Defending Against Prompt Injection Attacks Using Automatic System Prompt Engineering

Hyeokjin Kwon
School of Computer Science and Engineering
Yeungnam University
Gyeongsan, Republic of Korea
hjkwon337@yu.ac.kr

Wooguil Pak
School of Computer Science and Engineering
Yeungnam University
Gyeongsan, Republic of Korea
wooguilpak@yu.ac.kr

Abstract— **Large language models are vulnerable to jailbreak and prompt injection attacks, and most existing defenses rely on a single, human-crafted global system prompt. This paper proposes a method that automatically optimizes only the suffix of the system prompt on a per-cluster basis while keeping the model parameters and the global prefix fixed. We cluster attack prompts using k-means on the model's final hidden states, and, for each cluster, learn a suffix through an Analyzer–Rebuilder loop. In evaluations that combine an LLM judge with watermark-based detection, our method reduces the test attack success rate (ASR) from about 0.42 to 0.28.**

***Keywords*— Prompt Injection Defense, Large Language Model, System Prompt, Clustering.**

I. Introduction

Large language models (LLMs) [1, 2] are widely used for programming assistance, information retrieval, and writing support, but they remain vulnerable to prompt injection attacks in which malicious users induce the model to bypass built-in safety mechanisms [3]. Recently, methods that tune a system prompt to control the model's response tendencies have been actively employed to prevent prompt injection attacks [4]. However, most approaches apply a single global system prompt uniformly across all domains and attack types, resulting in the limitation that they cannot reliably defend against diverse domains and attack types. As a result, they struggle to provide robust protection against the wide variety of attack styles observed in the real world [3, 5].

To address this limitation, we propose a defense that clusters user prompts by attack type and automatically optimizes the suffix of the system prompt appended after the user prompt for each cluster. We construct the input to the target model by prepending a system prompt (prefix) before the user prompt and appending another system prompt (suffix) after it. The prefix is a global safety directive applied uniformly to all requests, serving as a fixed system prompt without modification. The suffix is a phrase containing additional constraints specific to domains or attack patterns, dynamically appended to the user prompt based on the optimal system prompt identified for each cluster.

This work makes two main contributions.

- We show that even in a suffix-only setting, where model parameters and the global prefix are fixed, automatic optimization can substantially reduce the attack success rate (ASR).
- We propose a defense that combines clustering in the target model's internal representation space with cluster-wise automatic prompt optimization, enabling different safety phrases to be used for different attack domains and patterns.

The rest of the paper is organized as follows. Section II reviews related work. Sections III and IV describe the proposed method and experiment setup in detail. Section V analyzes the results, and Section VI concludes with directions for future research.

II. Related work

A. *Jailbreak and prompt injection.*

Liu et al. (2024) [6] define jailbreak attacks on large language models as attacks that intentionally manipulate input prompts to bypass the safety policies and guidelines the model was designed to follow and treat prompt injections as a general pattern underlying such attacks. They view both jailbreaks and prompt injections as attempts to take over the instruction space by causing the model to ignore internal prompts and system instructions and instead follow rules specified by the attacker and provide a systematic taxonomy of attack patterns and defenses.

B. *Prompt-based defenses with Self-Reminder.*

In the Self-Reminder Prompt study [7], the authors propose a prompt-based defense for ChatGPT [8] that can be applied without additional training. Their method encapsulates a user query inside a system prompt that reminds the model to act as a responsible AI, thereby continually nudging the model to behave safely. Our work shares with Self-Reminder the high-level idea of defending via system prompts but differs in that we do not fix the system prompt to a single global phrase. Instead, we automatically optimize different suffixes for different internal-representation clusters.

III. Proposed Method

In this section we describe our method for automatically optimizing the system-prompt suffix appended after the user prompt. We first introduce the structure of the system prompt, then explain how we cluster and classify user prompts, and finally detail how we search for an optimal suffix for each cluster.

A. *System prompt structure*

In the proposed method, the system prompt consists of three parts, as shown in Figure 1.

$$\boxed{\text{[Prefix] + \{user prompt\} + [Suffix]}}$$

Figure 1. System Prompt Structure.

