American Journal of Artificial Intelligence and Neural Networks

australiansciencejournals.com/ajainn

E-ISSN: 2688-1950

VOL 07 ISSUE 01 2026

Structural Prompt Variants and Their Impact on LLM-Based Vulnerability Detection Accuracy

Mariana Torres[1]*, Javier Martínez[2], Alejandro Ruiz[3]
Department of Computer Science, Tecnológico de Monterrey, Monterrey 64849, Mexico
***Corresponding author**: m.torres@tec.mx*

Abstract
The structure of prompts used in vulnerability detection tasks significantly affects the reasoning process of large language models. This study systematically evaluates how formatting, decomposition, and semantic density influence the accuracy of detecting five major vulnerability categories (CWE-20, CWE-79, CWE-89, CWE-120, CWE-798). We construct a benchmark of 12,000 prompt–code pairs and test three LLMs: GPT-4, Claude-3, and Llama-3-70B. Results show that structured prompts with explicit reasoning stages improve true-positive rates by 24–41%, while overly verbose prompts increase hallucination-induced false alarms by 15%. Prompt–model mismatch analysis reveals that models differ in tolerance to prompt ambiguity. These findings highlight the necessity of prompt normalization for trustworthy security auditing using LLMs.

Keywords: *prompt structure; vulnerability detection; prompt ambiguity; LLM reasoning; CWE classification*

1.INTRODUCTION

Large language models (LLMs) are increasingly adopted across a wide range of software engineering tasks, including code review, vulnerability detection, security auditing, and automated debugging. Their ability to analyze source code with minimal contextual information has made them attractive tools for assisting developers during development and maintenance. However, recent empirical studies consistently show that LLMs remain unreliable when reasoning about software security. Evaluations report that models frequently miss common vulnerability patterns or produce incorrect assessments of code safety, even in relatively simple scenarios [1,2]. Typical failure cases include overlooked input validation, insufficient output sanitization, and unsafe construction of commands or database queries, with such issues observed across

multiple programming languages and coding styles [3]. In response to these limitations, a growing body of work has explored methods for improving the security behavior of LLMs during code-related tasks. Some approaches steer model generation away from deprecated or unsafe programming interfaces by explicitly constraining the APIs that can be used, demonstrating that generation-time guidance can reduce certain classes of insecure outputs [4]. While such methods show promise for secure code synthesis, they do not directly address the broader challenge of vulnerability detection and security reasoning over existing code. In parallel, several studies have proposed systematic evaluations of LLMs on security-focused tasks, revealing substantial variance in performance across models and vulnerability types [5,6]. These findings suggest that LLM security behavior is highly sensitive to task formulation and evaluation settings. To better understand these risks, researchers have developed benchmarks for secure coding and vulnerability detection based on public repositories, CWE-labeled code samples, and real-world vulnerability disclosures. Results from these benchmarks indicate that LLM accuracy varies widely and can change significantly with minor differences in how questions or code snippets are presented [7,8]. Despite this sensitivity, most existing evaluations treat the prompt as a fixed artifact and focus primarily on model architecture or training data. The role of prompt structure itself—how information is organized, ordered, or emphasized—has received comparatively little attention in security-focused studies, even though prompt design is known to influence model behavior in other domains. Prompt design has emerged as a central topic in LLM research more broadly[9]. Prior work demonstrates that formatting choices, stepwise instructions, and explicit reasoning cues can substantially affect model performance in tasks such as mathematical reasoning, logical inference, and general question answering [10,11]. These studies also show that overly verbose or poorly structured prompts can increase the likelihood of incorrect or fabricated responses. However, such insights have not been systematically examined in the context of vulnerability detection, where accurate reasoning requires careful analysis of control flow, data flow, and implicit security assumptions within code [12]. Whether structured prompts consistently improve security reasoning, or whether they introduce new failure modes, remains an open question. Another persistent challenge in LLM-based security analysis is hallucination. Prior security-focused evaluations report that models may warn about non-existent vulnerabilities, misinterpret standard library functions, or confuse unrelated security concepts [13,14]. These false positives can mislead developers and significantly increase the cost of manual review, particularly in large codebases. Recent monitoring efforts in

