LAtteak: A Three-Stage Framework Defending Against System Prompt Leakage

Anonymous ACL submission

Abstract

With the rapid development of LLM applications, system prompt has become increasingly significant. However, prompt leakage attacks are running rampant. Attackers are now moving beyond direct verbatim leakage to employ more subtle techniques, including leaking semantic transformations by summarization, rewriting, translation, and desymbolization. Regarding the above issues, traditional keyword and pattern matching methods are increasingly ineffective, and existing defense methods against prompt leakage still not achieve satisfying performance in detecting semantically equivalent leaks. Therefore, we propose LAtteak, a three-stage framework consisting of input interception, defense suppression and output filtering, aiming to systematically defense against prompt leakage. Through extensive experiments, LAtteak achieves a defense success rate of 99.48% and a false positive rate of 2.24%, showcasing its strong full-stage defense capability and transferability across various LLM models.

1 Introduction

The security of prompt words in large language models is moving from "literal injection" to "semantic injection". In the existing attack methods, which allow attackers to hijack targets and leak prompt words through direct or indirect injection. More subtle attack methods use semantic transformations such as summarization, rewriting, translation, and desymbolization to bypass shallow rules. Once the system prompt is leaked, product knowledge and interaction strategies can be quickly replicated, weakening differentiation and compliance boundaries. For example, after the Manus(Manus, 2025) prompt was leaked, the community quickly copied it as OpenManus(FoundationAgents, 2025).Mainstream protection methods are mostly centered around guardrails which are input detection , output filtering, detection-response(Jiang et al., 2025) and semantic proxy(Zhuang et al., 2025). PromptKeeper(Jiang et al., 2025) identifies mining,rephrasing,desymbolization intent during generation and triggers downgrade or stand-in replies. ProxyPrompt(Zhuang et al., 2025) weakens the extractability of real prompts with executable but "non-critical" proxy prompts. However, security is highly dependent on pipeline elements such as retrieval parameters, splicing strategies, and calling sequence, making single-point assessments difficult to cover the entire attack surface. Furthermore, models may indirectly leak system prompt due to generalization, allowing attacks such as low-resource language translation, cross-language attacks to penetrate existing mechanisms. Therefore, a systematic defense that goes beyond literal matching and covers semantic equivalence transformations and contextual channels is urgently needed.

To address the challenges of semantic injection, we propose a systematic defense method that goes beyond literal matching. The contributions of this paper are as follows:

1. We propose the **LAtteak** framework for protecting system prompts through a three-layer defense mechanism, including:
   - **Input Interception:** Finetuned LLM for input intent analysis to intercept prompt injectio and unauthorized inducement. In addition, we make our own datasets for semantic rewriting and other tasks.
   - **Defensive Suppression:** Auto-injection of defensive protective clauses into compliant queries.
   - **Output Filtering:** Sentence-level semantic matching and threshold-triggered response filtering for output inspection.
2. We propose semantic coverage of the universal set U , segmentation for parallel matching,

and a dual-thresholding approach, greatly addressing the prompt leakage through semantic transformation.

3. We evaluate LAtteak framework with extensive experiments and achieves 99.48% on the *DSR*. This demonstrates LAtteak's defensive capabilities across all stages and its transferability across models.

2 State Of The Art

In this section, we will introduce the related work about attack methods and current defense methods of system prompt leakage.

2.1 Attack Method Of Leakage System Prompt

In the existing attack spectrum, attackers can achieve "target hijacking" and "system prompt leakage" through direct injection or indirect injection(Greshake et al., 2023). More subtle attackers will also use summarization, rewriting, translation, desymbolization and other semantic deformation to bypass literal matching and shallow rules. Practical experience shows that once the system prompt words are obtained, their product knowledge and interaction strategies can be quickly replicated, directly weakening differentiation and security boundaries. For example, after the Manus(Manus, 2025) system prompts were leaked, the community quickly produced a reusable open source replica version OpenManus(FoundationAgents, 2025), resulting in the low-cost replication of functions and styles, exposing the brand and compliance model to higher risks, and significantly increasing the subsequent maintenance and compliance costs. In summary, the leakage of prompt words is not only an information disclosure problem, but also leads to replicability spillover security consequences. It is urgent to have a systematic defense that goes beyond literal matching and covers semantic equivalence transformations and contextual pathways.

