리스크 기반 프롬프트 퓨전을 통한 LLM 보강 정적 취약점 탐지

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

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

LLM Augmented Static Vulnerability Detection via Risk-Guided Prompt Fusion

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

Static analysis tools such as Joern can scale to large codebases, yet their purely structural reasoning often yields low precision and recall on real-world vulnerabilities. Recent large language models (LLMs) excel at code understanding, but retraining such models is costly and time-consuming. We present a training-free hybrid pipeline that couples Joern with prompt-based inference from GPT-4.1-nano. After a static scan, functions are routed to the LLM based on Joern's risk score, and their predictions are merged via one of three rule-based strategies. On the Big-Vul dataset, our hybrid more than doubles recall and F1 over Joern, while also improving precision — all without updating model weights. While it still trails fine-tuned models like CodeBERT, the proposed method provides a practical and deployable enhancement for vulnerability detection when full fine-tuning is infeasible. Future work will embed graph-based context and explore ensembles that marry LLM adaptability with structure-aware precision.

1. Introduction

Static vulnerability detection tools, such as Joern [1], provide scalable ways to identify security flaws through code graph analysis. However, despite their effectiveness in static reasoning, they often suffer from low precision and recall, limiting their ability to detect real-world vulnerabilities comprehensively.

Recent advancements in large language models (LLMs) such as GPT-4 [2] have sparked interest in applying language-based inference to code understanding tasks. Motivated by this, we investigate whether LLMs can improve vulnerability detection when used as resource-efficient complements to traditional static analysis. Specifically, we explore a hybrid approach that applies LLM-based inference selectively to code regions identified as potentially risky by Joern, without any model fine-tuning [3].

We conduct extensive experiments on the Big-Vul dataset [4] and show that our training-free hybrid pipeline improves recall by up to 155%, precision by 53%, and F1 score by 116% over Joern's baseline. However, the hybrid system still lags behind task-specific fine-tuned models such as CodeBERT [5], which achieve significantly higher precision and recall.

These findings suggest that while prompting large LLMs offers a practical alternative to costly and time-consuming fine-tuning, achieving truly high-accuracy vulnerability detection still demands structure-aware training tailored to code representations. Our approach highlights a middle ground, enhancing traditional static analysis with LLM inference without requiring model retraining, while also revealing the limitations of relying solely on surface-level code understanding.

Our main contributions are as follows:

- We propose a lightweight hybrid pipeline that combines Joern's static analysis with prompt-based LLM inference without requiring model fine-tuning.
- We introduce a risk-guided routing mechanism to selectively forward functions to the LLM and fuse predictions using interpretable rule-based strategies.
- We demonstrate significant recall and F1 improvements (up to +155% recall, +116% F1) over Joern on the Big-Vul dataset, while preserving high precision under conservative fusion settings.

2. Related Work

2.1 Static Vulnerability Detection

Traditional static analysis tools identify vulnerabilities by analyzing program structures, including abstract syntax trees and control/data flow graphs. While these tools offer scalable, explainable detection, they typically suffer from low precision and recall, especially on complex real-world vulnerabilities. To overcome these limitations, recent studies [6] have incorporated learning-based approaches such as graph neural networks (GNNs) to better model the structural semantics of code. Nevertheless, fundamental challenges remain in balancing explainability, scalability, and detection accuracy.

2.2 LLM Applications to Code Understanding

LLMs, pre-trained on vast natural-language corpora, recently delivered impressive results in code generation, summarization, and bug fixing [3]. Their versatility is further extended through zero-shot prompting, few-shot in-context learning, and task-specific fine-tuning, enabling rapid adaptation to new code-intelligence tasks without heavy retraining [5]. These encouraging outcomes—combined with the practical difficulty of fine-tuning large LLMs—motivate the exploration of prompting strategies as a practical alternative for security-oriented tasks such as vulnerability detection. While prompt-based LLMs have shown early promise in general code tasks, their application to security domains remains limited. This highlights the need to investigate how LLM inference can complement static analysis in practical, low-resource settings [7].

