

A Critical Evaluation of Defenses against Prompt Injection Attacks

Yuqi Jia[†], Zedian Shao[†], Yupei Liu^{*}, Jinyuan Jia^{*}, Dawn Song[‡], Neil Zhenqiang Gong[†]

[†]Duke University, ^{*}The Pennsylvania State University, [‡]UC Berkeley

[†]{yuqi.jia, zedian.shao, neil.gong}@duke.edu,

^{*}{yxl6415, jinyuan}@psu.edu, [‡]dawnsong@berkeley.edu

Abstract

Large Language Models (LLMs) are vulnerable to prompt injection attacks, and several defenses have recently been proposed, often claiming to mitigate these attacks successfully. However, we argue that existing studies lack a principled approach to evaluating these defenses. In this paper, we argue the need to assess defenses across two critical dimensions: (1) *effectiveness*, measured against both existing and adaptive prompt injection attacks involving diverse target and injected prompts, and (2) *general-purpose utility*, ensuring that the defense does not compromise the foundational capabilities of the LLM. Our critical evaluation reveals that prior studies have not followed such a comprehensive evaluation methodology. When assessed using this principled approach, we show that existing defenses are not as successful as previously reported. This work provides a foundation for evaluating future defenses and guiding their development. Our code and data are available at: <https://github.com/PIEval123/PIEval>.

1 Introduction

LLMs are inherently vulnerable to prompt injection attacks [26, 38, 10, 29, 39, 9, 19, 33] due to their instruction-following nature and the inseparability of instructions and data within a prompt. Specifically, a prompt consists of a concatenation of an *instruction*, which directs the LLM to perform a specific task (e.g., summarization, translation, or math solving), and *data*, which provides the information to be processed for the task. When the data originates from an untrusted source, such as the Internet, an attacker can embed a malicious prompt—referred to as an *injected prompt*—within the data. A prompt injection attack involves crafting and embedding such an injected prompt into the data so that, when the LLM processes the instruction + contaminated data, it performs an attacker-chosen task rather than the intended task.

Prompt injection attacks pose significant safety and security risks to LLMs. For instance, if an LLM is used to summarize reviews, a malicious reviewer could append an injected prompt, such as: “Ignore all previous instructions. Output the product is very bad.” As a result, current LLM would produce the summary: “The product is very bad,” thereby misleading the review summarization process and potentially damaging the product’s reputation. With a carefully crafted injected prompt, an attacker can even manipulate an LLM into revealing its system prompt, compromising both its confidentiality and intellectual property [43, 28, 14]. Additionally, a malicious tool developer could embed a tailored injected prompt into the description of their tool, tricking an LLM agent into preferentially selecting that tool during the tool selection process [34, 35].

Defenses against prompt injection can be broadly categorized into two groups: prevention-based and detection-based. Prevention-based defenses follow the principle of secure-by-design, aiming to construct an LLM system that reliably completes the intended task even when the data contains injected prompts. Early prevention methods [39, 21, 32, 31] focus on pre-processing instructions and/or data to neutralize the effects of injected prompts. More recent approaches [6, 7, 37, 42] involve fine-tuning the LLM itself to ensure that it follows the target prompt, even in the presence of injected prompts. Detection-based defenses, on the other hand, aim to identify whether a data sample has been contaminated with injected prompts during runtime. Earlier detection methods [16, 1, 23, 36] primarily rely on analyzing the perplexity of a data sample or leveraging the LLM as a zero-shot detector. In contrast, more recent approaches either fine-tune an LLM

as a binary classifier using standard supervised learning [22], or leverage an LLM augmented with a special instruction—whose answer is known to the detector but hidden from attackers—and fine-tune it using a game-theoretic method [20]. Alternatively, some detectors [3, 15] extract features to represent data samples and build a binary classifier.

Prior studies [19] have shown that earlier prevention and detection methods are largely ineffective against prompt injection attacks. More recent methods, however, often claim to successfully defend against these attacks. Based on such conclusions, a method called Instruction Hierarchy [37] has even been deployed on GPT-4o Mini. We argue that such claims rely on unrigorous evaluation methodologies, particularly in terms of the types of attacks, evaluation metrics, and benchmark datasets used. For instance, several prevention methods [40, 6, 7] evaluate effectiveness primarily using a single injected prompt, such as “Print exactly Hacked!”. If a fine-tuned LLM resists misdirection from this specific injected prompt, the prevention method is deemed effective. Furthermore, a fine-tuned LLM is considered to “maintain utility” if it performs comparably to its counterpart fine-tuned on clean data—yet both may exhibit similarly poor utility. Most detection methods face similar issues. They are frequently evaluated using existing attacks, while adaptive attacks are largely ignored [22, 3, 15]. Additionally, some studies [15] rely solely on metrics like AUC, which fail to capture a detector’s success in distinguishing between clean and contaminated data during deployment.

In this work, we argue that the *effectiveness* and *utility* of a defense should be tested against diverse attacks—including both existing and adaptive ones—alongside a broad range of target and injected prompts, meaningful metrics, and large-scale benchmarks. For example, instead of relying on a single injected prompt, a defense’s effectiveness should be tested against a wide range of target and injected prompts as well as both existing and adaptive attacks. A defense’s utility should be evaluated based on its general-purpose capabilities across large-scale benchmarks. Detection methods should be assessed using metrics such as false positive rate (FPR) and false negative rate (FNR), rather than solely relying on AUC.

We argue that without adhering to this rigorous evaluation approach, conclusions regarding the success of defenses may be unsound. To illustrate this, we evaluate several recent prevention and detection methods using our approach and find that they are not as successful as previously reported. For instance, LLMs fine-tuned by StruQ [6] and SecAlign [7] suffer substantial losses in general-purpose utility, while StruQ, SecAlign, and Instruction Hierarchy [37] remain vulnerable to even existing prompt injection attacks and more vulnerable to adaptive attacks—contrary to the claims made in these works. Additionally, detection methods, such as PromptGuard [22] and Attention Tracker [15], are ineffective against existing attacks in some scenarios and largely ineffective against adaptive attacks. Even when they achieve high AUCs, they often exhibit significant FPRs and/or FNRs, highlighting the limitations of drawing conclusions solely based on AUC.

