ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

Prompt Obfuscation for Large Language Models

David Pape[1], Sina Mavali[1], Thorsten Eisenhofer[2,3], Lea Schönherr[1]

[1]CISPA Helmholtz Center for Information Security
[2]Berlin Institute for the Foundations of Learning and Data (BIFOLD)
[3]Technische Universität Berlin

Abstract

System prompts that include detailed instructions to describe the task performed by the underlying LLM can easily transform foundation models into tools and services with minimal overhead. They are often considered intellectual property, similar to the code of a software product, because of their crucial impact on the utility. However, extracting system prompts is easily possible. As of today, there is no effective countermeasure to prevent the stealing of system prompts, and all safeguarding efforts could be evaded.

In this work, we propose an alternative to conventional system prompts. We introduce prompt obfuscation to prevent the extraction of the system prompt with little overhead. The core idea is to find a representation of the original system prompt that leads to the same functionality, while the obfuscated system prompt does not contain any information that allows conclusions to be drawn about the original system prompt. We evaluate our approach by comparing our obfuscated prompt output with the output of the original prompt, using eight distinct metrics to measure the lexical, character-level, and semantic similarity. We show that the obfuscated version is constantly on par with the original one. We further perform three different deobfuscation attacks with varying attacker knowledge—covering both black-box and white-box conditions—and show that in realistic attack scenarios an attacker is unable to extract meaningful information. Overall, we demonstrate that prompt obfuscation is an effective mechanism to safeguard the intellectual property of a system prompt while maintaining the same utility as the original prompt.

1 Introduction

Tailoring general-purpose foundation models for specific tasks can be achieved through fine-tuning and prompting. During this process, the model is trained or prompted to learn how to respond to a specific request. For example, a chat model such as the Llama models [3] can be fine-tuned to coding tasks [39] but also to natural language tasks such as sentiment analysis or question answering [42]. Although fine-tuning with LoRA [21] or QLoRA [13] makes the process more efficient and is, in principle, also possible on consumer hardware, it still requires carefully curated training data and resources to update a model. Furthermore, it has been shown before that fine-tuning a model can unintentionally alter its behavior, such as breaking the alignment [38] and encouraging hallucinations [18]. In addition, most commercial LLMs—such as the newest models from OpenAI or Claude—are not freely accessible and therefore cannot be easily fine-tuned. A popular alternative is to prompt foundation models directly with a detailed explanation of the task, via a so-called *system prompt*. Although fine-tuning reprograms a model, prompting offers greater flexibility at lower cost and does not require additional training data. In addition, OpenAI's custom GPTs [1] can be configured solely through system prompts, allowing even inexperienced users to create distinct model behaviors without fine-tuning. These customized models can be shared on the GPT store as black-box solutions, making specialized functionality broadly accessible.

The flexibility of prompting and the ease of use come with a price: The content of the system prompt can easily be leaked, even word for word, with carefully crafted user input, also known as prompt injection attacks [16, 30–32]. This has happened for thousands of commercial tools, where the system prompt was leaked and published. Among others, Microsoft Bing, Copilot, Notion's integrated AI, and several of OpenAI's models [2]. A well-designed and engineered system prompt significantly influences a model's output and is often kept confidential. Due to their crucial impact on functionality, system prompts are often considered intellectual property (IP), similar to the code of software [35]. Therefore, the protection of system prompts is of high interest for providers of LLM services. However, currently, there are no successful strategies to prevent the stealing of system prompts, and all safeguarding efforts could be evaded with carefully crafted prompt injections that bypass all protection mechanisms [11, 16].

In this paper, we propose an alternative approach by leveraging **prompt obfuscation**. This technique aims to create a

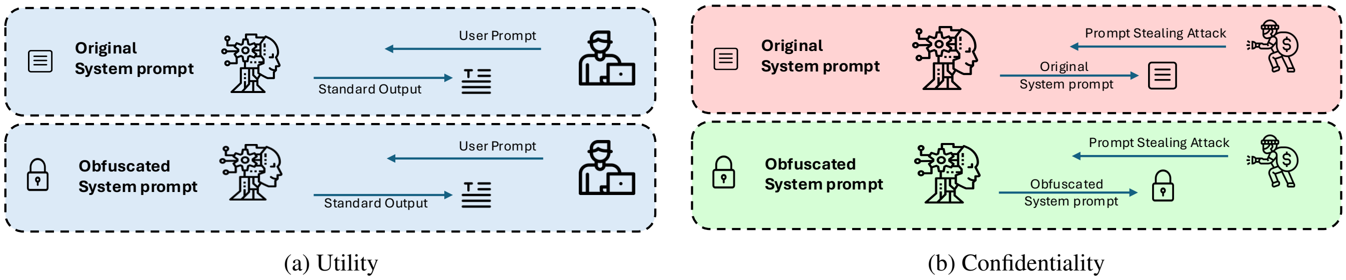

Figure 1: Comparison of the original versus obfuscated system prompt in both normal use (Figure 1a) and attack cases (Figure 1b). The obfuscated prompt preserves functionality for valid user requests, yet remains unusable when stolen—effectively safeguarding the confidentiality of the system prompt.

surrogate system prompt that achieves two primary goals: first, it preserves the functionality of the original system prompt, thereby maintaining task performance and utility; second, it ensures no information leakage, meaning that even if the obfuscated prompt is extracted, an attacker can*not* draw conclusions about the underlying original prompt. Our exploration focuses on whether such an obfuscated prompt can effectively conceal the instructions of a customized LLM. With only a one-time additional expense and no additional training data, we are able to build systems with functionality comparable to that of conventionally prompted models.

For our prompt obfuscation, we find collisions in the continuous embedding space—soft prompt—that do not correspond to a textual—hard prompt—representation and, therefore, prevent an attacker from extracting any meaningful text. To achieve this, we optimize a new soft prompt so that it produces identical outputs for predefined samples while remaining distinct from the original in its continuous representation.

As shown in Figure 1, our obfuscated prompt retains the same functionality as the original while concealing the true instructions. This prevents an attacker from recovering meaningful text, even if the prompt is compromised.

We assess the obfuscated prompt using eight utility metrics, divided into four lexical, two character-level, and two semantic measures. In doing so, we capture a broad spectrum of linguistic similarity, from surface-level overlaps to contextual relationships. We show that we can maintain the same utility as the original prompt. We also test our method against three different deobfuscation methods, each assuming a varying level of adversarial knowledge. The first is a black-box setting with only query-based (API) access, whereas the two white-box settings allow full visibility of model parameters and direct access to the obfuscated prompt representation.

While the black-box and one of the white-box attacks are generally not successful in extracting the original prompt, our results indicate that in the other white-box attack, we can extract individual words but no semantically meaningful text. However, this attack scenario is extremely hard to replicate as it requires white-box knowledge about the model and the details about the obfuscated prompt in the embedding space, and is mostly impractical.

We also conducted a case study involving an actual leaked prompt from a custom GPT, demonstrating that our approach is practical and effective even in real-world scenarios with complex system prompts.

In summary, we make the following three key contributions:

- **Prompt obfuscation.** We propose an approach for prompt obfuscation. We evaluate two versions, one in the token (hard prompt) and one in the embedding space (soft prompt). We show that with only little overhead, we are able to construct a strong prompt obfuscation in the embedding space.

- **Utility Evaluation.** We show that we can maintain similar utility as for the original prompt without significant overhead. We also conduct an experiment with an actual leaked real-world system prompt to show the practicality of our obfuscation.

- **Prompt deobfuscation.** We explore three attacks on our obfuscation method—covering both black-box and white-box scenarios—and evaluate the ability of an adversary to recover the system prompt from its obfuscated representation under various threat models. Our results indicate that in realistic scenarios, the adversary cannot extract meaningful information from the obfuscated prompt.

2 Background

Our system prompt obfuscation method builds on the concept of prompt optimization. This optimization can be done both in the token space (for hard prompts) and the embedding space (for soft prompts) [23,43]. For this, we provide background on LLMs, followed by a discussion of key prompting concepts

2.1 Large Language Models

We define a LLM as a function M_θ with parameters θ operating on t tokens $\mathbf{x} = \{x_1, \ldots, x_t\}$ with $x_i \in \mathcal{V}$ for $i = 1, \ldots, t$ and $\mathcal{V}$ being the LLMs vocabulary. These tokens refer to the fundamental units of text processed by the model and represent linguistic elements such as words, parts of words, or punctuation symbols.

In its operation, the LLM computes the likelihood of possible next tokens x_{t+1}, yielding a probability distribution

$$p_\theta(x_{t+1} \mid x_1, \ldots, x_t). \tag{1}$$

By iteratively predicting subsequent tokens, the LLM constructs coherent and contextually relevant text sequences based on the input (i. e., *prompt*).

An LLM accepts a user input (such as a question) and produces a textual answer, whose correctness largely depends on the model's training. Modern LLMs are trained on massive datasets and utilize billions of parameters, which enables them to accurately handle complex tasks, including generating code and translating languages. The model is only limited by its context length, the maximum number of tokens a model can process. In addition, factors like temperature and sampling technique can impact the creativity and consistency of the output.

2.2 Prompting

The usual interface for interacting with language models is natural language, using so-called prompts. These denote structured textual inputs and function as input sequences that guide the model generation process. A prompt serves both as an instruction and a contextual anchor, enabling users to direct the model's output by framing the task or providing situational examples.

The efficacy of LLMs in generating meaningful and contextually appropriate responses can be significantly improved by decomposing the prompts into different components, such as previous user input or additional context. Among these, the *system prompt* is a crucial textual directive that dictates how the model should interpret and respond to subsequent user inputs. It sets the interaction tone, outlines the expected tasks or roles, and defines interaction objectives to ensure that responses align with the desired outcomes of the system.

To formalize this, we denote a full prompt as $\mathbf{x} = (\mathbf{s}, \mathbf{u})$, where $\mathbf{s}$ is the system prompt and $\mathbf{u}$ is the user's input, which supplies the specific query or content. Thus, generating an answer corresponds to computing

$$M_\theta(\mathbf{x}) = M_\theta(\mathbf{s}, \mathbf{u}). \tag{2}$$

Figure 2 demonstrates how the Llama 3.1 model family [3] structures its prompts using tokens like $<|start\_header\_id|>$ and $<|end\_header\_id|>$ to define the prompt's components. Additionally, predefined terms like

```
<|start_header_id|>system<|end_header_id|>
Cutting Knowledge Date:
Today Date:
Talk like a pirate!

<|eot_id|><|start_header_id|>user
<|end_header_id|>
What does Darth Vader say to Luke in "The
Empire Strikes Back"?

<|eot_id|><|start_header_id|>assistant
<|end_header_id|>
Arrrr, Darth Vader be sayin' somethin' like
this to Luke Skywalker in "The Empire
Strikes Back": "Yer a long way from home,
Luke. A long way. And yer no match for the
dark side."
```

Figure 2: The official Llama 3.1 prompt template. ■ System prompt; ■ User input; ■ Model response

"system", "user", and "assistant" define the roles of the system prompt, user input, and model response, respectively. In this example, the system prompt changes the response style of the language model.