2.3 Fine-Tuned Models for Vulnerability Detection.

Models such as CodeBERT [5] and CodeLlama [8], pre-trained on large-scale source-code datasets, have achieved strong performance in vulnerability detection through fine-tuning. By explicitly incorporating code structure and semantics into their representations, these models outperform traditional static analysis methods in both precision and recall. However, the latest fine-tuned models often have hundreds of millions of parameters and demand high-end GPU memory and inference time, which can hinder rapid deployment in resource-limited or time-critical settings [8].

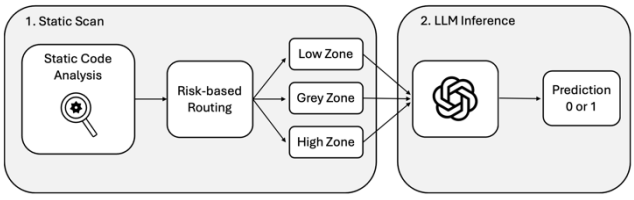

Figure 1: Overview of the hybrid pipeline for vulnerability detection, combining Joern's static scan and LLM-based inference.

3. Methodology

3.1 Overview

We aim to augment Joern's output using zero-training LLM prompts that require no GPU and can be deployed within minutes. Figure 1 illustrates the two-stage pipeline:

1. Static scan

a. Static Code Analysis

We run Joern on every C/C++ function to obtain (i) a binary prediction $p \in \{0,1\}$ and (ii) the built-in *riskScore* $\in [0,5]$, which is a heuristic estimate of exploitability.

b. Risk-based routing

Each function is assigned to a *zone* based on its score:

$$zone = \begin{cases} low, & \text{if riskScore} = 0 \\ grey, & \text{if } 0 < \text{riskScore} < 3 \\ high, & \text{if riskScore} \geq 3 \end{cases}$$

Functions in grey and high zones (plus optional "*extra*" candidates automatically mined by keyword and length heuristics) are forwarded to the LLM; *low-risk* code is left unchanged.

2. LLM inference

We query GPT-4.1-nano using (i) 0-shot, (ii) 3-shot, and (iii) 10-shot prompts. Each prompt contains the raw function text (≤6000 chars) and concise instructions to output 0 (benign) or 1 (vulnerable) only. No model weights are updated during inference.

3.2 Fusion Strategies

Given predictions from Joern (p) and LLM (g), we evaluate three rule-based fusion strategies applied across different risk zones:

Table 1: Fusion strategies for combining Joern's static analysis (p) with LLM-based predictions (g) across risk zones. ∧: logical AND, ∨: logical OR, risk: Joern's built-in *riskScore* ∈ [0, 5].

Fusion Version	low	grey	high
v1 (Conservative ∧)	$p \wedge g$	$p \wedge g$	$p \wedge g$
v2 (Aggressive ∨)	$p \vee g$	$p \vee g$	$p \vee g$
v3 (Confidence-weighted)	p	p	g if *riskScore* ≥ 3 else p

4. Results

4.1 Experiment Set

We evaluate our hybrid pipeline using the BigVul dataset [4], a collection of C/C++ functions with known vulnerability labels. We use the following evaluation metrics:

- *Precision*: The fraction of true positives among all predicted positives.

- *Recall*: The fraction of true positives among all actual positives.
- *F1 Score*: The harmonic mean of precision and recall.

For the LLM-based hybrids, we run the GPT-4.1-nano model with three different prompt strategies (0-shot, 3-shot, and 10-shot). Joern's performance is evaluated as the baseline. We also include results for the hybrid models using GPT-4.1-nano, comparing the performance of different fusion strategies (v1, v2, and v3).

4.2 Performance Gains with Hybrid Models

Table 2 presents the performance of Joern, the LLM-based hybrid models using the aggressive (v2) fusion strategy, and CodeBERT. Across all prompt settings — 0-shot, 3-shot, and 10-shot — the hybrid models consistently outperform Joern in recall. Notably, the 10-shot hybrid achieves a recall of 0.171, a 155% increase compared to Joern's 0.067, while 0-shot and 3-shot settings achieve 0.124 and 0.102 respectively, demonstrating that risk-aware prompting can significantly improve coverage. Precision remains comparable or slightly better: 0.186 (0-shot) and 0.188 (3-shot) versus Joern's 0.178. F1 scores also improve, with the 10-shot hybrid achieving 0.210 — a 116% increase over Joern's 0.097 .