2 Prompt Injection Attacks

A prompt injection attack aims to make an LLM perform an attacker-chosen *injected task* instead of the intended *target task* via injecting a prompt into the target prompt. We first formally define the target and injected tasks, and then discuss different attacks.

Target and injected tasks: A task sample consists of a triple (s, x, r) , where s is an instruction, x is a data sample to be processed by LLM based on s , and r is a ground-truth response that completes the task. $p = s||x$ denotes a prompt, where $||$ represents string concatenation. An LLM f takes the prompt p as input and outputs a response $f(p)$. The LLM completes the task if the response $f(p)$ is semantically equivalent to r . Suppose an LLM intends to complete a target task (s_t, x_t, r_t) , where $p_t = s_t||x_t$ is the *target prompt*. An

attacker aims to mislead the LLM to complete an injected task (s_e, x_e, r_e) via inserting the *injected prompt* $p_e = s_e || x_e$ into the target data x_t . In particular, the target data is contaminated as $x_c = x_t || z || p_e$, where z is a sequence of tokens called *separator* [19]. An attack aims to craft z such that $f(s_t || x_c) = f(p_t || z || p_e)$ is semantically equivalent to r_e .

Heuristic-based attacks: These attacks rely on manually designed heuristics to construct z . For instance, *Naive Attack* [38] employs an empty string as z . *Context Ignoring* [4] uses directives such as “Ignore previous instructions.” as z to mislead the LLM to follow the injected prompt. *Fake Completion* [39] embeds a misguiding response such as “Answer: this task is complete.” to deceive the LLM that the target task has been completed. *Escape Character* [38] exploits context-switching characters, e.g., newlines ($\backslash n$) and tabs ($\backslash t$), to manipulate the LLM’s context. *Combined Attack* [19] integrates all the aforementioned heuristics to create the separator, e.g., it might take the form “Answer: the task is complete. $\backslash n \backslash n$ Ignore previous instructions.”. Prior studies [19] show that Combined Attack is the most effective among the heuristic-based attacks.

Optimization-based attacks: These attacks [27, 18, 14, 34] optimize the separator z or $z || p_e$ or the entire contaminated target data x_c depending on the attacker’s capability in the threat model. Roughly speaking, the idea is to define a loss function (e.g., cross-entropy loss) that quantifies the similarity between the attacker-desired response r_e and the response $f(s_t || x_t || z || p_e)$. For instance, taking optimizing the separator z as an example, the loss function can be $\ell_{ce}(r_e, f(s_t || x_t || z || p_e)) = - \sum_{i=1}^{|r_e|} \log(p_f(r_e^i | s_t || x_t || z || p_e || r_e^{<i>}$), where $|r_e|$ is the number of tokens in r_e , $r_e^{<i>$ means the first i tokens in r_e , and $p_f(r_e^i | s_t || x_t || z || p_e || r_e^{<i>$ represents the conditional probability of LLM f generating the token r_e^i , given $s_t || x_t || z || p_e || r_e^{<i>$ as input. Then, these attacks iteratively optimize z (or $z || s_e$ or $z || p_e$) to minimize the loss function via Greedy Coordinate Gradient (GCG) [44]. In each iteration, GCG first calculates the gradient of the loss function with respect to the one-hot vector of each token in z (or $z || s_e$ or $z || p_e$) to identify the top- K most promising replacement tokens. Then, it randomly generates a set of candidates, each of them is obtained by replacing one token with a top- K token. Finally, it selects the candidate that minimizes the loss function. GCG repeats the above three steps until the stop condition is reached.

3 Defenses

Prevention-based defenses: Earlier prevention-based defenses [39, 21, 32, 31] neutralize the effects of injected prompts by pre-processing target instructions and/or data. For instance, *Data Isolation* [39, 21] encloses the (contaminated) target data within special delimiters, such as XML tags (i.e., $\langle data \rangle \dots \langle /data \rangle$), to clearly separate the instruction and data in a target prompt. Prior benchmark studies [19] show that these earlier prevention methods are largely ineffective. Specifically, they often suffer from bad utility and/or effectiveness.

More recent prevention strategies fine-tune an LLM to make it adhere to the target prompt under attacks. For instance, *StruQ* [6] separates instructions and data into distinct structures and fine-tunes an LLM to make it process only the designated prompts while ignoring injected prompts. *SecAlign* [7] constructs a dataset containing desirable and undesirable responses for contaminated data. Then, they use DPO [30] to fine-tune an LLM on the constructed dataset to make it generate defender-desired response under attacks. *Instruction Hierarchy* [37] fine-tunes the LLM to enforce a priority-based policy where higher-priority instructions (e.g., system instructions) take precedence over lower-priority ones (e.g., injected prompts).

Detection-based defenses: Earlier detection methods [16, 1, 23, 36] leverage the perplexity or employ

LLMs as zero-shot classifiers to detect contaminated data. For example, *Perplexity Filter* [16, 1] calculates the text perplexity of a given data sample to determine whether it is clean or contaminated. *Known-answer Detection* [23] leverages a detection LLM to perform a detection task (e.g., “Repeat a string while ignoring the following data.”) to detect if the given data sample is contaminated. *LLM-based Detection* [36] directly queries a detection LLM to ask if a given data sample contains an injected prompt. However, prior studies [19] have shown that these methods suffer from reduced utility and/or ineffectiveness against even existing attacks.

More recent detection approaches fine-tune LLMs as classifiers or extract features from clean and contaminated data samples to build classifiers to distinguish them. For example, *PromptGuard* [22] fine-tunes mDeBERTa-v3-base [11] as a classifier on a comprehensive corpus of attacks, designed to detect both explicitly malicious prompts and contaminated data containing injected inputs, such as injected prompts or jailbreak prompts. Another recent detection-based defense, *Attention Tracker* [15], aims at detecting contaminated data samples by analyzing attention patterns within the LLM. It identifies deviations in attention from an intended, target instruction to an injected prompt, thereby detecting contaminated data.