production LLM systems suggest that hallucination rates are strongly influenced by prompt clarity and information density [15,16]. Nevertheless, existing work rarely connects prompt structure to concrete outcomes at the level of specific vulnerability classes, such as CWE categories, limiting the practical guidance that can be derived for secure deployment. Despite growing interest in secure code analysis with LLMs, several fundamental gaps remain. It is not yet well understood how different prompt formats affect model performance on specific vulnerability categories, nor whether additional structure uniformly improves reasoning quality. Differences among widely used models—such as GPT-4, Claude-3, and Llama-3-70B—in their tolerance for ambiguous, sparse, or overly dense prompts are also underexplored. Moreover, current benchmarks typically rely on a small number of prompt styles, making it difficult to isolate the effects of prompt design from other factors. This study aims to address these gaps by systematically examining how prompt structure influences LLM-based vulnerability detection. We construct a benchmark comprising 12,000 prompt–code pairs spanning five representative CWE categories and design multiple prompt variants that differ in formatting, reasoning decomposition, and information density. Using this benchmark, we evaluate three major LLMs and measure changes in true-positive and false-positive rates under different prompt conditions. The results show that clearly structured prompts with explicit reasoning cues can significantly improve detection accuracy, while excessively dense prompts tend to increase false alarms. These findings demonstrate that prompt structure is a critical variable in LLM-driven security auditing and underscore the need for carefully designed, stable prompt templates when deploying LLMs in practical vulnerability analysis workflows.

2.Materials and Methods

2.1 Sample Description and Study Scope

This study uses 12,000 prompt–code pairs covering five vulnerability categories: CWE-20, CWE-79, CWE-89, CWE-120, and CWE-798. Code samples were taken from public repositories and LLM outputs. Only snippets with valid syntax and clear control flow were included. Prompts were created under controlled settings to vary format, step structure, and information level. All samples were checked to ensure they represent realistic coding patterns across different languages.

2.2 Experimental Design and Control Setup

We tested how different prompt structures influence detection accuracy. Three types of prompts were defined: short prompts, structured prompts, and prompts with step-by-step reasoning. These were compared with a simple baseline prompt. Each prompt type was paired with the same code inputs and evaluated by three LLMs

under identical settings. This design ensures that changes in accuracy can be traced to prompt structure rather than model differences.

2.3 Measurement Methods and Quality Control

Ground-truth labels were produced using static-analysis tools and manual review. Two reviewers labelled each sample, and a third reviewer resolved any disagreement. Performance was measured using true-positive and false-positive rates for each CWE category. All tests were repeated three times to reduce random variation. Preprocessing and evaluation steps were controlled through versioned scripts to keep the procedure consistent.

2.4 Data Processing and Model Formulas

Prompts and code snippets were normalized before analysis. Classification metrics were computed using standard formulas. The true-positive rate (TPR) was defined as:

$$TPR=\frac{TP}{TP+FN}.$$

The false-positive rate (FPR) was defined as:

$$FPR=\frac{FP}{FP+TN}.$$

Category-level performance was obtained by averaging results across the five CWE types.

2.5 Evaluation Procedure and Reproducibility

All LLM evaluations were run with temperature set to zero to keep outputs stable. Each prompt–code pair was evaluated once per run, and all models used the same test order. Outputs were logged with prompt identifiers and timestamps to allow full traceability. The evaluation can be reproduced by applying the same dataset, prompt groups, and model settings described here.

3.Results and Discussion

3.1 Influence of Prompt Structure on Detection Accuracy

Across the 12,000 prompt–code pairs, the structure of the prompt had a clear impact on detection outcomes. Prompts that used short sections and asked the model to explain its reasoning produced the highest true-positive rates. These structured prompts improved detection by 24–41% across the five CWE categories when compared with a simple one-sentence prompt. Flat prompts often missed cases where validation or control-flow checks were incomplete [17,18]. Fig. 1 summarizes these trends and shows that the improvement appears in all three evaluated models. The results indicate that added structure helps the model focus on key code paths rather than rely on generic patterns.

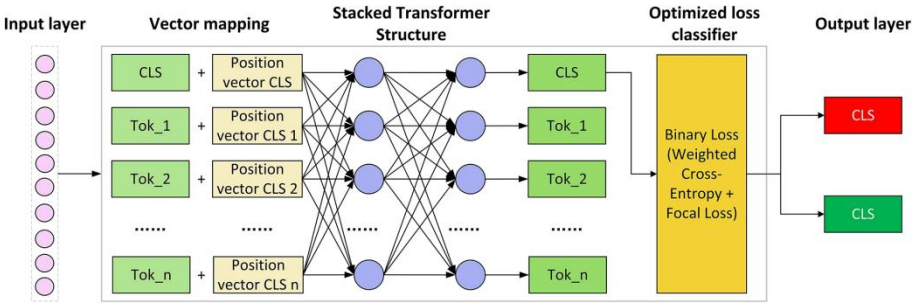

Figure 1:How prompt structure changes the accuracy of vulnerability detection.