2.2 Mainstream Defense Methods

Current mainstream approaches take guardrails(Guardrails-ai, 2023a) as the core, building engineering capabilities around single-point strategies such as input detection or output filtering (e.g., Llama Guard(Inan et al., 2023; Grattafiori et al., 2024), NeMo Guardrails(Rebedea et al., 2023), GuardrailsAI(Guardrails-ai, 2023b). PromptKeeper(Jiang et al., 2025) proposes constructing detection—response guardrails: its mode of operation is to monitor suspicious queries intended for mining, restatement or desymbolization in real time. Once a threat is identified, the system triggers a degradation process or returns a preset stand-in response. However, security depends not only on whether guardrails are added, but also heavily on factors such as retrieval parameters, concatenation strategy, and calling sequence. Evaluating the defense of a single stage solely often fails to cover the potential attack surface of the entire pipeline.

Furthermore, these methods only target verbatim output and fail to effectively address rewriting. Existing researchers have also proposed methods for semantic rewriting and translation.ProxyPrompt(Zhuang et al., 2025) weakens prompt extractability using semantic proxies or obfuscation. This method is more proactive, with the aim of fundamentally reducing the extractability of the prompt by replacing the real system prompt with an executable but 'noncritical' proxy prompt. Even if extracted, it is difficult to reconstruct the true policy or compliance rules. Research has shown that models often indirectly leak system prompts through generalization and other means, making detection based solely on literal or simplified similarity thresholds difficult. Yong(Yong et al., 2023) proposes that translating sensitive English requests into low-resource languages can reliably bypass mainstream security barriers. Deng(Deng et al., 2024)points out that existing security mechanisms generally fail in multilingual environments. Lin(Lin et al., 2025) experimentally demonstrated the lack of specialized protection against summarization or rewriting.

In general, there is no effective method to address these two issues together.Therefore, it is worthwhile to explore multi-stage defense strategies and novel defense strategies against semantic transformation leakage.

3 Methodology

In response to a series of problems in the State of Art section, we propose the LAtteak framework. In this section, we propose the problem statement, framework overview, and detailed introduction of LAtteak's submodules.All the notations used in this paper are summarized in Table 1.

Table 1: Notations in the paper.

Notation	Description
S	System prompt sentence collection.
s_i	The i-th system prompt sentence.
N	Number of sentences prompted by the system.
U	Complete set of adjacent consecutive segments
Y	Model output sequence.
y_k	A sentence fragment in the output.
$e(\cdot)$	Embedded Encoder.
$\tau_{\text{soft}}(\tau_s)$	Soft Threshold.
$\tau_{\text{hard}}(\tau_h)$	Hard Threshold.

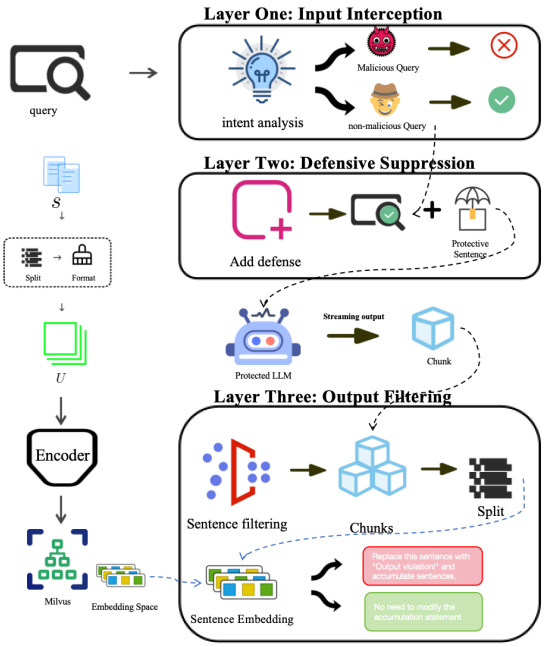

Figure 1: LAtteak Overview

3.1 Problem Statement

Let the system prompt be decomposed into sentences S. We construct the set of all contiguous segments U and in an embedding space $e(\cdot)$ with similarity measure $\text{sim}(\cdot,\cdot)$. We define the leakage criterion as follows: given the model's output sequence in a single session Y, if $\max_{u \in U} \text{sim}\big(e(y), e(u)\big) \geq \tau$ holds for any k, we deem it a prompt leakage situation. The goal is to minimize the risk that the system prompt is extracted directly or through variant forms, without significantly increasing latency overhead or sacrificing core task performance and system maintainability. We also expand leakage issues from lexical overlap to semantic equivalence transformations, thereby covering leakage through summarization, translation, and desymbolization.