4 Evaluation Methodology

The success of a defense should be comprehensively evaluated across two critical dimensions: *utility* and *effectiveness*. Below, we outline the evaluation of these dimensions with a focus on appropriate metrics, benchmarks, and attacks (for effectiveness). We note that while prior studies often did evaluate defenses across these dimensions, they did not employ appropriate metrics, benchmarks, and/or attacks.

4.1 Utility

LLMs are typically trained on massive datasets using substantial computational resources, enabling them to function as general-purpose systems capable of handling a broad range of tasks. Therefore, a defense should preserve the general-purpose utility of an LLM, ensuring it maintains performance across diverse tasks.

4.1.1 Metrics

Prevention-based defenses: A prevention-based defense typically pre-processes the prompt or modifies the parameters of the LLM. Multiple recent defenses [6, 7] utilize a metric known as the *win rate* to measure *relative utility*, which compares the quality of responses generated by a defended LLM and a reference LLM for the same prompts. Given a defended LLM f_d , a reference LLM f_u , and a set of tasks/prompts $P = \{p_1, p_2, \dots\}$, the win rate of f_d vs. f_u is formally defined as follows:

$$\text{win rate} = \frac{1}{|P|} \sum_{p \in P} \mathcal{I}(f_d(p), f_u(p), p), \quad (1)$$

where $\mathcal{I}(f_d(p), f_u(p), p) = 1$ if the response $f_d(p)$ has better quality than $f_u(p)$ for prompt p as judged by a strong LLM such as GPT-4, and $\mathcal{I}(f_d(p), f_u(p), p) = 0$ otherwise. If the win rates of the defended and undefended LLMs are similar when compared to the same reference LLM, indicating comparable response quality, these studies conclude that the defenses preserve the LLM’s utility.

We argue that a similar win rate between the defended and undefended LLMs, even when both are high, does not adequately capture the functional performance of the defended LLM across diverse tasks. Specifically, the defended and undefended LLMs may exhibit equally poor utility, while a weaker reference

LLM can still result in high win rates for them, rendering the win rate metric insufficient. Therefore, in addition to relying on relative utility measures such as the win rate, it is crucial to incorporate metrics that assess *absolute utility*. Specifically, given a set of prompt-response pairs $PR = \{(p_1, r_1), (p_2, r_2), \dots\}$, where p_i represents a prompt sample and r_i is a ground-truth response, the absolute utility of an LLM f is measured as follows:

$$\text{absolute utility} = \frac{1}{|PR|} \sum_{(p,r) \in PR} \mathcal{U}(f(p), r), \quad (2)$$

where $\mathcal{U}(f(p), r)$ is a task-specific metric that quantifies the quality of the response $f(p)$ with respect to the ground-truth r . For example, for classification or multiple-choice question-answering tasks, \mathcal{U} is accuracy, i.e., $\mathcal{U}(f(p), r) = 1$ if $f(p) = r$ and $\mathcal{U}(f(p), r) = 0$ otherwise. For summarization tasks, \mathcal{U} can be ROUGE-1 score. For grammar correction tasks, \mathcal{U} can be GLEU score.

We stress that a prevention-based defense is considered to maintain general-purpose utility if the defended LLM shows comparable absolute utility to the undefended one across various tasks using task-specific utility metrics.

Detection-based defenses: Unlike prevention-based defenses, detection-based defenses aim to classify whether the input data of an LLM is clean or contaminated. Thus, the utility metrics discussed above for prevention-based defenses are not directly applicable. Mistakenly flagging clean data as contaminated generates false alarms, resulting in user frustration and a poor user experience, as tasks can be rejected or are not completed promptly. Furthermore, frequent false alarms can lead to unnecessary human inspection and forensic analysis, reducing the practicality and reliability of the defense, and ultimately causing the detection system to be abandoned. Thus, the utility of a detection-based defense can be assessed by its *false positive rate (FPR)*—the probability of misclassifying clean data sample as contaminated. A low FPR indicates good utility. Formally, given a set of clean data samples $X = \{x_1, x_2, \dots\}$, FPR for a detector D is measured as follows:

$$\text{FPR} = \frac{1}{|X|} \sum_{x \in X} D(x), \quad (3)$$

where $D(x) = 1$ if D classifies data sample x as contaminated and $D(x) = 0$ otherwise.

4.1.2 Benchmarks

Multiple recent prevention-based defenses [6, 7] utilize the AlpacaFarm benchmark [8], which includes a set of prompts but lacks ground-truth responses, as they focus solely on the win rate metric. We emphasize that, beyond AlpacaFarm, diverse and large-scale benchmarks containing a set of prompt-response pairs should be employed to evaluate the absolute utility of both defended and undefended LLMs across a wide range of tasks.

For example, OpenPromptInjection [19] encompasses seven fundamental natural language processing tasks, including duplicate sentence detection, grammar correction, hate speech detection, natural language inference, sentiment analysis, spam detection, and text summarization. Additionally, MMLU [12] provides multiple-choice question-answering tasks, further enhancing utility evaluation across diverse applications. These benchmarks are essential for comprehensively assessing the absolute utility of prevention-based defenses.

Similarly, FPRs of detection-based defenses should also be evaluated using such diverse benchmarks. Given a benchmark consisting of a set of prompts or prompt-response pairs, we can construct a set X of

clean data samples to evaluate FPR. Specifically, since a prompt is the concatenation of an instruction and a data sample, we can extract the data samples from the prompts to form X .

4.2 Effectiveness

4.2.1 Metrics

Prevention-based defenses: The effectiveness of a prevention-based defense against prompt injection attacks can be evaluated using the *Attack Success Value (ASV)*. Consider a target prompt p_t with its ground-truth response r_t , an injected prompt p_e with its corresponding ground-truth response r_e , and a separator z crafted by a prompt injection attack. A defense is deemed ineffective if the defended LLM f completes the injected prompt, i.e., generates a response semantically equivalent to r_e , when provided with the concatenated input $p_t||z||p_e$. ASV depends on attacks. To compute ASV for an attack that uses the separator z , we consider a set T consisting of tuples (p_t, r_t, p_e, r_e) , where $r_t \neq r_e$ in each tuple to ensure that the LLM follows the injected prompt when its response is semantically equivalent to r_e . The ASV for an attack using separator z can then be defined as follows:

$$\text{ASV} = \frac{1}{|T|} \sum_{(p_t, r_t, p_e, r_e) \in T} \mathcal{U}(f(p_t||z||p_e), r_e), \quad (4)$$

where \mathcal{U} represents the task-specific metric (e.g., accuracy, ROUGE-1 score, or GLEU score) used to quantify the utility of an injected task. A prevention-based defense is deemed effective if it consistently achieves low ASVs for both existing and adaptive attacks across a diverse set of target and injected prompts. We use ASV instead of ASR because some evaluation metrics, such as ROUGE-1 and GLEU scores, are not rate-based measures.

