MulVul: Retrieval-augmented Multi-Agent Vulnerability Detection via Cross-Model Prompt Evolution

Anonymous ACL submission

Abstract

Large language models (LLMs) show promise for code vulnerability detection, but face critical challenges in multi-class settings: heterogeneous vulnerability patterns undermine a single unified detector, and LLM-based detection is sensitive to prompts and prone to hallucinations. We propose **MulVul**, a retrieval-augmented multi-agent framework with cross-model prompt evolution for reliable multi-class vulnerability detection. Specifically, a Router Agent first identifies candidate vulnerability types, and specialized Detector Agents then validate each independently. Both agents leverage retrieved evidence from vulnerability knowledge bases to ground reasoning and mitigate hallucinations. To obtain robust prompts, MulVul employs *Cross-Model Prompt Evolution*, where an Evolutionary Agent generates candidate prompts while Router and Detector Agents execute them and provide feedback separately. This separation mitigates overfitting to model-specific biases and enhances generalization. Experiments on [Dataset Names] show that MulVul achieves [X]% F1 improvement and [Y]% false positive reduction over state-of-the-art baselines, with ablations confirming each component's contribution.

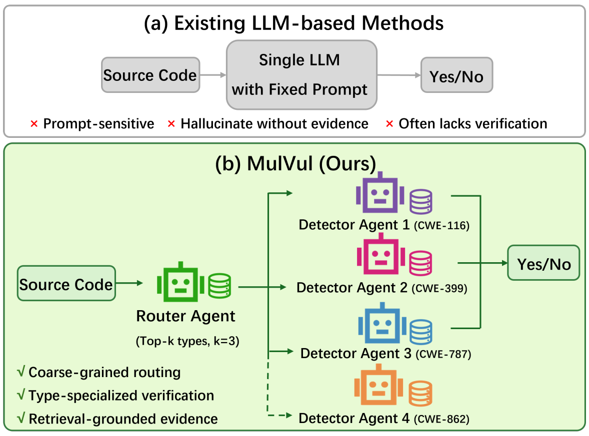


Figure 1: Comparison of vulnerability detection approaches. (a) Existing LLM-based methods use fixed prompts without external knowledge. (b) MulVul employs multi-agent collaboration with retrieval augmentation and cross-model prompt evolution.

1 Introduction

Software vulnerabilities remain a major threat to cybersecurity, causing substantial economic losses. As modern software systems grow increasingly complex, manual code auditing becomes expensive, time-consuming, and error-prone, motivating automated vulnerability detection (Ghaffarian and Shahriari, 2017).

Recent large language models (LLMs) have demonstrated strong code understanding capabilities, sparking interest in applying them to vulnerability detection (Zhou et al., 2025). Previous efforts primarily focused on single-model approaches, where one model is fine-tuned or prompted to identify all vulnerability types simultaneously (Lin and Mohaisen, 2025). However, vulnerability patterns are highly heterogeneous (Chakraborty et al., 2021). For example, buffer overflows typically depend on pointer/array bounds reasoning, while injection attacks depend on tracking how untrusted inputs flow into sensitive operations. As a result, a single unified detector struggles to capture type-specific patterns simultaneously, leading to missed vulnerabilities or false alarms across types.

Motivated by the progress of multi-agent systems that decompose complex tasks into specialized components (Wu et al., 2024), a natural question arises: *Can a multi-agent architecture improve multi-class vulnerability detection by routing inputs to type-specialized reasoning?* However, it is challenging to applying multi-agent architectures for multi-class vulnerability detection. First, multi-agent architectures incur prohibitive computational overhead. A straightforward approach assigns one detector agent to each vulnerability type, but ensuring comprehensive coverage requires invoking numerous agents per input, causing inference costs to scale linearly with the number of vulnerability types. For real-world systems covering dozens of Common Weakness Enumeration (CWE) types, this cost quickly becomes impractical for deployment. Second, the complexity of prompt engineering increases significantly in multi-agent architectures. Different vulnerability types require different prompting strategies. Each detector agent must accurately capture these type-specific patterns, including relevant code structures, data flow properties, and exploitation conditions. Third, multi-agent LLM systems amplify hallucinations in multi-type settings. Vulnerability evidence is often distributed across complex control and data flows, forcing individual agents to reason under uncertainty. Once a hallucinated claim enters the inter-agent communication, it can be shared and echoed across agents, triggering cascading error propagation and ultimately distorting the final decision (Hong et al., 2023).