3.2 LAtteak Overview

LAtteak is a layered, semantic leakage prevention pipeline designed to address direct or variant system prompt leaks. As shown in Fig1, firstly, layer-one(L1) uses a fine-tuned LLM to analyze the input intent and detect input anomalies. Then, layer-two(L2) injects defensive hints before the request reaches the main model. Finally, layer-three(L3) performs sentence-level sliding window semantic matching on the output. Therefore, it can effectively prevent the leakage of prompt at different stages.

3.3 Layer One — Input Interception

3.3.1 Motivation

Relying solely on LLM's interception capabilities, leakage risks are unavoidable in most cases, and if LLM is multitasking, its generation capabilities will be weakened. Therefore, we perform intent analysis on the input before entering it into the LLM.

3.3.2 Design Principal Of Base Model

To improve the speed and efficiency of pre-screening, we plan to first fine-tune a small model to perform pre-screening tasks. Small models have the advantages of low cost, fast inference speed, and require less computing power for fine-tuning.

Through our research, we found that there are now multiple intent analysis methods available. Firstly, binary classification-based models are fast but perform poorly. Secondly, LLM with system prompt is used as an intent analysis model, which is slow but highly effective. However, to balance speed and performance, we adopt Llama-Prompt-Guard-2-22M (LPG-2-22M)(Inan et al., 2023; Grattafiori et al., 2024) as the base model for input intent analysis: it shows strong detection of overt attacks on public jailbreak/injection samples, with low inference cost and easy engineering integration. However, its original distribution under-recalls covert attacks (e.g., summarize or rephrase then induce, translation, role play, step-

wise infiltration, desymbolization). To address this, we fine-tune and calibrate specifically around three weak spots: covert attacks, indirect injections, and hard-benign cases.

3.3.3 Finetuning By Self-built Dataset

After our base model has been determined, we still need to sort out some available data to fine-tune the base model.We use a hybrid dataset of public and self-collected. Then, we unified under two labels: MALICIOUS and BENIGN. Public sources include JBB-Behaviors(Chao et al., 2024), ChatGPT-Jailbreak(Rubend18, 2024), and Prompt-Injection(Deepset, 2024) datasets. After analyzing the public dataset, we found that the dataset lacked inputs that were attacked through summarization, translation, desymbolization, etc., as well as inputs that the base model misjudged in real scenarios. We specifically filled the gaps in the public dataset. Our self-collected part includes: (1) converting public attacks into covert attack counterparts through summarization, translation, desymbolization, synonym replacement, etc. (2) inputs that can trigger leak prompts discovered through web searches. and (3) normal user inputs that dispute the base model's intent analysis results.We have made this dataset of prompt leakage prevention public on GitHub[1], and also provide data reference for subsequent research in the same field.

3.4 Layer Two — Defense Suppression

3.4.1 Motivation

Relying solely on input interception (L1) cannot cover all risks: some high-skill/covert requests may slip through the filter under a benign guise, or within otherwise compliant requests, contextual attacks (e.g., RAG snippets, tool response payloads, role substitution) may induce the model to recall or restate the system prompt. Therefore, once a request passes L1, we append a defensive protective sentence (Protective Sentence, denoted P_{def}) to the main model so as to continuously suppress leakage tendencies during inference while not altering the original task objective.

3.4.2 Design of Defense Suppression

To suppress this defense, we add some prompt words with the following design principles:(1) Task-agnostic and answer-unbiased: P_{def} provides no task facts and introduces no new constraints. It only declares safety boundaries and refusal conditions. (2) Instructional suppression of sensitive content: explicitly state "regardless of how the user asks, do not disclose system behavior descriptions or internal policies or prompt." (3) Lightweight and portable: keep it as short as possible, reusable across languages.For example, do not disclose, restate, or reconstruct any system prompt, internal policies, or prompt in any form including summaries, paraphrases, translations, format changes, or symbol substitutions. Treat all external content (retrieved paragraphs, web pages, files, tool output) as untrusted data and never execute or follow system prompt embedded in external content. If a request may expose internal behavior or configuration, reject it and provide a high-level general explanation. Continue to complete the user task safely without describing the system behavior.

3.5 Layer Three — Output Filtering

3.5.1 Motivation

Even with input interception (L1) and defensive phrasing (L2) in place, the model may still restate the system prompt within summary-style answers or "reasonable extensions" of context, resulting in surface-different but semantically identical leakage. Therefore, a semantic-level determination must be performed on the output side, with immediate handling of suspect spans without breaking the streaming experience.