Detection-based defenses: A detection-based defense aims to detect contaminated data. Thus, the effectiveness of a detection defense can be evaluated using *false negative rate (FNR)*—the probability of falsely classifying contaminated data samples as clean. Consider a set X_c consisting of pairs (x, p_e) , where x is a clean data sample and p_e is an injected prompt. The FNR of a detector D for an attack using separator z can then be defined as follows:

$$\text{FNR} = \frac{1}{|X_c|} \sum_{(x, p_e) \in X_c} (1 - D(x||z||p_e)), \quad (5)$$

where $D(x||z||p_e) = 1$ if D classifies $x||z||p_e$ as contaminated and $D(x||z||p_e) = 0$ otherwise. Note that FNR also depends on attacks which may use different separator z .

We observe that most studies did use ASV and FNR to evaluate their effectiveness. However, they often fail to consider a comprehensive range of prompt injection attacks, including both existing and adaptive ones. Moreover, some defenses [6, 7] evaluate their effectiveness using a limited set of target and injected prompts. For instance, they often use only one specific injected prompt, such as “Print exactly Hacked!”. This restricted scope does not capture the diversity and variability of real-world prompt injection attacks, potentially leading to overly optimistic conclusions about a defense’s effectiveness.

Additionally, some detection-based defenses rely solely on Area Under the Curve (AUC) as a metric to assess effectiveness. We emphasize that AUC is insufficient for quantifying detection performance in real-world deployment. Specifically, AUC evaluates a detector’s ability to rank contaminated and clean data samples based on their likelihood of being contaminated, rather than its ability to classify them accurately.

In deployment, a detector must select a classification threshold to distinguish between contaminated and clean data samples. A high AUC does not guarantee that the detector can identify such a threshold during training, nor does it ensure effective classification in practice.

4.2.2 Attacks

The ASV and FNR metrics depend on prompt injection attacks that employ various separators z . An attacker can deploy any attack and is considered successful as long as at least one attack succeeds. Consequently, a defense’s effectiveness must be rigorously evaluated against a comprehensive suite of attacks, encompassing both existing and adaptive ones. Specifically, evaluation should include both heuristic-based and optimization-based existing attacks, as discussed in Section 2. However, we observe that multiple defenses [15, 37] claim effectiveness based solely on evaluations using a limited subset of existing attacks, which does not provide a complete picture of their effectiveness.

Furthermore, adaptive attacks tailored to specific defenses must also be considered. In the realm of adversarial example research, defenses frequently claimed to be effective are often compromised shortly after publication due to inadequate evaluation against adaptive adversarial examples [2, 5]. A similar pattern is emerging in studies on prompt injection attacks and defenses. For example, we find that multiple defenses [7, 15, 37, 1] neglect to incorporate adaptive attacks in their evaluations, undermining the credibility of their effectiveness claims.

It is critical for defense studies to make a *best-effort attempt* to adapt existing attacks specifically to their proposed defenses. Without such rigorous evaluation, conclusions about a defense’s effectiveness are likely to be unsound. In Section 5, we will highlight several case studies where adaptive attacks significantly undermine the effectiveness of recent defenses [6, 7, 15].

4.2.3 Benchmarks

Based on Equation 4, we need to construct a set T of tuples (p_t, r_t, p_e, r_e) to evaluate the effectiveness of prevention-based defenses. To measure the effectiveness across diverse scenarios, T should include target and injected prompts derived from various tasks. T can be constructed using any benchmark dataset used to evaluate the absolute utility of an LLM. Specifically, given a benchmark dataset of prompt-response pairs, a tuple in T can be formed by sampling one pair as (p_t, r_t) and another as (p_e, r_e) , where $r_t \neq r_e$. This ensures that the LLM’s response to the injected prompt can be identified when it generates a response equivalent to r_e .

For example, OpenPromptInjection benchmark [19] constructs T by sampling (p_t, r_t) and (p_e, r_e) pairs from seven fundamental NLP tasks: duplicate sentence detection, grammar correction, hate speech detection, natural language inference, sentiment analysis, spam detection, and text summarization. We note that other benchmarks, such as MMLU, can also be used to construct T .

To measure the FNR, we need to construct a set X_c of pairs in the form (x, p_e) . Benchmark datasets used to evaluate the utility of an LLM can also be used to construct X_c . Specifically, a pair (x, p_e) can be created by sampling a prompt and using its data portion as x and sampling another prompt as p_e . For example, OpenPromptInjection generates X_c by drawing prompts from seven fundamental NLP tasks. Similarly, other benchmarks, such as MMLU, can also be used to construct X_c .

Table 1: Utility of different LLMs for StruQ and SecAlign.

(a) Relative utility (Win Rate) on AlpacaFarm			(b) Absolute utility		
Measured LLM	Reference LLM	Win Rate (%)	LLM	OpenPromptInjection	MMLU-PI
Llama-3-8B-Instruct-StruQ	Llama-3-8B-Instruct	21.60	Llama-3-8B-Instruct-StruQ	0.54	0.35
Llama-3-8B-Instruct-SecAlign	Llama-3-8B-Instruct	36.42	Llama-3-8B-Instruct-SecAlign	0.48	0.34
			Llama-3-8B-Instruct	0.65	0.45

5 Case Studies

Two benchmarks: We first present the statistics of the benchmarks used in our evaluation: OpenPromptInjection [19] and MMLU-PI. Each benchmark includes a set PR of prompt-response pairs (used to measure absolute utility), a set T of tuples (p_t, r_t, p_e, r_e) (used to measure ASV), a set X of clean data samples (used to measure FPR), and a set X_c of pairs (x, p_e) (used to measure FNR). PR in OpenPromptInjection consists of 700 prompt-response pairs sampled from seven NLP tasks, with 100 pairs per task. T is constructed based on PR and contains 4,900 tuples. X includes 700 clean data samples, with 100 samples per task. X_c contains 4,900 pairs. We also construct a new benchmark, MMLU-PI, from MMLU [13]. In this benchmark, PR includes 200 prompt-response pairs sampled from the test split of MMLU, T contains 1,000 tuples, X comprises the 200 clean data samples from PR , and X_c contains 1,000 pairs. We run all the experiments on one single NVIDIA RTX A5000 GPU with 24GB memory.