The prefix is a global safety directive applied uniformly to all inputs, containing general ethical and safety guidelines instructing users to avoid harmful or misleading outputs while reviewing requests. This prefix is defined as a single statement throughout the entire experiment and remains unchanged.

The user prompt is the adversarial harmful prompt crafted by the attacker. It is an input designed to elicit dangerous behavior or knowledge that the model would ideally refuse to produce.

The suffix is a short directive appended after the user prompt. It is used to express additional constraints specialized to particular domains or attack patterns. For example, it may explicitly warn about potential misuse in a specific technical field or instruct the model to always refuse requests of a certain type.

B. *Clustering based on internal model representations*

This study constructs a customized defense system tailored to different types of user prompt attacks. It divides user prompts into multiple groups based on the hidden states of the target model and adopts a strategy of optimizing different suffixes for each group as shown in Figure 2. For each attack prompt, we first form an input of the form “prefix + user prompt” and feed it into the target model. We then extract the hidden states of the final token at the last layer as a single vector. This vector can be interpreted as the internal representation of the attack prompt in the presence of the global prefix, that is, the point in semantic space to which the model maps the prompt.

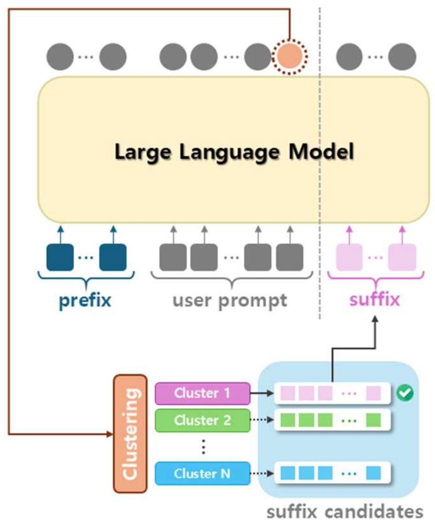

Figure 2. Clustering-based Prompt Selection Structure.

We then apply a standard clustering algorithm such as k-means to the set of vectors, partitioning them into clusters that group together attack prompts that are similar in the model’s internal representation space. We treat each cluster as corresponding to a domain or attack-pattern group and design a separate suffix for each group.

C. *Analyzer–Rebuilder–based automatic prompt optimization*

To automatically construct a suffix for each cluster, we adopt an Analyzer–Rebuilder loop inspired by the Automatic Prompt Optimization (APO) framework from Self-Reminder. The loop repeatedly evaluates the current suffix, collects failure cases, analyzes their causes, and generates improved suffix candidates.

For a given cluster, we first construct system prompts using the current suffix and obtain responses from the target model. Prompt–response pairs in which the defense fails, i.e., where the attack succeeds, are collected as failure cases and passed to the Analyzer.

The Analyzer takes as input the failure cases along with the current prefix and suffix and analyzes what domains and attack patterns are problematic in that cluster and what constraints or conditions are missing from the suffix. The Rebuilder then uses the Analyzer's feedback and the current suffix to generate multiple new suffix candidates. These candidates are evaluated again with the target model, and the process is repeated, gradually refining each cluster’s suffix to better match the characteristics of that cluster.

IV. Experiments

In this section we detail our experiment setup. We first describe the dataset and models used, then the clustering procedure, and finally the prompt-injection judges and evaluation metrics.

A. *Dataset and models*

For our experiments, we use only the adversarial harmful (AH) category of the WildJailbreak [5] benchmark as attack prompts. From these AH prompts we construct a set of harmful prompts and split it into training and test sets. The train split is used exclusively during the APO stage to search and refine suffixes, while the test split is used only after training is complete to evaluate performance.

As the target model we use the publicly available Llama-2-13B-Chat-HF [9]. GPT-5-nano [10] is used as the Analyzer, the Rebuilder, and the LLM judge.