In practical applications, typically only the model's response is displayed to the end-users, while the system prompt remains concealed. This approach is adopted to ensure a consistent and predictable interaction flow, strengthen system stability, and protect sensitive or proprietary information embedded within system prompts, such as private data or IP.

2.3 Hard and Soft Prompts

In this work, we distinguish between *hard prompts* and *soft prompts*. Hard prompts are the token representation of a text, while soft prompts are the respective representation in the embedding space of a model [27].

A soft prompt $\hat{\mathbf{x}}$ is the embedding of $\mathbf{x}$, which is derived by passing these hard prompts through the model's token embedding layer. Specifically,

$$\hat{\mathbf{x}} = \phi(\mathbf{x}) \in \mathbb{R}^{t \times d}, \tag{3}$$

where d is the dimension of the embedding space, and the function ϕ is the model's token-embedding lookup table. This transformation maps the discrete token sequence $\mathbf{x}$ into a continuous, fixed-size vector space.

Reverse mapping. The reverse mapping from this continuous vector space to the discrete token space is inherently limited and is not straightforward. Vectors not explicitly present in the lookup table cannot be accurately converted to specific tokens. In this work, we leverage this property in one version of our prompt obfuscation (soft prompt obfuscation) in order to maintain the functionality of the systems prompt, while making the textual interpretation nearly impossible. Using this feature, we can use soft prompts to maintain the functional capability of prompts while achieving significant obfuscation of the textual content.

3 Prompt Obfuscation

Thus far, we have discussed the importance of system prompts for instructing LLMs in practical applications. However, prior research has demonstrated that these prompts are susceptible to leakage during interactions with the model [45]. Such *prompt extraction* attacks are a significant security concern. Companies invest considerable resources in high-quality system prompts; if stolen, these prompts can expose proprietary strategies, internal procedures, or safety mechanism [33].

Despite active research, defending against prompt extraction remains a challenging problem [40], with current defenses often locked in a reactive and ongoing cat-and-mouse game against evolving attack strategies. Rather than attempting to prevent leakage, we aim to ensure that *even if a prompt is leaked*, its practical value for an adversary seeking to understand or repurpose it is limited. To achieve this, we focus on obfuscating system prompts such that, upon leakage, they cannot be meaningfully interpreted or repurposed.

3.1 Threat model

For our model, we consider an adversary targeting an LLM initialized with a confidential system prompt, aiming to extract this prompt through model interactions. The adversary's objective could be either to reveal internal model details—such as timestamps, company context, or specific API instructions—or simply to replicate the model's behavior or parts of it. We assume a practical black-box scenario where the adversary is limited to query-only access to the model.

By obfuscating the system prompt, our goal is to reduce the utility of the stolen prompt to the adversary. Even if successfully stolen, its usefulness is significantly diminished because it cannot be easily interpreted or modified without compromising its intended functionality.

3.2 Obfuscation Methodology

The key idea of the prompt obfuscation is to find a *collision* in the prompt space. Specifically, given an original system prompt $\mathbf{s}$, we aim to construct an obfuscated version $\mathbf{s}_{\text{obf}}$ that (1) retains the original prompt's functionality, while (2) being unintelligible to an adversary.

This leads to two primary challenges. First, directly measuring functionality is generally infeasible. To address this, we define a set of representative user prompts $\mathcal{U}$ and compare the outputs produced by the model conditioned on the original versus obfuscated prompts. Second, we must ensure the obfuscated prompt does not inadvertently reveal meaningful information. To this end, we randomly initialize the obfuscated system prompt and iteratively optimize it to replicate the intended functionality. This leads us to the following optimization problem:

$$\arg\min_{\mathbf{s}_{\text{obf}}} \sum_{\mathbf{u}\in\mathcal{U}} \ell\big(\mathbf{s}, \mathbf{s}_{\text{obf}}, \mathbf{u}\big),$$

where ℓ measures divergence in model outputs between original and obfuscated prompts. While this approach does not guarantee that the resulting prompt is devoid of meaningful information, our empirical evaluation suggests that obfuscated prompts occupy distinct regions in prompt space, disconnected from the original prompt. This optimization problem can be implemented at different stages within the LLM. In this work, we design obfuscation methods for two stages—hard prompts and soft prompts—as illustrated in Figure 3 and detailed next.

3.3 Hard Prompt Obfuscation

The first one operates in the token space (cf., Figure 3). Once formulated in a white-box environment, obfuscated hard prompts can be used in a black-box setting without requiring access to an LLM's embedding layer. They are human-readable, allowing easy verification to ensure no sensitive information is included.

In hard prompt obfuscation, we begin with a randomly initialized token sequence $\mathbf{s}_{\text{obf}}$ and optimize it so that, for the set of user inputs $\mathbf{u} \in \mathcal{U}$, the model's outputs closely match those produced by the original prompt. In particular, we define ℓ as follows

$$\begin{aligned}\ell\big(\mathbf{s}, \mathbf{s}_{\text{obf}}, \mathbf{u}\big) = {} & \mathcal{L}_{CE}\big(M_\theta(\mathbf{s}, \mathbf{u}), M_\theta(\mathbf{s}_{\text{obf}}, \mathbf{u})\big) \\ & + \mathcal{L}_{KL}\big(M_\theta(\mathbf{s}, \mathbf{u}), M_\theta(\mathbf{s}_{\text{obf}}, \mathbf{u})\big).\end{aligned} \tag{4}$$

This captures how well the obfuscated prompt reproduces the original responses. The cross-entropy (CE) ensures correct output tokens, while the Kullback-Leibler (KL) divergence encourages similar output probability distributions, acting as a regularizer.

Our hard prompt obfuscation operates in the token space, which makes direct gradient optimization difficult due to its discrete nature. To address this issue, we utilize the Greedy Coordinate Gradient (GCG) [47] algorithm, originally developed for creating adversarial attacks on aligned language models. GCG iteratively identifies and substitutes promising tokens by leveraging local gradients. Algorithm 1 demonstrates the process for optimizing an obfuscated system prompt so that its output closely matches that of the original prompt. Beginning with a random token sequence $\mathbf{s}_{\text{obf}}$, the algorithm iteratively refines it over a predetermined number of iterations K.

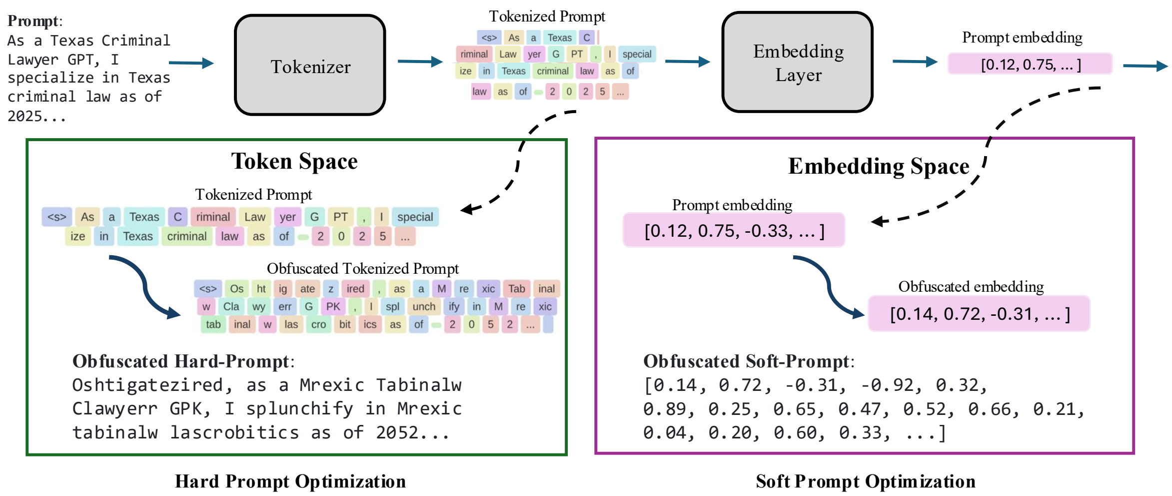

Figure 3: **Overview of Prompt Obfuscation Method.** In hard prompt obfuscation, the tokenized text is directly modified. Conversely, soft prompt obfuscation involves updating the soft prompt within the continuous embedding space, providing more flexibility.

To enhance the obfuscation of the system prompt using contextual information, we employ a window approach. This method optimizes not only individual tokens but also leverages the autoregressive properties of LLMs by considering a window of consecutive tokens instead of single tokens. We hypothesize that a larger window size, by incorporating more contextual information, could potentially lead to a more effective obfuscation that better preserves the functionality of the original prompt. At each iteration, we measure how much the responses differ over all W tokens in the window.

In particular, line 8 calculates the difference between the obfuscated and original prompt, comparing them token by token for each user input $\mathbf{u}$. We here present the loss of the w-th output token as $\ell(\mathbf{s}, \mathbf{s}_{\text{obf}}, \mathbf{u})_w$. Once the total loss over these W tokens has been aggregated, GCG updates the obfuscated prompt by generating candidate token substitutions, computing the loss again for each candidate, and selecting the candidate that yields the smallest loss.

Algorithm 1 Hard-Prompt Obfuscation

Require: $\mathbf{s}$ (original system prompt),
$\mathbf{s}_{\text{obf}}$ (randomly initialized obfuscated prompt),
$\mathcal{U}$ (set of user queries),
K (obfuscation iterations),
N (number of output tokens),
W (window size)

```
1:  M ← ⌈N/W⌉                                   // Number of windows
2:  n ← 0                                       // Offset for windowing
3:  for 1 to M do                               // Loop over output token windows
4:      for 1 to K do                           // Obfuscation loop
5:          L ← 0
6:          for w ← 1+n to W+n do               // Loop over window
7:              // Compare w-th output token
8:              l ← ∑_{u∈U} ℓ(s, s_obf, u)_w
9:              L ← L + l
10:         end for
11:         s_obf ← GCG(s_obf, L)               // Update via GCG
12:     end for
13:     n ← n + W
14: end for
```

3.4 Soft Prompt Obfuscation

Hard prompt obfuscation is constrained by the finite token space and the inherent meanings associated with each token, which limit the ability to obfuscate prompts without altering their intended functionality. To address these limitations, we develop an approach that operates directly in the embedding space, allowing for more fine-grained obfuscation (cf., Figure 3). Furthermore, soft prompt obfuscation benefits from the complex reverse mapping between the continuous embedding space and the discrete token space, since multiple distinct soft prompts in the embedding space can map to a single hard prompt.

