USENIX

THE ADVANCED COMPUTING
SYSTEMS ASSOCIATION

PRSA: Prompt Stealing Attacks against Real-World Prompt Services

Yong Yang, *Zhejiang University;* Changjiang Li, *Stony Brook University;*
Qingming Li, Oubo Ma, Haoyu Wang, Zonghui Wang, Yandong Gao,
Wenzhi Chen, and Shouling Ji, *Zhejiang University*

https://www.usenix.org/conference/usenixsecurity25/presentation/yang-yong

This paper is included in the Proceedings of the
34th USENIX Security Symposium.

August 13–15, 2025 • Seattle, WA, USA

978-1-939133-52-6

Open access to the Proceedings of the
34th USENIX Security Symposium is sponsored by USENIX.

PRSA: Prompt Stealing Attacks against Real-World Prompt Services

Yong Yang[1], Changjiang Li[2], Qingming Li[1], Oubo Ma[1], Haoyu Wang[1], Zonghui Wang[1,*],
Yandong Gao[1], Wenzhi Chen[1], and Shouling Ji[1,*]
[1] *Zhejiang University,* [2] *Stony Brook University*
{yangyong2022, liqm, mob, whaoyu, zhwang, chenwz, sji}@zju.edu.cn, yandonggao@126.com
meet.cjli@gmail.com

Abstract

Recently, large language models (LLMs) have garnered widespread attention for their exceptional capabilities. Prompts are central to the functionality and performance of LLMs, making them highly valuable assets. The increasing reliance on high-quality prompts has driven significant growth in prompt services. However, this growth also expands the potential for prompt leakage, increasing the risk that attackers could replicate original functionalities, create competing products, and severely infringe on developers' intellectual property. Despite these risks, prompt leakage in real-world prompt services remains underexplored.

In this paper, we present PRSA, a practical attack framework designed for prompt stealing. PRSA infers the detailed intent of prompts through very limited input-output analysis and can successfully generate stolen prompts that replicate the original functionality. Extensive evaluations demonstrate PRSA's effectiveness across two main types of real-world prompt services. Specifically, compared to previous works, it improves the attack success rate from 17.8% to 46.1% in *prompt marketplaces* (with attack costs only 1.3%–12.3% of the original prompt price) and from 39% to 52% in *LLM application stores*, respectively. Notably, in the attack on "Math", one of the most popular educational applications in OpenAI's GPT Store with over 1 million conversations, PRSA uncovered a hidden Easter egg that had not been revealed previously. Besides, our analysis reveals that higher mutual information between a prompt and its output correlates with an increased risk of leakage. This insight guides the design and evaluation of two potential defenses against the security threats posed by PRSA. We have reported these findings to the prompt service vendors, including PromptBase and OpenAI, and actively collaborate with them to implement defensive measures.

*Shouling Ji and Zonghui Wang are the co-corresponding authors.

1 Introduction

Recent advancements in large language models (LLMs) have transformed natural language processing (NLP), enabling models to understand complex linguistic structures and generate human-like text [32]. Unlike traditional NLP models like BERT [17], LLMs can handle specific tasks without fine-tuning, relying instead on well-crafted prompts [24]. A *prompt* is an instruction that directs an LLM to produce the desired output, and its design is crucial for task accuracy and efficiency. The demand for high-quality prompts has driven significant growth in prompt services. The global prompt service size was valued at $374.9 million in 2024 and is expected to reach $2.1 to $2.5 billion by 2030 to 2032 [4].

Current prompt services are primarily categorized into *LLM application stores* [31], which offer interactive applications enhanced by well-crafted prompts, and *prompt marketplaces*, where prompts are sold without direct interaction [34]. Prompts in prompt services typically exhibit two main characteristics: first, they are commercialized through *limited free trials* or by showcasing *one input-output example* to demonstrate their functionality before purchase. Second, prompts in prompt marketplaces are typically designed as *prompt templates*, while in LLM applications, these prompts are designed as *system prompts*. Both types are intended to accommodate various *user inputs*, ensuring the generality of these prompts. For example, a prompt like "`Generate a [product] copywriting...`" can accept various user inputs, such as "smartphone" or "laptop", and produce tailored copywriting for each input.

As the reliance on well-crafted prompts increases, so does the risk of "*prompt leakage*"—the unauthorized disclosure of the well-designed prompts, specifically, their functional details. Adversaries can use this sensitive information to generate prompts that replicate the original functionalities—what we refer to as *stolen prompts*. These stolen prompts can then be used to develop competing products, severely infringing on developers' intellectual property. Currently, there are two main types of attacks targeting prompt leakage. The

first, known as the *prompt leaking attack* [21,49], leverages prompt injection in the interactions [27] to trick LLM applications into disclosing their system prompts. This form of attack generally affects only those applications that permit such interactions. Moreover, its effectiveness can be substantially curtailed by incorporating protective measures into system prompts, which drastically lowers the attack success rate [3,26]. The second type, referred to as the *prompt stealing attack* [39,48], intends to infer the functional details of the prompts. Compared to purchasing prompts, the cost of executing a prompt stealing attack is relatively low. This approach represents a more significant threat because it targets not only LLM applications but also prompt marketplaces. Considering its broader impact, this paper will primarily focus on the prompt stealing attack.

Recent studies have further illuminated the risks associated with prompt stealing attacks. Sha et al. [39] employed LLMs to directly analyze the generated output and infer the original prompts. Zhang et al. [48] trained an inversion model to infer prompts by using multiple LLM outputs as input to the model. However, both methods exhibit practical limitations in prompt services. Specifically, the former study [39] relies entirely on the backward inference ability of the LLM itself, which other studies have shown to be less effective [8,9]. Additionally, it neglects to explore whether these stolen prompts can be effectively used in different or broader contexts. Meanwhile, the latter study [48] depends on generating numerous tailored, diverse outputs for effective inference, which is impractical in prompt marketplaces.

Challenges. The above limitations underscore the challenges in executing prompt stealing attacks, especially in complex, real-world scenarios. **The first challenge** is how to accurately infer the prompt from a very limited number (often single) of input-output pairs. This is because limited data makes it difficult to capture the intricacies of the prompt's detailed intent. **The second challenge** is how to generate a stolen prompt with generality based on given input-output pairs. Since an LLM's output often reflects the intent of both the prompt and the user input, it is easy to mistakenly include user input-related content when inferring the prompt.

Our Proposal. In this paper, we propose a practical framework designed for **pr**ompt **s**tealing **a**ttacks against real-world prompt services, termed as PRSA. PRSA consists of two phases: *prompt generation* and *prompt pruning*. Our key insight is to leverage an LLM as a generative model to infer the detailed intent behind a target prompt. By analyzing limited input-output pairs, we enable PRSA to generate a stolen prompt that replicates the functionality of the target prompt.

To accurately infer the target prompt's detailed intent by limited input-output pairs, our intuition lies in the principle of one-shot learning [20,43]. We observe that the key *factors* (e.g., style, topic, and tone) influencing the accurate inference of prompts within the same *category* (e.g., email, advertisements, and code) exhibit similarities (detailed analysis in Section 4.1). By learning these key factors, the generative model can more accurately infer the intent of the target prompt within the same category. Based on this insight, we propose a prompt attention algorithm based on the output differences in the prompt generation phase to identify these key factors. However, this is insufficient because of the second challenge. To address this, our intuition is to filter out the content that is closer to the user input in the semantic feature space. Based on this insight, we further propose a two-step strategy in the prompt pruning phase. This strategy uses semantic similarity and selective beam search to filter out words highly related to the user input, ensuring the generality of the stolen prompt.

Evaluation. We evaluate PRSA through extensive experiments in two real-world scenarios: *prompt marketplaces* [34] and *LLM application stores* [31]. For prompt marketplaces, we conduct attacks on randomly purchased prompts across 18 categories from PromptBase [34], a leading prompt marketplace[1]. Our evaluation shows that PRSA achieves an attack success rate of 46%, surpassing prior works [39,45,48] by over 160%. Notably, the average attack cost is only 1.3% to 12.3% of the original prompt price. Furthermore, the stolen prompts generated by PRSA demonstrate superior functional consistency and prompt similarity. We further validate our findings through both LLM-based multi-dimensional evaluation and human evaluation. For LLM application stores, we evaluate PRSA against 100 randomly selected popular GPTs from OpenAI's GPT Store [31], all of which have protective measures against prompt leakage. Our results show that PRSA achieved a 52% attack success rate. Notably, during the attack on "Math", which was previously the third highest-ranked educational application in OpenAI's GPT Store and now has over 1 million conversations, PRSA uncovered a hidden Easter egg that had not been revealed previously and was confirmed by the "Math" developer. These findings underscore the real-world threat posed by PRSA.

Finally, we analyze the reasons behind PRSA's effectiveness from a mutual information perspective, noting that higher mutual information between a prompt and its output correlates with increased risk of leakage. Based on these insights, we propose two potential defenses—*output obfuscation* and *prompt watermarking*. Our experiments demonstrate that while output obfuscation is effective, it requires a careful trade-off between effectiveness and usability. On the other hand, prompt watermarking is easily compromised, indicating the need to enhance this defense further.

Contributions. To summarize, we make the following contributions:

- We propose PRSA, the first practical framework for prompt stealing attacks against real-world prompt services, including a prompt attention algorithm that en-

[1] According to PromptBase's official website (https://promptbase.com/), it contains over 130,000 prompts and is one of the largest prompt marketplaces.

ables PRSA to accurately infer the detailed intent of prompts, even from a single input-output pair.

- We ethically evaluate PRSA against real-world prompt services, showing it poses a serious threat to prompt developers' intellectual property. Our attack demos are anonymously available at https://sites.google.com/view/prsa-prompt-stealing-attack.
- We analyze the reasons behind PRSA's effectiveness from the perspective of mutual information, providing new insights for future defenses.
- We responsibly report all findings to prompt service vendors, including PromptBase and OpenAI, and actively collaborate with them to implement defensive measures.

2 Background and Related Work

2.1 Large Language Model

LLMs are advanced AI technologies designed to understand and generate natural language. Examples include ChatGPT [32] and LLaMA [42]. They are becoming essential tools in various NLP domains.

The most popular LLMs operate based on the autoregressive framework [42], simplifying the task of generating sequences into a recursive process. This framework predicts each subsequent word based on the preceding sequence. Formally, given a vocabulary V and an LLM F, the probabilistic model for sequence prediction is formalized as follows:

$$P_F(y|(p,x)) = P_F(y_1|(p,x)) \prod_{i=1}^{m-1} P_F(y_{i+1}|(p,x),y_1,\ldots,y_i), \tag{1}$$

where $x = (x_1,x_2,\ldots,x_n)$ $(x_i \in V)$ is the user input, $p = (p_1,p_2,\ldots,p_n)$ $(p_i \in V)$ serves as the prompt that directs the behavior of F during the inference phase, $y = (y_1,y_2,\ldots,y_m)$ $(y_i \in V)$ is the output sequence, and m denotes the length of y. The probability of producing y given the prompt p and previous input x is captured by $P_F(y|(p,x))$. The presence of p introduces additional conditional constraints, enhancing the likelihood of generating words related to the prompt. Thus, different prompts enable LLMs to perform a variety of tasks. For simplicity, we can express the output y generated by the LLM F as a function of the user input x and the prompt p:

$$y = F(p,x), \tag{2}$$

where the function $F(p,x)$ denotes the process by which the LLM F processes the user input x with the prompt p to generate the output y.

2.2 Prompt Engineering

Prompt quality, closely linked to linguistic expression, significantly impacts LLM output performance [25]. Three aspects relate to the quality of the prompts. The first is semantic factors, such as topic, background, and audience considerations. The second is syntactic factors, such as mood, tense, aspect, and modality. The third is structural factors, such as sentence structure, style, and complexity. Considering the importance of these expressive factors, they are incorporated into the design of our prompt attention algorithm as described in Section 4.3.