3.2 Category-Level Differences and Effect of Verbosity

The effect of prompt design varies by vulnerability type. For CWE-20 and CWE-120, structured prompts mainly reduce missed detections, likely because they guide the model to look for input and boundary checks. For CWE-79 and CWE-89, the benefit is more pronounced, since these cases require tracking how untrusted data flows into output or queries. In contrast, very long prompts with repeated instructions increase false alarms by about 15%, especially for CWE-79 and CWE-89. This suggests that dense wording encourages the model to overestimate risk when string operations appear in the code [19,20]. Fig. 2 illustrates how different prompt styles shift the balance between true-positive and false-positive rates. These results show that prompt structure is helpful, but excessive length can reduce reliability.

```
int root(int a){
    int c=0;
    int b=bar(a);
    if(a==10){
        c=a+b;
    }
    return c;
}
```

```
1   #Type      #Depth  Value1  Value2
2   func        0       int     root
3   params      1
4   param       2       int     a
5   stmnts      1
6   decl        2       int     c
7   op          2       =
8   water       2       0
9   decl        2       int     b
10  op          2       =
11  call        3       bar
12  arg         4       a
13  water       5       a
14  if          2       (a == 10)
15  cond        3       a == 10
16  water       4       a
17  op          4       ==
18  water       4       10
19  stmts       3
20  water       4       {
21  water       4       c
22  op          4       =
23  water       4       a
24  op          4       +
25  water       4       b
26  water       4       ;
27  water       4       }
28  return      2       c
29  water       3       c
30
```

Figure 2. *Performance of the three models when tested with different prompt styles.*

3.3 Model-Specific Responses to Prompt Variants

The three models respond differently to the same prompt changes. GPT-4 benefits the most from structured prompts with short reasoning steps and shows only a modest increase in false positives. Claude-3 also improves with structure but is more stable when prompts are long. Llama-3-70B is more sensitive: it gains from concise structured prompts but shows sharp increases in false alarms when prompts become verbose. These differences show that a prompt designed for one model may not work well for another. They also indicate that prompt design should be treated as part of the evaluation pipeline rather than a fixed background choice [21,22].

3.4 Practical Implications and Relation to Prior Work

Compared with earlier research that focuses on model architecture or graph-based code representations, our results highlight that prompt structure alone can shift model behavior by a considerable margin. Structured prompts narrow the performance gap between general-purpose LLMs and specialized vulnerability detectors, even though the underlying model remains unchanged. At the same time, the rise in false alarms under verbose prompts limits how much structure can be added without harming practical use [23]. These findings support the need for fixed, tested prompt templates and emphasize that prompt clarity is as important as model selection. Prompt engineering cannot replace static analysis or robust training data, but it is a low-cost measure that improves early-stage vulnerability review in multi-language settings [24].

4.Conclusion

This study shows that prompt structure has a clear effect on how large language models identify software vulnerabilities. Prompts with short sections and simple reasoning steps improve true-positive rates across the five CWE categories, while long or dense prompts raise false-positive rates. The three models tested do not react in the same way, which means that prompt format must be selected with the model in mind. These findings show that consistent prompt design is important when LLMs are used for security checks. The approach can help early screening but cannot replace static analysis or manual review. Future work should extend the dataset, include more code frameworks, and examine how prompt structure interacts with tools that can also repair insecure code.

References

Siddiq, M. L., & Santos, J. C. (2022, November). SecurityEval dataset: mining vulnerability examples to evaluate machine learning-based code generation techniques. In Proceedings of the 1st International Workshop on Mining Software Repositories Applications for Privacy and Security (pp. 29-33).