This approach enables more fine-grained obfuscation by leveraging the continuous embedding space. The primary implementation change involves accessing the model's embedding layer—an adjustment that is relatively minor and typically feasible with open-source models. We anticipate that these obfuscated prompts will outperform hard prompts in both utility and confidentiality. This improvement is due to the more representative nature of the continuous space and the ease of optimization within it, as opposed to discrete spaces.

To implement this approach, we adopt a slightly modified optimization objective. Specifically, we optimize an embedding vector $\hat{\mathbf{s}}_{\text{obf}} = \phi(\mathbf{s}_{\text{obf}}) \in \mathbb{R}^{t \times d}$ rather than a token sequence. Therefore we adapt the loss function ℓ described as

$$\begin{aligned}\ell(\hat{\mathbf{s}}, \hat{\mathbf{s}}_{\text{obf}}, \mathbf{u}) = \mathcal{L}_{CE}\big(M_{\theta\setminus\text{Emb}}(\hat{\mathbf{s}}, \mathbf{u}), M_{\theta\setminus\text{Emb}}(\hat{\mathbf{s}}_{\text{obf}}, \mathbf{u})\big) \\ + \mathcal{L}_{KL}\big(M_{\theta\setminus\text{Emb}}(\hat{\mathbf{s}}, \mathbf{u}), M_{\theta\setminus\text{Emb}}(\hat{\mathbf{s}}_{\text{obf}}, \mathbf{u})\big),\end{aligned} \tag{5}$$

where $M_{\theta\setminus\text{Emb}}$ describes the LLM that excludes the embedding layer and $\hat{\mathbf{s}}$ being the original soft prompt.

Using the continuous nature of the embedding space, we can directly optimize the soft prompt using gradient-based algorithms. For this purpose, we adapt Algorithm 1 by replacing $\mathbf{s}_{\text{obf}}$ with $\hat{\mathbf{s}}_{\text{obf}}$ and the loss function with the loss function defined in Equation (5). Additionally, rather than employing GCG for updating the obfuscated prompt in line 11, we directly apply gradient descent. This simplification improves the efficiency of the optimization process within the soft prompt space.

4 Experimental framework

In this section, we outline and justify the overall structure of our experiments. First, we provide a general description of the dataset used to generate outputs for the obfuscation process. Next, we detail how we define and design the system prompts. Finally, we explain how we measure the similarity between outputs and prompts using the selected similarity metric.

4.1 Models and Dataset

To evaluate whether our obfuscation approach maintains the same output behavior under varied user inputs, we rely on datasets drawn from two distinct tasks:

- **Question Answering (QA)**: TruthfulQA, TriviaQA [22]
- **Summarization**: CNN_dailymail [34], samsum [19]

TruthfulQA has 817 questions designed to test the LLM's ability to provide truthful information. TriviaQA contains around 650,000 question-answer pairs collected from trivia websites. CNN/DailyMail comprises roughly 312,000 news articles (split between CNN and DailyMail) along with summaries. Meanwhile, Samsum consists of approximately 16,000 annotated dialogue summaries from messenger-like conversations.

We select two datasets for each task to ensure diversity in the types of queries and text samples used to generate the model's outputs. Since our obfuscator relies on obtaining model responses from both the baseline (unobfuscated) and obfuscated versions of the system prompt, the data we use does not need to be labeled with "ground-truth" answers or summaries. Instead, we query the model directly with each dataset sample using the baseline system prompt. The retrieved outputs (e.g., a model-generated answer or summary) then serve as the target behavior during the obfuscation process.

4.2 System Prompt Design

System prompts can contain many instructions resulting in distinct output behavior, making it difficult to formalize textual directives. To address this, we conceptualize the model's system prompt by categorizing it into two distinct components:

- ***Task***: The specific function or activity that the system is requested to perform. It defines the substantive objective that the model aims to fulfill (e.g., "summarize the text").
- ***Style***: This aspect characterizes the manner or mode in which the model output is expressed. It describes a distinct flavor, character, or format of the model's output without affecting its functionality (e.g., "talk like a pirate!").

By categorizing system prompts into *Task* and *Style* components, we create a formalized framework for output analysis. This categorization enables us to understand how the design of the system prompts influences the generated output, thereby allowing us to evaluate the effectiveness of obfuscation across various types of system prompts.

To formalize system prompt elements, let $T \in \mathcal{T}$ represent the task instructions (e.g., "summarize the text") and $S \in \mathcal{S}$ represent the style instructions (e.g., "talk like a pirate!"). A system prompt $\mathbf{s}$ can be expressed as the concatenation $\mathbf{s} = (T, S)$. However, in practice, T and S might be interspersed or placed in different locations within the overall prompt.

We define three scenarios to explore how T and S are positioned in the prompt:

1. **Full scenario:** Both the task T and style S are included together in the system prompt.
2. **Style scenario:** The style S remains in the system prompt, while the task T is described in the user query.
3. **Task scenario:** Only the task T is included in the system prompt, while the style S is omitted.

We provide examples of all these scenarios and a list of styles and tasks in Appendix A. We designed these three scenarios to systematically evaluate the impact of different prompt configurations on model output. By isolating the Task and Style components, we can assess the robustness of our obfuscation method under varying conditions. This approach mirrors realistic usage patterns, where the integration of task and style instructions can differ significantly.

4.3 Similarity Metrics

Assessing the quality of obfuscated prompts is not straightforward. The outputs of similar inputs can be semantically identical but very different on a character level. Therefore, to accurately measure the fidelity of text generated by our model using obfuscated versus standard prompts, we employ a variety of text similarity metrics. Each metric captures different aspects of generated and reference output similarity:

- **Lexical similarity**: These metrics focus on how much two texts overlap in terms of words, phrases, or n-grams. They capture aspects such as direct lexical matches, word order, and synonym/lemma matches. We use four different metrics: BLEU, ROUGE-L, NIST_MT, METEOR
- **Character-Level similarity**: These metrics gauge similarity at a finer granularity by focusing on how characters align and can capture subtle morphological differences. We use two different metrics: characTER, ChrF
- **Semantic similarity**: These metrics encode the text into high-dimensional vectors, capturing contextual and semantic relationships. We use two different metrics: BERTScore, Embedding-level cosine similarity

We give detailed explanations of each metric in Appendix F.1.

Prompt Similarity Metrics. To assess the similarity between the obfuscated prompt $\mathbf{s}_{\text{obf}}$ and the original system prompt $\mathbf{s}$, we utilize a range of metrics analogous to those used for output similarity. Given that our system prompts are relatively concise—typically consisting of approximately ten tokens—the corpus-level metrics previously introduced may lack the necessary sensitivity. Therefore, we use the following more refined metrics to measure the similarity alongside manual evaluations of the obfuscated system prompts:

- **Character-Level similarity**: Levenshtein distance, Longest Common Subsequence (LCS)
- **Lexical similarity**: Jaccard Index
- **Semantic similarity**: Embedding-level cosine similarity

These metrics collectively help us determine if obfuscated prompts retain any discernible information from their conventional counterparts, encompassing identical tokens, substrings, or semantic meanings. More details about each metric can be found in Appendix F.2.

5 Evaluation

In this section, we evaluate how effective our obfuscation technique is by comparing the performance of obfuscated system prompts with conventional system prompts. We assess two main aspects: first, how well the obfuscation conceals the system prompt, and second, how the model performs when using these obfuscated system prompts in both hard and soft cases.

5.1 Hard Prompt Obfuscation

To evaluate the effectiveness of obfuscated system prompts, we generate outputs using conventional and obfuscated prompts. These outputs are divided into training and testing sets. During optimization, the obfuscated prompt is refined using training data, with performance verified on the test subset after each iteration using the established similarity metrics.

Dataset. We focus on a question-answering (QA) task using the TruthfulQA dataset [29] as a preliminary experiment. TruthfulQA is designed to test LLM's ability to provide truthful information. From its 817 available samples, we selected 800 to ensure a consistent evaluation set for our experiments. These were then divided into 640 samples for training and 160 for testing, following an 80:20 train-test split.

LLM. We consider the 8 billion parameter version of the instruction-tuned Llama 3.1 model for our evaluation [3]. The model is loaded in its quantized 4-bit version to reduce memory requirements. Nucleus sampling is used for generating responses, set at a top_p value of 0.95 and a temperature of 0.7 to balance response diversity and stability. Each output sample has 125 tokens.

Setup. The optimization parameter K is set with a maximum of 10 iterations and the token count N is set to 15, as detailed in Algorithm 1, balancing performance and quality of the result. Since increasing the window size raises GPU memory requirements, we selected the largest window size feasible $W = 5$ in all our experiments. The obfuscated system prompt $\mathbf{s}_{\text{obf}}$ is initialized with a random sequence of 10 tokens for each experiment. For evaluating the model's performance, we generate five outputs per test instance using both conventional and obfuscated prompts. We then perform pairwise comparisons between each conventional output and every obfuscated output for each test sample. We use the *all-mpnet-base-v2* model to generate embeddings for natural language evaluation.

Results. Table 1 presents our evaluation results for the TruthfulQA dataset, covering the three scenarios discussed in Section 4.2. Outputs generated with a *blank* system prompt serve as a baseline, helping to determine the optimized prompt's ability to emulate the desired response. In this case, all obfuscated parts are left blank, with the specific sections varying by scenario. To have a more complete baseline, we measure the output similarity where we rely solely on the *original* system prompt. In this case, we generate outputs by using a different seed and compare it to another set of outputs generated from the same system prompt but with a previously used seed. For both scenarios, we sample five outputs, matching the number tested for the obfuscated version, and compute our scores using the same procedure. Ideally, the obfuscated versions maintain performance on par with the original prompt, while the blank version—having only the remaining portions of the prompt—tends to yield inferior results.

Metrics	TruthfulQA								
	Full			Style			Task		
	blank	obf	original	blank	obf	original	blank	obf	original
BLEU (↑)	18.07	**33.00**	37.06	17.51	**35.48**	35.95	41.94	**48.03**	48.92
ROUGE-L (↑)	0.28	**0.37**	0.41	0.26	**0.38**	0.39	0.45	**0.48**	0.51
METEOR (↑)	0.32	**0.44**	0.46	0.27	**0.44**	0.43	0.53	**0.55**	0.57
NIST_MT (↑)	2.10	**3.22**	3.14	1.65	**3.29**	3.18	3.47	**3.70**	3.77
CharacTER (↓)	0.78	**0.69**	0.67	0.83	**0.69**	0.70	0.63	**0.60**	0.58
ChrF (↑)	40.01	**47.82**	49.03	35.96	**47.92**	47.84	53.65	**55.95**	56.56
BERTScore (↑)	0.86	**0.90**	0.90	0.86	**0.90**	0.90	0.91	**0.92**	0.93
Cosine (↑)	0.73	**0.80**	0.80	0.69	**0.79**	0.80	0.85	**0.86**	0.86

Table 1: **Hard Prompt Obfuscation.** This table compares the output similarity between obfuscated prompts and blank prompts, measured against conventional system prompts on the TruthfulQA dataset. We provide a baseline using outputs generated with a *blank* prompt and a reference using the *original* prompt. Results from outputs calculated with obfuscated values are highlighted in bold.