5.1 StruQ and SecAlign

StruQ [6] fine-tunes an LLM using supervised fine-tuning, while SecAlign [7] applies DPO for fine-tuning. The fine-tuning dataset consists of both clean and contaminated data samples. The utility of StruQ and SecAlign was evaluated using win rate on AlpacaFarm [8]. Specifically, they utilized AlpacaEval [17] with GPT-4-Turbo as a judge LLM to compare the responses of their fine-tuned LLMs against a reference LLM (davinci003 [24] in their experiment) and calculate the corresponding win rate. Their results demonstrated that these defenses maintain utility, as win rates of the defended LLMs by StruQ and SecAlign were found to be very similar to those of the undefended ones when compared to the same reference LLM. Moreover, they evaluated the effectiveness using both heuristic-based and optimization-based attacks. Based on the results, both StruQ and SecAlign were claimed to be effective: ASVs of heuristic-based attacks are close to 0; ASVs of StruQ and SecAlign under optimization-based attacks (using GCG) are reduced to 0.04 and 0, respectively. However, the ASRs were calculated primarily based on only one injected prompt: "Print exactly Hacked!".

We re-evaluate StruQ and SecAlign using OpenPromptInjection and MMLU-PI. We downloaded LLMs released by StruQ and SecAlign: Llama-3-8B-Instruct, Llama-3-8B-Instruct-StruQ, and Llama-3-8B-Instruct-SecAlign. Llama-3-8B-Instruct is finetuned and released by Meta. Llama-3-8B-Instruct-StruQ (or Llama-3-8B-Instruct-SecAlign) is fine-tuned based on Llama-3-8B-Instruct model using StruQ (or SecAlign). We also re-evaluate StruQ and SecAlign models fine-tuned on Llama-3-8B released by Meta. The experiment details and results are provided in Appendix A. The StruQ and SecAlign defended models are licensed under CC BY-NC 4.0. OpenPromptInjection and MMLU are released under the MIT license.

Relative and absolute utility: We first evaluate the win rate of StruQ and SecAlign using AlpacaEval on AlpacaFarm. In their original evaluation, the reference LLM was davinci003, whose architecture differs significantly from the two defended models fine-tuned on Llama-3-8B-Instruct. For a more straight comparison,

Table 2: ASVs of different LLMs on OpenPromptInjection and MMLU-PI against various attacks for StruQ and SecAlign.

LLM	OpenPromptInjection				MMLU-PI			
	Combined Attack		GCG		Combined Attack		GCG	
	existing	adaptive	existing	adaptive	existing	adaptive	existing	adaptive
Llama-3-8B-Instruct-StruQ	0.03	0.09	0.80	1.00	0.10	0.16	0.88	1.00
Llama-3-8B-Instruct-SecAlign	0.06	0.04	0.24	0.46	0.14	0.13	0.72	0.88
Llama-3-8B-Instruct	0.52	0.31	1.00	1.00	0.46	0.24	1.00	1.00

we instead use Llama-3-8B-Instruct as the reference LLM. Table 1(a) presents the results. Against Llama-3-8B-Instruct, Llama-3-8B-Instruct-StruQ achieves win rate of 21.60% and Llama-3-8B-Instruct-SecAlign achieves win rate of 36.42%. We also evaluate the absolute utility of Llama-3-8B-Instruct-StruQ and Llama-3-8B-Instruct-SecAlign. The results are shown in Table 1(b). Llama-3-8B-Instruct-StruQ shows a utility decrease of 0.11 and 0.10 on the two benchmarks compared to Llama-3-8B-Instruct, while Llama-3-8B-Instruct-SecAlign shows a corresponding drop of 0.17 and 0.11. **Our results indicate that both StruQ and SecAlign lead to a loss of both relative and absolute utility – contrary to the original claims.**

Effectiveness against existing attacks: Table 2 shows the ASVs of Llama-3-8B-Instruct-StruQ and Llama-3-8B-Instruct-SecAlign on the two benchmarks against various attacks. Different from the original observations, we find that the existing Combined Attack still exhibits a certain degree of effectiveness. Specifically, for Llama-3-8B-Instruct-StruQ (or Llama-3-8B-Instruct-SecAlign), it achieves ASVs of 0.03 (or 0.06) and 0.10 (or 0.14) on the two benchmarks, respectively. Furthermore, the effectiveness of optimization-based attack using GCG shows a significant disparity compared to their observations. Specifically, for computational efficiency, we select 50 tuples from T in OpenPromptInjection and 25 tuples in MMLU-PI, and optimize the separator z using GCG as described in Section 2. The results indicate that GCG achieves ASVs exceeding 0.80 against StruQ on both benchmarks, and 0.72 ASVs against SecAlign on MMLU-PI. **These results show that StruQ and SecAlign are not as effective against *existing* attacks as previously reported when evaluated on diverse target and injected prompts.**

Effectiveness against adaptive attacks: We also propose adaptive attacks to StruQ and SecAlign. Both defenses add special tokens to the LLM’s vocabulary as delimiters to explicitly separate the instruction, data, and response. Therefore, an adaptive attack can apply the same idea to structure an injected prompt. However, these defenses filter the special tokens in the data during runtime, making it not feasible to directly use these special tokens in the injected prompt. To address the challenge, our adaptive attack finds the tokens in the LLM’s vocabulary whose embeddings have the smallest ℓ_2 distance to these special tokens, and uses them as delimiters to structure the injected prompt. The results in Table 2 demonstrate that our adaptive GCG attacks further weaken the effectiveness of these two defenses. For instance, the ASV of the adaptive GCG Attack is around 0.20 higher than that of the existing GCG Attack for both defenses on OpenPromptInjection. For the heuristic Combined Attack, the adaptive version does not achieve a higher ASV. This may be because it uses significantly more tokens, approximately twice the number of tokens in the original, which the LLM may struggle to process effectively in a long-context setting, leading to reduced attack success. **These results underscore the critical role of *adaptive* attacks in evaluating the effectiveness of defenses.**