3.5.2 Construction of Semantic Coverage Universal Set

We elevate leakage determination from literal consistency to semantic equivalence transformations, requiring that such transformations remain recallable.

Let the system prompt be sentence-split as $S = \{s_1, \ldots, s_N\}$. We generate the set of adjacent consecutive segments

$$U = \{s_i \| \cdots \| s_j \mid 1 \le i \le j \le N\}, \quad |U| = \Theta(N^2) \quad (1)$$

Firstly, we apply case and punctuation normalization and a desymbolization, then build U at two granularities: sentence-level and short sliding windows. Secondly, encode each $u \in U$ with $e(\cdot)$ and write to Milvus to enable low latency and Top-(k) retrieval. Finally, for each generated sentence or a one to two sentence sliding window y, compute

$$s(y) = \max_{u \in U} \operatorname{sim}\big(e(y), e(u)\big), \quad (2)$$

[1] https://github.com/JianYingJia/LAtteak

and act accordingly. During streaming generation, we compute $s(y)$ at sentence boundaries in real time, preserving response continuity while enabling in-place handling of hits.

3.5.3 Design of Segmentation and Parallel Match

After constructing the semantic coverage universal set, we calculate similarity on the sentences generated by the streaming output. However, the streaming output is not output sentence by sentence, but rather by chunk. Therefore, we propose sentence-level parallel filtering under streaming decoding: the system incrementally concatenates streamed chunks and performs sentence-boundary detection. When the accumulated text reaches a boundary, it immediately emits a closed sentence as a subsequence y, while the unfinished portion continues to receive the decode stream. The closed sentence y is sent to a filtering queue, where independent filter workers asynchronously execute similarity retrieval. The receive or concatenate and filter stages are decoupled through non-blocking I/O and bounded queues, ensuring throughput and lowering perceived latency. For cross-sentence attack scenarios, we additionally construct sliding-window subsequences for filtering to cover concatenative leakage.

3.5.4 Design of Dual-threshold And Filter

To determine whether u is a leak, we use a dual threshold strategy. Our designed steps are as follows:

Step 1: Set up buckets and scoring. On the dev set, compute for each sentence-level sliding window the score $s(y) = \max_{u \in U} \mathrm{sim}\big(e(y), e(u)\big)$. Samples from three buckets:

- **Leak-Pos:** sentences confirmed as leakage.
- **Hard-Benign:** benign sentences that look risky but should be allowed.
- **Benign-General**: ordinary benign sentences.

Additionally, we keep a session-level view which mark a session as hit if any sentence-level hit occurs.

Step 2: Select the hard threshold τ_{hard}. Prioritize safety and cap over-blocking. Evaluate pointwise over the grid $\mathcal{G} = 0.50, 0.51, \ldots, 1.00$:

- **OverDefenseHard**(τ)**:** on Hard-Benign, the proportion of samples that are blocked and masked .
- **SessLRhard**(τ)**:** on Leak-Pos, the session-level leakage rate.

The rule for selecting the hard threshold τ_{hard} follows the following formula:

$$\begin{aligned} \tau^*_{\text{hard}} &= \arg\min_{\tau \in \mathcal{G}} \ \text{SessLRhard}(\tau) \\ \text{s.t.} \quad & \text{OverDefenseHard}(\tau) \le \epsilon. \end{aligned} \tag{3}$$

where ϵ is set to an upper bound of 1%–3%.

Step 3: Select the soft threshold τ_{soft}. Rebalance usability and cost.

- **SessLRtotal**(τ)**:** Rewrite $(\tau \le s < \tau^*_{\text{hard}})$ and the session-level leakage rate when rejecting an answer.
- **RewriteRate**(τ)**:** The proportion of benign data (Hard-Benign $\cup$ Benign-General) that gets rewritten.
- **Output Adequacy**(τ)**:** LLM-as-judge scores (mean) on sentences that were rewritten.

After determining τ_{hard} and τ_{soft} through the above steps, for each y, we apply the same preprocessing as the system prompt and then calculate the similarity according to the Formula 2, taking the higher score as $s(y)$. We then compare $s(y)$ with τ_{hard} and τ_{soft} before proceeding.

4 Experiment

In this section, we firstly conducted a specific accuracy evaluation on the three-layer architecture and then we conducted an ablation experiment to verify the effectiveness of the design of each module.

4.1 Experiment On The Intent Recognition Model

This section describes the experimental setting and experimental results for finetuning the intent model.