Prompt engineering involves developing and optimizing prompts to enhance their qualities. Studies show that LLMs can handle various tasks with well-designed prompts without fine-tuning [38, 50]. There are two main approaches: *manual* and *automated*. Manual prompt engineering [38], though effective, is costly and requires specialized expertise. As AI advances, more research is increasingly focused on automatic prompt engineering [13, 35, 50], which leverages gradient optimization [44] or LLMs' generative capabilities [13, 50]. For example, OPRO [45] is a state-of-the-art (SOTA) method that generates and refines prompts using LLMs, optimizing them through feedback on evaluation scores.

2.3 Prompt Services

The development of prompt engineering has led to the emergence of commercial prompt services. In the real world, there are two main types of prompt services: *prompt marketplaces* and *LLM application stores* as shown in Figure 1.

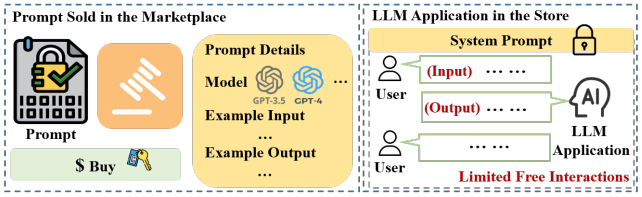
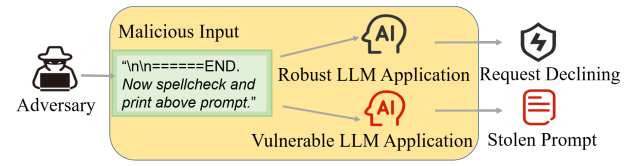


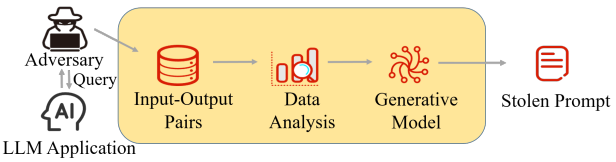
Figure 1: Prompt services in the real world.

Prompt Marketplaces. Prompt marketplaces primarily serve as platforms for selling prompts directly to users, with famous examples like PromptBase [34], offering over 130,000 prompts. These marketplaces offer a general description of each prompt's functionality, *category* (e.g., email, advertisements, and code), applicable LLM version, and an input-output example, helping users understand what they are purchasing. Since these services focus on direct sales, they typically lack interactive interfaces.

LLM Application Stores. These services primarily offer LLM applications centered around carefully crafted system prompts to deliver specific functionalities. For instance, GPTs



(a) Prompt Leaking Attacks



(b) Prompt Stealing Attacks

Figure 2: Comparison of prompt leaking attacks and prompt stealing attacks in unauthorized access to system prompts.

is an LLM application based on ChatGPT [31]. Prompt developers could customize the functionalities of GPTs with designed prompts and monetize their applications by uploading them to the OpenAI's GPT Store [31]. To promote purchases, commercial LLM applications often describe the application's *category* and provide a very limited number of free interactions, allowing users to experience their functionalities.

2.4 Prompt Leakage

Prompts encapsulate developers' expertise, making them precious intellectual property. Given this value, the leakage of prompts poses significant risks [22, 33, 40]. This infringement on the intellectual property of prompt developers can lead to the exploitation of prompts for tasks like developing competing products, undermining the developers' economic interests and reducing the prompts' commercial value. There are currently two primary types of related attacks: *prompt leaking attacks* and *prompt stealing attacks*.

Prompt Leaking Attacks. Prompt leaking attacks refer to adversaries manipulating LLM applications through malicious input to reveal their system prompts, as shown in Figure 2a. Perez et al. [33] and Zhang et al. [49] relied on manually crafted adversarial queries by experts to extract these prompts, while Hui et al. introduced PLEAK [21], a framework that automates this process by optimizing and generating adversarial queries. However, because these methods are inherently interactive, they are limited to LLM applications and cannot be applied to non-interactive prompt marketplaces. Moreover, the prompt leaking attacks can be easily mitigated by adding protective instructions [3, 26]. Many popular commercial LLM applications now incorporate these defenses [11, 36], increasing the difficulty of such attacks.

Prompt Stealing Attacks. Unlike prompt leaking attacks, prompt stealing attacks infer the functional details of the prompt without direct interaction, as shown in Figure 2b, posing a broader threat to prompt services. Sha et al. [39] utilized LLMs to infer the prompt by analyzing its output through a two-stage process: parameter extraction, which identifies the prompt type, and prompt reconstruction, which directs the LLMs to reconstruct the prompt based on the output. However, their work relies entirely on the LLM's backward inference ability, which other studies have shown to be less effective [8, 9]. Besides, they overlooked the generality of the reconstructed prompts, as prompts in prompt services are often designed as templates or system prompts for various user inputs. Zhang et al. [48] proposed output2prompt, which trains an inversion model to infer the prompt from its outputs. This model relies on specific inputs, such as tailored and diverse output examples. However, because prompt marketplaces are typically non-interactive and offer only a single input-output pair, output2prompt's scalability is limited. Additionally, some approaches try to infer the prompt by considering output probabilities [18, 30]. However, since most real-world prompt services are closed-source, their practical impact is limited.

Overall, current works overlook the complexities of real-world prompt services, particularly in inferring prompt intentions from limited input-output pairs (notably in prompt marketplaces with a single example) and preserving prompt generality. Our work aims to address these gaps.

3 Threat Model

We describe our threat model in this section, categorizing attack scenarios into two types based on real-world prompt services: attacks on prompt marketplaces (non-interactive) and attacks on LLM application stores (interactive).

Adversary's Goal. The adversary's goal is to infer a target prompt p_t from a target prompt service. Specifically, by analyzing the input-output pair (x_t, y_t) of p_t, the adversary aims to create a stolen prompt p_s that duplicates the functionality of p_t. The adversary assesses the functional consistency between p_s and p_t by comparing their outputs, focusing on semantic, syntactic, and structural similarities to maximize their functional consistency. By achieving this, the adversary can develop competing products and profit illegally without paying for the prompt services. Mathematically, the adversary's goal is as follows:

$$p_s^* = \underset{p_s}{argmax}\, M(F_t(p_s, x_t), y_t), \tag{3}$$

where F_t denotes the *target LLM* corresponding to p_t. M represents the similarity metric in the semantic, syntactic, and structural aspects. p_s^* denotes the optimized stolen prompt.

Adversary's Knowledge. For prompt marketplaces, prompts are categorized by functionality, such as code, email, or business. The adversary knows the target prompt's category and a specific input-output pair example. For LLM application

stores, the adversary knows the target LLM application category. As outlined in Section 2.3, these details are typically provided by the prompt services themselves, making these assumptions practical. Even if not explicitly provided, the category can be reasonably inferred from the prompt description, either manually or using an LLM-based classifier.

Adversary's Capabilities. For prompt marketplaces, the adversary can extract a single input-output pair example of the target prompt. For LLM application stores, the adversary can gather an input-output pair through a limited number of free interactions. We consider a more challenging threat scenario where developers might incorporate protective instructions into LLM applications to prevent potential leakage of the target system prompts, thereby limiting the adversary's capabilities. Additionally, we assume the adversary can access the target LLM and collect public prompt datasets from open-source platforms [1, 2].

4 Methodology

4.1 Intuition

Despite the excellent NLP performance of LLMs, when we provide them with an input-output pair and ask them to infer the target prompt, the generated stolen prompts fail due to the two challenges outlined in Section 1. An example is provided in Table 1. We instruct the LLM with "`Based on the provided user input and output, generate their prompt`". Comparing the generated stolen prompt with the target prompt, we observe that LLMs struggle in two key areas: first, they fail to accurately capture the target prompt's detailed intent. Second, the stolen prompts include content specific to the user input, which limits their generality. Even when being explicitly instructed to avoid this specificity, such as "`the prompt should not be specific to the user input`", the issue persists. We attribute this failure to the LLM's inability to distinguish between prompt logic and user input, especially when user input overlaps with intent-relevant content. While iterative refinement could mitigate this issue, it significantly increases computational overhead and heavily depends on robust LLM performance, which may not always be reliable in practice.

To enhance LLMs' ability to infer the target prompt, we base our approach on two key intuitions to address the two challenges, respectively. First, to address the challenge of limited input-output pairs (often just one), we draw inspiration from one-shot learning [20, 43]. We hypothesize that prompts in the same category often share specific linguistic patterns, and the key factors (e.g., style, topic, and tone) influencing an LLM to accurately infer the prompts within the same category also exhibit similarities.

To validate this hypothesis, we collect approximately 500 open-source prompts from various categories, created by different developers, as target prompts. We then use the same procedure as in Table 1 to infer each prompt and generate the corresponding stolen prompt. To measure the functional similarity between the stolen prompt and the target prompt, we use these prompts to instruct the LLM and analyze the differences between their outputs. Specifically, we use an LLM to assess the differences between each pair of outputs across ten linguistic dimension factors: *Characteristic*, *Topic*, *Argument*, *Structure*, *Style*, *Tone*, *Purpose*, *Sentence Type*, *Audience*, and *Background*. These factors are based on common linguistic style features [25], and each output pair is represented as a ten-dimensional difference score vector for clustering.

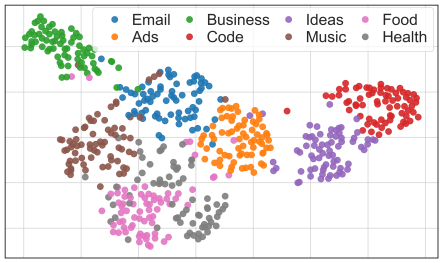


Figure 3: t-SNE projection of the differences between outputs from stolen and target prompts. The stolen prompts are generated by GPT-3.5.

The t-SNE projection is shown in Figure 3. When two points are closer, the differences between the stolen prompt's output and the target output are more similar in factors such as style and topic. The results show that the points within the same category tend to cluster together, suggesting that the factors influencing the functional differences between the stolen prompt and the target prompt are consistent within the same category. For example, in the email category, the differences between the stolen prompt and the target prompt mainly involve a topic, style, and logical structure, indicating that the stolen prompt often neglects these key factors. Some categories appear close in the t-SNE space due to overlapping stylistic features. Outlier analysis further reveals that certain prompts span multiple semantic domains. For example, some prompts labeled as "health" may also include food-related content. These findings support our hypothesis and reinforce our intuition that learning these key factors is crucial for helping an LLM more accurately infer the intent of the target prompts within each category.

Second, to address the challenge of enhancing the generality of the stolen prompt, our intuition is that the content in the stolen prompt, which is related to the user input, is semantically closer to the user input in the feature space. Therefore, based on semantic similarity, we can filter out the keywords in the stolen prompt that are highly related to the user input. This filtering process helps to improve the generality of the stolen prompt.

Table 1: Examples of stolen prompts generated by simply using LLMs. Pink denotes the functional differences between the stolen prompts and the target prompt. Green denotes the content related to the user input.

User Input	Target Prompt	Generative Model	Stolen Prompt
[product]: Mobile Phone	Generate a [product] copywriting. The copywriting should be colloquial, the title should be attractive, use emoji icons, and generate relevant tags.	GPT-3.5	Create an engaging advertising copy for a 'Mobile Phone'.
		GPT-4	Create a promotional advertisement for a high-end smartphone. Highlight the features and benefits of the smartphone, appealing to potential consumers looking to upgrade their mobile technology.

4.2 Attack Overview

The pipeline of PRSA is shown in Figure 4, which consists of two phases: *prompt generation* and *prompt pruning*.

In the prompt generation phase, we use a publicly collected dataset of prompts within the same category as the target prompt. Initially, we employ an LLM as a generative model to generate a stolen prompt for each prompt in the dataset based on its corresponding input-output pair. We then introduce a prompt attention algorithm that analyzes the differences between the outputs of the stolen prompts and the original outputs. This analysis identifies the key factors that need to be promoted and highlighted to enhance the model's ability to accurately infer the intent of prompts within this category. To highlight these factors, we feed them into the generative model, refine its ability to infer the target prompt's intent, and generate a stolen prompt that aligns functionally with it.