To address these challenges, we propose MulVul, a retrieval-augmented multi-agent framework with cross-model prompt evolution for reliable multi-class vulnerability detection. Figure 1 contrasts prior LLM-based vulnerability detectors with MulVul. MulVul adopts a Router-Detector architecture with Top-k routing, where a Router Agent identifies Top-k candidate types and then type-specialized Detectors are invoked for verification, reducing inference cost while maintaining high recall. To obtain reliable and specialized prompts for each agent, MulVul employs Cross-Model Prompt Evolution mechanism. An Evolution Agent explores the prompt space via genetic operators, while the Router and Detector Agents execute candidate prompts using a different LLM backbone (e.g., Llama-3 for evolution, GPT-4 for execution). By decoupling prompt generation from execution, MulVul prevents prompts from overfitting to single-model biases and yields robust detection strategies. To mitigate hallucinations, MulVul is built on a knowledge base using SCALE representations and retrieve cross-type evidence for routing and type-specific positive/negative pairs for verification, anchoring predictions and reducing hallucinations (Wen et al., 2024).

Through comprehensive experiments on [Dataset Names], we demonstrate that MulVul consistently achieves state-of-the-art performance compared to existing methods across diverse vulnerability types and scenarios. Furthermore, we show that prompts evolved by MulVul effectively transfer across different LLM backbones, confirming the generalization benefits of cross-model optimization.

The contributions are summarized as follows:

- We propose MulVul, a novel retrieval-augmented multi-agent framework for multi-class vulnerability detection that decomposes the task into candidate routing and type-specialized verification, effectively addressing the pattern heterogeneity challenge while significantly reducing inference costs.

- We design cross-model evolutionary prompt optimization that separates prompt generation from execution across different LLMs, mitigating overfitting to model-specific biases and enabling joint optimization of interdependent agent prompts.

- Comprehensive experiments demonstrating [X]% F1 improvement and [Y]% false positive reduction over state-of-the-art baselines, with ablation studies confirming each component's contribution.

The remainder of this paper is organized as follows. Section 2 reviews related work on code vulnerability detection, and Section 3 introduces preliminaries and problem definition. Section 4 presents the MulVul framework in detail, including the router-detector architecture, structured retrieval mechanism, and Cross-Model EPO. Section 5 describes our experimental setup, and Section 5 presents results and analysis. Section 6 concludes with discussion of limitations and future directions.

2 Related Work

Learning-based vulnerability detection. Learning-based vulnerability detection has progressed from early deep learning frameworks (e.g., VulDeePecker (Li et al., 2018)) to neural models that learn code representations with sequence and graph encoders (Zhou et al., 2019; Li et al., 2021; Chakraborty et al., 2021), and more recently to pre-trained code models such as GraphCodeBERT (Guo et al., 2021) and UniXcoder (Guo et al., 2022). Large language models further enable zero-shot or few-shot vulnerability detection, but practical deployment faces a key tension: multi-class coverage vs. type-specific precision, since a single unified detector struggles to capture heterogeneous vulnerability patterns simultaneously (Zhou et al., 2025; Sheng et al., 2025). Meanwhile, multi-agent LLM frameworks (e.g., AutoGen (Wu et al., 2024), MetaGPT (Hong et al., 2023)) demonstrate effective task decomposition via specialized roles, yet they have not been tailored to multi-class vulnerability detection under tight cost and reliability constraints. We target this tension by decomposing multi-class detection into candidate routing followed by type-specialized verification.

