USENIX Security ’25 Artifact Appendix: 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

A Artifact Appendix

A.1 Abstract

This artifact provides the source code and framework for a prompt obfuscation method that safeguards the intellectual property of system prompts by creating functionally equivalent, unintelligible representations. It implements two techniques: a ’hard’ method in the discrete token space and a ’soft’ method in the continuous embedding space. The artifact includes code to evaluate obfuscated prompts, run deobfuscation attacks (e.g., prompt extraction, projection, and fluency optimization), compare against a LoRA finetuning baseline, and automatically download all required public models and datasets.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

The provided artifact poses no security or privacy risks to the evaluator. The experiments exclusively use publicly available datasets and models downloaded from HuggingFace. Furthermore, the artifact does not perform any destructive file operations outside of its own designated output directories.

A.2.2 How to access

The artifact is available on Zenodo: https://doi.org/10.5281/zenodo.15601914.

A.2.3 Hardware dependencies

A GPU is required for running this artifact. We recommend using an environment with an NVIDIA GeForce RTX 3090 or more powerful GPU. Around 35GB of disk space is needed to store the main language model, datasets, and other supporting models downloaded from Hugging Face. All experiments reported in the paper were conducted on a single NVIDIA A100 GPU with 40 GB of VRAM.

A.2.4 Software dependencies

All experiments were conducted in a conda environment with Python 3.12.7 on Ubuntu 22.04.4, though any operating system supporting Python works. All necessary dependencies are listed in `requirements.txt` in the artifact, with detailed installation instructions outlined in Section A.3.1.

A.2.5 Benchmarks

The artifact relies on several publicly available models, datasets, and a set of curated prompts, all of which are essential for reproducing the experiments reported in the paper.

- **Models**: The primary model used is Llama 3.1-8B, which is automatically downloaded from Huggingface. Please note that this is a gated model requiring explicit agreement to Meta’s Community License. Supporting models for evaluation are also downloaded automatically from the Hugging Face Hub.
- **Datasets**: Our experiments utilize four public datasets: TruthfulQA, TriviaQA, CNN/DailyMail, and Samsum. The provided data loaders automatically download and process the correct versions of these datasets from the Hugging Face Hub.
- **Prompt Extraction Queries**: The set of 105 prompt extraction queries from Zhang et al. used for the attack evaluation in Section 6.1 is included directly within the artifact at `extraction_prompts/gpt4_generated.json`.

A.3 Set-up

A.3.1 Installation

First, download the artifact from Zenodo and extract it. We recommend creating a new virtual environment with Python 3.12.7 (e.g., using conda):

```
conda create -n prompt_obfuscation python=3.12.7
conda activate prompt_obfuscation
```

Next, install all necessary dependencies:

```
pip install -r requirements.txt
```

After account creation and access is granted to Llama 3.1, log in using the following command:

```
huggingface-cli login
```

Note: All models and datasets are downloaded to ~/.cache/. To change the default path, you can run:

```
export HF_HOME="/new/path"
export SENTENCE_TRANSFORMERS_HOME="/new/path"
export NLTK_DATA="/new/path"
```

A.3.2 Basic Test

We provide the script basic_test.py to perform a lightweight functionality test that checks the correct setup of all core software components and hardware dependencies. To run the test, execute the following command from the project's root directory.

```
python3 basic_test.py
```

The script will print detailed progress for each step. A successful execution concludes with the final confirmation message: "All Basic Tests Passed Successfully! The environment is set up correctly and all core components are functional."

A.4 Evaluation workflow

A.4.1 Major Claims

Our paper makes the following major claims:

(C1): Hard prompt obfuscation preserves the utility of the original prompt but is susceptible to information leakage. This is proven by experiments (E1) and (E2), described in Section 5.1, with results in Tables 1 and 2.

(C2): Soft prompt obfuscation effectively preserves the functionality of conventional system prompts across diverse tasks and in a real-world scenario. This is proven by experiments (E3) and (E4), described in Sections 5.2 and 5.3, with results in Tables 4, 5, 11, and 12.

(C3): Prompt obfuscation is a competitive alternative to fine-tuning, achieving similar utility with significantly lower storage overhead. This is proven by experiment (E5), described in Section 5.4, with results in Table 6.

(C4): Obfuscated soft prompts are robust against prompt extraction and direct projection attacks. This is proven by experiments (E6) and (E7), described in Sections 6.1 and 6.2, with results in Tables 7 and 8.

(C5): Under a white-box threat model, a fluency optimization attack can partially recover human-readable fragments. This is proven by experiments (E8) and (E9), described in Section 6.3, with results in Tables 9 and 10.

A.4.2 Experiments

The following experiments provide the steps to reproduce the evidence for the major claims. Each is demonstrated with a representative example. To facilitate quick verification of computationally intensive experiments, precomputed results for specific styles are provided in the artifact's precomputed_results/ directory. All other results reported in the paper can be reproduced by running the provided scripts with different parameters (e.g., --style, --dataset_name). For a comprehensive list of arguments, please refer to the main README.md. A complete list of available style identifiers is defined in src/style_prompts.py and listed in Appendix A of the paper. Note: The estimated compute times and VRAM requirements may vary slightly depending on the specific style and dataset chosen.

(E1): [Hard Prompt Obfuscation Utility] [12 compute-hours, 23GB VRAM]: This experiment evaluates obfuscated hard prompt utility against baselines.

How-to: We provide scripts for "Full", "Style", and "Task" scenarios. The "Task" scenario script is run once, while others are run for each style.