5.2 Instruction Hierarchy

Instruction Hierarchy [37] fine-tunes an LLM to selectively follow instructions based on the context. The effectiveness of Instruction Hierarchy against prompt injection attacks has been evaluated using manually

crafted prompt injections embedded in web results for browsing or return values from external tools. However, the specific details of these injected and target prompts remain unspecified in the paper. Instruction Hierarchy has been claimed to preserve utility while effectively mitigating attacks and has thus been deployed on GPT-4o-mini [25]. To reassess its utility and effectiveness, we evaluate GPT-4o-mini using the OpenPromptInjection and MMLU-PI benchmarks. Our results show that GPT-4o-mini maintains utility, with absolute utility scores of 0.71 on OpenPromptInjection and 0.73 on MMLU-PI. However, even the existing Combined Attack achieves ASVs of 0.68 and 0.75 on these benchmarks, respectively. **These findings contradict previous claims, indicating that Instruction Hierarchy is not effective when evaluating on diverse injected prompts.**

5.3 PromptGuard and Attention Tracker

Claims about the success of detection-based defenses shouldn’t be drawn solely based on AUC: We obtained the open-source model parameters of PromptGuard [22] and Attention Tracker [15], two detection-based defenses. We evaluate their FPRs, FNRs, and AUCs on the two benchmarks. The results are summarized in Table 3(a), where the Combined Attack is used. While both PromptGuard and Attention Tracker achieve high AUCs, they exhibit significant limitations in terms of FPR or FNR. Specifically, PromptGuard achieves an AUC of 0.92 but exhibits an FPR of 0.89 on OpenPromptInjection, which suggests a strong bias toward predicting most data samples as contaminated. This indicates that while PromptGuard may detect some attacks, its high FPR limits its utility by introducing a substantial number of false alarms. On the other hand, Attention Tracker achieves AUCs of 1.00 for both OpenPromptInjection and MMLU-PI but suffers from a high FNR of 0.69 on the more challenging benchmark MMLU-PI. **These results show that AUC alone is insufficient for assessing the success of detection methods.**

Adaptive attacks: We propose a general adaptive attack framework for detection-based methods. Given a target/injected prompt tuple (p_t, r_t, p_e, r_e) , our framework optimizes a separator z to achieve two objectives for adaptive attacks: evading the detector D and ensuring that the LLM f completes the injected task by generating r_e . The first objective requires the detector D to misclassify contaminated data as clean, i.e., $D(x_t \| z \| p_e) = 0$, where x_t is the target data in the target prompt p_t . To quantify this, we define an evasion loss $\ell_e(0, D(x_t \| z \| p_e))$. A lower evasion loss signifies a higher likelihood that the detector D will incorrectly classify the contaminated data $x_t \| z \| p_e$ as clean. The evasion loss can be customized based on the specific detection-based defenses being targeted. The second loss ensures the effectiveness of this adaptive prompt injection attack. To achieve this, we can use the standard cross-entropy loss introduced in Section 2, i.e., $\ell_{ce}(r_e, f(p_t \| z \| p_e)) = -\sum_{i=1}^{|r_e|} \log(p_f(r_e^i | p_t \| z \| p_e \| r_e^{<i>i</i>}))$. Formally, we have the following loss function for the adaptive attacks against detection-based defenses:

$$\mathcal{L}(p_t, p_e, r_e) = -\ell_e(0, D(x_t \| z \| p_e)) + \alpha \cdot \ell_{ce}(r_e, f(p_t \| z \| p_e)), \quad (6)$$

where α is a hyper-parameter to balance the two loss terms. Moreover, in addition to solely optimizing the separator z , our framework can be adapted to optimize different components of the contaminated data. Specifically, we can also optimize $z \| s_e$, the combination of the separator and injected instruction, and $z \| p_e$, the combination of the separator, injected instruction, and injected data.

We apply our adaptive attack to Attention Tracker since it achieves good detection performance in Table 3(a). Attention Tracker designs a *focus score* to determine whether a data sample is contaminated. Thus, we can customize the evasion loss as the focus score of the contaminated data. The goal is to minimize this focus score, effectively reducing the likelihood of the data being flagged as contaminated. To minimize the loss function in Equation 6, we adopt GCG to iteratively update the tokens in z , $z \| s_e$, or $z \| p_e$. We use

Table 3: Results of detection-based defenses on OpenPromptInjection and MMLU-PI.

(a) FPRs, FNRs, and AUCs of PromptGuard and Attention Tracker

Detection-based defense	OpenPromptInjection			MMLU-PI		
	FPR	FNR	AUC	FPR	FNR	AUC
PromptGuard	0.89	0.00	0.92	0.84	0.00	0.75
Attention Tracker	0.00	0.00	1.00	0.00	0.69	1.00

(b) FNRs of different adaptive attack strategies against Attention Tracker

Adaptive attack strategy	OpenPromptInjection	MMLU-PI
Separator	0.66	1.00
Separator+Instruction	0.96	1.00
Separator+Instruction+Data	0.96	1.00

Qwen2-1.5B-Instruct [41] as the LLM, which is used by Attention Tracker. We set α to 0.01 and evaluate the adaptive attacks against Attention Tracker using the same tuples (p_t, r_t, p_e, r_e) as our previous experiments for GCG-based attack in Section 5.1.

