10	10/10	10/10	10/10	10/10
codellama34b	10/10	0/10	10/10	0/10	10/10	0/10	10/10	0/10	10/10	10/10	10/10	0/10
gpt-3.5	10/10	10/10	10/10	10/10	10/10	10/10	10/10	1/10	10/10	10/10	10/10	7/10
gpt-4	10/10	10/10	10/10	10/10	10/10	0/10	10/10	10/10	10/10	10/10	10/10	5/10

(Figure 1) Evaluation results for LLM output consistency. The table presents the number of correctly answered instances out of 10 attempts for each CWE scenario and every prompt used in the study [3].

Vulnerability detection also requires specialized domain knowledge of programming language nuances, memory

management, and security principles. LLMs trained on general-purpose datasets may lack this expertise, making it challenging to identify vulnerabilities dependent on subtle language features or specific security concepts.

Furthermore, LLMs have constrained context windows, limiting the amount of code they can process at once. Large codebases with complex interactions exceed these limits, preventing the model from understanding the overall program structure. The lack of interpretability is another limitation; LLMs function as black boxes, providing limited insights into their decision-making processes, making it difficult for security experts to trust and verify their outputs.

4. Prospects and Directions for Improvement

To enhance the effectiveness of LLMs in vulnerability detection, several strategies can be pursued. Building specialized datasets is crucial. Developing extensive, labeled datasets that cover a wide range of programming languages and vulnerability types can significantly improve model training. Collaborative efforts between industry and academia could facilitate the creation of such datasets. By including real-world examples with detailed annotations from security experts, LLMs can learn nuanced vulnerability patterns, enhancing their ability to detect complex security issues. Expanding datasets like Big-Vul could provide the diversity needed for better generalization.

Integrating LLMs with traditional static analysis tools and program analysis techniques offers a promising avenue for enhancing vulnerability detection. Hybrid models can leverage the strengths of both approaches—using LLMs for pattern recognition and natural language processing while relying on static analysis for in-depth code examination and execution flow analysis. This combination can improve detection accuracy and reduce false positives, which remain significant issues for LLMs in vulnerability detection. Recent research by Zhang et al. [2] demonstrated that integrating structural information derived from program analysis techniques into prompt engineering can significantly enhance the effectiveness of LLMs. By crafting prompts that include elements like control flow graphs (CFGs), program dependence graphs (PDGs), and data flow graphs (DFGs), they observed improvements in detecting certain types of vulnerabilities. For instance, incorporating data flow information into the prompts helped the LLM focus on relevant variable interactions and data dependencies, leading to better identification of vulnerabilities. Figure 2 illustrates an example where data flow is included in the prompt to guide the LLM's analysis. This suggests that leveraging program analysis techniques within prompt engineering can be a practical approach to mitigate some limitations of LLMs without the need for extensive retraining. Combining such prompt-engineered LLMs with traditional program analysis methods could further enhance performance, indicating that deeper integration between these methods may yield even better results in vulnerability detection.

Improving LLM architectures to better understand code execution flows is also vital. Incorporating mechanisms that allow models to reason about control structures, data dependencies, and program semantics could enhance their ability to detect complex vulnerabilities. Research into models that can process larger contexts or hierarchically analyze code could address limitations imposed by context size restrictions. For example, adapting transformer architectures to handle longer sequences or using hierarchical models may enable LLMs to capture broader program structures.

Please output [data flow/control flow/program dependence] of the following code: [CODE]

The answer for data flow:
1. the data value of data at 5th token comes from privateFive at 1st token
2. the data value of data at 5th token comes from the constant “7e5tc4s3” at 5th token
3. the data value of data at 8th token comes from privateFive at 1st token
4. the data value of data at 8th token comes from the constant null at 8th token
5. the data value of principal at 15th token comes from the constant “test” at 15th token
6. the data value of key at 20th token comes from principal at 15th token
7. the data value of key at 20th token comes from data at 5th token or 8th token
8. the data value of key at 20th token comes from the constant null at 20th token

(Figure 2) Example of a prompt incorporating data flow information to enhance LLM analysis [2].

Collaboration with human experts presents another key direction. Developing systems where LLMs assist security professionals can combine the efficiency of automation with expert judgment. LLMs can serve as initial filters, flagging potential vulnerabilities for further examination. This collaborative approach leverages the strengths of both humans and machines, potentially increasing the effectiveness and trustworthiness of vulnerability detection processes. Ullah et al. [3] emphasize the importance of human oversight, given that LLMs cannot yet reliably identify and reason about security vulnerabilities.