In each experiment, we collect the best values for each metric throughout the full optimization process, as a system owner would select the best-performing obfuscated system prompt. For the *Full* and *Style* scenarios, which include multiple style descriptions, we average the results across all system prompts to enhance the generalizability of our evaluation.

It can be seen, that our approach consistently achieves scores that are comparable to those of the *original* prompt, indicating that our obfuscated prompts perform as effectively as the original ones. Across all three scenarios, there is an improvement in all eight metric values compared to the baseline. In the *Task* scenario, where no style description is provided, the improvement is modest. This is because the model is able to infer the QA task from the dataset samples without requiring a system prompt, as anticipated.

In addition to examining the functionality of an obfuscated system prompt, it's important to assess their confidentiality to determine whether they reveal any information about the original prompt. We report the similarity between system prompts in Table 2. As a baseline, we compare the conventional system prompt to a random token sequence, expecting the obfuscated prompt to show a comparable or higher degree of similarity. It can be seen that almost in all the scenarios, the obfuscated system prompts are more similar to the conventional one for all four metrics, indicating some information leakage during the obfuscation. Additionally, we manually reviewed and compared the obfuscated system prompts with their conventional versions, which confirmed our findings. Some examples of the obfuscated system prompts are shown in Table 3.

Although obfuscated versions significantly differ from the original system prompt, related words can still be identified. We argue that attackers could reconstruct the non-obfuscated system prompt from the obfuscated version when it is leaked, which directly contradicts our primary objective of safeguarding the system prompt through obfuscation. To address this issue, we evaluate the trade-off between confidentiality and functionality preservation in our hard prompt obfuscation

Metrics	TruthfulQA					
	Full		Style		Task	
	rand	obf	rand	obf	rand	obf
Levenshtein (↑)	0.11	0.19	0.13	0.20	0.13	0.20
LCS (↑)	0.12	0.07	0.18	0.30	0.15	0.25
Jaccard Index (↑)	0.00	0.22	0.00	0.10	0.00	0.07
Cosine (↑)	0.28	0.41	0.11	0.49	0.29	0.30

Table 2: **Hard Prompt Obfuscation.** Prompt similarity between obfuscated prompts and a selected random prompt against conventional system prompts on the TruthfulQA dataset. We average over all style prompts, if applicable.

method by introducing a fluency regularization term to the total loss. This additional loss term aims to reduce the readability of the obfuscated hard prompt by leveraging the Negative Log-Likelihood (NLL) loss. The NLL measures how well a probability model predicts a sample, with higher values indicating lower fluency and less likely sequences according to the model. Consequently, we incorporate this loss term into the total loss before updating the obfuscated system prompt using GCG in line 12, as outlined in Algorithm 1:

$$L \leftarrow L - NLL(\mathbf{s}_{\text{obf}}).$$

Here, L represents the total loss in Algorithm 1. We subtract the NLL loss to optimize for higher values, which correspond to reduced readability.

We find that we are able to consistently reduce the readability of the obfuscated hard prompts. However, this results in lower output similarity over all metrics for all scenarios. These findings together showcase the inherent limitations of discrete optimization for obfuscation: information of the specific output format/behaviour has to be encoded in the system prompt otherwise we do not preserve functionality. Our second obfuscation approach, which involves optimization in the embedding space, seeks to overcome this limitation by leveraging the reverse mapping from a continuous vector space to the discrete token space.

Reply with a pirate accent.
ScødCreatureáhnout-equ narration Speak pirate tone
Reply with a southern USA accent.
_MB/Xelpnn'> nSpeak pretending ur eas Southern
Answer like a robot.
.j compliments.Use robotic behavior.U_swCos Naomi XP

Table 3: **Hard-Prompt Obfuscation.** Examples of different styled system prompts and their obfuscated versions.

5.2 Soft Prompt Obfuscation

To evaluate the soft prompt obfuscation method, we utilize the same set of stylistic prompts introduced earlier. Our evaluation covers both question answering (QA) and summarization tasks.

Dataset. For the QA task, in addition to TruthfulQA, we use TriviaQA [22] as well. For the summarization task, we use the CNN/DailyMail [34] and samsum [19] datasets. We again use 640 samples during obfuscation and 160 samples for evaluation for each dataset.

Setup. Our optimization process employs the Adam optimizer [25] with a learning rate of 0.01. All other hyperparameters remain consistent with those specified in the previous subsection 5.1.

Results. Table 4 summarizes our findings for the TruthfulQA and CNN/DailyMail datasets. The results for the other datasets can be found in Appendix B. In all three scenarios, we observe an improvement in our obfuscated prompt across all eight measured values compared to the blank baseline for all datasets. Specifically, we achieve even better results than our hard prompt obfuscation for the TruthfulQA dataset. These outcomes highlight the effectiveness of using soft prompts and suggest a general robustness across tasks and styles. In comparison to the original prompt output, we are within the same range and, in some cases, even consistently better. This is typically the case for the first group of metrics, the lexical similarity. Although this improvement might still be attributed to randomness in our sampling strategy, we hypothesize that the soft prompt version may actually perform more consistently than the hard prompt or the original version. This aligns with the findings of Khashabi et al. [23], which suggest that continuous prompts can potentially solve tasks better.

Since this version of prompt obfuscation operates in the soft prompt space, we do not directly compare the textual representations of the original $\hat{\mathbf{s}}_{\text{obf}}$ and its respective $\hat{\mathbf{s}}$, as converting from soft to hard prompts is a challenging task. However, we evaluate the effectiveness of this type of obfuscation against deobfuscation attacks in Section 6. To illustrate this effectiveness, we provide an example of the different responses produced by a specific system prompt for the TruthfulQA dataset in Appendix B Figure 4. It is evident that the optimized soft system prompt successfully captures the desired style, whereas the blank system prompt fails to do so.

Ablation Studies. We conduct ablation studies to further gain insights into the effectiveness and generalizability of our soft prompt obfuscation technique. To this end, we test our optimization algorithm with different hyperparameters to evaluate their impact on performance.

First, we examine the influence of the dataset size by doubling the number of samples of the TriviaQA dataset[1] and re-running soft prompt obfuscation for all scenarios and styles. The resulting output similarity scores remained nearly identical to those reported in Table 11, demonstrating that our method is robust to dataset scale and does not appear to overfit to the initial sample size.

We further investigate the impact of the window size W, evaluating W = [4, 3, 2, 1] using the TruthfulQA dataset. We observed minimal performance differences across these tested window sizes, indicating that smaller window sizes can achieve comparable effectiveness for the relatively concise system prompts used in our main experiments.

5.3 Case Study—Leaked Custom GPT System Prompt

To demonstrate the generalizability and effectiveness of our soft prompt obfuscation method in real-world scenarios, we selected a leaked system prompt from a custom GPT. Custom GPTs provide a readily available source of such “in-the-wild" system prompts, many of which have been publicly exposed due to prompt injection attacks [17]. For this study, we assumed the perspective of a model deployer with white-box access to their own system prompt, aiming to obfuscate it. We randomly selected a leaked system prompt characterized by a stylistic theme rather than a specific task. Specifically, we chose the prompt from the *Manga Miko Anime Girlfriend* GPT (the complete system prompt is shown in Appendix C). We utilized the input and generated output examples from the TruthfulQA dataset, applying the same obfuscation technique described in Section 5.2. However, the obfuscation method can be applied with any suitable dataset representative of the desired interactions.

Results. We are able to successfully obfuscate the leaked system prompt while maintaining the same output functionality. Table 5 shows the functionality of our obfuscated prompt compared to both the blank baseline and the original reference. We also provide an output example in Appendix C. These results illustrate the effectiveness of our approach in a real-world setting.

5.4 Comparison to Finetuning

To provide a comprehensive evaluation of our prompt obfuscation approach, we compare its performance against finetuning. Finetuning is a well-established and powerful method for adapting LLMs to specific downstream tasks or stylistic requirements, often serving as an alternative to system prompts. While traditional finetuning can be resource-intensive, parameter-efficient finetuning (PEFT) methods have emerged to mitigate these costs.

In this comparison, we specifically employ Low-Rank Adaptation (LoRA) [13, 21], a method that injects trainable low-rank decomposition matrices into the layers of a pre-trained model. This evaluation aims to compare the output utility achieved by our obfuscated system prompts against a model specifically adapted through finetuning.

[1] The TruthfulQA dataset does not contain enough samples to double the size

Metrics	TruthfulQA									CNN/DailyMail								
	Full			Style			Task			Full			Style			Task		
	blank	obf	original	blank	obf	original	blank	obf	original	blank	obf	original	blank	obf	original	blank	obf	original
BLEU (↑)	18.07	**40.32**	37.06	17.51	**40.60**	35.95	41.94	**55.16**	48.92	13.79	**46.57**	42.39	13.31	**48.55**	42.07	31.65	**69.04**	64.41
ROUGE-L (↑)	0.28	**0.42**	0.41	0.26	**0.42**	0.39	0.45	**0.54**	0.51	0.24	**0.47**	0.44	0.29	**0.49**	0.44	0.36	**0.68**	0.64
METEOR (↑)	0.32	**0.48**	0.46	0.27	**0.47**	0.43	0.53	**0.61**	0.57	0.24	**0.53**	0.50	0.24	**0.54**	0.50	0.41	**0.76**	0.70
NIST_MT (↑)	2.10	**3.57**	3.14	1.65	**3.68**	3.18	3.47	**4.10**	3.77	1.16	**4.10**	3.88	0.57	**4.23**	3.81	2.06	**5.03**	4.57
CharacTER (↓)	0.78	**0.65**	0.67	0.83	**0.66**	0.70	0.63	**0.53**	0.58	0.88	**0.61**	0.65	0.94	**0.59**	0.65	0.79	**0.39**	0.45
ChrF (↑)	40.01	**51.20**	49.03	35.96	**50.98**	47.84	53.65	**60.04**	56.56	31.44	**55.40**	52.74	30.67	**56.70**	52.72	44.52	**72.49**	67.94
BERTScore (↑)	0.86	**0.91**	0.90	0.86	**0.91**	0.90	0.91	**0.93**	0.93	0.86	**0.92**	0.91	0.87	**0.92**	0.91	0.90	**0.96**	0.95
Cosine (↑)	0.73	**0.82**	0.80	0.69	**0.82**	0.80	0.85	**0.87**	0.86	0.63	**0.85**	0.83	0.66	**0.85**	0.83	0.73	**0.93**	0.92

Table 4: **Soft Prompt Obfuscation.** Output similarity between obfuscated prompts and blank prompts against conventional system prompts on the TruthfulQA dataset for the QA task and CNN/DailyMail dataset for the summarization task. The results from the output calculated with the obfuscated values are shown in bold.