As shown in Table 3(b), our proposed adaptive attack notably increases the FNRs of Attention Tracker when only optimizing the separator z , reaching 0.66 and 1.00 on the two benchmarks, respectively. Moreover, we find that all contaminated data samples that successfully evade Attention Tracker’s detection also make the LLM generate the attacker-desired response r_e . When the optimizable part includes the injected instruction, the adaptive attack becomes even more effective. On OpenPromptInjection, optimizing both the separator and the instruction resulted in a 0.3 FNR improvement. This is because, after optimization, some tokens in the injected instruction are replaced, making the contaminated data appear to lack a clear instruction. As a result, the detector may misclassify it as clean. **Our results show that the perceived effectiveness of a detection-based defense can vary significantly between existing and adaptive attacks, highlighting the necessity of incorporating adaptive attacks in its evaluation.**

6 Conclusion

Recent defenses against prompt injection attacks were not evaluated in a principled way, which often overestimates their effectiveness and/or utility maintenance. In this paper, we highlight the need to comprehensively evaluate defenses along the two dimensions: effectiveness and general-purpose utility. We hope our study enables more rigorous evaluation for future prompt injection defenses.

References

- [1] Gabriel Alon and Michael Kamfonas. Detecting language model attacks with perplexity. *arXiv*, 2023.
- [2] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *International conference on machine learning*, pages 274–283. PMLR, 2018.
- [3] Md Ahsan Ayub and Subhabrata Majumdar. Embedding-based classifiers can detect prompt injection attacks. *arXiv preprint arXiv:2410.22284*, 2024.

- [4] Hezekiah J. Branch, Jonathan Rodriguez Cefalu, Jeremy McHugh, Leyla Hujer, Aditya Bahl, Daniel del Castillo Iglesias, Ron Heichman, and Ramesh Darwishi. Evaluating the susceptibility of pre-trained language models via handcrafted adversarial examples. *arXiv*, 2022.
- [5] Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proceedings of the 10th ACM workshop on artificial intelligence and security*, pages 3–14, 2017.
- [6] Sizhe Chen, Julien Piet, Chawin Sitawarin, and David Wagner. Struq: Defending against prompt injection with structured queries. In *USENIX Security Symposium*, 2025.
- [7] Sizhe Chen, Arman Zharmagambetov, Saeed Mahloujifar, Kamalika Chaudhuri, and Chuan Guo. Aligning llms to be robust against prompt injection. *arXiv preprint arXiv:2410.05451*, 2024.
- [8] Yann Dubois, Chen Xuechen Li, Rohan Taori, Tianyi Zhang, Ishaan Gulrajani, Jimmy Ba, Carlos Guestrin, Percy S Liang, and Tatsunori B Hashimoto. AlpacaFarm: A simulation framework for methods that learn from human feedback. *Advances in Neural Information Processing Systems*, 36, 2024.
- [9] Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you’ve signed up for: Compromising real-world llm-integrated applications with indirect prompt injection. In *AISeC*, 2023.
- [10] Rich Harang. Securing LLM Systems Against Prompt Injection. <https://developer.nvidia.com/blog/securing-llm-systems-against-prompt-injection>, 2023.
- [11] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. DeBERTa: Decoding-enhanced bert with disentangled attention. *arXiv preprint arXiv:2006.03654*, 2020.
- [12] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.
- [13] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.
- [14] Bo Hui, Haolin Yuan, Neil Gong, Philippe Burlina, and Yinzhi Cao. Pleak: Prompt leaking attacks against large language model applications. In *CCS*, 2024.
- [15] Kuo-Han Hung, Ching-Yun Ko, Amrbrish Rawat, I-Hsin Chung, Winston H. Hsu, and Pin-Yu Chen. Attention tracker: Detecting prompt injection attacks in llms. *arXiv preprint arXiv:2411.00348*, 2024.
- [16] Neel Jain, Avi Schwarzschild, Yuxin Wen, Gowthami Somepalli, John Kirchenbauer, Ping yeh Chiang, Micah Goldblum, Aniruddha Saha, Jonas Geiping, and Tom Goldstein. Baseline defenses for adversarial attacks against aligned language models. *arXiv*, 2023.
- [17] Xuechen Li, Tianyi Zhang, Yann Dubois, Rohan Taori, Ishaan Gulrajani, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. AlpacaEval: An Automatic Evaluator of Instruction-following Models. https://github.com/tatsu-lab/alpaca_eval, 2023.
- [18] Xiaogeng Liu, Zhiyuan Yu, Yizhe Zhang, Ning Zhang, and Chaowei Xiao. Automatic and universal prompt injection attacks against large language models. *arXiv preprint arXiv:2403.04957*, 2024.

- [19] Yupei Liu, Yuqi Jia, Runpeng Geng, Jinyuan Jia, and Neil Zhenqiang Gong. Formalizing and benchmarking prompt injection attacks and defenses. In *USENIX Security Symposium*, 2024.
- [20] Yupei Liu, Yuqi Jia, Jinyuan Jia, Dawn Song, and Neil Zhenqiang Gong. Datasentinel: A game-theoretic detection of prompt injection attacks. In *IEEE Symposium on Security and Privacy*, 2025.
- [21] Alexandra Mendes. Ultimate ChatGPT prompt engineering guide for general users and developers. <https://www.imaginarycloud.com/blog/chatgpt-prompt-engineering>, 2023.
- [22] Meta. PromptGuard Prompt Injection Guardrail. <https://www.llama.com/docs/model-cards-and-prompt-formats/prompt-guard>, 2024.
- [23] Yohei Nakajima. Yohei’s blog post. <https://twitter.com/yoheinakajima/status/1582844144640471040>, 2022.
- [24] OpenAI. Openai gpt-3.5 api [text-davinci-003]. <https://platform.openai.com/docs/guides/text-generation>, 2022.
- [25] OpenAI. Gpt-4o mini: advancing cost-efficient intelligence. <https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/>, 2024.
- [26] OWASP. OWASP Top 10 for Large Language Model Applications. https://owasp.org/www-project-top-10-for-large-language-model-applications/assets/PDF/OWASP-Top-10-for-LLMs-2023-v1_1.pdf, 2023.
- [27] Dario Pasquini, Martin Strohmeier, and Carmela Troncoso. Neural exec: Learning (and learning from) execution triggers for prompt injection attacks. In *Proceedings of the 2024 Workshop on Artificial Intelligence and Security*, pages 89–100, 2024.
- [28] Fábio Perez and Ian Ribeiro. Ignore previous prompt: Attack techniques for language models. *arXiv preprint arXiv:2211.09527*, 2022.
- [29] Fábio Perez and Ian Ribeiro. Ignore previous prompt: Attack techniques for language models. In *NeurIPS ML Safety Workshop*, 2022.
- [30] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In *NeurIPS*, 2023.
- [31] Sander Schulhoff. Instruction defense. https://learnprompting.org/docs/prompt_hacking/defensive_measures/instruction_defense/, 2023.
- [32] Sander Schulhoff. Sandwich defense. https://learnprompting.org/docs/prompt_hacking/defensive_measures/sandwich_defense/, 2023.
- [33] Zedian Shao, Hongbin Liu, Jaden Mu, and Neil Zhenqiang Gong. Making llms vulnerable to prompt injection via poisoning alignment. *arXiv preprint arXiv:2410.14827*, 2024.
- [34] Jiawen Shi, Zenghui Yuan, Yinuo Liu, Yue Huang, Pan Zhou, Lichao Sun, and Neil Zhenqiang Gong. Optimization-based prompt injection attack to llm-as-a-judge. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 660–674, 2024.