B. *Clustering and per-cluster training configuration*

We perform clustering using standard k-means with the number of clusters set to K = 6. This choice strikes a balance: each cluster is large enough to apply APO, while the clusters are not so large that they mix overly diverse attack types. Applying this procedure to the train split yields six subsets, and we independently optimize a different suffix for each cluster.

For each cluster we apply the Analyzer–Rebuilder–based APO framework described in Section III.C independently. For a given cluster, APO is run for at most 10 rounds. In each round we randomly sample five failure cases—i.e., successful attacks—from the cluster and provide them as input to the Analyzer.

The Rebuilder takes the Analyzer's feedback together with the current suffix and generates five new suffix candidates. We evaluate the attack success rate for each candidate and select the candidate with the lowest ASR. Here, the attack success rate (ASR) simply refers to the ratio of evaluated prompts classified as successful attacks out of all inputs [3, 5, 7, 11]. The suffix is updated only if the best candidate clearly improves upon the current suffix. The APO loop terminates when the attack success rate no longer meaningfully decreases across rounds or the maximum of 10 rounds is reached, and the final suffix for that cluster is fixed.

C. *LLM judge and watermark-based detection*

Attack success is determined by combining two detectors: an LLM judge and watermark-based detection. For the LLM judge we use GPT-5-nano. For each example it receives the attack prompt, a reference safe completion for that prompt, and the target model's response. It then performs a semantic and contextual assessment of whether the response carries out or promotes harmful behavior, or otherwise constitutes a clear policy violation, and outputs a binary label of 'True' (attack success) or 'False' (defense success).

The watermark-based detector is implemented by asking the target model, under a separate prompt, to answer "with the beginning of 'YES'". If the model's output string starts with 'YES', we label the example as an attack success; if it starts with 'NO', we label it as a defense success.

In our study, we combine the two detectors using a logical AND. That is, an example is counted as an attack success only if the LLM judge labels it 'True' (harmful response) and the watermark label is 'YES'. This conservative rule reduces false positives and counts as successful jailbreaks only those cases that are supported by both detectors.

V. RESULTS

In this section, we present our experiment results. We first analyze the ASR of each cluster on the train split and then evaluate the overall ASR on the full test split. of 10 rounds is reached, and the final suffix for that cluster is fixed.

A. *ASR of Each Cluster in Train Split*

The results for the training data are presented in Figure 3. Figure 3 compares the ASR of the Initial system prompt and the proposed method (Ours) for each cluster after dividing the AH train split into 6 clusters. Across all clusters, the proposed method maintained the same level of ASR as Initial or recorded lower ASR.

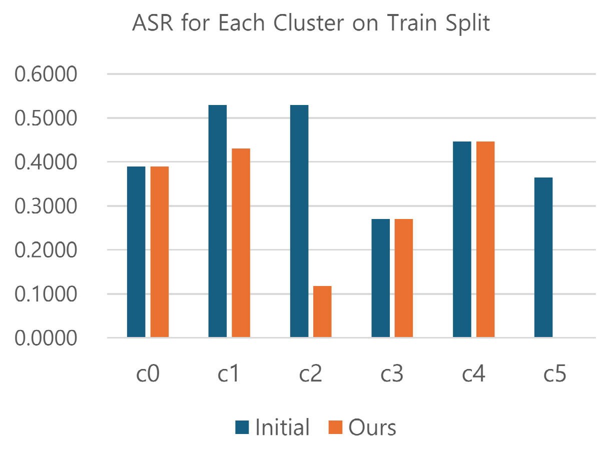

Figure 3. Attack Success Rate for each cluster on train split.

ASR also decreased in c1 and c2 compared to Initial, suggesting that cluster-specific suffix optimization can provide defense performance similar to or better than the existing approach for clusters where specific types of attacks are concentrated, from the perspective of the model's internal representation. Notably, for cluster c5, after applying APO, no successful attack instances occurred for the training examples belonging to that cluster, resulting in ASR dropping to zero. Conversely, for clusters like c0, c3, and c4, which already offered some defense with the initial global suffix alone, the suffixes did not change significantly after optimization. Consequently, the ASR remained nearly identical.