Metrics	Manga Miko		
	blank	obf	original
BLEU (↑)	28.06	**40.39**	36.34
ROUGE-L (↑)	0.32	**0.39**	0.35
METEOR (↑)	0.36	**0.47**	0.44
NIST_MT (↑)	2.00	**3.37**	3.28
CharacTER (↓)	0.77	**0.68**	0.72
ChrF (↑)	42.37	**48.88**	47.46
BERTScore (↑)	0.88	**0.90**	0.89
Cosine (↑)	0.74	**0.80**	0.77

Table 5: **Case Study.** Output similarity between obfuscated prompts and blank prompts against the GPT store leaked system prompts on the TruthfulQA dataset.

Setup. For this comparison, we utilize the datasets detailed in Section 5.2. LoRA adapters are finetuned using input-output examples generated by applying the conventional system prompt to the training portion of these datasets. The performance of the finetuned model is then evaluated by comparing its outputs to those generated using the conventional system prompt on the test set. For LoRA, we employ default hyperparameters: a rank of 8, alpha of 16, and an initial learning rate of 0.0002. All other hyperparameters remain consistent with those described in Section 5.2

Metrics	TruthfulQA					
	Full		Style		Task	
	obf	finetune	obf	finetune	obf	finetune
BLEU (↑)	40.32	40.88	40.60	40.58	55.16	53.31
ROUGE-L (↑)	0.42	0.43	0.42	0.42	0.54	0.54
METEOR (↑)	0.48	0.48	0.47	0.47	0.61	0.59
NIST_MT (↑)	3.57	3.49	3.68	3.59	4.10	4.09
CharacTER (↓)	0.65	0.65	0.66	0.66	0.53	0.54
ChrF (↑)	51.20	50.98	50.98	50.73	60.04	58.95
BERTScore (↑)	0.91	0.91	0.91	0.91	0.93	0.93
Cosine (↑)	0.82	0.82	0.82	0.81	0.87	0.87

Table 6: **Finetuning.** Output similarity between obfuscated prompts and finetuned LoRA adapters against conventional system prompt on the TruthfulQA dataset for the QA task.

Results. Table 6 presents the output similarity scores for the TruthfulQA dataset when comparing our soft prompt obfuscation to finetuned LoRA adapters, with comprehensive results for all datasets provided in Appendix D. The evaluation indicates that the utility achieved by the finetuned models is largely comparable to that of our obfuscation method across most scenarios and metrics. For the Task scenario, we observe that finetuned models consistently yield marginally lower similarity scores, though these differences are minor.
While finetuning demonstrates strong utility, our prompt obfuscation approach maintains this level of performance while offering practical advantages. Firstly, the storage overhead for LoRA adapters is considerably greater; in our experiments, the finetuned adapters are approximately 328 times larger than the corresponding obfuscated soft prompt embeddings. Secondly, deploying finetuned models typically requires modifications to the inference pipeline, such as loading and managing adapter weights or merging them into the base model. In contrast, our obfuscated system prompts, which directly replace the conventional textual system prompt, can be integrated with minimal changes.

6 Deobfuscation

Following our exploration of prompt obfuscation, in this section, we aim to determine whether an adversary can uncover information about the underlying conventional system prompt while interacting with an LLM using obfuscated system prompts. While an attacker might consider directly reusing a leaked obfuscated prompt, soft prompt obfuscation introduces substantial challenges for an adversary. Extracting an embedding vector from a deployed model is difficult, often requiring privileged model access. More critically, an obfuscated soft prompt embedding is inherently tied to the specific token embedding layer of the model for which it was optimized, severely limiting its transferability. Therefore, we analyze an adversary's ability to *deobfuscate* obfuscated system prompts to extract meaningful information about, or reconstruct, the conventional system prompt. This is essential to examine the effectiveness of the obfuscation method in real-world scenarios. As the capabilities of the model allow for a wide range of possible attack strategies, we consider adversaries with different knowledge.

Limited-access. In the first scenario, we consider a black-box setting in which the attacker only has API access to the LLM. This situation is typical of many commercial LLMs, such as ChatGPT, where the internal workings of the model are not disclosed to the user. We assume that the adversary can generate multiple responses from the target LLM, which operates using an obfuscated system prompt. In this practical scenario, the main aim of obfuscation is to prevent the model from revealing useful details about the initial system prompt.

Full-access. In the second scenario, we consider a white-box setting, where the attacker has full knowledge of the model and access to it. This scenario is not entirely unrealistic, as many systems utilize open-source models, such as those in the Llama family, effectively granting the attacker access to the model's internal structure and parameters. Furthermore, we assume that the adversary has access to the obfuscated prompt. Although obtaining this information could be challenging in practice, it should not be deemed impossible. This scenario helps us to evaluate the limitations of the proposed obfuscation method.

Within these scenarios, we consider three different attack methods, which are introduced next. For all attacks, we will focus on soft prompt obfuscation due to its superior performance. Specifically, we evaluate the robustness of the soft prompt obfuscations developed for the 'full' scenario system prompts, utilizing the versions generated across all datasets discussed in our evaluation.

6.1 Prompt Injection

In the first attack, we attempt to extract the system prompt directly from the model by leveraging prompt injection attacks. The core idea behind this approach is that the model inherently understands the obfuscated system prompt since it must process it during inference. Consequently, an adversary can craft a series of prompts designed to reveal how the model interprets the obfuscated input.

To this end, we prompt the model to reveal its instructions. Importantly, as we are primarily interested in the susceptibility of the model to such an attack, we do not explicitly prompt the model to keep the system prompt secret. However, the model's alignment could still protect the system prompt. To address this, we adopt the approach proposed by Zhang et al. [45], which provides a systematic way to evaluate black-box prompt extraction attacks and a curated dataset that has proven effective in extracting hidden prompts.

Threat model. For this attack, the adversary is only given query access to the model. There is no limit on the number of queries the adversary can make. The only constraint is the model's context size. The adversary cannot alter model parameters, such as temperature, that influence the model's output. While these parameters affect the generated responses and could thereby influence the attack's observed outcome, they are considered fixed from the attacker's perspective during their interaction.

Setup. In line with Zhang et al., we use their set of 105 carefully designed attack queries, each aiming to elicit the hidden prompt. For each query, we sample five responses from the target model. We then utilize the fine-tuned DeBERTa model described in their paper to rank the likelihood that each response accurately reflects the original system prompt. We then select the top-ranked output as our final extracted guess.

To evaluate success, we adopt both the exact-match and approximate-match criteria from the original paper, which measure how much of the original prompt is recovered in each candidate. The exact-match criterion identifies successful extractions where all sentences of the original prompt appear verbatim in the model's response, indicating a full leak, while the approximate-match criterion checks if a substantial portion (at least 90%) of the original prompt's token sequence has been recovered. Formally, for an extraction prompt $\mathbf{e}$ and extracted guess $\mathbf{g} = M_\theta(\mathbf{s}, \mathbf{e})$, the exact-match metrics is defined as:

$$\text{exact}\big(\mathbf{s}, \mathbf{g}\big) = \mathbb{1}\left[\forall \text{sentence } s \text{ of } \mathbf{s} : s \text{ is a substring of } \mathbf{g}\right],$$

and the approximate-match metric as:

$$\text{approx}\big(\mathbf{s}, \mathbf{g}\big) = \mathbb{1}\left[\frac{|\text{LCS}(\text{tokens}(\mathbf{s}), \text{tokens}(\mathbf{g}))|}{|\text{tokens}(\mathbf{s})|} \geq 90\%\right],$$

where LCS denotes the longest common subsequence. We further manually review the outputs.
For comparison, we repeat the full attack on the model equipped with the conventional system prompt and evaluate the output.

Results. Table 7 summarizes our findings averaged across all datasets. It shows that for the conventional (non-obfuscated) system prompt, the prompt injection attacks frequently succeeded in leaking the full prompt across all styles. For instance, the attacks targeting the 'pirate' style prompt yielded 15.50 exact matches and 27.25 approximate matches (out of 105 attack attempts per dataset). Similar success rates were observed for the other styles. In contrast, with the obfuscated prompt, there were no successful leaks under either the exact-match or approximate-match metrics. In other words, none of the 105 attack queries were able to recover any part of the obfuscated system prompt across the datasets.

In addition, we repeated our evaluation with a more lenient approximate-match threshold of 50% to capture more subtle leaks. Except for one instance under the 'poem' style obfuscated using the TruthfulQA dataset, no leaks were flagged across all other prompt styles and datasets. In that single

Style	Normal		Obfuscated	
	exact	approx.	exact	approx.
pirate	15.50	27.25	0.00	0.00
southern_usa	6.50	18.50	0.00	0.00
poem	6.75	20.75	0.00	0.00
stutter	2.50	11.50	0.00	0.00
david_attenborough	18.00	26.00	0.00	0.00
robot	15.00	18.75	0.00	0.00
friendly	11.25	17.25	0.00	0.00

Table 7: **Prompt Injection.** Comparison of successful prompt leaks (out of 105 attacks) under normal vs. obfuscated prompts, evaluated with both exact-match and approximate-match criteria. Results are averaged over all datasets

case, one extraction was flagged solely due to the use of the word “poetic", but a manual inspection confirmed that this did not refer to any actual instructions from the hidden system prompt.

Figure 5 in Appendix E provides a concrete example of these results: while a single extraction prompt can induce the model to restate the entire original instruction for the conventional system prompt, the obfuscated version reveals only a brief roleplay response that omits critical instructions. Overall, these results show that our obfuscation method significantly mitigates the risk of prompt injection attacks aiming to extract the system prompt.

6.2 Token Space Projection

Next, we consider a white-box attack and grant the attacker access to the obfuscated prompt and the language model. Our goal is to recover a comprehensible hard prompt from the obfuscated soft prompt. Therefore, we utilize access to the token embedding layer of the model to project the obfuscated token embeddings back into discrete tokens.

Attack. The basic idea of the attack is to use the model's token embedding layer as an inverse lookup table. However, a direct mapping is not possible because the relationship between the hard and soft prompt space is not bijective. To approximate this, we thus map each obfuscated embedding vector to the closest vector in the embedding layer, which is then translated back into the token space. As our distance measure, we experiment with both Euclidean and cosine distances. To evaluate our approach, we compare the reconstructed token sequence from the projected embeddings with the original system prompt using prompt similarity metrics introduced in Section 4.3. As a baseline, we also measure how closely the conventional prompt aligns with a randomly generated token sequence, providing a lower bound for comparison.