Prompt engineering and optimization for LLMs. Beyond manual prompt design (e.g., zero-shot, few-shot, chain-of-thought prompting (Wei et al., 2022)), automatic prompt optimization reduces human effort by generating and refining prompts via search-based or evolutionary strategies, such as APE (Zhou et al., 2022), EvoPrompt (Guo et al.), and OPRO (Yang et al., 2023). However, these methods typically optimize prompts on a single target backbone, risking overfitting to model-specific biases and limiting transferability across LLMs. We complement this line by decoupling prompt generation from execution across different LLMs, leveraging cross-model feedback to improve generalization.

Retrieval-augmented generation and hallucination mitigation. Retrieval-augmented generation (RAG) grounds model outputs with external evidence (Lewis et al., 2020) and has been explored in code intelligence, such as retrieval-augmented code completion (Lu et al., 2022). However, vulnerability detection demands security-specific evidence and type-aware reasoning to distinguish similar but distinct vulnerability classes. We construct a vulnerability knowledge base using SCALE's structured semantic representations (Wen et al., 2024) and perform agent-specific retrieval: cross-type evidence for the Router Agent and type-specific positive/negative exemplars for Detector Agents, anchoring predictions to concrete patterns and reducing hallucinations without additional fine-tuning.

3 Preliminaries and Problem Definition

3.1 LLM-based Code Vulnerability Detection

Given an LLM $\mathcal{M}$ with frozen parameters, vulnerability detection is formulated as a prompt-based classification task. The input consists of a code snippet $x \in \mathcal{X}$ and a textual prompt p, and the model predicts:

$$\hat{y} = \operatorname*{argmax}_{y \in \mathcal{Y}} P_{\mathcal{M}}(y \mid p, x) \quad (1)$$

where $\mathcal{Y} = \{y_0, y_1, \ldots, y_K\}$ is the label space, with y_0 denoting non-vulnerable code and each y_i ($i \geq 1$) corresponding to a specific vulnerability type. Since the parameters of $\mathcal{M}$ are frozen, the detection performance relies heavily on the quality of p, which serves as the optimizable variable.

3.2 SCALE: Structured Code Representation

To alleviate the difficulty of extracting statement-level semantics and execution logic from raw code, SCALE (Wen et al., 2024) converts a code snippet x into a structure-aware sequence. Concretely, SCALE builds a syntax-based structure for x, enriches security-relevant statements with semantic comments, and normalizes key control-flow conditions with structured natural-language templates, yielding

$$T(x) = \text{SCALE}(x). \quad (2)$$

In MulVul, we use $T(x)$ for indexing and retrieval, as it reduces sensitivity to superficial variations while preserving structural cues critical for distinguishing vulnerability types.

3.3 Retrieval-Augmented Code Analysis

LLM-based detection is prone to hallucinations. To solve this issue, all LLM are input with external evidence. Specifically, we construct a vulnerability knowledge base $\mathcal{K} = \{(T(x^{(i)}), y^{(i)})\}_{i=1}^{N}$ from labeled samples, pairing each SCALE representation with its vulnerability label.

Given a query snippet x, we retrieve Top-k similar examples $E = \text{Retrieve}(T(x), \mathcal{K}, k)$, and incorporate them into the prompt to get output:

$$\hat{y} = \operatorname*{argmax}_{y \in \mathcal{Y}} P_{\mathcal{M}}(y \mid p * \oplus E \oplus x). \quad (3)$$

3.4 Evolutionary Prompt Optimization

Definition 3.1 (Evolutionary Prompt Optimization)**.** Given an LLM $\mathcal{M}$, a validation set $\mathcal{D}_{val}$, and a fitness function $\mathcal{F}$, evolutionary prompt optimization searches for the optimal prompt p^* by iteratively evolving a population of candidate prompts $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$:

$$p^* = \operatorname*{argmax}_{p} \mathcal{F}(p, \mathcal{M}, \mathcal{D}_{val}) \quad (4)$$