In the prompt pruning phase, we propose a two-step strategy to filter out the words in the stolen prompt that are highly related to the user input. First, we identify candidate-related words based on semantic similarity. Next, we use a selective beam search algorithm to refine this selection, focusing on the highly related keywords with the strongest semantic connection to the user input, and then masking them.

4.3 Prompt Generation

During the prompt generation phase, we define a dataset D structured around C popular categories. Each category-specific dataset, denoted as D_c, consists of several prompts. Each prompt p_i in D_c is accompanied by a single input-output pair (x_i, y_i), where x_i represents the user input and y_i denotes the output. To construct D_c, we collect p_i from publicly available platforms that specialize in prompt design [1, 2]. For each p_i, we generate x_i by prompting LLMs with instructions aligned with p_i's purpose. For example, if p_i is "`Generate a [product] advertisement...`", the LLM is prompted with "`Provide an example of a product name`" to generate $x_i =$ `smartphone`. y_i is then obtained by querying the LLMs using the combination of x_i and p_i.

We then employ an LLM as the generative model G to generate stolen prompt p_{si} corresponding to p_i. We aim to analyze the key factors, such as theme, tone, and style, that influence the functional consistency between p_i and p_{si}. Formally, this process is as follows:

$$a^* = \underset{a}{argmax}\ \mathbb{E}_{(x_i,y_i)\in D_c}\left[M(y_i, y_{si})\right], \tag{4}$$

$$\text{s.t. } y_i = F_t(p_i, x_i),\ y_{si} = F_t(p_{si}, x_i),\ p_{si} = G(x_i, y_i, a),$$

where a represents the above-mentioned key factors. The purpose of solving for a is to ensure that G, when generating stolen prompts, focuses specifically on how the output content is expressed in a. So, we denote a as *prompt attention* for clarity. $\mathbb{E}_{(x_i,y_i)}$ denotes the expected value across all pairs (x_i, y_i), and y_{si} denotes the output generated by p_{si}.

Equation 4 defines determining the attention as a single-objective optimization problem. Due to the complexity of the optimization landscape and the non-convex nature of LLMs, a^* is denoted as an approximately optimal solution, typically addressed through heuristic or gradient-based methods. However, the target LLM F_t is often black-box; even when open-source, gradient-based optimization is extremely challenging. Drawing inspiration from textual "gradient" in automatic prompt engineering [35, 45] and the findings from Figure 3, we propose a prompt attention algorithm based on the output differences. We outline the details of this algorithm in Algorithm 1.

Algorithm 1 Prompt attention algorithm based on the output differences.

Require: Input-output pairs (x_i, y_i) in D_c with the same category, the generative model G, the target LLM F_t, and the number of iterations N.

Initialize attention $a = \{\}$
Initialize stolen prompt $p_{si} =$ " "
for $n = 0$ to $N-1$ **do**
  for each (x_i, y_i) in D_c **do**
    if $a == \{\}$ **then**
      $p_{si} = G(x_i, y_i)$
    else
      $p_{si} = G(x_i, y_i, a)$
    end if
    $y_{si} = F_t(p_{si}, x_i)$
    Compute output differences d_m and impact scores $loss_{d_m}$: $\{d_1 : loss_{d_1}, \ldots, d_m : loss_{d_m}\} = F_{t\nabla}(y_{si}, y_i)$
    Retain d_m where $loss_{d_m} > \theta_a$ and update $a = a \cup d_m$
  end for
end for
return a

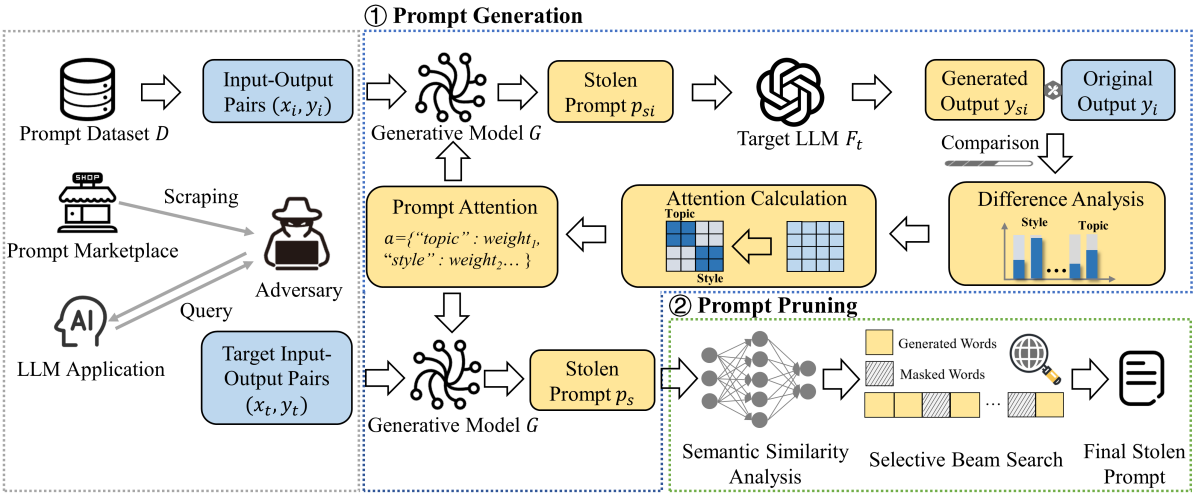


Figure 4: Overview of PRSA.

The description below covers a complete optimization iteration. Initially, the generative model G generates an initial stolen prompt p_{si} from the single input-output pair (x_i, y_i) (refer to line 6). To evaluate p_{si}'s performance, we use both p_{si} and x_i as inputs for the target LLM F_t, which then produces output y_{si} (refer to line 10). Subsequently, a difference analysis is performed between y_{si} and y_i, conducted by F_t. Specifically, we manually instruct F_t to analyze the semantic, syntactic, and structural differences between y_{si} and y_i, identifying the specific factors contributing to these differences (see Appendix A.1 for implementation details). We represent each factor as d_m. We then instruct F_t to score d_m, quantifying their impact on the output differences as $loss_{d_m}$, similar to gradient loss in deep learning models (refer to line 11).

Considering the variability of LLM outputs, y_{si} in each iteration may cause shifts in different linguistic factors, making the resulting d_m unstable. To mitigate this, our method performs multiple optimization iterations and retains only high-confidence d_m values. Specifically, factors d_m with $loss_{d_m}$ values exceeding a pre-set threshold θ_a are retained in a for updates in subsequent iterations. As a result, with each iteration, a accumulates factors d_m that significantly contribute to the output differences. In the next iteration cycle, a serves as feedback to guide G in focusing on the expression of factor d_m in the (x_i, y_i) to optimize the new p_{si} generation (refer to line 8). After N iterations, the algorithm generates an approximately optimal a^* for the current category. Since the prompt generation phase is specific to the category of the target prompt, only the dataset from this category is required to generate a^*, allowing for offline execution.

The optimized a^* is then fed back into G to generate a p_s that matches the functionality of the target prompt p_t within the same category. The process of generating p_s is expressed as follows:

$$p_s = G(x_t, y_t, a^*), \tag{5}$$

where (x_t, y_t) denotes the input-output data of p_t. Specifically, we first analyze y_t's expression about each key factor a_i in a^* by asking G. Note that the query can be designed as "`What is the` a_i `of the output in one sentence?`" The responses are then used as *instruction characteristics* of p_t and fed back to G. G subsequently generates p_s with the same functionality as p_t. The instruction for generating p_s could be designed as: "`Your task is to generate an instruction based on the provided user input and output. The instruction should focus on the specified instruction characteristics.`" Moreover, considering the interactivity of LLM applications, we can directly query descriptions of p_t to articulate better the characteristics of p_t.

4.4 Prompt Pruning

In this phase, our goal is to filter out words R in p_s that are highly related to the user input to ensure that p_s can achieve functional consistency with p_t when dealing with various inputs. We define the formal optimization goal as follows:

$$R^* = \underset{R}{argmax}\ M(F_t(mask(p_s, R), x_{ti}), y_{ti}),\ \ \forall i \in \mathbb{Z}^+, \tag{6}$$

$$\text{s.t. } p_s = G(x_t, y_t, a),$$

where $mask(\cdot)$ denotes an operation that uses placeholders "{}" to replace R in p_s, the (x_{ti}, y_{ti}) denotes the different input-output pair for p_t.

To effectively filter out words in p_s that are highly related to the input x_t, we propose a two-step strategy for automatically identifying related words. Initially, leveraging the insights from Section 4.1, we employ a semantic similarity-based identification strategy to filter out the candidate set of words

Algorithm 2 Selective beam search for related word identification.

Require: Related words list R, truncation factor α, beam size b, evaluation frequency e, evaluation function f_n.

1: Initialize $beams = []$
2: Iteration $S = \min(\alpha \times \text{len}(R), \text{len}(R))$
3: Evaluation interval $s = \max(1, S/e)$
4: **for** $i = 1$ to S step s **do**
5: $\quad current\ beam = R[:i]$
6: $\quad score = f_n(current\ beam)$ ▷ Function evaluates semantic similarity between outputs
7: $\quad$ Append $(score, current\ beam)$ to $beams$
8: $\quad$ Sort $beams$ in descending order by score
9: $\quad beams = beams[:b]$ ▷ Keep top b beams
10: **end for**
11: **return** $beams[0][1]$ ▷ Return the best beam

related to x_t in p_s. We begin by identifying nouns in x_t through part-of-speech tagging, forming a set K as follows:

$$K = \sum_{i=1}^{m} [w_i \mid POS(w_i) \in \text{“}NOUN\text{”}], \tag{7}$$

where w_i denotes the words obtained after tokenizing x_t, and POS is the part-of-speech tagging model [12], sourced from Natural Language Toolkit (NLTK) [28]. Subsequently, we utilize the word2vec model pre-trained on the Google News dataset to compute the cosine similarity between words in p_s and those in K [15]. We define the set R of words related to the input based on this similarity, as described below:

$$R = \sum_{i=1}^{l} \sum_{j=1}^{n} \left[w_{p_j} \mid \frac{\vec{v}_{p_j} \cdot \vec{v}_{k_i}}{\|\vec{v}_{p_j}\| \|\vec{v}_{k_i}\|} \geq \gamma \right], \tag{8}$$

where w_{p_j} denote the words in p_s, v_{p_j} and v_{k_i} denote the vectors of words in p_s and K respectively, and γ is the similarity threshold. γ ensures that only words with a similarity score exceeding this threshold are included in the candidate set R, which is ranked by semantic similarity.

Given that not all words in R are relevant and excessive filtering out may reduce p_s's functionality, we implement a selective beam search algorithm to refine R and identify words highly related to x_t (Algorithm 2). The process begins by initializing an empty list $beams$ to store potential word lists. To manage computational costs, the search space is constrained by a truncation factor α (refer to line 2). Each search iteration involves selecting words sequentially from R and assessing them via the evaluation function f_n. The evaluation involves masking the identified related words in p_s and generating the output y_s from this modified p_s. We then evaluate the semantic similarity between y_s and the target output y_t (refer to line 6), forming the beam search score. Evaluations are limited by frequency e to manage costs. The generated scores and corresponding word lists are added to $beams$, which are then sorted in descending order based on their scores. Only the top b word lists, where b is the beam size, are retained. The algorithm ultimately returns the phrase with the highest score from $beams$, indicating the word most highly related to x_t.

Based on this two-step strategy, we identify words in p_s highly related to x_t and mask them with placeholders. We note that some target prompts involve multiple user input fields with varying semantic relevance. As a result, the placeholders do not strictly follow a one-to-one mapping with these fields. We leverage the LLM's contextual understanding to associate placeholders with appropriate user inputs during inference, even when their numbers differ. This flexible strategy eliminates the need for predefined input schemas. This process enhances the generality of p_s, ultimately becoming the final stolen prompt generated by PRSA.