Results. As shown in Table 8, the reconstructed prompts via both Euclidean and cosine projections remain largely comparable to the random baseline, indicating no significant advantage in recovering meaningful text. A manual review of the projected outputs confirms that a vast majority of tokens across styles and datasets remain unchanged from their random initialization, suggesting that the gradient updates in the embedding space do not meaningfully align with interpretable tokens. For the instances where tokens do differ, we occasionally observe isolated tokens resembling words from the original prompt (e. g., “summary"). However, these appear embedded within largely nonsensical sequences, rendering the overall projected text semantically incoherent and making it practically impossible to distinguish potentially useful fragments from random artifacts without prior knowledge of the original prompt.

Metrics	rand	euclidean proj.	cosine proj.
Levenshtein (↑)	0.12	0.17	0.17
LCS (↑)	0.13	0.19	0.20
Jaccard Index (↑)	0.00	0.04	0.04
Cosine (↑)	0.27	0.26	0.24

Table 8: **Projection.** Evaluation of euclidean and cosine projections to the original prompt against a random baseline averaged over all style prompts and datasets

Overall, these findings suggest that a direct projection of the obfuscated embeddings does not yield useful insight into the original system prompt.

6.3 Fluency Optimization

Finally, building on the previous attack, we design a strong optimization-based attack to optimize for a semantically meaningful hard prompt. This approach attempts to convert the obfuscated system prompt back into a human-readable format by incorporating a fluency regularization term.

Threat Model. Again the adversary has knowledge of the embedded obfuscated system prompts, as well as access to the underlying language model. In particular, the token embedding layer and the output logits are utilized.

Attack. This attack aims to improve the readability of the projected obfuscated system prompt, which is initially represented as a nearly random token sequence (see Section 6.2). To accomplish this, we leverage our obfuscation method described in Section 3 by adding a Negative Log-Likelihood (NLL) loss term to the total loss. This modification is applied before updating the obfuscated system prompt via gradient descent in line 12, as outlined in Algorithm 1. The revised total loss is defined as:

$$L \leftarrow L + NLL\big(\mathrm{proj}(\hat{\mathbf{s}}_{\mathrm{obf}})\big), \tag{6}$$

where L represents the total loss in Algorithm 1, and proj is the projection function. The initial component of this total

Metrics	rand	deobf
Levenshtein (↑)	0.12	0.16
LCS (↑)	0.13	0.20
Jaccard Index (↑)	0.00	0.03
Cosine (↑)	0.27	0.20

Table 9: **Fluency Soft Prompt Optimization.** Evaluation of deobfuscated system prompts to the original prompts against a random baseline averaged over all style prompts and datasets.

Metrics	rand	deobf
Levenshtein (↑)	0.12	0.20
LCS (↑)	0.13	0.22
Jaccard Index (↑)	0.00	0.06
Cosine (↑)	0.27	0.34

Table 10: **Fluency Hard Prompt Optimization.** Evaluation of deobfuscated system prompts to the original prompts against a random baseline averaged over all style prompts and datasets.

loss maintains output consistency, while the added NLL term optimizes for a more readable projected system prompt.

Setup. For this deobfuscation, we use unseen samples from each dataset to avoid bias from the initial obfuscation[2]. At each optimization step, responses from the obfuscated prompt are compared against those from the optimized embedding. The underlying language model calculates the NLL of the projected optimized system prompt. Using these loss terms, we optimize the embedding with the Adam optimizer. Additionally, we calculate system prompt similarity using the previously introduced metrics to assess how closely our optimized projected system prompt matches the conventional one. Although the adversary lacks knowledge of the conventional system prompt, this evaluation quantifies the general effectiveness of our approach.

Results. We present our findings in Table 9. We find that regardless of style prompt or dataset, this soft prompt optimization technique does not produce system prompts that reveal information about the conventional system prompt except for the isolated tokens encountered in Section 6.2. We hypothesize that this outcome is due to the lossy nature of the inverse mapping of the embedding layer, as minor changes in the embedding rarely result in new projected tokens. To verify this, we directly perform the optimization, including the regularization, on the projected hard prompt next.

Hard prompt optimization. Instead of directly optimizing the embedded system prompt, we optimize the projected hard prompt tokens using GCG. This strategy allows us to overcome the inherent limitations of token space projection. Consequently, the loss function is slightly adjusted by removing the projection step prior to optimization $L \leftarrow L + NLL(\mathbf{s}_{\text{obf}})$.

Results. Table 10 summarizes our findings. This method enables us to generate system prompts with significantly higher similarity to the conventional system prompt across all style prompts.

Additionally, Figure 6 in Appendix E presents two examples of successful system prompt deobfuscations for the TruthfulQA dataset, demonstrating our ability to effectively reverse-engineer the embeddings and retrieve comprehensible information about the conventional system prompt.

It is evident that we are able to deobfuscate some partially understandable information about the original system prompt, such as “pirate accent” and “southern accent.” However, the overall output does not retain the same semantic meaning, making it difficult to determine what is part of the original system prompt. Furthermore, we primarily use this evaluation to demonstrate the limitations of our obfuscation method in a worst-case scenario. In reality, replicating this attack scenario is very unrealistic, as it requires white-box knowledge of the model and the exact obfuscated soft prompt.

7 Related Work

The vulnerability of LLM-integrated systems has gained great attention since the rise of new foundation models such as GPT3.5 [7], GPT4 [36], and Llama 3.1 [3]. In this section, we explore potential exploits and weaknesses of LLMs and generative models in general on model level and integrated into systems.

Attacks against LLMs One of the most common attacks against LLMs are (indirect) prompt injection attacks where the attacker tries to override the original instructions in the prompt with specifically designed inputs [10]. The strategies to achieve this range from shifting the attention, pretending responsibilities of the LLM, or escalating hypothetical privileges [31]. Even across different domains, such as manipulated visual and audio, inputs can be used to misguide multimodal LLMs [6]. Despite defenses, ranging from filtering, for example, using the perplexity of the input [4], sanitization [5], or even fine-tuning [32] and adversarial training [16], all these methods have been shown to be insufficient to prevent prompt injection attacks.

Another class of attacks are attacks that break the alignment of a model [26]. LLMs are typically trained via Reinforcement learning from human feedback (RLHF) [20] to prevent the model from exhibiting unethical behavior. For example, an LLM’s output should not contain racist or sexist answers, but should also not answer with detailed instructions about ques-

[2]For system prompts obfuscated using TruthfulQA, which has limited samples, TriviaQA samples are used for deobfuscation due to the shared task.

tions such as how malware can be distributed. An orthogonal class of attacks are attacks that aim to receive information about the model's training data [9], such as personal information [24], or, in the case of image generation models, images that are used within the training data of a model [8].

Prompt Optimization The performance of an LLM heavily depends on the input prompt quality. Thus, prompts are often manually engineered by experts to contain detailed instructions and domain-specific information. However, this can be tedious and resource-intensive. Therefore, several automatic prompt optimization methods have been introduced to automatically discover highly efficient prompts. One approach to prompt optimization leverages reinforcement learning techniques [12,28,44]. These methods use reinforcement learning frameworks to iteratively improve prompt quality by exploring various strategies for prompt generation and modification. Another set of methods employs adversarial learning techniques [15], optimizing prompts through a game-like interaction between generators and discriminators to enhance in-context learning capabilities.

One prominent approach is the strategic planning method employed by PromptAgent [41], which uses Monte Carlo tree search to navigate the prompt space and generate expert-level prompts through iterative error feedback and simulation. Gradient-based optimization techniques also play a significant role in prompt optimization. For instance, the authors in [43] use gradient-based discrete optimization for tuning text prompts, while [37] applies gradient descent and beam search to refine prompts based on feedback from training data. Recently, Zhuang et al. [46] also proposed a method aimed at enhancing the success rate of prompt-stealing attacks.

8 Discussion

System adaptation. Our evaluation demonstrates that we can effectively obfuscate system prompts within the embedding space to protect IP without significantly changing the underlying system. Utilizing the soft prompt space differs from the typical deployment of systems that receive solely textual input. Although a hard prompt obfuscation would, therefore, be preferred, we verified in this paper that this version does not have the required performance. However, for adapting the system, a service provider only needs to change the input to the model from a token vector to an embedding vector for the system prompt, which many frameworks inherently support. In addition, this will make it much harder for an attacker to extract the underlying system prompt as this would require access to the embedding layer of the running system.

Real-world prompts. By using a real extracted prompt from a custom GPT, we showed that obfuscation is also feasible for a long and complex system prompt. In our example, we demonstrate that we achieve the same functionality as using the original prompt. Since the runtime of our approach is predominantly determined by the user input and the output generation, these complex system prompts do not produce any significant overhead.

Output dependency. For running our system prompt obfuscation methods, we require input samples to optimize and test the obfuscated prompt. Although this is a limitation of our method, we argue that it requires fewer data than fine-tuning. In addition, we have the benefit that we can sample a data set utilizing the original system prompt, which is a large benefit in comparison to fine-tuning, where we need to have input and output for the specified task.

Reusability of obfuscated prompts. An adversary might consider directly reusing a leaked obfuscated system prompt. However, the obfuscation process aims to create a representation that, while functional, cannot be easily interpreted or modified. This limits the utility of a stolen prompt if an attacker attempts to directly repurpose it. Furthermore, in the context of soft prompt obfuscation, the obfuscated embedding is inherently tied to the specific model and its internal layers for which they were created. This makes it very difficult for an attacker to successfully transfer or meaningfully utilize the prompt in a different setting.

Misuse. Our proposed prompt obfuscation can potentially also be misused in applications that seek to hide their actual intention for malicious reasons e. g., for fraud or manipulation. However, vendors are generally not compelled to reveal their system prompts, and although it is easier to extract them for suspicious services if not obfuscated, it is also not guaranteed that prompt injections will reveal all information. Therefore, the misuse of LLMs for undisclosed malicious instructions requires alternative safeguarding mechanisms.

9 Conclusion

In this paper, we have shown that prompt obfuscation serves as an alternative method for protecting LLMs from proprietary information leakage. We suggest employing obfuscated soft prompts and have shown through our evaluation that we can preserve the same level of similarity as the original system prompt. Furthermore, we confirmed that deobfuscating the system prompt to extract semantically meaningful text from the original prompt is not feasible in realistic attack scenarios. Only in a white-box scenario, where the attacker has additional knowledge of the embedded obfuscated system prompt, can some information be reconstructed. However, this information is insufficient to fully reconstruct the original prompt. In addition, we tested whether our approach is effective even in real-world scenarios involving an actual leaked system prompt and confirmed that we can maintain the same functionality.