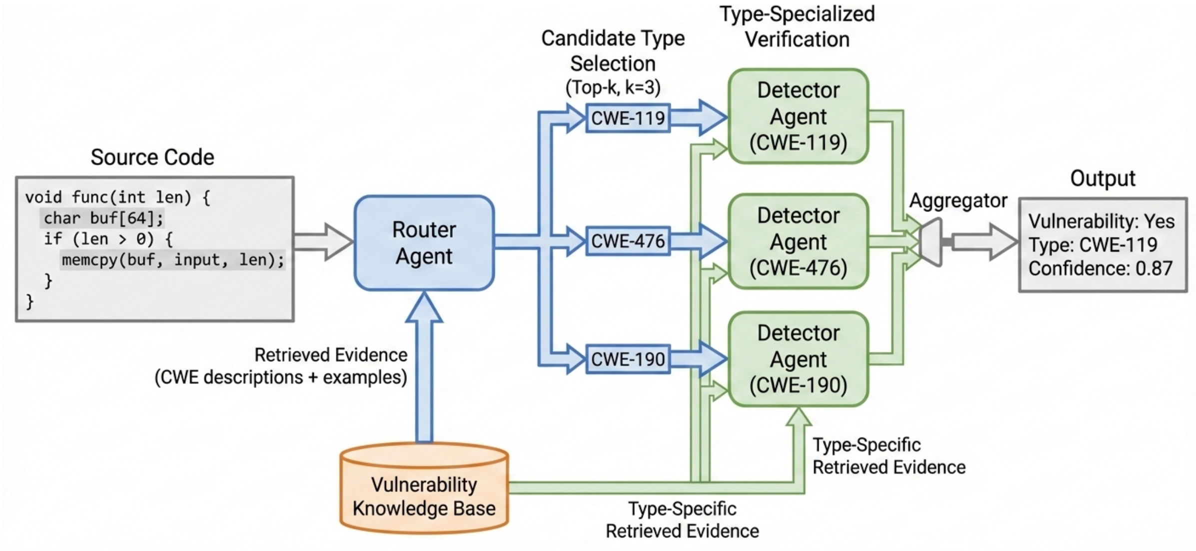


Figure 2: Overview of MulVul. (a) **Router-Detector Architecture**: The Router Agent identifies top-k candidate types, and corresponding Detector Agents perform independent verification. (b) **Agent-Specific Retrieval**: The Router receives cross-type evidence while Detectors receive type-specific positive/negative exemplars. (c) **Cross-Model Prompt Evolution**: Prompt generation and execution are decoupled across different LLMs to improve robustness.

Each generation applies genetic operators to generate new candidates, which are then evaluated and selected based on fitness scores.

Evolutionary methods explore the discrete prompt space more broadly than gradient-based approaches. However, existing methods typically optimize on a single LLM, risking overfitting to model-specific biases.

3.5 Problem Formulation

Our goal is to design a reliable and efficient multi-class vulnerability detection framework. Formally, given a code snippet $x \in \mathcal{X}$ and a vulnerability label space $\mathcal{Y} = \{y_0, y_1, \ldots, y_K\}$, we aim to design a detection system $\mathcal{A}$ that produces a prediction $\hat{y} = \mathcal{A}(x)$ satisfying: (i) high type-specific precision across all K vulnerability categories, (ii) robustness to prompt variations and transferability across different LLM backbones, and (iii) computational efficient and scalable.