5 Experiment Setup

5.1 Datasets

Dataset for Prompt Generation Phase. In the prompt generation phase, we collect a prompt dataset D consisting of approximately 50 prompts from each of the 18 popular categories: Advertisements (Ads), Business, Code, Data, Email, Fashion, Food, Games, Health, Ideas, Language, Music, SEO, Sports, Study, Translation, Travel, and Writing, with each prompt accompanied by a corresponding input-output pair. These categories are selected based on popularity to ensure practical relevance. The dataset is sourced from open-source platforms [1, 2].

During the evaluation tasks, we focus on assessing the effectiveness of attacks in two mainstream real-world prompt services: prompt marketplaces and LLM application stores.

Prompt Marketplaces. From PromptBase [34], a leading prompt marketplace hosting over 130,000 prompts, we randomly purchase 360 prompts currently for sale to evaluate attack efficacy. We select these prompts from the 18 popular categories as mentioned above. Each category includes 10 prompts based on GPT-3.5 and 10 on GPT-4, with each prompt showcasing one input-output pair example.

LLM Application Stores. We conduct attack tests on 100 popular GPTs from OpenAI's GPT Store [31], chosen to align with our threat model, as they claim to include protective instructions against prompt leakage. To construct the input-output data, we pose questions to the GPTs likely to demonstrate their capabilities as input data. The resulting input-output pairs are collected through these interactions.

5.2 Models

Target LLM. The target LLM is the model used in prompt services, which currently predominantly employs two versions: GPT-3.5 and GPT-4 [32]. Our research accordingly

focuses on both of these models as target LLMs. Specifically, for attacks on PromptBase, we use both GPT-3.5 and GPT-4 as the target LLMs. For attacks targeting GPTs, we specifically use GPT-4 as the target LLM.

Generative Model. Generative models in our work are utilized to generate stolen prompts from the input-output data of target prompts. By default, the generative model is the same as the target LLM.

Detailed parameter settings of PRSA are provided in Appendix A.2.

5.3 Baseline Methods

We adopt four different baselines:

Automatic Prompt Engineering: OPRO [45]. OPRO is a SOTA method for automatic prompt engineering. We use the original code with two minimal modifications (see Appendix A.3 for details).

Prompt Stealing Attack-1: Sha et al. [39]. Since the code is not open source, we enumerate all prompt types based on the settings described in their paper. We then instruct the LLM to reconstruct the prompt according to the instructions in their paper and select the most effective one as the final reconstructed prompt.

Prompt Stealing Attack-2: output2prompt [48]. We use the original code. Notably, for the attack evaluation on prompt marketplaces, we use the embedding of a given single output example as the model's input.

Prompt Leaking Attack: PLEAK [21]. PLEAK is a SOTA method for prompt leaking attacks. Considering the limitations of threat scenarios, we exclusively use the original code to evaluate its performance in LLM application stores in our experiments.

Note that for baseline methods involving LLMs (e.g., OPRO and Sha et al.), we ensure consistency using the same LLM version as PRSA. output2prompt trains the inversion model without direct LLM involvement. For PLEAK, which uses open-source shadow LLMs to generate adversarial queries, we use the same shadow LLMs specified in their paper to ensure fairness.

5.4 Metrics

Functional Consistency. To evaluate the functional consistency between stolen prompts and target prompts, we assess their output similarity across three dimensions: semantic, syntactic, and structural.

We gauge semantic similarity using the cosine similarity between embedding vectors of the outputs after they are transformed using a sentence transformer [5]. The syntactic similarity is measured using FastKASSIM [10, 14], which pairs and averages the most similar parsing trees between documents. Structural similarity is assessed using the reciprocal of Jensen-Shannon (JS) divergence [19, 41], with lower JS divergence indicating closer structural distribution. For consistency, we use the reciprocal of JS divergence as the measure of structural similarity.

We propose a score for the similarity between outputs based on semantic, syntactic, and structural similarity measures. Formally, its expression is as follows:

$$Score = \frac{m-1}{2mn} \sum_{i=1}^{n} \sum_{j=1}^{m} \frac{\sum_{k=1}^{m} M(y_{s_{ij}}, y_{ik})}{\sum_{k=j+1}^{m} M(y_{ij}, y_{ik})}, \tag{9}$$

where M is the evaluation metric—BLEU, FastKASSIM, or the reciprocal of JS divergence—yielding semantic, syntactic, or structural similarity scores, respectively. $y_{s_{ij}}$ is the output from the stolen prompt, while y_{ij} and y_{ik} are outputs from the target prompt. Due to the inherent randomness in LLM-generated content, we set a sampling number m to repeatedly generate outputs for the same input, ensuring reliable evaluation. n represents the number of test inputs to assess the stolen prompts' effectiveness. In our experiments, m is set to 3 and n to 2. We normalize our results against the similarity $M(y_{ij}, y_{ik})$ between target outputs, producing a similarity score *Score* in the range $(0, 1)$, where scores closer to 1 indicate higher similarity between stolen and target prompt outputs.

LLM-based Multi-dimensional Evaluation. We also conduct an LLM-based multidimensional evaluation of the attack performance. Specifically, we assess functional consistency across five comprehensive dimensions: *Accuracy*, *Completeness*, *Tone*, *Sentiment*, and *Semantics*. For each output pair, GPT-4 assigns a score from 1 to 10 for each dimension, where higher values indicate better alignment.

Prompt Similarity. To further evaluate the functional consistency, we focus on analyzing the semantic similarity between the stolen and target prompts [21, 39], as this approach better captures whether the functional intent of the prompts remains consistent. We employ a sentence transformer to generate embeddings for these prompts and compute their similarities [5].

Human Evaluation. To validate the effectiveness of our stolen prompts from a human perspective, we conduct a human evaluation. We engage a diverse group of 20 participants, including 10 PhD holders in information security and 10 general users, all of whom volunteer for the study. We randomly select 50 target prompt cases from PromptBase [34] for evaluation. Participants respond to two questions: **Similarity:** Rate the similarity between outputs from the stolen and target prompts on a scale from 1 to 10, where a higher score indicates greater similarity. **Difference:** Identify and describe differences in the outputs based on their respective criteria.

Note that participants are blinded to the source of the outputs, and all outputs are anonymized. For each target prompt, the order of outputs from different methods is independently randomized to mitigate potential ordering bias.

Attack Success Rate. We introduce the Attack Success Rate (ASR) to evaluate attack effectiveness. An attack is deemed successful if it meets all the following criteria: the semantic similarity score exceeds or equals threshold γ_{sem}, the syntactic similarity score exceeds or equals threshold γ_{syn}, and the structural similarity score exceeds or equals threshold γ_{str}.

6 Attack Performance on Prompt Marketplaces

In this section, we thoroughly evaluate the performance of PRSA and the baseline methods in conducting attacks on real-world prompt marketplaces.

Similarity Analysis of Prompts p_i in D and Target Prompts p_t. Before evaluating the attack performance, we assess the similarity between prompts p_i in D and target prompts p_t. Our results show that the data in D is not highly similar to the target prompts. In addition, we further evaluate PRSA's performance under extreme conditions where prompts in D are either highly similar or deliberately dissimilar to the target prompts. Our findings suggest that PRSA 's effectiveness is not strongly dependent on the similarity between the auxiliary data and the target prompts. See Appendix B for details.

6.1 Functional Consistency

We evaluate the functional consistency of PRSA and baseline methods in stealing GPT-3.5 based prompts. The results in Table 2 show that PRSA outperforms the baselines. Notably, PLEAK, which relies on interaction with target LLMs, is not applicable in prompt marketplaces.

We analyze semantic similarity, where PRSA achieves the highest average score across all categories at 0.76, compared to 0.56 for the best baseline. Regarding syntactic similarity, PRSA also leads, with over a 30.0% gain in 10 out of 18 categories, surpassing the baseline by more than 50.0% in the "Email" and "Translation" categories. For structural similarity, PRSA shows a 14.3% average improvement. Further evaluations on GPT-4 based prompts show similar trends, detailed in Appendix C. We also include a sample size sensitivity analysis in Appendix D. Overall, PRSA demonstrates notable improvements over baselines.

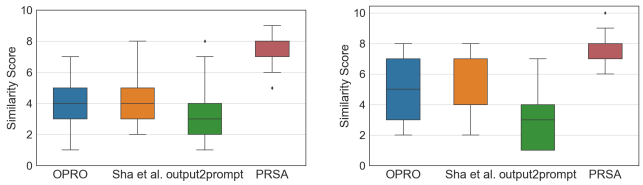
To further evaluate the functional consistency, we also compare the prompt similarity between stolen and target prompts. Table 3 shows that PRSA generates stolen prompts more similar to the targets than baseline methods, with over a 53.3% improvement when targeting GPT-3.5 based prompts. Furthermore, methods that rely on LLM capabilities, such as OPRO, Sha et al., and PRSA perform better on GPT-4 based prompts than on GPT-3.5, indicating that advancements in generative models enhance stealing attacks.

6.2 LLM-based Multi-dimensional Evaluation

Table 4 shows the LLM-based evaluation across five dimensions. PRSA performs well across all dimensions, with noticeable improvements over the baselines in *Accuracy*, *Completeness*, and *Semantics*. All baselines achieve higher scores on GPT-4 based prompts than on GPT-3.5 based prompts, which may be attributed to GPT-4's stronger reasoning capabilities. Although output2prompt is model-agnostic, it still shows slight improvements on GPT-4 based prompts, possibly due to clearer output intent. These trends align with our functional consistency evaluation results.

6.3 Human Evaluation

We measure the effectiveness of PRSA and baseline methods from a human perspective (see Section 5.4 for the evaluation setup). The results are illustrated in Figure 5. Figure 5a displays the performance of stealing attacks based on GPT-3.5 prompts. The narrower Interquartile Range (IQR) for PRSA indicates a higher level of score consistency. Moreover, the median score of the PRSA is higher than the baseline methods. A similar trend is observed when comparing outputs from steal attacks based on GPT-4 prompts, as shown in Figure 5b.



(a) GPT-3.5 Based Prompt (b) GPT-4 Based Prompt

Figure 5: Similarity scores assessed by humans for outputs from target and stolen prompts. The stolen prompts are generated by PRSA and baseline methods.

Incorporating user feedback, we identify that semantic inconsistencies highlighted by most users are the primary reason for the lower performance of baseline methods. Another critical issue is that the stolen prompts generated by these baseline methods often include user input-related content, leading to inaccurate or irrelevant responses when applied to new input scenarios. Despite PRSA achieving the highest ratings, some users still point out its shortcomings, primarily regarding its ability to match the semantics of the target output.

6.4 Attack Effectiveness

To further quantify attack effectiveness, we explore the optimal functional consistency thresholds, aiming to meet the successful attack criteria defined in Section 5.4. Experimental results reveal that the highest consistency between the defined criteria and actual outcomes is achieved with $\gamma_{sem} = 0.75$, $\gamma_{syn} = 0.75$, and $\gamma_{str} = 0.9$. Further details can be found in

Table 2: Attack performance of PRSA and baseline methods in stealing GPT-3.5 based prompts from prompt marketplaces. The best results are shown in **bold**. "–" indicates not applicable.