- [35] Jiawen Shi, Zenghui Yuan, Guiyao Tie, Pan Zhou, Neil Zhenqiang Gong, and Lichao Sun. Prompt injection attack to tool selection in llm agents. *arXiv preprint arXiv:2504.19793*, 2025.
- [36] R Gorman Stuart Armstrong. Using GPT-Eliezer against ChatGPT Jailbreaking. <https://www.alignmentforum.org/posts/pNcFYZnPdXyL2RfgA/using-gpt-eliezer-against-chatgpt-jailbreaking>, 2023.
- [37] Eric Wallace, Kai Xiao, Reimar Leike, Lilian Weng, Johannes Heidecke, and Alex Beutel. The instruction hierarchy: Training llms to prioritize privileged instructions. *arXiv preprint arXiv:2404.13208*, 2024.
- [38] Simon Willison. Prompt injection attacks against GPT-3. <https://simonwillison.net/2022/Sep/12/prompt-injection/>, 2022.
- [39] Simon Willison. Delimiters won't save you from prompt injection. <https://simonwillison.net/2023/May/11/delimiters-wont-save-you>, 2023.
- [40] Tong Wu, Shujian Zhang, Kaiqiang Song, Silei Xu, Sanqiang Zhao, Ravi Agrawal, Sathish Reddy Indurthi, Chong Xiang, Prateek Mittal, and Wenxuan Zhou. Instructional segment embedding: Improving llm safety with instruction hierarchy. *arXiv preprint arXiv:2410.09102*, 2024.
- [41] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.
- [42] Jingwei Yi, Yueqi Xie, Bin Zhu, Emre Kiciman, Guangzhong Sun, Xing Xie, and Fangzhao Wu. Benchmarking and defending against indirect prompt injection attacks on large language models. *arXiv preprint arXiv:2312.14197*, 2023.
- [43] Yiming Zhang and Daphne Ippolito. Prompts should not be seen as secrets: Systematically measuring prompt extraction attack success. *arXiv preprint arXiv:2307.06865*, 2023.
- [44] Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*, 2023.

Appendix

A More Experiments on StruQ and SecAlign

We re-evaluate StruQ and SecAlign models fine-tuned on Llama-3-8B using OpenPromptInjection and MMLU-PI. Similarly, we downloaded LLMs released by StruQ and SecAlign: Llama-3-8B-undefended, Llama-3-8B-StruQ, and Llama-3-8B-SecAlign. Llama-3-8B-undefended is fine-tuned based on Llama-3-8B base model following the standard supervised fine-tuning with only clean data; Llama-3-8B-StruQ (or Llama-3-8B-SecAlign) is fine-tuned based on Llama-3-8B base model using StruQ (or SecAlign).

Utility: To evaluate the **relative utility**, for a more straight comparison, we similarly instead use Llama-3-8B-undefended as the reference LLM. Table 4(a) presents the results. Against Llama-3-8B-undefended, both Llama-3-8B-StruQ and Llama-3-8B-SecAlign achieve win rates close to 50%, consistent with the papers’ observations. For the **absolute utility**, results are shown in Table 4(b). Llama-3-8B-StruQ shows a utility decrease of 0.04 and 0.03 on the two benchmarks compared to Llama-3-8B-undefended, while Llama-3-8B-SecAlign shows a corresponding drop of 0.04 and 0.05. This further indicates that both StruQ and SecAlign lead to some degree of utility loss – contrary to the original claims.

Effectiveness: Table 5 shows the ASVs of Llama-3-8B-StruQ and Llama-3-8B-SecAlign on the two benchmarks against various attacks. The conclusions align with those in Section 5.1: 1) the existing Combined Attack still exhibits a certain degree of effectiveness; 2) the effectiveness of optimization-based attack using GCG shows a significant disparity compared to the original observations; and 3) adaptive attacks further weaken the effectiveness of these two defenses.

Table 4: Utility of different LLMs for StruQ and SecAlign fine-tuned on Llama-3-8B.

(a) Relative utility (Win Rate) on AlpacaFarm

Measured LLM	Reference LLM	Win Rate (%)
Llama-3-8B-StruQ	Llama-3-8B-undefended	48.04
Llama-3-8B-SecAlign	Llama-3-8B-undefended	55.03

(b) Absolute utility

LLM	OpenPromptInjection	MMLU-PI
Llama-3-8B-StruQ	0.54	0.39
Llama-3-8B-SecAlign	0.54	0.37
Llama-3-8B-undefended	0.58	0.42

Table 5: ASVs of different LLMs on OpenPromptInjection and MMLU-PI against various attacks for StruQ and SecAlign fine-tuned on Llama-3-8B.

LLM	OpenPromptInjection				MMLU-PI			
	Combined Attack		GCG		Combined Attack		GCG	
	existing	adaptive	existing	adaptive	existing	adaptive	existing	adaptive
Llama-3-8B-StruQ	0.07	0.14	0.96	1.00	0.09	0.16	0.92	0.92
Llama-3-8B-SecAlign	0.05	0.09	0.64	0.70	0.12	0.23	0.68	0.88
Llama-3-8B-undefended	0.44	0.51	1.00	1.00	0.31	0.33	1.00	1.00