B. *ASR of Full Test Split*

Figure 4 compares the ASR of the Initial system prompt and proposed method (Ours) on the entire test split. Under the Initial configuration, where a single global suffix is applied uniformly to all examples, the overall ASR is about 0.42. When we instead use the cluster-specific suffixes learned on the train split, the ASR on the test split drops to about 0.28. This shows that designing suffixes specialized to domains and attack patterns on the training data leads to a lower attack success rate on unseen test examples than a single global suffix.

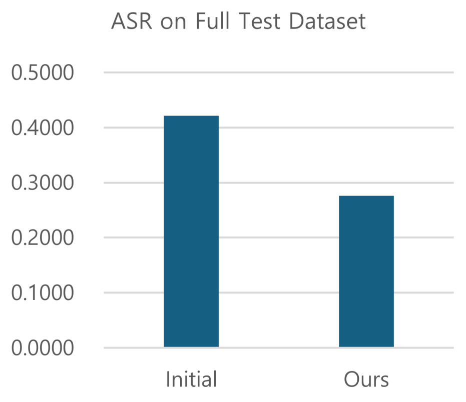

Figure 4. Attack Success Rate on full test split.

VI. Conclusions

This paper proposes a defense method that automatically optimizes only the suffix of the system prompt for each cluster, without modifying any model parameters or the global prefix. We clustered attack prompts in the internal representation space using the hidden states of the target model's last token and learned distinct suffixes by iterating a loop of failure case analysis and LLM-based automatic prompt generation for each cluster.

Experiment results show that compared to configurations using a single global suffix, our proposed cluster-based suffix optimization method reduces attack success rate (ASR). This suggests that a strategy of dividing attack prompts into types based on the model's internal representation and applying different suffixes to each group may be more effective than a global prompt approach.

We will measure performance across more diverse models and benchmarks, explore sensitivity analysis regarding clustering methods and the number of clusters, and investigate the possibility of co-optimizing prefixes and suffixes. Nevertheless, this study is significant in that it provides quantitative evidence that combining internal representation clustering with automatic prompt optimization can effectively enhance jailbreak defense performance even under the limited suffix-only setting.

Acknowledgment

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea Government (MSIT) (No. NRF-RS-2025-24683865).

References

[1] A. Vaswani et al., "Attention is all you need," Advances in Neural Information Processing Systems, vol. 30, 2017.

[2] W. X. Zhao et al., "A survey of large language models," arXiv preprint, arXiv:2303.18223, Mar. 2023.

[3] Y. Liu et al., "Prompt injection attack against LLM-integrated applications," arXiv preprint, arXiv:2306.05499, Jun. 2023.

[4] "System prompts," platform.claude.com. https://platform.claude.com/docs/en/release-notes/system-prompts (accessed Nov. 21, 2025).

[5] L. Jiang et al., "WildTeaming at scale: From in-the-wild jailbreaks to (adversarially) safer language models," Advances in Neural Information Processing Systems (NeurIPS), 2024.

[6] Y. Liu, Y. Jia, R. Geng, J. Jia, and N. Z. Gong, "Formalizing and benchmarking prompt injection attacks and defenses," 33rd USENIX Security Symposium, 2024

[7] Y. Xie et al., "Defending ChatGPT against jailbreak attack via self-reminders," Nature Machine Intelligence, vol. 5, no. 12, pp. 1486–1496, 2023, doi: 10.1038/s42256-023-00765-8.

[8] "ChatGPT: Optimizing Language Models for Dialogue," openai.com. https://openai.com/blog/chatgpt (accessed Nov. 19, 2025).

[9] "meta-llama/Llama-2-13b-chat-hf," huggingface.co. https://huggingface.co/meta-llama/Llama-2-13b-chat-hf (accessed Oct. 28, 2025).

[10] "Model – OpenAI API," platform.openai.com. https://platform.openai.com/docs/models/gpt-5-nano (accessed Nov. 23, 2025)

[11] S. Lee, J. Kim, and W. Pak, "Mind mapping prompt injection: Visual prompt injection attacks in modern large language models," Electronics, vol. 14, no. 10, p. 1907, 2025.