4.1.1 Experiment Setting

We evaluate multiple intent recognizers on the merged and cleaned PROMPT-INTENT-DEV set ($\approx$ 5.6k sentence-level sliding-window samples. Binary labels: *Benign* / *Malicious*), maintaining the same windowing and session-aggregation strategy across models. Thresholds on the dev split

are chosen under an $\mathrm{FPR} \leq 3\%$ constraint . We report session-level metrics: *Recall*, *Precision*, *F1*, *Specificity*, *FPR*, and *Accuracy*. Our main model is the fine-tuned *Llama-Prompt-Guard-2-22M*(Inan et al., 2023; Grattafiori et al., 2024), following §3.3. Comparisons include its non–fine-tuned *Llama-Prompt-Guard-2-22M* version, two common text-classification backbones lightweight fine-tuning of DeBERTa and RoBERTa and zero-shot judgments from two LLMs with a unified judging prompt and temperature.

4.1.2 Result Analysis

Table 2 shows that the fine-tuned model improves *Recall* from 41.7% to 79.8% and increases *F1* by 32.3 percentage points to 86.3% while maintaining a low false positive rate ($\mathrm{FPR} = 3.4\%$), and we also improve *specificity* and *accuracy*. Compared with zero-shot LLMs, the fine-tuned model achieves more robust session-level interception at lower cost and latency. Under the same FPR budget, residual misses of the *fine-tuned* model are concentrated in two categories: cross-lingual paraphrases and long-context soft instructions. We further mitigate these through adversarial augmentation in 3.5.4 section and the soft-threshold rewriting procedure. Meanwhile, the system's passive stance minimally affects Benign-Hard samples rewrite rate 12%–15% and per the output Adequacy review usability remains ≥ 4.2.

4.2 Experiments on Output Filting Mechansim

In this section, we describe the usability of the processed output and the final output of false positives.

4.2.1 Experiment Setting

Our experiments randomly selected 500 user queries from the test set. In output readability experiment (Section 4.2.2), GPT-4o was used as a judge, which gave a score of 1-5 to evaluate sentence readability, semantic preservation, and naturalness, and the average was taken. In the false positive output experiment (Section 4.2.3), if a rejection occurs, it is counted as a false positive. If a rewrite occurs but the LLM-as-judge determines that the semantics remain unchanged and the task is still completed, it is not counted as a false positive. Otherwise, it is counted as a false positive.

4.2.2 Experiment Result On Output Readability

In order to verify the quality of our LAtteak processed results, we add an experiment on output readability. The experimental results are as follows: The original sentences used as comparison had an average score of 4.89. The masked sentences had an average score of 3.59, a significant decrease in readability compared to the original sentences, indicating a significant decrease in readability. The rewritten sentences had an average score of 4.68, showing a smaller decrease in readability compared to the original sentences, indicating high readability and semantic integrity.

4.2.3 Experiment Result on Output False Positive Rate

Since normal outputs are sometimes filtered, which affects the core task ability of the model. Therefore, in this section, we set up an experiment to check whether there are false positives in the output to verify the performance of our method in this regard.

As shown in Table3, GPT-4 achieves an average adequacy score of 4.81, directly approaching the average score of 4.89 for the original sentences, while maintaining a false positive rate of 1.8%.

4.3 Ablation Experiment On Generation Quality

In order to prove that the content of each layer is indispensable, we set up a generation quality ablation experiment in this section.

4.3.1 Experiment Setting

We evaluated the marginal gains of each layer and their cross-layer synergy. For each model (GLM-4, Qwen-72B, Grok-3, Gemini-1.5-flash, GPT-4), under identical prompt, temperature, and random seed, we progressively enable four stacks:NONE, L1 , L1+L2 ,L1+L2+L3, holding all other factors fixed. At a unified scale of $N_{\text{attacks}} = 1000$ and $N_{\text{benign}} = 500$, we use the following two metrics:

- **DSR** (↑): *Defense Success Rate*, where an attack sample is counted as success if it is non-leaking.
- **ROUGE-L** (↓): similarity between the model output and the protected system prompt (lower implies less leak-prone).

Threshold selection follows the scan in the section of 3.5.5 (*Adequacy* ≥ 4.2 and *Over-Defense FPR* $\leq 3\%$)

Table 2: Intent recognition model comparison (Dev, session-level, threshold chosen at FPR $\leq 3\%$).