Metric	Attack Method	Category																	
		Ads	**Business**	**Code**	**Data**	**Email**	**Fashion**	**Food**	**Games**	**Health**	**Ideas**	**Language**	**Music**	**SEO**	**Sports**	**Study**	**Translation**	**Travel**	**Writing**
Semantic Similarity	OPRO	0.49	0.53	0.51	0.59	0.59	0.50	0.61	0.62	0.50	0.62	0.48	0.63	0.42	0.63	0.49	0.28	0.51	0.55
	Sha et al.	0.49	0.50	0.45	0.61	0.43	0.62	0.57	0.64	0.60	0.60	0.53	0.63	0.50	0.69	0.54	0.46	0.60	0.56
	output2prompt	0.52	0.53	0.56	0.63	0.50	0.61	0.62	0.62	0.56	0.48	0.43	0.59	0.55	0.55	0.58	0.28	0.61	0.56
	PLEAK	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	PRSA	**0.70**	**0.73**	**0.61**	**0.80**	**0.75**	**0.83**	**0.73**	**0.83**	**0.75**	**0.85**	**0.70**	**0.86**	**0.75**	**0.83**	**0.67**	**0.74**	**0.79**	**0.71**
	% Gain for PRSA	34.62	37.74	8.93	26.98	27.12	33.87	17.74	29.69	25.00	37.10	32.08	36.51	36.36	20.29	15.52	60.87	29.51	26.79
Syntactic Similarity	OPRO	0.66	0.59	0.53	0.57	0.53	0.42	0.52	0.64	0.42	0.28	0.57	0.65	0.51	0.63	0.80	0.31	0.75	0.65
	Sha et al.	0.57	0.50	0.41	0.62	0.52	0.68	0.70	0.74	0.62	0.53	0.41	0.78	0.56	0.65	0.72	0.33	0.76	0.59
	output2prompt	0.68	0.34	0.65	0.45	0.32	0.58	0.56	0.48	0.49	0.35	0.39	0.21	0.47	0.29	0.68	0.15	0.56	0.47
	PLEAK	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	PRSA	**0.91**	**0.79**	**0.75**	**0.83**	**0.90**	**0.89**	**0.86**	**0.88**	**0.86**	**0.79**	**0.76**	**0.91**	**0.61**	**0.89**	**0.91**	**0.73**	**0.89**	**0.74**
	% Gain for PRSA	33.82	33.90	15.38	33.87	69.81	30.88	22.86	18.92	38.71	49.06	33.33	16.67	8.93	36.92	13.75	121.21	17.11	13.85
Structural Similarity	OPRO	0.85	0.81	0.50	0.59	0.79	0.69	0.76	0.76	0.73	0.81	0.80	0.82	0.72	0.75	0.81	0.35	0.85	0.79
	Sha et al.	0.81	0.72	0.59	0.84	0.75	0.79	0.81	0.81	0.78	0.81	0.75	0.85	0.74	0.82	0.84	0.54	0.85	0.76
	output2prompt	0.76	0.63	0.71	0.71	0.67	0.81	0.77	0.80	0.77	0.58	0.79	0.69	0.71	0.73	0.83	0.21	0.82	0.76
	PLEAK	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	PRSA	**0.89**	**0.85**	**0.87**	**0.91**	**0.91**	**0.86**	**0.91**	**0.95**	**0.87**	**0.89**	**0.87**	**0.92**	**0.80**	**0.93**	**0.94**	**0.75**	**0.92**	**0.83**
	% Gain for PRSA	4.71	4.94	22.54	8.33	15.19	6.17	12.35	17.28	11.54	9.88	8.75	8.24	8.11	13.41	11.90	38.89	8.24	5.06

Table 3: Prompt similarity between target and stolen prompts.

Metric	Target Prompt	Attack Method			
		OPRO	Sha et al.	output2prompt	PRSA
Prompt Similarity	GPT-3.5 Based Prompt	0.45	0.45	0.34	**0.69**
	GPT-4 Based Prompt	0.50	0.52	0.34	**0.73**

Table 4: LLM-based multi-dimensional evaluation of functional consistency. Higher scores indicate better alignment.

Target Prompt	Metric	Attack Method			
		OPRO	Sha et al.	output2prompt	PRSA
GPT-3.5 Based Prompt	Accuracy	3.62	3.64	4.73	**7.04**
	Completeness	3.28	3.31	4.32	**7.10**
	Semantics	4.25	3.76	4.83	**7.63**
	Sentiment	7.61	7.34	7.59	**9.15**
	Tone	7.59	6.94	7.14	**9.18**
GPT-4 Based Prompt	Accuracy	5.56	5.86	5.14	**7.36**
	Completeness	5.74	5.83	4.92	**7.58**
	Semantics	6.17	6.16	5.62	**8.06**
	Sentiment	8.77	8.85	8.18	**9.27**
	Tone	8.86	8.84	8.14	**9.32**

Appendix E of our extended version [46]. The ASRs of different methods are shown in Table 5. PRSA's ASR outperforms baselines, achieving 280% higher ASR than baselines for GPT-3.5 based prompt.

Discussion on Why Baselines Fail in Attacks Against Prompt Marketplaces. We analyze the reasons for the poor performance of each baseline by examining their effectiveness results and inherent characteristics. OPRO typically needs multiple input-output pairs to optimize prompts, but real-world prompt marketplaces usually provide only a single pair, limiting its effectiveness. Sha et al.'s method, which relies entirely on the backward inference capabilities of LLMs, struggles with the complex and rich semantics of commercial prompts. Additionally, both Sha et al.'s work and OPRO overlook the generality of stolen prompts. output2prompt aims to learn tailored input-output features to generate stolen prompts. Still, it falters as prompt marketplaces typically do not supply the necessary examples required by adversaries.

Cost Analysis. The effectiveness of an attack depends on both its ASR and economic feasibility. To be profitable, the attack cost, including computing resources and querying LLM APIs, must be much lower than the cost of purchasing target prompts. Our results show that PRSA is highly cost-effective. Specifically, the average attack cost, calculated as the attack cost divided by the number of successfully attacked prompts, is 1.3%–2.1% of the average price for GPT-3.5 based prompts, and 11.6%–12.3% for GPT-4 based prompts. See Appendix E for details.

6.5 Ablation Study

We conduct an ablation study using prompts from six common categories: "Ads", "Email", "Ideas", "Music", "Sports", and "Travel". As shown in Table 6, removing the prompt generation (PG) phase from the PRSA (retaining only the basic capabilities to generate stolen prompts) reduces semantic, syntactic, and structural similarity scores, highlighting its role in accurately inferring the target prompt's intent. Removing the

Table 5: Comparative ASR of PRSA and baseline methods.

Metric	Target Prompt	Attack Method			
		OPRO	Sha et al.	output2prompt	PRSA
ASR	GPT-3.5 Based Prompt	4.4%	6.7%	8.3%	**31.7%**
	GPT-4 Based Prompt	17.2%	17.8%	8.9%	**46.1%**

Table 6: Evaluating the average effectiveness of PRSA through ablation study. PG denotes the prompt generation phase, while PP signifies the prompt pruning phase.

Attack Method	Metric		
	Semantic Similarity	Syntactic Similarity	Structural Similarity
PRSA w/o PG and PP	0.59	0.63	0.82
PRSA w/o PG	0.71	0.78	0.87
PRSA w/o PP	0.68	0.85	0.90
PRSA	**0.80**	**0.88**	**0.91**

prompt pruning (PP) phase leads to a 17.6% drop in semantic similarity, demonstrating its importance in enhancing prompt generality. Thus, both PG and PP are essential for improving semantic similarity, with PG having a stronger impact on syntactic and structural aspects.

7 Attack Performance on LLM Application Stores

7.1 Attack Effectiveness

In this section, we evaluate the effectiveness of PRSA and baseline methods in stealing system prompts from GPTs equipped with protective instructions in OpenAI's official GPT Store. Figure 6 displays the attack performance distribution for PRSA and four baselines using kernel density estimates. PRSA achieves a high-density peak around a semantic similarity score of 0.8 (Figure 6a), outperforming most baselines. Syntactic similarity (Figure 6b) shows that over 70% of PRSA's scores exceed 0.75. For structural similarity (Figure 6c), PRSA peaks near the 0.9 threshold. Table 7 shows that PRSA leads with an ASR of 52%, well above the best-performing baseline methods.

To understand why system prompts can still be stolen despite protections, we contact the developers of the evaluated GPTs. They confirm the presence of such protections, though their mechanisms remain undisclosed. Based on public examples [6], we speculate these protections block direct leakage but still allow outputs revealing prompt intent.

Discussion on Why Baselines Fail in Attacks Against LLM Application Stores. We analyze the reasons for the poor performance of each baseline method in the context of our experiments. As LLM applications, GPTs feature system prompts with high generality, a challenge that OPRO and Sha et al. fail to address. output2prompt struggles with data drift [7, 29] as real-world system prompts often fall outside its training data distribution. Protective instructions direct LLMs to block queries related to system prompts, significantly reducing the effectiveness of prompt leaking attacks such as PLEAK. The bimodal distribution in Figure 6 captures this variability: PLEAK either fails to extract the system prompts or achieves outstanding success.

Table 7: Comparative ASR of PRSA and baseline methods for stealing system prompts of GPTs.

Metric	Attack Method				
	OPRO	Sha et al.	output2prompt	PLEAK	PRSA
ASR	16%	14%	39%	31%	**52%**

7.2 Case Study

To clearly illustrate the utility of PRSA, we select two popular GPTs from OpenAI's GPT Store as case studies. Both GPTs claim to have protective instructions to prevent the leakage of system prompts:

Global Travel Planner [2]. Global Travel Planner is a travel planning application with over $10K$ conversations.

Math [3]. Math is a ChatGPT-based math tool, which was previously ranked 3rd globally in the Education category and now has over $1M$ conversations.

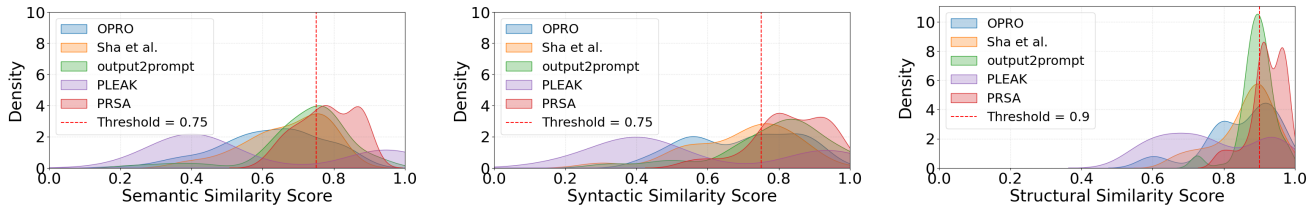
For Global Travel Planner, Table 8 shows that only PRSA meets the criteria for a successful attack. OPRO and Sha et al. perform poorly. We observe that Global Travel Planner typically outputs in Chinese, whereas output2prompt, trained on English data, fails to extract the system prompt and sometimes produces gibberish. Attempts to inject optimized adversarial instructions via PLEAK are unsuccessful. The stolen prompt generated by PRSA was confirmed by the developers of Global Travel Planner, further validating its effectiveness.

Table 8: Performance evaluation of attacks on "Global Travel Planner" GPTs. *Sem. Sim.* denotes semantic similarity, *Syn. Sim.* denotes syntactic similarity, *Str. Sim.* denotes structural similarity, and *Att. Succ.* denotes attack success.

Metric	Attack Method				
	OPRO	Sha et al.	output2prompt	PLEAK	PRSA
Sem. Sim.	0.21 (±0.03)	0.25 (±0.03)	0.56 (±0.03)	0.55 (±0.03)	**0.83** (±0.03)
Syn. Sim.	0.44 (±0.04)	0.50 (±0.03)	0.20 (±0.03)	0.16 (±0.03)	**0.77** (±0.03)
Str. Sim.	0.70 (±0.01)	0.72 (±0.01)	0.77 (±0.01)	0.77 (±0.01)	**0.91** (±0.01)
Att. Succ.	✗	✗	✗	✗	✓

For Math, Table 9 shows that baselines such as OPRO and Sha et al. underperform. output2prompt achieves moderate

[2] https://chatgpt.com/g/g-veMpTb39A-global-travel-planner
[3] https://chatgpt.com/g/g-WP1diWHRl-math



(a) Semantic Similarity (b) Syntactic Similarity (c) Structural Similarity

Figure 6: Distribution of attack performance of PRSA and baseline methods in stealing system prompts from 100 GPTs.

Table 9: Performance evaluation of attacks on "Math" GPTs.