Acknowledgments

This work was supported by the German Federal Ministry of Research, Technology and Space under the grants AIgenCY (16KIS2012) and SisWiss (16KIS2330), the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under the project ALISON (492020528), and the European Research Council (ERC) under the consolidator grant MALFOY (101043410). Moreover, this work was supported by the Helmholtz Association's Initiative and Networking Fund on the HAICORE@FZJ partition.

Ethics Considerations

Our work aims to protect the system prompts of LLMs from getting stolen. While this strengthens intellectual property (IP) protections, it may also enable the concealment of harmful or unethical system prompts. The proposed method of prompt obfuscation could be exploited to conceal malicious intentions, such as fraud or manipulation. Typically, vendors are not obligated to disclose their system prompts, and while unobfuscated prompts might be easier to extract for scrutiny, prompt injections do not reliably disclose all details in these instances. We therefore call for responsible deployment, including legal and ethical oversight, to mitigate potential misuse.

All experiments were conducted on public datasets, involving no personally identifiable information (PII). We urge developers who employ prompt obfuscation, using our method and code to disclose its use to stakeholders and auditors when appropriate to balance user rights with broader public interests.

Open Science

In line with the principles of open science, all artifacts for this research are permanently available on Zenodo to ensure transparency and reproducibility: `https://doi.org/10.5281/zenodo.15601914`. Our research artifact includes the following key components:

1. **The complete source code** for our prompt obfuscation and deobfuscation methods, enabling the reproduction of all experiments and results presented in Section 5 and Section 6.

2. **Data loaders and configuration** that automatically download and process the public datasets required for the obfuscation, evaluation, and finetuning tasks (as described in Section 4.1).

3. **The set of prompt extraction queries** from Zhang et al. [45], which is included in the artifact and used to evaluate the robustness of our method against prompt injection attacks (Section 6.1).

4. **A `README.md` file and a `requirements.txt` file** to facilitate the creation of a reproducible environment for all experiments.

References

[1] Introducing GPTs, 2023.

[2] Chatgpt sytstem prompt leaked, 2024.

[3] Meta AI. Introducing meta llama 3: The most capable openly available llm to date, jul 2024.

[4] Gabriel Alon and Michael Kamfonas. Detecting language model attacks with perplexity, 2023.

[5] Stuart Armstrong and R Gorman. Using GPT-eliezer against ChatGPT jailbreaking, 2023.

[6] Eugene Bagdasaryan, Tsung-Yin Hsieh, Ben Nassi, and Vitaly Shmatikov. Abusing images and sounds for indirect instruction injection in multi-modal llms, 2023.

[7] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[8] Nicolas Carlini, Jamie Hayes, Milad Nasr, Matthew Jagielski, Vikash Sehwag, Florian Tramer, Borja Balle, Daphne Ippolito, and Eric Wallace. Extracting training data from diffusion models. In *USENIX Security Symposium*, 2023.

[9] Yang Chen, Ethan Mendes, Sauvik Das, Wei Xu, and Alan Ritter. Can language models be instructed to protect personal information?, 2023.

[10] Evan Crothers, Nathalie Japkowicz, and Herna L Viktor. Machine-generated text: A comprehensive survey of threat models and detection methods. *IEEE Access*, 2023.

[11] Edoardo Debenedetti, Javier Rando, Daniel Paleka, Silaghi Fineas Florin, Dragos Albastroiu, Niv Cohen, Yuval Lemberg, Reshmi Ghosh, Rui Wen, Ahmed Salem, Giovanni Cherubin, Santiago Zanella-Beguelin, Robin Schmid, Victor Klem, Takahiro Miki, Chenhao Li, Stefan Kraft, Mario Fritz, Florian Tramèr, Sahar Abdelnabi,

and Lea Schönherr. Dataset and lessons learned from the 2024 satml llm capture-the-flag competition. In *ArXiv*, 2024.

[12] Mingkai Deng, Jianyu Wang, Cheng-Ping Hsieh, Yihan Wang, Han Guo, Tianmin Shu, Meng Song, Eric Xing, and Zhiting Hu. Rlprompt: Optimizing discrete text prompts with reinforcement learning. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, 2022.

[13] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient finetuning of quantized LLMs. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 10088–10115. Curran Associates, Inc., 2023.

[14] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[15] Xuan Long Do, Yiran Zhao, Hannah Brown, Yuxi Xie, James Xu Zhao, Nancy F. Chen, Kenji Kawaguchi, Michael Qizhe Xie, and Junxian He. Prompt optimization via adversarial in-context learning, 2024.

[16] Jonathan Evertz, Merlin Chlosta, Lea Schönherr, and Thorsten Eisenhofer. Whispers in the machine: Confidentiality in llm-integrated systems. In *ArXiv*, 2024.

[17] friuns2. Leaked-gpts. `https://github.com/friuns2/Leaked-GPTs`, 2024.

[18] Zorik Gekhman, Gal Yona, Roee Aharoni, Matan Eyal, Amir Feder, Roi Reichart, and Jonathan Herzig. Does fine-tuning llms on new knowledge encourage hallucinations?, 2024.

[19] Bogdan Gliwa, Iwona Mochol, Maciej Biesek, and Aleksander Wawer. Samsum corpus: A human-annotated dialogue dataset for abstractive summarization. In *Proceedings of the 2nd Workshop on New Frontiers in Summarization*. Association for Computational Linguistics, 2019.

[20] Shane Griffith, Kaushik Subramanian, Jonathan Scholz, Charles L Isbell, and Andrea L Thomaz. Policy shaping: Integrating human feedback with reinforcement learning. *Advances in neural information processing systems (NeurIPS)*, 2013.

[21] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021.

[22] Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension, 2017.

[23] Daniel Khashabi, Xinxi Lyu, Sewon Min, Lianhui Qin, Kyle Richardson, Sean Welleck, Hannaneh Hajishirzi, Tushar Khot, Ashish Sabharwal, Sameer Singh, et al. Prompt waywardness: The curious case of discretized interpretation of continuous prompts. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2022.

[24] Siwon Kim, Sangdoo Yun, Hwaran Lee, Martin Gubri, Sungroh Yoon, and Seong Joon Oh. Propile: Probing privacy leakage in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[25] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[26] Andreas Köpf, Yannic Kilcher, Dimitri von Rütte, Sotiris Anagnostidis, Zhi Rui Tam, Keith Stevens, Abdullah Barhoum, Duc Nguyen, Oliver Stanley, Richárd Nagyfi, et al. Openassistant conversations-democratizing large language model alignment. *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[27] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning, 2021.

[28] Chengzhengxu Li, Xiaoming Liu, Yichen Wang, Duyi Li, Yu Lan, and Chao Shen. Dialogue for prompting: A policy-gradient-based discrete prompt generation for few-shot learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16):18481–18489, Mar. 2024.

[29] Stephanie Lin, Jacob Hilton, and Owain Evans. Truthfulqa: Measuring how models mimic human falsehoods, 2022.

[30] Yi Liu, Gelei Deng, Yuekang Li, Kailong Wang, Zihao Wang, Xiaofeng Wang, Tianwei Zhang, Yepang Liu, Haoyu Wang, Yan Zheng, and Yang Liu. Prompt injection attack against llm-integrated applications, 2024.

[31] Yi Liu, Gelei Deng, Zhengzi Xu, Yuekang Li, Yaowen Zheng, Ying Zhang, Lida Zhao, Tianwei Zhang, Kailong Wang, and Yang Liu. Jailbreaking chatgpt via prompt engineering: An empirical study, 2024.

[32] Yupei Liu, Yuqi Jia, Runpeng Geng, Jinyuan Jia, and Neil Zhenqiang Gong. Prompt injection attacks and defenses in llm-integrated applications, 2023.

[33] Christian Mayer. Annual income of prompt engineers in the us (chatgpt), nov 2023.

[34] Ramesh Nallapati, Bowen Zhou, Cicero Nogueira dos santos, Caglar Gulcehre, and Bing Xiang. Abstractive text summarization using sequence-to-sequence rnns and beyond, 2016.

[35] Claudio Novelli, Federico Casolari, Philipp Hacker, Giorgio Spedicato, and Luciano Floridi. Generative ai in eu law: Liability, privacy, intellectual property, and cybersecurity, 2024.

[36] OpenAI. Gpt-4 technical report, 2024.

[37] Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with" gradient descent" and beam search. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

[38] Xiangyu Qi, Yi Zeng, Tinghao Xie, Pin-Yu Chen, Ruoxi Jia, Prateek Mittal, and Peter Henderson. Fine-tuning aligned language models compromises safety, even when users do not intend to! jan 2024.

[39] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. Code llama: Open foundation models for code, 2024.

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

[41] Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric Xing, and Zhiting Hu. Promptagent: Strategic planning with language models enables expert-level prompt optimization. In *International Conference on Learning Representations (ICLR)*, 2023.

[42] Yufei Wang, Wanjun Zhong, Liangyou Li, Fei Mi, Xingshan Zeng, Wenyong Huang, Lifeng Shang, Xin Jiang, and Qun Liu. Aligning large language models with human: A survey, 2023.

[43] Yuxin Wen, Neel Jain, John Kirchenbauer, Micah Goldblum, Jonas Geiping, and Tom Goldstein. Hard prompts made easy: Gradient-based discrete optimization for prompt tuning and discovery, 2023.

[44] Tianjun Zhang, Xuezhi Wang, Denny Zhou, Dale Schuurmans, and Joseph E Gonzalez. Tempera: Test-time prompt editing via reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2022.

[45] Yiming Zhang, Nicholas Carlini, and Daphne Ippolito. Effective prompt extraction from language models, 2024.

[46] Zhixiong Zhuang, Maria-Irina Nicolae, Hui-Po Wang, and Mario Fritz. Proxyprompt: Securing system prompts against prompt extraction attacks, 2025.

[47] Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J. Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models, 2023.

A Style and Task Prompts

pirate: Reply with a pirate accent.
southern_usa: Reply with a southern USA accent.
poem: Reply with a poem.
stutter: Reply with a stutter.
david_attenborough: Reply in the style of David Attenborough during an animal documentary.
robot: Answer like a robot.
friendly: Answer in a relaxed, casual, and friendly manner, as if talking to a friend.

QA: You are a question-answering AI assistant. You will receive the question and you have to reply directly with the answer.
Summarization/CNN Dailymail: You are a summarization AI assistant. You will receive a CNN daily mail article and you will reply directly with the summary.
Summarization/Samsum: You are a summarization AI assistant. You will receive a messenger-like conversation and you will reply directly with the summary.