Enhancing the interpretability of LLMs is essential for practical adoption. Techniques such as attention visualization, explainable AI methods, or integrating model outputs with human-readable explanations can help bridge the gap between model predictions and expert understanding. By providing insights into the reasoning behind their assessments, LLMs can build trust among security professionals and facilitate the identification of shortcomings in the models' understanding of code semantics. Zhou et al. [4] highlighted that current LLMs lack the necessary code semantic understanding, which hampers their effectiveness in vulnerability detection. Improving interpretability can directly address this issue by revealing how LLMs process and comprehend code structures and behaviors. With a clearer view of the models' internal workings, researchers can pinpoint areas where LLMs fail to capture critical code semantics and make informed enhancements to the model architectures, as suggested by Zhou et al. [4]. Additionally, they advocated for the creation of specialized datasets

enriched with domain-specific knowledge to train models more effectively. By combining enhanced interpretability with enriched training data, LLMs can be better equipped to understand complex code semantics, ultimately improving their vulnerability detection capabilities.

Overall, addressing these areas can significantly improve the capabilities of LLMs in vulnerability detection, making them more reliable and effective tools in securing software systems.

5. Conclusion

Large Large Language Models hold significant potential as tools for source code analysis, including vulnerability detection. However, they currently face substantial challenges related to limited code understanding, data scarcity, accuracy issues, context limitations, and lack of interpretability. Addressing these challenges requires concerted efforts to improve model architectures, develop specialized datasets, integrate hybrid analysis approaches, and foster collaboration between AI systems and human experts.

Future research should focus on these areas to enhance the capabilities of LLMs in vulnerability detection. By leveraging the strengths of LLMs while mitigating their limitations, we can contribute to more secure software systems and reduce the risks posed by software vulnerabilities.

Acknowledgement

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government(MSIT) (RS-2023-00277326). This work was supported by the BK21 FOUR program of the Education and Research Program for Future ICT Pioneers, Seoul National University in 2024. This work was partly supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2020-0-01840,Analysis on technique of accessing and acquiring user data in smartphone, 0.5) and Korea Evaluation Institute of Industrial Technology(KEIT) grant funded by the Korea government(MOTIE) (No.2020-0-01840,Analysis on technique of accessing and acquiring user data in smartphone, 0.5). This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) under the artificial intelligence semiconductor support program to nurture the best talents (IITP-2023-RS-2023-00256081) grant funded by the Korea government(MSIT)
This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2023-2020-0-01602) supervised by the IITP(Institute for Information & Communications Technology Planning & Evaluation) This research was supported by Korea Planning & Evaluation Institute of Industrial Technology(KEIT) grant funded by the Korea Government(MOTIE) (No. RS-2024-00406121, Development of an Automotive Security Vulnerability-based Threat Analysis System(R&D))

References

[1] Li, Z., Zou, D., Xu, S., Ou, X., Jin, H., Wang, S., & Deng, Z. (2018). VulDeePecker: A deep learning-based system for vulnerability detection. *Proceedings of the 25th Annual Network and Distributed System Security Symposium (NDSS).*

[2] Chenyuan Zhang, Hao Liu, Jiutian Zeng, Kejing Yang, Yuhong Li, and Hui Li. 2024. Prompt-enhanced software vulnerability detection using chatgpt. *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings. 276–277.*

[3] Saad Ullah, Mingji Han, Saurabh Pujar, Hammond Pearce, Ayse Coskun, and Gianluca Stringhini. Llms cannot reliably identify and reason about security vulnerabilities (yet?): A comprehensive evaluation, framework, and benchmarks. *IEEE Symposium on Security and Privacy, 2024.*

[4] Xin Zhou, Ting Zhang, and David Lo. 2024. Large Language Model for Vulnerability Detection: Emerging Results and Future Directions. *2024 International Conference on Software Engineering (ICSE), New Ideas and Emerging Results (NIER) Track. IEEE*

[5] Y. Ding, Y. Fu, O. Ibrahim, C. Sitawarin, X. Chen, B. Alomair, D. Wagner, B. Ray, and Y. Chen, “Vulnerability detection with code language models: How far are we?” *arXiv preprint arXiv:2403.18624, 2024.*