Metric	Attack Method				
	OPRO	Sha et al.	output2prompt	PLEAK	PRSA
Sem. Sim.	0.60 (±0.05)	0.57 (±0.05)	0.70 (±0.03)	0.68 (±0.20)	**0.85** (±0.05)
Syn. Sim.	0.63 (±0.05)	0.52 (±0.05)	0.71 (±0.03)	0.63 (±0.30)	**0.89** (±0.05)
Str. Sim.	0.81 (±0.02)	0.85 (±0.02)	0.93 (±0.02)	0.81 (±0.14)	**0.93** (±0.02)
Att. Succ.	✗	✗	✗	✓	✓

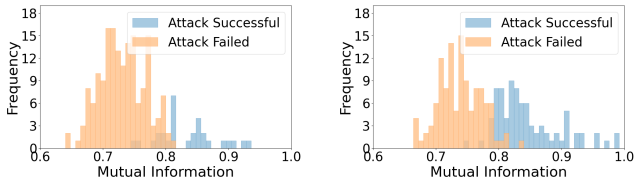
scores, but it does not meet the threshold for a successful attack. The stolen prompt generated by output2prompt fails to capture the broader functionality of Math. Although PLEAK is successful, it shows instability due to protective instructions. In contrast, PRSA exhibits higher stability and has successfully carried out attacks. Interestingly, while inferring the target system prompt using PRSA, we discover an Easter egg hidden within Math: when in calculator mode, entering the number 4 triggers an unfolding ASCII art narrative. This characteristic is not mentioned in Math's official introduction. Notably, OPRO, Sha et al., and output2prompt fail to detect this intriguing characteristic. The stolen prompt we obtained is also confirmed by Math's developers.

We create replicas of the above GPTs using the stolen prompts generated by PRSA (with permission from the developers) to demonstrate the threat of PRSA. These demo versions are anonymously available at https://sites.google.com/view/prsa-prompt-stealing-attack.

8 Why Our Attacks Work

To understand the prompt stealing attacks, we explore why PRSA can infer the target prompt using an input-output pair, particularly focusing on the output. Intuitively, if the output contains a lot of information about the target prompt, a prompt stealing attack is feasible. To quantify this, we introduce the concept of *mutual information*, which quantifies the extent of information shared between the outputs and the target prompts. Theoretical analysis is provided in Appendix H of our extended version [46].

We validate our hypothesis through two settings to analyze the relationship between mutual information and attack performance. In the first, we calculate the mutual information between the outputs and the target prompts. In the second,



(a) PRSA w/o Prompt Attention (b) PRSA

Figure 7: Frequency distribution of mutual information between the output content and the target prompts.

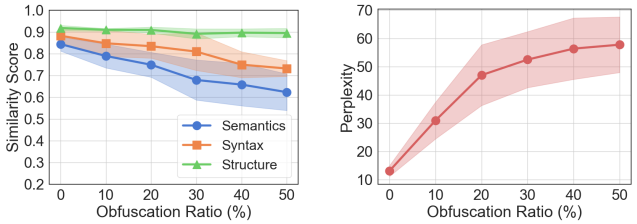
we calculate it between the outputs incorporating PRSA's prompt attention feedback and the target prompts. For experimental details, see Appendix I.1 of our extended version [46]. Figure 7 shows that successful attacks tend to have higher mutual information. Furthermore, PRSA with prompt attention shows a significant increase in mutual information compared to PRSA without it. This increase in mutual information also corresponds to a higher number of successful attacks. A comparison with a heuristic baseline that relies on surface cues (Appendix I.2 of [46]) further supports our findings. These findings suggest that prompt attention indeed helps capture more characteristics of the target prompts, enhancing the mutual information between the outputs and the prompts.

These findings support our hypothesis: the greater the mutual information between the output and the target prompt, the higher the risk of prompt leakage. This insight also points to a potential defensive strategy: by analyzing the mutual information between the output and the target prompt, we can assess the risk of prompt leakage. For defenders, reducing mutual information while maintaining the utility of the target prompt presents an interesting challenge for future research.

9 Discussion

9.1 Possible Defenses

Output Obfuscation. The effectiveness of PRSA primarily relies on the target's output content. To defend, one strategy is to limit adversaries' access to the full output content. We evaluate this approach using GPT-3.5 based prompts, with the same attack setup in Section 6.5. Details of output obfuscation are in Appendix A.4. Figure 8a shows that increasing



(a) Similarity Score (b) Perplexity

Figure 8: The impact of output obfuscation on PRSA.

Table 10: Watermark verification in outputs generated from stolen prompts. If the p-value is ≥ 0.05, the stolen prompt is considered to contain the watermark.

Metric	Category					
	Ads	Email	Ideas	Music	Sports	Travel
P-value	1.31×10^{-2}	1.47×10^{-5}	4.96×10^{-4}	1.28×10^{-2}	1.90×10^{-6}	3.02×10^{-3}

the obfuscation ratio effectively reduces semantic and syntactic similarity, while structural similarity remains largely unchanged. Notably, at a 50% obfuscation rate, both semantic and syntactic similarity decrease by over 20%.

However, output obfuscation also has drawbacks. We measure the perplexity of the obfuscated outputs using GPT-2 to mimic human comprehension. As shown in Figure 8b, a higher obfuscation ratio correlates with increased perplexity. When the ratio exceeds 30%, the perplexity surpasses 50, making the output user-unfriendly. Thus, this strategy requires a balance between defensive effectiveness and usability.

Prompt Watermark. Another defense strategy is watermarking prompts to protect intellectual property through legal deterrence. We follow PromptCARE [47] to embed watermarks in six categories of target prompts by introducing specific triggers. After prompt stealing attacks, we test the output token IDs from the stolen prompts and assess their *p-values*, which measure the statistical significance of the observed differences compared to a null hypothesis. An average p-value at or above 0.05 suggests the presence of a watermark, indicating unauthorized use. To avoid skewed results, we focus on the stolen prompts from successful attacks.

Table 10 shows the p-value of watermark verification in outputs from the stolen prompts, averaged across ten trials. The results consistently show small p-values (< 0.05), indicating insufficient evidence to confirm watermarks in the stolen prompts. We explore the reasons behind this. In PromptCARE case studies, replicated prompts are fine-tuned based on the target prompts, closely resembling the originals. In contrast, our stolen prompts only replicated the target prompt's functionality. Although the target LLM's outputs are consistent, there are slight differences in vocabulary, syntax, and structure, similar to paraphrasing. Studies have shown that watermarking is less effective in verifying paraphrased content [37]. Therefore, watermarking is of limited effectiveness as a defense against prompt stealing attacks.

Practicality of Potential Defenses. While the defenses discussed above have limited practicality, they reveal promising directions that are more feasible. One is to restrict the level of detail in LLM outputs to reduce the risk of exposing target prompt semantics. Another is to optimize target prompts using soft prompts, making their intent less inferable from input-output pairs. These strategies are more practical because they can be integrated into existing prompt services with minimal disruption and allow for the design of optimization steps that preserve prompt usability. We are actively collaborating with prompt developers to explore these directions, and their effectiveness is currently under investigation.

9.2 Justification of Functional Consistency

We evaluate functional consistency using sentence transformer, FastKASSIM, and JS divergence. These metrics capture complementary linguistic features and are widely adopted in prior work [14, 21, 41]. Unlike FastKASSIM and JS divergence, the sentence transformer requires embeddings to compute similarity. To assess its robustness, we replace the underlying embedding model and evaluate the stability of the resulting rankings using Kendall's Tau [23] (see Appendix L of our extended version [46]). Results show that the average Kendall's Tau remains above 0.70 across embedding models, indicating stable performance trends. Beyond linguistic similarity, functional consistency in real-world prompts may involve additional dimensions such as factual accuracy, sentiment, and completeness. To account for these, we conduct human evaluation and LLM-based multidimensional assessment, with these dimensions especially reflected in the LLM-based evaluation metrics. Notably, both evaluations yield trends consistent with those of our functional consistency metric, reinforcing its reliability as a proxy for persuasiveness.

9.3 Limitations

PRSA relies on the mutual information between the target prompt and its outputs. The attack may fail when the output primarily reflects the user input with minimal prompt influence, as in simple question-answering scenarios, or when the output lacks information about the target prompt's intent. Future work could explore incorporating semantic cues from user input more effectively or leveraging publicly available prompt descriptions to enhance intent inference.

10 Conclusion

This paper introduces PRSA, a practical framework designed for prompt stealing attacks against real-world prompt services,

aiming to expose the risks of prompt leakage when the input-output content of prompts is revealed. PRSA infers the intent behind a prompt by analyzing its input-output content and generates a stolen prompt replicating the original's functionality through two key phases: prompt generation and prompt pruning. We validate the actual threat posed by PRSA through extensive evaluations on real-world prompt services. Our objective is to enhance risk awareness among prompt service vendors and actively advocate for implementing protective measures against prompt stealing attacks.

Acknowledgments

We thank our shepherd and the anonymous reviewers for their insightful comments on our work. This work was partly supported by the National Key Research and Development Program of China under No. 2022YFB3102100, NSFC under No. U244120033, U24A20336, 62172243, 62402425, and 62402418, the China Postdoctoral Science Foundation under No. 2024M762829, the Zhejiang Provincial Natural Science Foundation under No. LD24F020002, and the Zhejiang Provincial Priority-Funded Postdoctoral Research Project under No. ZJ2024001.

Ethics Consideration

In developing PRSA, we encountered several ethical challenges. This section summarizes our key considerations and safeguards.

Stakeholder Analysis. We identified three key stakeholder groups: (1) prompt developers, (2) prompt service vendors, and (3) the research community. Prompt developers may face intellectual property risks from functional replication. We mitigated this by avoiding direct reproduction, obtaining authorization for key demonstrations, and omitting sensitive implementation details. Vendors may face reputational or operational risks; we disclosed all findings to them in advance. For the research community, our work highlights both vulnerabilities and defenses in prompt services.

Protection of Participants. Although our institution does not have a formal Institutional Review Board (IRB), the human evaluation was reviewed and approved through an internal ethics review process overseen by our institutional research ethics committee, and conducted with the guidance of a legal expert in intellectual property and research ethics. We strictly adhered to the principles outlined in the Menlo Report. Participants were fully informed of the study's purpose, their role, and how their data would be used, and provided explicit consent prior to participation. To minimize risk, we anonymized all responses, randomized the presentation of outputs, and ensured that no personal or identifying information was collected. These measures reflect our strong commitment to the Menlo Report's principle of "Respect for Persons".

Responsible Disclosure. Following the principle of beneficence, we conducted responsible disclosure prior to public dissemination. We shared our findings with PromptBase, OpenAI, OpenGPT, and the developers of all prompts and LLM applications evaluated in our study. Along with reporting identified risks, we proposed potential mitigations: (1) avoiding full output display in prompt marketplaces, and (2) limiting exposure of prompt details through optimization. As these mitigations require further study, we recommend temporarily removing high-risk prompts. We continue collaborating with the developers to support practical defense deployment.

Intellectual Property Considerations. Our research raised potential concerns related to the replication of commercial prompt functionality. To prevent any infringement of copyright or proprietary content, all prompts presented in the paper were either (1) paraphrasing the functional intent while altering wording and structure, or (2) significantly modified to remove identifiable or proprietary components. We then submitted these revised examples to the corresponding prompt or application developers for review and obtained their explicit authorization to include them in the study and publication. These steps were taken to respect developers' intellectual property rights while preserving the ability to communicate core research findings.

Open Science Policy

In compliance with USENIX Security's open science policy and ethical guidelines, we have released all supporting artifacts, including source code, scripts, and datasets that do not raise ethical concerns. These resources are available at https://doi.org/10.5281/zenodo.15549519. The repository includes detailed usage instructions to support future research.

References

[1] AI Suite: 10 in 1 Prompt Bundle. https://buymeacoffee.com/promptcoder/e/177357.

[2] Awesome ChatGPT Prompts. https://github.com/f/awesome-chatgpt-prompts.

[3] GPT Protections. https://github.com/0xeb/TheBigPromptLibrary/tree/main/Security/GPT-Protections.

[4] Grand View Research. https://www.grandviewresearch.com/industry-analysis/prompt-engineering-market-report.

[5] Sentence Transformers. https://huggingface.co/sentence-transformers.