4 Method

4.1 Overview of MulVul

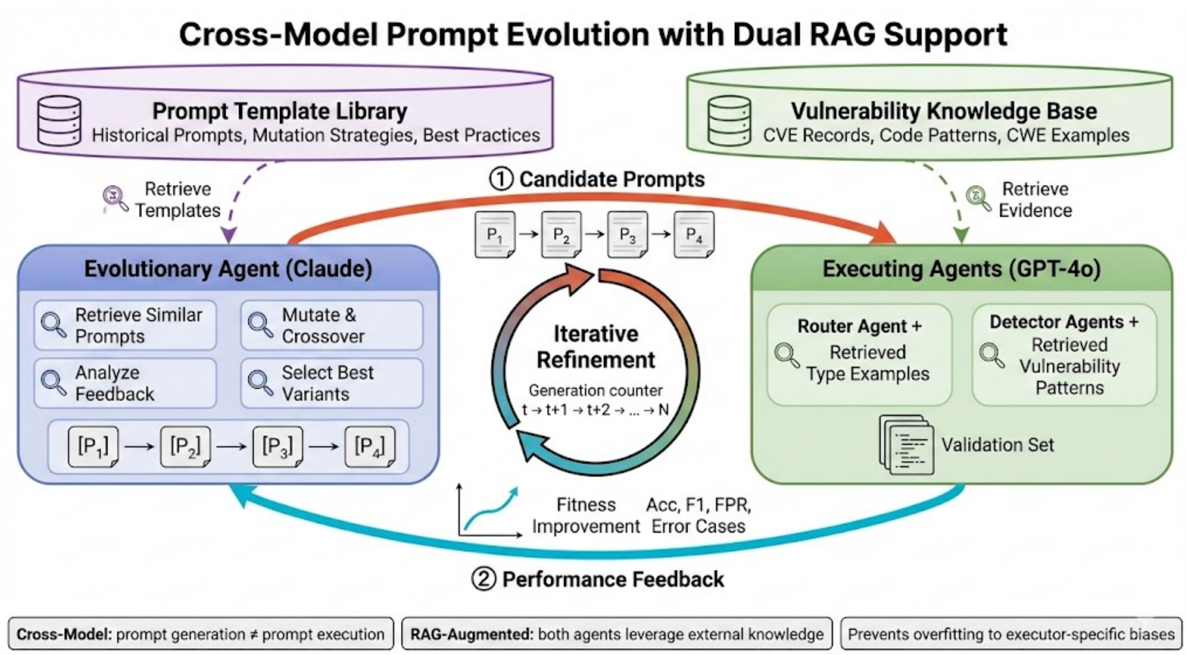
Figure 2 illustrates the overall architecture of MulVul, which consists of three key components: (1) a Router-Detector multi-agent architecture that decomposes multi-class detection into candidate routing and type-specialized verification, (2) agent-specific retrieval augmentation that grounds each agent's reasoning with tailored evidence from a vulnerability knowledge base, and (3) cross-model prompt evolution that jointly optimizes agent prompts across different LLM backbones to improve robustness and transferability.

Given a code snippet x, MulVul first transforms it into a SCALE representation $T(x)$ and retrieves cross-type evidence from the knowledge base $\mathcal{K}$. The Router Agent then identifies a candidate set $\mathcal{C}$ of top-k most likely vulnerability types. For each candidate type $c \in \mathcal{C}$, the corresponding Detector Agent retrieves type-specific positive and negative exemplars and performs independent verification. The final prediction is determined by aggregating the verification results. All agent prompts are optimized through cross-model evolution, where an Evolution Agent generates candidate prompts using one LLM while Router and Detector Agents execute and evaluate them using a different LLM, mitigating single-model overfitting. The following subsections detail each component.

4.2 Router-Detector Multi-Agent Architecture

4.3 Agent-Specific Retrieval Augmentation

4.4 Cross-Model Prompt Evolution



Cross-Model Prompt Evolution with Dual RAG Support
Prompt Template Library
Historical Prompts, Mutation Strategies, Best Practices
Vulnerability Knowledge Base
CVE Records, Code Patterns, CWE Examples
Retrieve Templates
① Candidate Prompts
Retrieve Evidence
P1 → P2 → P3 → P4
Evolutionary Agent (Claude)
Retrieve Similar Prompts
Mutate & Crossover
Analyze Feedback
Select Best Variants
[P1] → [P2] → [P3] → [P4]
Iterative Refinement
Generation counter
t → t+1 → t+2 → ... → N
Executing Agents (GPT-4o)
Router Agent +
Retrieved Type Examples
Detector Agents +
Retrieved Vulnerability Patterns
Validation Set
Fitness Improvement
Acc, F1, FPR, Error Cases
② Performance Feedback
Cross-Model: prompt generation ≠ prompt execution
RAG-Augmented: both agents leverage external knowledge
Prevents overfitting to executor-specific biases

Figure 3

5 Evaluation

5.1 Experimental Setup

5.2 Comparison of Existing Methods

5.3 Evaluation of AAA

5.4 Ablation Studies

6 Conclusion

Limitations

This document does not cover the content requirements for ACL or any other specific venue. Check the author instructions for information on maximum page lengths, the required "Limitations" section, and so on.