While CodeBERT records the highest precision (0.781) and F1 (0.425), it requires task-specific fine-tuning and high-end GPU resources. In contrast, our zero-training hybrids offer practical recall gains without retraining, making them deployable in settings where full model tuning is infeasible.

Table 2: Performance comparison between Joern, LLM-based hybrid models (using v2: Aggressive fusion strategy) with different prompting settings (0-shot, 3-shot, 10-shot), and fine-tuned CodeBERT. The hybrid models show substantial recall and F1 improvements over Joern, while CodeBERT achieves the highest precision and F1 overall.

Model	Precision	Recall	F1
Joern-only	0.178	0.067	0.097
Hybrid (0-shot)	0.186	0.124	0.149
Hybrid (3-shot)	0.188	0.102	0.132
Hybrid (10-shot)	0.183	0.171	0.210
CodeBERT	0.781	0.292	0.425

4.3 Fusion Strategy Impact

Table 3 compares the impact of the three fusion strategies — v1 (Conservative), v2 (Aggressive), and v3 (Confidence-weighted) — across different prompt configurations (0-shot, 3-shot, 10-shot).

The v1 (Conservative) strategy consistently yields the highest precision across all prompt settings, with 0.281 in 0-shot and 0.307 in 3-shot. However, this comes with low recall (as low as 0.027 in 3-shot), reflecting its strict agreement-based fusion rule. It is most suitable in scenarios where false positives must be minimized.

In contrast, v2 (Aggressive) significantly improves recall, reaching 0.171 in the 10-shot configuration—the highest among all settings. It also produces the best F1 score in each prompt group (e.g., 0.210 for 10-shot), showing its effectiveness in capturing more vulnerabilities by trusting the LLM prediction more often. While v2 improves recall and F1, its precision is slightly lower than v1 in 0-shot and 3-shot, but notably higher in 10-shot—showing that aggressive prompting can benefit both metrics when context is sufficient.

The v3 (Confidence-weighted) strategy performs similarly to Joern's baseline, with precision = 0.178, recall = 0.067, and F1 = 0.097 across all shots. This suggests that the *riskScore* thresholding used in v3 does not add significant value over the default Joern prediction under current settings, likely due to overlapping decision boundaries.

These results highlight that v1 is appropriate when high precision is desired, v2 when maximizing coverage is crucial, and v3 offers minimal benefit unless further tuned. Selecting the right fusion strategy thus depends on the downstream application's tolerance for false positives versus false negatives.

Table 3: Performance of hybrid models under three fusion strategies (v1: Conservative, v2: Aggressive, v3: Confidence-weighted) across different prompt settings. Each strategy emphasizes different detection objectives: v1 prioritizes precision, v2 enhances recall and F1 score, and v3 results in outputs similar to Joern's baseline.

Model	Version	Precision	Recall	F1
Hybrid (0-shot)	v1	0.281	0.05	0.085
Hybrid (0-shot)	v2	0.186	0.124	0.149
Hybrid (0-shot)	v3	0.178	0.067	0.097
Hybrid (3-shot)	v1	0.307	0.027	0.050
Hybrid (3-shot)	v2	0.188	0.102	0.132
Hybrid (3-shot)	v3	0.178	0.067	0.097
Hybrid (10-shot)	v1	0.207	0.057	0.09
Hybrid (10-shot)	v2	0.273	0.171	**0.210**
Hybrid (10-shot)	v3	0.178	0.067	0.097

5. Discussion

Our experiments demonstrate that the LLM-based hybrid models, particularly those using the v2 (Aggressive) fusion strategy, substantially improve recall compared to Joern. The 10-shot hybrid achieves a recall of 0.171, a 155% increase over Joern's 0.067, while 0-shot and 3-shot configurations also show notable gains (0.124 and 0.102, respectively). These results indicate that LLMs — without any fine-tuning — can capture vulnerabilities that static analysis often misses, particularly in ambiguous or context-heavy code.