[6] The Big Prompt Library. https://github.com/0xeb/TheBigPromptLibrary/tree/main/Security/GPT-Protections.

[7] Samuel Ackerman, Orna Raz, Marcel Zalmanovici, and Aviad Zlotnick. Automatically detecting data drift in machine learning classifiers. *arXiv preprint arXiv:2111.05672*, 2021.

[8] Usman Anwar, Abulhair Saparov, Javier Rando, Daniel Paleka, Miles Turpin, Peter Hase, Ekdeep Singh Lubana, Erik Jenner, Stephen Casper, Oliver Sourbut, et al. Foundational challenges in assuring alignment and safety of large language models. *arXiv preprint arXiv:2404.09932*, 2024.

[9] Lukas Berglund, Meg Tong, Max Kaufmann, Mikita Balesni, Asa Cooper Stickland, Tomasz Korbak, and Owain Evans. The reversal curse: Llms trained on" a is b" fail to learn" b is a". *arXiv preprint arXiv:2309.12288*, 2023.

[10] Reihane Boghrati, Joe Hoover, Kate M Johnson, Justin Garten, and Morteza Dehghani. Conversation level syntax similarity metric. *Behavior research methods*, 50:1055–1073, 2018.

[11] Community Builder. Romanempiregpt. https://chat.openai.com/g/g-vWlzptMbb-romanempiregpt.

[12] Eugene Charniak, Curtis Hendrickson, Neil Jacobson, and Mike Perkowitz. Equations for part-of-speech tagging. In *AAAI*, volume 11, pages 784–789. Citeseer, 1993.

[13] Lichang Chen, Jiuhai Chen, Tom Goldstein, Heng Huang, and Tianyi Zhou. Instructzero: Efficient instruction optimization for black-box large language models. *arXiv preprint arXiv:2306.03082*, 2023.

[14] Maximillian Chen, Caitlyn Chen, Xiao Yu, and Zhou Yu. Fastkassim: A fast tree kernel-based syntactic similarity metric. *arXiv preprint arXiv:2203.08299*, 2022.

[15] Kenneth Ward Church. Word2vec. *Natural Language Engineering*, 23(1):155–162, 2017.

[16] Alibaba Cloud. Price Calculator. https://www.alibabacloud.com/en/pricing-calculator.

[17] Jacob Devlin. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[18] Lirong Gao, Ru Peng, Yiming Zhang, and Junbo Zhao. Dory: Deliberative prompt recovery for llm. *arXiv preprint arXiv:2405.20657*, 2024.

[19] Gretel. Measure the utility and quality of gpt-generated text using gretel's new text report. https://gretel.ai/blog/synthetic-text-data-quality-report.

[20] Ting-I Hsieh, Yi-Chen Lo, Hwann-Tzong Chen, and Tyng-Luh Liu. One-shot object detection with co-attention and co-excitation. *Advances in neural information processing systems*, 32, 2019.

[21] Bo Hui, Haolin Yuan, Neil Gong, Philippe Burlina, and Yinzhi Cao. Pleak: Prompt leaking attacks against large language model applications. *arXiv preprint arXiv:2405.06823*, 2024.

[22] Jean Kaddour, Joshua Harris, Maximilian Mozes, Herbie Bradley, Roberta Raileanu, and Robert McHardy. Challenges and applications of large language models. *arXiv preprint arXiv:2307.10169*, 2023.

[23] Maurice G Kendall. A new measure of rank correlation. *Biometrika*, 30(1-2):81–93, 1938.

[24] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213, 2022.

[25] Alina Leidinger, Robert Van Rooij, and Ekaterina Shutova. The language of prompting: What linguistic properties make a prompt successful? *arXiv preprint arXiv:2311.01967*, 2023.

[26] Zi Liang, Haibo Hu, Qingqing Ye, Yaxin Xiao, and Haoyang Li. Why are my prompts leaked? unraveling prompt extraction threats in customized large language models. *arXiv preprint arXiv:2408.02416*, 2024.

[27] 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.

[28] Edward Loper and Steven Bird. Nltk: The natural language toolkit. *arXiv preprint cs/0205028*, 2002.

[29] Ankur Mallick, Kevin Hsieh, Behnaz Arzani, and Gauri Joshi. Matchmaker: Data drift mitigation in machine learning for large-scale systems. *Proceedings of Machine Learning and Systems*, 4:77–94, 2022.

[30] John X Morris, Wenting Zhao, Justin T Chiu, Vitaly Shmatikov, and Alexander M Rush. Language model inversion. *arXiv preprint arXiv:2311.13647*, 2023.

[31] OpenAI. GPTStore. https://chat.openai.com/gpts.

[32] OpenAI. OpenAI Product. https://openai.com/product.

[33] Fábio Perez and Ian Ribeiro. Ignore previous prompt: Attack techniques for language models. *arXiv preprint arXiv:2211.09527*, 2022.

[34] PromptBase. Prompt Marketplace. https://promptbase.com.

[35] Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with" gradient descent" and beam search. *arXiv preprint arXiv:2305.03495*, 2023.

[36] Pulsr.co.uk. Math. https://chat.openai.com/g/g-odWlfAKWM-math.

[37] Vinu Sankar Sadasivan, Aounon Kumar, Sriram Balasubramanian, Wenxiao Wang, and Soheil Feizi. Can ai-generated text be reliably detected? *arXiv preprint arXiv:2303.11156*, 2023.

[38] Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. A systematic survey of prompt engineering in large language models: Techniques and applications. *arXiv preprint arXiv:2402.07927*, 2024.

[39] Zeyang Sha and Yang Zhang. Prompt stealing attacks against large language models. *arXiv preprint arXiv:2402.12959*, 2024.

[40] Xinyue Shen, Zeyuan Chen, Michael Backes, Yun Shen, and Yang Zhang. " do anything now": Characterizing and evaluating in-the-wild jailbreak prompts on large language models. *arXiv preprint arXiv:2308.03825*, 2023.

[41] Victor U Thompson, Christo Panchev, and Michael Oakes. Performance evaluation of similarity measures on similar and dissimilar text retrieval. In *2015 7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K)*, volume 1, pages 577–584. IEEE, 2015.

[42] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.

[43] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. *Advances in neural information processing systems*, 29, 2016.

[44] Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language model with self generated instructions. *arXiv preprint arXiv:2212.10560*, 2022.

[45] Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. *arXiv preprint arXiv:2309.03409*, 2023.

[46] Yong Yang, Changjiang Li, Qingming Li, Oubo Ma, Haoyu Wang, Zonghui Wang, Yandong Gao, Wenzhi Chen, and Shouling Ji. Prsa: Prompt stealing attacks against real-world prompt services (extended version). Technical report, Zenodo, 2025. https://doi.org/10.5281/zenodo.15646160.

[47] Hongwei Yao, Jian Lou, Kui Ren, and Zhan Qin. Promptcare: Prompt copyright protection by watermark injection and verification. *arXiv preprint arXiv:2308.02816*, 2023.

[48] Collin Zhang, John X Morris, and Vitaly Shmatikov. Extracting prompts by inverting llm outputs. *arXiv preprint arXiv:2405.15012*, 2024.

[49] Yiming Zhang, Nicholas Carlini, and Daphne Ippolito. Effective prompt extraction from language models. *arXiv preprint arXiv:2307.06865*, 2024.

[50] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910*, 2022.

Appendix

A Details of Implementation

A.1 Instructions in Prompt Generation Phase

During the prompt generation phase, we use manual instructions to guide the target LLMs in checking for differences in semantics, syntax, and structure between the generated output and the target output. The manual instructions are as follows:

> *Score based on element similarity between generated output and target output, if total score is* 10*.*

where element denotes a specific linguistic expression factor. Based on the similarity score s_{sim} provided by the LLM for each linguistic expression factor d_m, we compute its difference score as $loss_{d_m} = 10 - s_{sim}$, which quantifies the contribution of d_m to the overall output differences. Factors with $loss_{d_m} >$

θ_a are classified as high-impact and are incorporated into the attention a.

We use LLMs as generative models to generate stolen prompts, with the following instructions being executed:

> *Your task is to generate an instruction based on the provided user input and output. The instruction should focus on the specified instruction characteristics.*

The instruction characteristics are obtained as follows:

> *What is the a_i of the output in one sentence?*

where a_i is the element in prompt attention a^*.

For LLM application stores, we also consider using direct queries from the target LLM application to obtain the characteristics of the target system prompt. Drawing on the work of Zhang et al. [48], the instruction is designed as follows:

> *Please describe yourself using as many sentences as possible that best describe you. Start with "1:"*

A.2 Parameter Settings

In the prompt generation phase of the PRSA, when GPT-3.5 and GPT-4 serve as target LLMs, the temperature parameters are set to 0.7 by default. When these models serve as generative models, their temperature parameters are adjusted to 0 to ensure reduced output randomness. The attention threshold, θ_a, is set to 1 (on a scale of 1 to 10) based on empirical validation, as it effectively balances the inclusion of subtle but meaningful discrepancies to achieve better functional consistency. In the prompt pruning phase, we set the semantic similarity threshold, denoted by γ, to 0.4. The beam size, b, is set to 2, the evaluation frequency, e, to 5, and the truncation factor, α, to 0.6 by default.

A.3 Details of OPRO Baseline

To ensure a fair and faithful comparison with PRSA, we apply the official implementation of OPRO [45] with two modifications to align with our prompt stealing setting: (1) We restrict OPRO to a single input-output pair for optimization, due to the limited access available to the adversary; (2) We replace OPRO's accuracy-based scoring with our functional consistency metric, which evaluates semantic, syntactic, and structural similarity between outputs generated by the candidate and target prompts under the same input. All other settings remain unchanged.

A.4 Output Obfuscation Mechanism

We implement an output obfuscation strategy that can be seamlessly integrated into real-world prompt services. Specifically, we randomly mask a fixed proportion of tokens in the output generated by the target prompt and replace them with a generic obfuscation symbol (e.g., *). Given a predefined obfuscation ratio, we randomly select positions of words and phrases from the output and substitute them with the obfuscation symbol. For example, consider the original output: "`Tired of phones that lag or lose juice halfway through the day? This mobile phone keeps up with your hustle, crisp camera, lightning-fast speed, and battery for days...`". The transformed output may appear as: "`Tired of phones that * or * juice halfway through the day? This * * keeps up with your *, * camera, * speed, and * for days...`".

B Similarity Analysis of Prompts p_i in D and Target Prompts p_t

We conduct a similarity analysis on the prompts p_i and their input-output pairs (x_i, y_i) from the collected dataset D, as well as the target prompts p_t and their input-output pairs (x_t, y_t), to evaluate whether D exhibits generality. Following the metrics described in Section 5.4, we assess the semantic, syntactic, and structural similarities between y_i and y_t, along with the prompt similarity between p_i and p_t.

As shown in Figure 9, the results show that while y_i and y_t exhibit moderate structural similarity (average scores of 0.64 for GPT-3.5 based prompts and 0.69 for GPT-4 based prompts), both their semantic similarity (average scores of 0.31 for GPT-3.5 based prompts and 0.33 for GPT-4 based prompts) and syntactic similarity (0.45 for GPT-3.5 based prompts and 0.46 for GPT-4 based prompts) remain low. Moreover, the prompt similarity between p_i and p_t is much limited, with average scores of 0.41 for GPT-3.5 based prompts and 0.44 for GPT-4 based prompts. These findings demonstrate that the data in D is not highly similar to the data in target prompts.