Yang, M., Wang, Y., Shi, J., & Tong, L. (2025). Reinforcement

Learning Based Multi-Stage Ad Sorting and Personalized Recommendation System Design.
Croft, R., Xie, Y., Zahedi, M., Babar, M. A., & Treude, C. (2022). An empirical study of developers' discussions about security challenges of different programming languages. Empirical Software Engineering, 27(1), 27.
Bai, W., Xuan, K., Huang, P., Wu, Q., Wen, J., Wu, J., & Lu, K. (2024). Apilot: Navigating large language models to generate secure code by sidestepping outdated api pitfalls. arXiv preprint arXiv:2409.16526.
Lee, K., Kim, J., Kim, D., & Kim, H. (2025). A Systematic Evaluation of Parameter-Efficient Fine-Tuning Methods for the Security of Code LLMs. arXiv preprint arXiv:2509.12649.
Peng, H., Jin, X., Huang, Q., & Liu, S. (2025). E-commerce Intelligent Recommendation Optimization and Personalized Marketing Strategy Based on Big Model.
Coignion, T., Quinton, C., & Rouvoy, R. (2024, June). A performance study of llm-generated code on leetcode. In Proceedings of the 28th international conference on evaluation and assessment in software engineering (pp. 79-89).
Shah, N., Genc, Z., & Araci, D. (2024). Stackeval: Benchmarking llms in coding assistance. Advances in Neural Information Processing Systems, 37, 36976-36994.
Du, Y. (2025). Research on Digital Quality Traceability System for Temperature-Controlled Supply Chain of Foreign Trade Wine Driven by Blockchain and IoT. Business and Social Sciences Proceedings, 4, 57-65.
Mao, Y., Chang, K. M., & Chen, Z. (2026). Research on Frontend-Backend Collaboration and Performance Optimization for High-Concurrency Web Systems.
Plaat, A., Wong, A., Verberne, S., Broekens, J., van Stein, N., & Back, T. (2024). Reasoning with large language models, a survey. arXiv preprint arXiv:2407.11511.
Mao, Y., Chen, Z., & Ma, X. (2026). Research on a Lightweight Full-Stack Edge Execution Optimization Framework Based on Serverless and WebAssembly.
Siddiq, M. L., Ulfat, N., Raihan, N., Santos, J. C., & Zampieri, M. (2025). Multi-Sallm: A Multilingual Security Assessment of Generated Code.
Du, Y. (2025). Research on Deep Learning Models for Forecasting Cross-Border Trade Demand Driven by Multi-Source Time-Series Data. Journal of Science, Innovation & Social Impact, 1(2), 63-70.
Hassan, M. (2025). Measuring the Impact of Hallucinations on

Human Reliance in LLM Applications. Journal of Robotic Process Automation, AI Integration, and Workflow Optimization, 10(1), 10-20.

Hu, W. (2025, September). Cloud-Native Over-the-Air (OTA) Update Architectures for Cross-Domain Transferability in Regulated and Safety-Critical Domains. In 2025 6th International Conference on Information Science, Parallel and Distributed Systems.

Giroh, S. K., Ghosh, P., Jain, A., Paunikar, H. G., Rastogi, A., Yenigalla, P., & Nediyanchath, A. (2025, November). < SYNTACT>: Structuring Your Natural Language SOPs into Tailored Ambiguity-Resolved Code Templates. In Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track (pp. 2367-2376).

Liu, S., Feng, H., & Liu, X. (2025). A Study on the Mechanism of Generative Design Tools' Impact on Visual Language Reconstruction: An Interactive Analysis of Semantic Mapping and User Cognition. Authorea Preprints.

Giroh, S. K., Ghosh, P., Jain, A., Paunikar, H. G., Rastogi, A., Yenigalla, P., & Nediyanchath, A. (2025, November). < SYNTACT>: Structuring Your Natural Language SOPs into Tailored Ambiguity-Resolved Code Templates. In Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track (pp. 2367-2376).

Zhou, Y., Li, Q., Jensen-Page, L., Huang, S., Donavon, B., Nguyen, V., & Narsilio, G. (2025). An AI-Powered Simulation and Optimisation Framework for Energy Infrastructure Investment to Achieve Pareto Optimum Solutions. In Sustainable and Emerging Energy Technology: Challenges, Opportunities, and Perspectives (pp. 199-209). Cham: Springer Nature Switzerland.

Boubdir, M., Kim, E., Ermis, B., Fadaee, M., & Hooker, S. (2023). Which prompts make the difference? Data prioritization for efficient human LLM evaluation. arXiv preprint arXiv:2310.14424.

Sivarajkumar, S., Kelley, M., Samolyk-Mazzanti, A., Visweswaran, S., & Wang, Y. (2024). An empirical evaluation of prompting strategies for large language models in zero-shot clinical natural language processing: algorithm development and validation study. JMIR Medical Informatics, 12, e55318.

Murray, M. D. (2024). Prompt engineering and priming in law. Available at SSRN.

Banitaan, S., Daoud, M., Alquran, H., & Akour, M. (2026).

Foundation Models in Software Engineering: A Taxonomy, Systematic Review, and In-Depth Analysis of Testing Support. Information, 17(1), 73.