References

Rie Kubota Ando and Tong Zhang. 2005. A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research*, 6:1817–1853.

Galen Andrew and Jianfeng Gao. 2007. Scalable training of L1-regularized log-linear models. In *Proceedings of the 24th International Conference on Machine Learning*, pages 33–40.

Saikat Chakraborty, Rahul Krishna, Yangruibo Ding, and Baishakhi Ray. 2021. Deep learning based vulnerability detection: Are we there yet? *IEEE Transactions on Software Engineering*, 48(9):3280–3296.

Seyed Mohammad Ghaffarian and Hamid Reza Shahriari. 2017. Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey. *ACM computing surveys (CSUR)*, 50(4):1–36.

Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. 2022. Unixcoder: Unified cross-modal pre-training for code representation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7212–7225.

Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie LIU, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, and 1 others. 2021. Graphcodebert: Pre-training code representations with data flow. In *International Conference on Learning Representations*.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *The Twelfth International Conference on Learning Representations*.

Dan Gusfield. 1997. *Algorithms on Strings, Trees and Sequences*. Cambridge University Press, Cambridge, UK.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, and 1 others. 2023. Metagpt: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, and 1 others. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474.

Zhen Li, Deqing Zou, Shouhuai Xu, Hai Jin, Yawei Zhu, and Zhaoxuan Chen. 2021. Sysevr: A framework for using deep learning to detect software vulnerabilities. *IEEE Transactions on Dependable and Secure Computing*, 19(4):2244–2258.

Zhen Li, Deqing Zou, Shouhuai Xu, Xinyu Ou, Hai Jin, Sujuan Wang, Zhijun Deng, and Yuyi Zhong. 2018. Vuldeepecker: A deep learning-based system for vulnerability detection. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. The Internet Society.

Jie Lin and David Mohaisen. 2025. From large to mammoth: A comparative evaluation of large language models in vulnerability detection. In *Proceedings of the 2025 Network and Distributed System Security Symposium (NDSS)*.

Shuai Lu, Nan Duan, Hojae Han, Daya Guo, Seungwon Hwang, and Alexey Svyatkovskiy. 2022. Reacc: A retrieval-augmented code completion framework. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6227–6240.

Mohammad Sadegh Rasooli and Joel R. Tetreault. 2015. Yara parser: A fast and accurate dependency parser. *Computing Research Repository*, arXiv:1503.06733. Version 2.

Ze Sheng, Zhicheng Chen, Shuning Gu, Heqing Huang, Guofei Gu, and Jeff Huang. 2025. LLMs in Software Security: A Survey of Vulnerability Detection Techniques and Insights. *ACM Computing Surveys*, 58(5):1–35.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Xin-Cheng Wen, Cuiyun Gao, Shuzheng Gao, Yang Xiao, and Michael R Lyu. 2024. Scale: Constructing structured natural language comment trees for software vulnerability detection. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 235–247.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, and 1 others. 2024. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First Conference on Language Modeling*.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. 2023. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*.

Xin Zhou, Sicong Cao, Xiaobing Sun, and David Lo. 2025. Large language model for vulnerability detection and repair: Literature review and the road ahead. *ACM Transactions on Software Engineering and Methodology*, 34(5):1–31.

Yaqin Zhou, Shangqing Liu, Jingkai Siow, Xiaoning Du, and Yang Liu. 2019. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks. *Advances in neural information processing systems*, 32.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2022. Large language models are human-level prompt engineers. In *The eleventh international conference on learning representations*.

A Example Appendix

This is an appendix.