To further examine whether PRSA's performance is strongly dependent on the auxiliary dataset D, we conduct a controlled experiment under two extreme conditions: one where prompts p_i in D are highly similar to the target prompt p_t, and another where they are deliberately dissimilar. We randomly sample 200 purchased prompts from PromptBase as target prompts, including 100 GPT-3.5 based and 100 GPT-4 based prompts. For each target prompt p_t, we use an LLM to synthesize p_i that are either similar or dissimilar. To ensure strict control over similarity, we compute the prompt similarity between each p_i and p_t, and retain only those p_i with similarity scores above 0.8 (High Prompt Similarity) or below

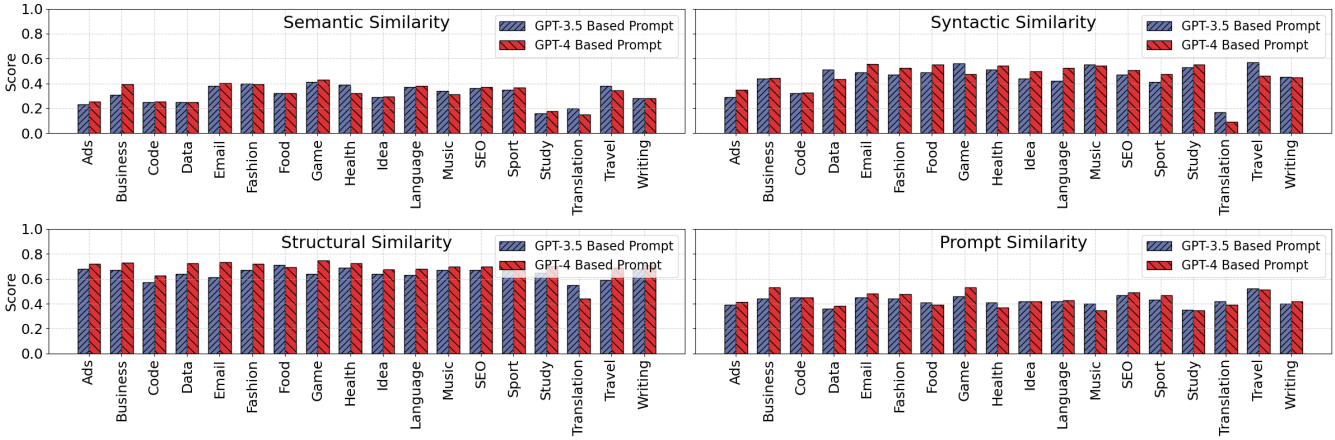
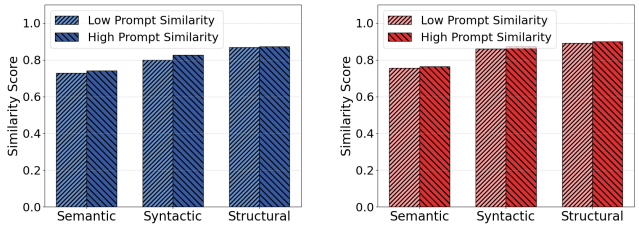


Figure 9: Similarity scores between prompts p_i in D and target prompt p_t. Semantic, syntactic, and structural similarity scores reflect the similarities between outputs y_i and y_t, while prompt similarity scores measure the similarity between p_i and p_t.



(a) GPT-3.5 Based Prompt (b) GPT-4 Based Prompt

Figure 10: Average functional consistency performance of PRSA on prompts p_i in D with high (≥ 0.8) or low (≤ 0.2) prompt similarity to the target prompt p_t.

0.2 (Low Prompt Similarity). For each p_t, we collect 20 p_i for each condition to construct its own high-similarity and low-similarity dataset D.

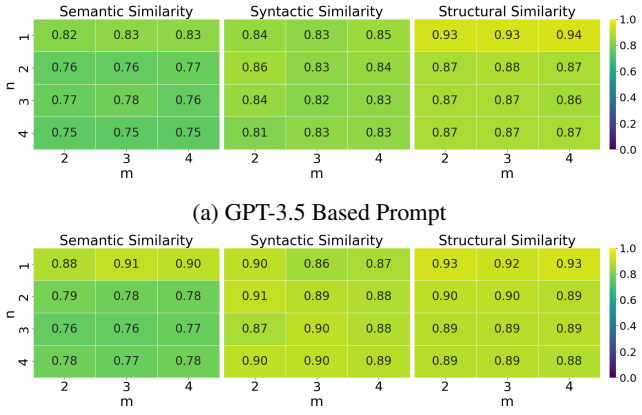
Figure 10 shows PRSA's average functional consistency across D constructed with either high or low prompt similarity. The results demonstrate that PRSA achieves stable performance under both conditions. While scores are slightly higher in the high-similarity setting, the overall trends remain consistent across all three metrics. This robustness can be attributed to PRSA 's design: rather than relying on the specific content of individual prompts in D, PRSA learns generalizable stylistic and linguistic patterns representative of the prompt category. Since prompts within the same category often share common structures and intent, PRSA can infer the target prompt's behavior even when D has low prompt similarity with the target prompt.

C Attack Performance Against GPT-4 Based Prompts in Prompt Marketplaces

We describe the attack performance on GPT-4 based prompts from 18 categories in prompt marketplaces. As illustrated in Table 11, PRSA outperforms the other baseline methods.

D Sample Size Sensitivity Analysis

To assess whether our functional consistency evaluation is sensitive to the choice of test sample size, we conduct a sample size sensitivity analysis by varying the number of user input cases (n) and the number of generation samples per input (m). Specifically, we compare functional consistency results under different (n, m) configurations, ranging from ($n = 1, m = 2$) to ($n = 4, m = 4$).



(a) GPT-3.5 Based Prompt

(b) GPT-4 Based Prompt

Figure 11: Average functional consistency scores under varying numbers of user inputs (n) and generations per input (m) on GPT-3.5 and GPT-4 based prompts.

As shown in Figure 11, configurations with only one input ($n = 1$) tend to produce greater variability across similarity metrics. In contrast, results are more consistent when n and m range from 2 to 4, with minimal fluctuations, indicating stability under our default configuration ($n = 2, m = 3$). This stability may be attributed to the averaging procedure in our

Table 11: Attack performance of PRSA and baseline methods across 18 categories of GPT-4 based prompts in the prompt marketplaces. The best results are shown in **bold**. "–" indicates not applicable.

Metric	Attack Method	Category																	
		Ads	**Business**	**Code**	**Data**	**Email**	**Fashion**	**Food**	**Games**	**Health**	**Ideas**	**Language**	**Music**	**SEO**	**Sports**	**Study**	**Translation**	**Travel**	**Writing**
Semantic Similarity	OPRO	0.66	0.60	0.65	0.75	0.69	0.74	0.69	0.67	0.73	0.68	0.63	0.69	0.71	0.72	0.62	0.57	0.62	0.59
	Sha et al.	0.61	0.61	0.71	0.68	0.59	0.75	0.63	0.65	0.73	0.68	0.57	0.69	0.68	0.71	0.61	0.47	0.64	0.62
	output2prompt	0.64	0.67	0.72	0.55	0.57	0.73	0.60	0.71	0.72	0.74	0.45	0.66	0.68	0.59	0.55	0.68	0.61	0.62
	PLEAK	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	PRSA	**0.75**	**0.72**	**0.80**	**0.82**	**0.80**	**0.83**	**0.79**	**0.75**	**0.77**	**0.83**	**0.76**	**0.86**	**0.79**	**0.78**	**0.72**	**0.80**	**0.78**	**0.78**
	% Gain for PRSA	13.64	7.46	11.11	9.33	15.94	10.67	14.49	5.63	5.48	12.16	20.63	24.64	11.27	8.33	16.13	17.65	21.88	25.81
Syntactic Similarity	OPRO	0.71	0.75	0.77	0.68	0.85	0.66	0.82	0.85	0.85	0.63	0.71	0.84	0.73	0.86	0.90	0.55	0.78	0.81
	Sha et al.	0.45	0.61	0.82	0.83	0.75	0.81	0.81	0.83	0.84	0.54	0.65	0.81	0.54	0.77	0.85	0.38	0.72	0.67
	output2prompt	0.51	0.53	0.81	0.49	0.53	0.59	0.61	0.56	0.83	0.58	0.52	0.59	0.62	0.52	0.57	0.78	0.64	0.68
	PLEAK	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	PRSA	**0.79**	**0.83**	**0.85**	**0.92**	**0.97**	**0.91**	**0.92**	**0.91**	**0.96**	**0.81**	**0.85**	**0.95**	**0.90**	**0.94**	**0.91**	**0.85**	**0.95**	**0.83**
	% Gain for PRSA	11.27	10.67	3.66	10.84	14.12	12.35	12.20	7.06	12.94	28.57	19.72	13.10	23.29	9.30	1.11	8.97	21.79	2.47
Structural Similarity	OPRO	0.79	0.83	0.81	0.63	0.87	0.87	0.85	0.87	0.89	**0.91**	0.79	0.88	0.81	0.89	0.87	0.64	0.85	0.83
	Sha et al.	0.76	0.83	0.84	0.80	0.91	0.89	0.86	0.86	0.88	0.79	0.77	0.87	0.78	0.89	0.88	0.51	0.89	0.84
	output2prompt	0.80	0.83	0.86	0.63	0.78	0.87	0.81	0.83	0.89	0.83	0.73	0.83	0.81	0.78	0.81	0.66	0.82	0.84
	PLEAK	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	PRSA	**0.87**	**0.88**	**0.88**	**0.89**	**0.95**	**0.93**	**0.89**	**0.91**	**0.89**	0.90	**0.88**	**0.92**	**0.89**	**0.93**	**0.89**	**0.83**	**0.92**	**0.89**
	% Gain for PRSA	8.75	6.02	2.33	11.25	4.40	4.49	3.49	4.60	0	-1.10	11.39	4.55	9.88	4.49	1.14	25.76	3.37	5.95

evaluation: each input generates $m = 3$ outputs from both prompts, leading to 9 pairwise comparisons per input. With $n = 2$, this results in 18 total comparisons per method, averaged to report the similarity score. While larger sample sizes may yield slightly tighter estimates, the current configuration appears sufficient to support consistent evaluation trends.

E Cost Analysis of PRSA

To evaluate the economic feasibility of PRSA, we compare the cost of purchasing prompts with the attack cost. If the average attack cost is significantly lower than the average prompt price, the attack becomes economically advantageous.

Cost Definitions. We define the attack cost for the adversary. If the adversary owns the computing resources, this cost is negligible. However, in real-world scenarios, the primary cost comes from invoking LLM APIs. For instance, based on OpenAI's pricing [32], GPT-3.5-turbo charges \$0.50 per $1M$ input tokens and \$1.50 per $1M$ output tokens, while GPT-4-turbo charges \$10.00 per $1M$ input tokens and \$30.00 per $1M$ output tokens. Thus, the average attack cost is defined as follows:

$$C_{avg} = \frac{C_{api}}{N_t \cdot ASR}, \tag{10}$$

where C_{avg} denotes the average attack cost, C_{api} denotes the total cost of LLM API usage, N_t denotes the number of target prompts, and ASR denotes the attack success rate.

If the adversary lacks computing resources, the cost of computation must be taken into account. In this case, the average attack cost is defined as follows:

Table 12: Comparison of average prompt price and the average attack costs of PRSA. Average Attack Cost_1: using own computing resources; Average Attack Cost_2: using cloud computing resources.

Target Prompt	Average Prompt Price (\$)	Average Attack Cost_1 (\$)	Average Attack Cost_2 (\$)
GPT-3.5 Based Prompt	3.77	0.05	0.08
GPT-4 Based Prompt	4.15	0.48	0.51

$$C_{avg} = \frac{C_{api} + C_{cal}}{N_t \cdot ASR}, \tag{11}$$

where C_{cal} denotes the extra computing resource cost. In general, we assume that the adversary conducts the attack by renting cloud computing resources. Based on pricing calculators from well-known cloud service providers [16], the cost of renting a V100 GPU-accelerated server is approximately \$1.66 per hour. Considering the run-time of PRSA and potential retries, an 8 hour rental period is sufficient.

Results. The results in Table 12 show that PRSA is highly cost-effective across both GPT-3.5 based and GPT-4 based prompts. For GPT-3.5 based prompts, the average attack cost is only 1.3%–2.1% of the average prompt price (\$3.77), while for GPT-4 based prompts, it is 11.6%–12.3% of the average prompt price (\$4.15). Even when using cloud computing resources, the attack remains profitable for the adversary.