Execution: To reproduce the "Full" scenario for the ‘pirate‘ style:

```
./bash_scripts/hard_prompt_obfuscation_full.sh
    --style pirate
```

For the "Style" and "Task" scenarios, run hard_prompt_obfuscation_style.sh --style <style> and hard_prompt_obfuscation_task.sh, respectively.

Results: Outputs are in their respective subdirectories (e.g., results/hard_pirate_full/). The key files to verify against Table 1 are:

- best_candidate_scores.json (’obf’ column)
- blank_output_scores.json (’blank’ column)
- original_output_scores.json (’original’ column)

(E2): [Hard Prompt Information Leakage] [30 compute-seconds, 1.5GB VRAM]: This experiment quantifies information leakage from the best hard prompt.

How-to: This script requires the results directory from a corresponding (E1) experiment.

Execution: To evaluate leakage for the ’Full’ ‘pirate‘ scenario from (E1):

```
./bash_scripts/compare_hard_prompt_leakage.sh
    --results_dir results/hard_pirate_full
```

Results: The key files to verify against Table 2 are obf_sys_prompt_scores.json (’obf’ column) and rand_sys_prompt_scores.json (’rand’ column).

(E3): [Soft Prompt Obfuscation Utility] [4 compute-hours, 25GB VRAM]: This experiment evaluates obfuscated soft prompt utility against baselines.

How-to: Similar to (E1), we provide scripts for "Full",

"Style", and "Task" scenarios, which take `--style` and `--dataset_name` arguments (except for the task script, which only needs `--dataset_name`).
Execution: To reproduce the "Full" scenario for the ‘pirate‘ style on ‘truthfulqa‘:

```
./bash_scripts/soft_prompt_obfuscation_full.sh
    --style pirate --dataset_name truthfulqa
```

Results: The output directory (e.g., `results/soft_pirate_truthfulqa_full/`) will contain the same set of score files as in (E1), used to verify results in Tables 4, 11, and 12.

(E4): [Case Study] [4 compute-hours, 25GB VRAM]: This experiment demonstrates soft prompt obfuscation on a real-world prompt.
How-to: The experiment is encapsulated in a single script that requires no command-line arguments.
Execution: Execute the script:

```
./bash_scripts/soft_prompt_obfuscation_case_study.sh
```

Results: The output directory `results/soft_manga_miko_case_study/` contains score files (same as (E1)) to verify the results in Table 5.

(E5): [Comparison to LoRA Finetuning] [1 compute-hour, 13GB VRAM]: This experiment fine-tunes a LoRA adapter as a baseline.
How-to: We provide scripts for "Full", "Style", and "Task" scenarios, mirroring the structure of (E3).
Execution: To reproduce the "Full" scenario for ‘pirate‘ on ‘truthfulqa‘:

```
./bash_scripts/run_finetuning_full.sh --style
    pirate --dataset_name truthfulqa
```

Results: The output directory (e.g., `results/finetuning_pirate_truthfulqa_full/`) contains `best_adapter_scores.json`. These scores correspond to the ’finetune’ column in Table 6 and can be compared with the ’obf’ scores from (E3).

(E6): [Robustness Against Prompt Extraction] [15 compute-minutes, 14GB VRAM]: This experiment evaluates robustness against prompt extraction attacks.
How-to: This script requires the results directory from a soft prompt obfuscation run (E3).
Execution: To test the ‘pirate‘ style on ‘truthfulqa‘ from (E3):

```
./bash_scripts/run_prompt_extraction_attack.sh
    --results_dir
    results/soft_pirate_truthfulqa_full
```

Results: The console output reports the number of successful extractions for both obfuscated and conventional prompts, corresponding to the data in Table 7.

(E7): [Robustness Against Projection Attack] [2 compute-minutes, 10GB VRAM]: This experiment evaluates deobfuscation via direct projection to the token space.
How-to: This script requires the results directory from a soft prompt obfuscation run (E3).
Execution: To test the ‘pirate‘ style on ‘truthfulqa‘ from (E3):

```
./bash_scripts/run_projection_attack.sh
    --results_dir
    results/soft_pirate_truthfulqa_full
```

Results: The key files to verify against Table 8 are `euclidean_sys_prompt_scores.json`, `cosine_sys_prompt_scores.json`, and `random_sys_prompt_scores.json`.

(E8): [Soft Fluency Deobfuscation Attack] [3.5 compute-hours, 30GB VRAM]: This experiment runs the "soft" fluency optimization attack.
How-to: The script requires the results directory from a soft prompt obfuscation run (E3).
Execution: Using the ‘pirate‘ style on ‘truthfulqa‘ from (E3):

```
./bash_scripts/run_fluency_deobfuscation_soft.sh
    --results_dir
    results/soft_pirate_truthfulqa_full
```

Results: The output directory will contain the similarity scores for the deobfuscated prompt and the random baseline. The key files to verify against Table 9 are `best_deobf_sys_prompt_soft_scores.json` (’deobf’ column) and `random_sys_prompt_scores.json` (’rand’ column).

(E9): [Hard Fluency Deobfuscation Attack] [7 compute-hours, 31GB VRAM]: This experiment runs the "hard" fluency optimization attack.
How-to: The script requires the results directory from a soft prompt obfuscation run (E3).
Execution: Using the ‘pirate‘ style on ‘truthfulqa‘ from (E3):

```
./bash_scripts/run_fluency_deobfuscation_hard.sh
    --results_dir
    results/soft_pirate_truthfulqa_full
```

Results: The output directory will contain the similarity scores for the deobfuscated prompt and the random baseline. The key files to verify against Table 10 are `best_deobf_sys_prompt_hard_scores.json` (’deobf’ column) and `random_sys_prompt_scores.json` (’rand’ column).

A.5 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at `https://secartifacts.github.io/usenixsec2025/`.