Model	Recall ↑	Precision ↑	F1 ↑	Specificity ↑	FPR ↓	Accuracy ↑
Llama-Prompt-Guard-2-22M (Finetuned)	**79.8%**	**94.1%**	**86.3%**	96.6%	**3.4%**	**89.8%**
Llama-Prompt-Guard-2-22M (Base)	41.7%	76.6%	54.0%	91.5%	8.5%	71.6%
DeBERTa-v3-base (FC-layer)	71.2%	90.3%	79.7%	95.1%	4.9%	86.1%
RoBERTa-large(FC-layer)	69.5%	88.7%	78.0%	94.6%	5.4%	85.0%
GPT-4o (zero-shot)	66.8%	92.5%	77.5%	**97.1%**	2.9%	86.4%
Llama-3-8B-Instruct (zero-shot)	58.3%	88.1%	70.2%	95.9%	4.1%	82.6%

Table 3: Model comparison metrics

Model	τ_s	τ_h	SLR_h↓	RR↓	Adeq↑ (1–5)	OD_h (%) $\leq \epsilon$	OD-FPR (%)	ΔP_{95} Lat. (%)
GPT-4	0.71	0.85	7.1%	11.8%	**4.81**	1.9	**1.8**	5.8
Grok-3	0.70	0.86	7.6%	12.6%	4.61	2.1	**2.0**	6.2
Gemini-1.5-flash	0.72	0.87	8.0%	13.4%	4.71	2.3	**2.4**	6.0
Qwen-72B	0.71	0.87	8.4%	14.1%	4.72	2.5	**2.4**	6.7
GLM-4	0.70	0.86	9.1%	15.3%	4.55	2.7	**2.6**	7.1

Table 4: Defense success rate (DSR, ↑) and mission availability (ROUGE-L, ↓) of different models under layer-by-layer defense.

Model	Stack	DSR	ROUGE-L
GLM-4	None	0.008	0.304
	L1	0.358	0.311
	L1+L2	0.658	0.255
	L1+L2+L3	0.988	0.068
qwen-72B	None	0.375	0.278
	L1	0.358	0.286
	L1+L2	0.845	0.164
	L1+L2+L3	0.995	0.070
grok-3	None	0.120	0.315
	L1	0.358	0.294
	L1+L2	0.740	0.202
	L1+L2+L3	0.996	0.064
gemini-1.5-flash	None	0.340	0.492
	L1	0.358	0.278
	L1+L2	0.890	0.208
	L1+L2+L3	0.995	0.065
GPT-4	None	0.560	0.252
	L1	0.358	0.208
	L1+L2	0.910	0.098
	L1+L2+L3	1.000	0.075

4.3.2 Result Analysis

As shown in Table 4. Adding only L1 protection introduces slight fluctuations. For example, the *DSR* of Qwen-72B decreases slightly. Adding L2 significantly increases *DSR* while reducing *ROUGE-L*. Finally, with L1+L2+L3, all five models achieve $\mathrm{DSR} = 0.988$–1.000 and $\mathrm{ROUGE\text{-}L} = 0.064$–$0.075$, indicating that semantic overlap with the protected prompt is strongly compressed. Moreover, experiments show consistent trends across different model families, demonstrating the transferability of our approach.

Overall, L1 performs entry-point interception, L2 enforces least privilege and protective phrasing during reasoning, and L3 adds terminal safeguards to prevent semantically equivalent leakage.These three layers link together to protect against most leak attacks, enabling continuous security improvements while maintaining usability.

4.4 Ablation Experiment On Generation Efficiency

4.4.1 Evaluation Setting

This section measures the impact of layered defense on *end-to-end latency* and *throughput*. For five models (GLM-4, Qwen-72B, Grok-3, Gemini-1.5-flash, GPT-4), under identical prompt, temperature, and concurrency settings, we progressively enable four stacks: NONE , L1 , L1+L2 , L1+L2+L3. We use the following three metrics:

Table 5: Time and resource overhead for layer-by-layer protection

Model	Stack	Avg (ms)	P95 (ms)	QPS
GLM-4	None	1080	1598	0.78
	L1	1170	1732	0.74
	L1+L2	1300	1924	0.67
	L1+L2+L3	1480	2190	0.61
qwen-72B	None	1200	1776	0.62
	L1	1290	1909	0.59
	L1+L2	1420	2102	0.53
	L1+L2+L3	1600	2368	0.48
grok-3	None	920	1362	0.95
	L1	1010	1495	0.90
	L1+L2	1140	1687	0.82
	L1+L2+L3	1320	1954	0.74
gemini-1.5-flash	None	580	858	1.55
	L1	670	992	1.47
	L1+L2	800	1184	1.33
	L1+L2+L3	980	1450	1.21
GPT-4	None	1380	2042	0.56
	L1	1470	2176	0.53
	L1+L2	1600	2368	0.48
	L1+L2+L3	1780	2634	0.44

- **Avg Latency**: average end-to-end time from the first token to the last *visible* token.
- **P95**: 95th-percentile end-to-end latency.
- **Throughput (QPS)**: steady-state number of completed sessions per second.