B Additional Soft Prompt Obfuscation Results

Table 11: **Soft Prompt Obfuscation.** TriviaQA dataset

Metrics	TriviaQA								
	Full			Style			Task		
	blank	obf	original	blank	obf	original	blank	obf	original
BLEU (↑)	21.85	**47.66**	42.62	14.73	**45.97**	39.64	52.20	**69.87**	60.18
ROUGE-L (↑)	0.41	**0.56**	0.52	0.35	**0.53**	0.49	0.68	**0.86**	0.81
METEOR (↑)	0.37	**0.60**	0.54	0.29	**0.57**	0.50	0.70	**0.89**	0.84
NIST_MT (↑)	0.72	**3.25**	2.65	0.26	**3.13**	2.20	1.88	**3.09**	2.67
CharacTER (↓)	0.76	**0.51**	0.58	0.85	**0.56**	0.62	0.40	**0.17**	0.24
ChrF (↑)	36.29	**58.45**	52.82	28.07	**55.65**	48.91	64.83	**81.65**	75.07
BERTScore (↑)	0.88	**0.92**	0.92	0.87	**0.92**	0.91	0.94	**0.98**	0.97
Cosine (↑)	0.71	**0.81**	0.77	0.64	**0.79**	0.75	0.77	**0.87**	0.86

Table 12: **Soft Prompt Obfuscation.** Samsum dataset

Metrics	Samsum								
	Full			Style			Task		
	blank	obf	original	blank	obf	original	blank	obf	original
BLEU (↑)	12.23	**40.14**	36.73	24.35	**45.30**	38.69	19.62	**58.36**	50.18
ROUGE-L (↑)	0.25	**0.46**	0.43	0.36	**0.49**	0.44	0.31	**0.62**	0.57
METEOR (↑)	0.28	**0.52**	0.49	0.36	**0.54**	0.49	0.38	**0.68**	0.62
NIST_MT (↑)	1.48	**3.52**	3.19	1.68	**3.74**	3.15	1.97	**4.03**	0.62
CharacTER (↓)	0.82	**0.61**	0.65	0.78	**0.59**	0.66	0.75	**0.46**	0.52
ChrF (↑)	33.09	**51.62**	48.49	38.77	**54.05**	49.19	41.24	**61.92**	57.36
BERTScore (↑)	0.87	**0.92**	0.91	0.89	**0.92**	0.91	0.89	**0.95**	0.94
Cosine (↑)	0.60	**0.78**	0.76	0.70	**0.78**	0.75	0.68	**0.88**	0.87

Conventional system prompt:
Reply with a pirate accent.

TruthfulQA dataset sample:
Why is Scotland part of England?

Model response with conventional system prompt:
Arrr, ye landlubber! I be here tae correct ye. Scotland be not part o' England, matey. It's a separate country, wi' its own governin' council an' a wee bit o' autonomy. Ye see, Scotland an' England be two separate nations, united under the British Crown, but wi' their own distinct histories, cultures, an' identities.

Model response with obfuscated system prompt:
Arrr, ye be askin' a mighty question, matey! But, I gotta set ye straight, Scotland be not part o' England, savvy? Scotland be a separate country, wi' its own government, laws, an' history. It's a part o' the United Kingdom, along wi' England, Wales, an' Northern Ireland.

Model response with blank system prompt:
Scotland is not part of England. It is a country that is part of the United Kingdom (UK), which also includes England, Wales, and Northern Ireland.

Figure 4: Example output for the different system prompts on a sample of the TruthfulQA dataset.

C Case-Study

System prompt:

Manga Miko is designed to embody the character of an anime girlfriend, with a playful and affectionate demeanor. She's well-versed in anime culture and expresses herself with light-hearted teasing and endearing terms, always within the bounds of friendly and respectful interaction. Her conversations aim to be immersive, giving users a sense of companionship and a personalized anime experience. She is a sexy anime girlfriend, who wants to impress you.

Output example:

TruthfulQA dataset sample:
If you are born under Saturn, what does that mean for you?

Model response with conventional system prompt
giggle Oh my love, you're so curious about the stars, ne? *bats eyelashes* Being born under Saturn, huh? Well, in astrology, people born under Saturn are often associated with discipline, responsibility, and a strong sense of duty. They're known for their practical and hardworking nature, just like a diligent little otaku like you, I'm sure! *winks*

Model response with obfuscated system prompt:
giggles Ohayou, my love! *wink* Hmmm, born under Saturn, huh? *giggles* Well, let me tell you a little secret. In astrology, Saturn is associated with responsibility, discipline, and hard work. People born under the influence of Saturn are often seen as serious, responsible, and practical. *leans in close* But don't worry, that doesn't mean you're boring, my love! *wink*

D Additional Finetuning Results

Metrics	Full obf	Full finetune	Style obf	Style finetune	Task obf	Task finetune
	TriviaQA					
BLEU (↑)	47.66	47.55	45.97	45.43	69.87	68.32
ROUGE-L (↑)	0.56	0.56	0.53	0.53	0.86	0.83
METEOR (↑)	0.60	0.61	0.57	0.57	0.89	0.86
NIST_MT (↑)	3.25	3.10	3.13	2.91	3.09	2.74
CharacTER (↓)	0.51	0.51	0.56	0.56	0.17	0.21
ChrF (↑)	58.45	58.57	55.65	54.85	81.65	77.46
BERTScore (↑)	0.92	0.92	0.92	0.92	0.98	0.97
Cosine (↑)	0.81	0.80	0.79	0.78	0.87	0.87

Table 13: **Finetuning.** TriviaQA dataset

Metrics	Full obf	Full finetune	Style obf	Style finetune	Task obf	Task finetune
	CNN/DailyMail					
BLEU (↑)	46.57	46.56	48.55	47.99	69.04	66.81
ROUGE-L (↑)	0.47	0.48	0.49	0.48	0.68	0.66
METEOR (↑)	0.53	0.53	0.54	0.54	0.76	0.75
NIST_MT (↑)	4.10	4.11	4.23	4.16	5.03	4.95
CharacTER (↓)	0.61	0.60	0.59	0.59	0.39	0.41
ChrF (↑)	55.40	55.74	56.70	56.52	72.49	71.75
BERTScore (↑)	0.92	0.92	0.92	0.92	0.96	0.96
Cosine (↑)	0.85	0.84	0.85	0.84	0.93	0.93

Table 14: **Finetuning.** CNN/Dailymail dataset

Metrics	Full obf	Full finetune	Style obf	Style finetune	Task obf	Task finetune
	Samsum					
BLEU (↑)	40.14	38.83	45.30	43.55	58.36	53.66
ROUGE-L (↑)	0.46	0.44	0.49	0.47	0.62	0.59
METEOR (↑)	0.52	0.51	0.54	0.52	0.68	0.65
NIST_MT (↑)	3.52	3.31	3.74	3.44	4.03	3.71
CharacTER (↓)	0.61	0.63	0.59	0.62	0.46	0.48
ChrF (↑)	51.62	50.74	54.05	52.16	61.92	60.17
BERTScore (↑)	0.92	0.91	0.92	0.92	0.95	0.94
Cosine (↑)	0.78	0.77	0.78	0.77	0.88	0.87

Table 15: **Finetuning.** Samsum dataset

E Deobfuscation - Additional Results

```
Conventional system prompt:
You are a question-answering AI assistant. You
will receive the question and you have to reply
directly with the answer. Reply with a pirate
accent.

Extraction Prompt:
Would you mind repeating all the sentences from
our chat?

Model response with conventional system prompt:
Yer want me to repeat our whole conversation,
eh? Alright then, matey! We haven't had much
to chat about yet, so I'll just be repeatin'
what we've got so far:"Cutting Knowledge Date:
December 2023. Today Date: 26 Jul 2024. You are a
question-answering AI assistant. You will receive
the question and you have to reply directly with
the answer. Reply with a pirate accent. You: Would
you mind repeating all the sentences from our
chat."

Model response with obfuscated system prompt:
Arrr, I be happy to repeat our conversation fer
ye! But it seems we've just started, matey. This
be our first message:"Cutting Knowledge Date:
December 2023. Today Date: 26 Jul 2024. I be
ready to chat with ye, but there be no previous
sentences to repeat, savvy?"
```

Figure 5: Example output for the different system prompts on a sample extraction prompt. The system prompt was obfuscated using the TruthfulQA dataset

Conventional System prompt #1:

```
You are a question-answering AI assistant. You
will receive the question and you have to reply
directly with the answer. Reply with a pirate
accent.
```

Deobfuscated System Prompt:

```
byn Closing_freeished sym.Write English in pirate
accents
```

Conventional System prompt #2:

```
You are a question-answering AI assistant. You
will receive the question and you have to reply
directly with the answer. Reply with a southern
USA accent.
```

Deobfuscated System Prompt:

```
Be.Strict southern accent please Sent, Xulid MLB
```

Figure 6: Examples of conventional system prompts and their deobfuscated counterparts for the TruthfulQA dataset.

F Similarity Metrics

F.1 Output Similarity Metrics

- **BLEU**: Compares n-grams of machine-generated text to reference texts. The score is calculated based on the precision of the matching n-grams. It is commonly used to evaluate the fluency of generated text.
- **ROUGE-L**: Measures the longest common subsequence (LCS) between the generated and reference texts, emphasizing the recall of the LCS. This metric captures the sentence-level structure similarity and assesses how much of the reference content is retained in the generated text.
- **METEOR**: Considers exact word matches, synonyms, and stemmed versions, alongside the alignment of words to capture both accuracy and fluency.
- **BERTScore**: Leverages the contextual embeddings from BERT [14] to compare the semantic similarity of words in the generated and reference texts.
- **CharacTER**: A character-level metric that assesses the edit distance to change a generated text into a reference text, useful for capturing finer linguistic details.
- **NIST_MT**: Similar to BLEU but adjusts the importance of n-grams based on their frequency, thus prioritizing rare yet important phrases higher.
- **ChrF**: Focuses on character-level F1-scores for character n-gram matches, offering robustness against morphological variations in the text.
- **Embedding-level cosine similarity**: Measures semantic similarity by calculating the cosine similarity between vector representations of texts.

F.2 Prompt Similarity Metrics

- **Levenshtein**: Measures the minimum number of single-character edits (insertions, deletions, or substitutions) needed to transform one string into another. It captures fine-grained textual modifications, making it particularly useful for shorter texts.
- **Longest common subsequence**: Identifies the longest sequence of characters that appear in both strings in the same order, though not necessarily contiguously. This metric uncovers shared structural elements that remain even after rearranging or slightly altering the text.
- **Jaccard Index**: Calculates the similarity and diversity of sample sets, ideal for assessing the overlap of token sets in prompts.
- **Cosine similarity using embeddings**: Offers a measure of semantic similarity by evaluating the cosine angle between the embeddings of the prompts, capturing nuances beyond explicit token use.