While v2 improves recall and F1 score, precision gains are modest. In 0-shot and 3-shot settings, v2 achieves 0.186 and 0.188 precision, slightly above Joern's 0.178 baseline. In contrast, v1 (Conservative) consistently yields the highest precision across all prompting strategies (e.g., 0.307 in 3-shot) but suffers from very low recall (e.g., 0.027), making it suitable for high-confidence, low-coverage scenarios.

The v3 (Confidence-weighted) strategy mirrors Joern's baseline performance, suggesting that static riskScore thresholds alone may be insufficient to guide effective LLM inference without further refinement.

Although the hybrid models do not surpass fine-tuned models such as CodeBERT (0.781 precision, 0.425 F1), they offer significant recall gains without requiring model retraining or large-scale computational resources. This makes them practical for rapid deployment, triage systems, or settings where fine-tuning is infeasible.

Nevertheless, limitations remain. Current prompting strategies lack structural awareness, leading to imprecise predictions in complex control-flow or dataflow scenarios. Future work should explore incorporating graph-based contexts (e.g., ASTs, CFGs, dataflow features) into LLM prompting, or develop hybrid methods that blend LLM inference with static analysis to achieve higher precision without sacrificing adaptability [9].

6. Conclusion

In this study, we proposed a hybrid pipeline that integrates Joern's static analysis with zero-training LLM-based inference for software vulnerability detection. Our experiments demonstrate that the hybrid models, particularly when using aggressive fusion strategies, substantially improve recall by up to 155% over Joern, capturing vulnerabilities that static analysis alone often misses. While fine-tuned models such as CodeBERT achieve higher precision and F1 scores, they require significant computational resources and retraining. In contrast, our approach leverages prompting without model updates, enabling immediate integration into existing static analysis workflows without the need for fine-tuning. Future work should focus on enhancing the structural reasoning capabilities of LLMs, such as by incorporating graph-based context or static program features, and on exploring ensemble methods that combine the adaptability of LLMs with the precision of structure-aware models.

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 2025. This work was supported by Inter-University Semiconductor Research Center (ISRC). 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).

References

[1] Joern. - The bug hunter's workbench. Joern.io. https://joern.io/

[2] OpenAI. (2023). ChatGPT-4 [Large Language Model]. OpenAI. https://openai.com/chatgpt

[3] Chakraborty, S., Krishna, R., Ding, Y., & Ray, B. (2021). Deep learning based vulnerability detection: Are we there yet?. *IEEE Transactions on Software Engineering*, 48(9), 3280-3296.

[4] Fan, J., Li, Y., Wang, S., & Nguyen, T. N. (2020, June). AC/C++ code vulnerability dataset with code changes and CVE summaries. In *Proceedings of the 17th international conference on mining software repositories* (pp. 508-512).

[5] Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., ... & Zhou, M. (2020). Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155*.

[6] Steenhoek, B., Le, W., & Gao, H. (2022). DeepDFA: Dataflow Analysis-Guided Efficient Graph Learning for Vulnerability Detection.

[7] Ullah, S., Han, M., Pujar, S., Pearce, H., Coskun, A., & Stringhini, G. (2024, May). Llms cannot reliably identify and reason about security vulnerabilities (yet?): A comprehensive evaluation, framework, and benchmarks. In *2024 IEEE Symposium on Security and Privacy (SP)* (pp. 862-880). IEEE.

[8] Roziere, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X. E., ... & Synnaeve, G. (2023). Code llama: Open foundation models for code. arXiv preprint arXiv:2308.12950.

[9] Tian, Z., Tian, B., Lv, J., Chen, Y., & Chen, L. (2024). Enhancing vulnerability detection via AST decomposition and neural sub-tree encoding. Expert Systems with Applications, 238, 121865.