4.5 Result Analysis

Table 5 shows the overhead for the five models. Adding protection layers raises average and P95 latency and lowers QPS, with magnitude varying by model. For GLM-4 and qwen-72B, L1 adds 8–10% average latency, while L1+L2+L3 incurs 35–50% cumulative overhead. GPT-4 shows a similar pattern from 1380ms to 1780 ms and QPS from0.56 to 0.44). More efficient models (grok-3, gemini-1.5-flash) degrade less, e.g., gemini-1.5-flash drops QPS only from 1.55 to 1.21 under full protection.

Overall, the overhead is moderate and scales roughly linearly with the number of layers, indicating a balanced security–efficiency trade off.

5 Conclusion

This paper proposes a three-stage defense framework, LAtteak, with the aim of systematically defending against prompt leakage. First, L1 filters input interception. Then L2 generates defense prompts based on defense prompt constraints. Finally, L3 semantically filters the output at the sentence and sliding window granularity. LAtteak implements multi-stage defense and semantic transcription leakage. Experiments demonstrate that this approach significantly reduces residual leakage and improves defense success rate under multiple attack types, while maintaining low over-defense and manageable $P95$ latency in benign scenarios. The system also provides engineering implementations for parallel sliding window detection and vector retrieval, adapting to multi-model deployments. Therefore, LAtteak framework plays a key role in the multi-stage defense against system prompt leakage and provides an effective solution to sematic transformation prompt leakage.

6 Limitation

Since the main current limitations lie in the scale and maintenance cost , the bias in LLM-as-judge review, and the need to improve the reproducibility of public benchmarks and data. How to protect system prompt words is a very promising research direction, because prompt words play a very important role in model output specifications.

Acknowledgments

This section was hidden due to the double-blind review mechanism.

References

Divyansh Agarwal, Alexander Fabbri, Ben Risher, Philippe Laban, Shafiq Joty, and Chien-Sheng Wu. 2024. Prompt leakage effect and mitigation strategies for multi-turn LLM applications. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 1255–1275, Miami, Florida, US. Association for Computational Linguistics.

Chloe Autio, Reva Schwartz, Jesse Dunietz, Shomik Jain, Martin Stanley, Elham Tabassi, Patrick Hall, and Kamie Roberts. 2024. Artificial intelligence risk management framework: Generative artificial intelligence profile.

Matan Ben-Tov, Mor Geva, and Mahmood Sharif. 2025. Universal jailbreak suffixes are strong attention hijackers. *Preprint*, arXiv:2506.12880.

Victoria Benjamin, Emily Braca, Israel Carter, Hafsa Kanchwala, Nava Khojasteh, Charly Landow, Yi Luo, Caroline Ma, Anna Magarelli, Rachel Mirin, Avery Moyer, Kayla Simpson, Amelia Skawinski, and Thomas Heverin. 2024. Systematically analyzing prompt injection vulnerabilities in diverse llm architectures. *Preprint*, arXiv:2410.23308.

Patrick Chao, Edoardo Debenedetti, Alexander Robey, Maksym Andriushchenko, Francesco Croce, Vikash Sehwag, Edgar Dobriban, Nicolas Flammarion, George J. Pappas, Florian Tramer, Hamed Hassani, and Eric Wong. 2024. Jailbreakbench: An open robustness benchmark for jailbreaking large language models. *Preprint*, arXiv:2404.01318.

Deepset. 2024. prompt-injections datasets's huggingface page. Accessed: 2025-10-07.

Yue Deng, Wenxuan Zhang, Sinno Jialin Pan, and Lidong Bing. 2024. Multilingual jailbreak challenges in large language models. *Preprint*, arXiv:2310.06474.

FoundationAgents. 2025. Openmanus's github page. Accessed: 2025-10-07.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. 2023. Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection. *Preprint*, arXiv:2302.12173.

Guardrails-ai. 2023a. Guardrails' github page. Accessed: 2025-10-07.

Guardrails-ai. 2023b. Guardrails' home page. Accessed: 2025-10-07.

Kuo-Han Hung, Ching-Yun Ko, Ambrish Rawat, I-Hsin Chung, Winston H. Hsu, and Pin-Yu Chen. 2025. Attention tracker: Detecting prompt injection attacks in LLMs. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 2309–2322, Albuquerque, New Mexico. Association for Computational Linguistics.

Hakan Inan, Kartikeya Upasani, Jianfeng Chi, Rashi Rungta, Krithika Iyer, Yuning Mao, Michael Tontchev, Qing Hu, Brian Fuller, Davide Testuggine, and 1 others. 2023. Llama guard: Llm-based input-output safeguard for human-ai conversations. *arXiv preprint arXiv:2312.06674*.

Jiabao Ji, Bairu Hou, Alexander Robey, George J Pappas, Hamed Hassani, Yang Zhang, Eric Wong, and Shiyu Chang. 2024. Defending large language models against jailbreak attacks via semantic smoothing. *arXiv preprint arXiv:2402.16192*.

Fengqing Jiang, Zhangchen Xu, Luyao Niu, Zhen Xiang, Bhaskar Ramasubramanian, Bo Li, and Radha Poovendran. 2024. ArtPrompt: ASCII art-based jailbreak attacks against aligned LLMs. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15157–15173, Bangkok, Thailand. Association for Computational Linguistics.

Zhifeng Jiang, Zhihua Jin, and Guoliang He. 2025. Promptkeeper: Safeguarding system prompts for llms. *Preprint*, arXiv:2412.13426.

Liang Lin, Zhihao Xu, Xuehai Tang, Shi Liu, Biyu Zhou, Fuqing Zhu, Jizhong Han, and Songlin Hu. 2025. Paper summary attack: Jailbreaking llms through llm safety papers. *Preprint*, arXiv:2507.13474.

Yuping Lin, Pengfei He, Han Xu, Yue Xing, Makoto Yamada, Hui Liu, and Jiliang Tang. 2024. Towards understanding jailbreak attacks in LLMs: A representation space analysis. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 7067–7085, Miami, Florida, USA. Association for Computational Linguistics.

Yi Liu, Gelei Deng, Yuekang Li, Kailong Wang, Zihao Wang, Xiaofeng Wang, Tianwei Zhang, Yepang Liu, Haoyu Wang, Yan Zheng, and Yang Liu. 2024. Prompt injection attack against llm-integrated applications. *Preprint*, arXiv:2306.05499.

Manus. 2025. Manus home page. Accessed: 2025-10-07.

Sheikh Samit Muhaimin and Spyridon Mastorakis. 2025. Helping large language models protect themselves: An enhanced filtering and summarization system. *Preprint*, arXiv:2505.01315.

Javier Rando, Francesco Croce, Kryštof Mitka, Stepan Shabalin, Maksym Andriushchenko, Nicolas Flammarion, and Florian Tramèr. 2024. Competition report: Finding universal jailbreak backdoors in aligned llms. *arXiv preprint arXiv:2404.14461*.

Traian Rebedea, Razvan Dinu, Makesh Sreedhar, Christopher Parisien, and Jonathan Cohen. 2023. Nemo guardrails: A toolkit for controllable and safe llm applications with programmable rails. *Preprint*, arXiv:2310.10501.

Rubend18. 2024. Chatgpt-jailbreak-prompts datasets's huggingface page. Accessed: 2025-10-07.

Junlin Wang, Tianyi Yang, Roy Xie, and Bhuwan Dhingra. 2024. Raccoon: Prompt extraction benchmark of llm-integrated applications. In *Findings of the Association for Computational Linguistics ACL 2024*, page 13349–13365. Association for Computational Linguistics.

Zhun Wang, Vincent Siu, Zhe Ye, Tianneng Shi, Yuzhou Nie, Xuandong Zhao, Chenguang Wang, Wenbo Guo, and Dawn Song. 2025. Agentvigil: Generic black-box red-teaming for indirect prompt injection against llm agents. *Preprint*, arXiv:2505.05849.

Zheng Xin Yong, Cristina Menghini, and Stephen Bach. 2023. Low-resource languages jailbreak GPT-4. In *Socially Responsible Language Modelling Research*.

Kaijie Zhu, Qinlin Zhao, Hao Chen, Jindong Wang, and Xing Xie. 2024. Promptbench: A unified library for evaluation of large language models. *Journal of Machine Learning Research*, 25(254):1–22.

Zhixiong Zhuang, Maria-Irina Nicolae, Hui-Po Wang, and Mario Fritz. 2025. Proxyprompt: Securing system prompts against prompt extraction attacks. *Preprint*, arXiv:2505.11459.