USENIX Security '25 Artifact Appendix: Generated Data with Fake Privacy: Hidden Dangers of Fine-tuning Large Language Models on Generated Data

Atilla Akkus,[1*] Masoud Poorghaffar Aghdam, [1*] Mingjie Li,[2*] Junjie Chu,[2]
Michael Backes,[2] Yang Zhang,[2] Sinem Sav[1]

[1]*Bilkent University*
[2]*CISPA Helmholtz Center for Information Security*

April 15, 2025

A Artifact Appendix

This artifact appendix provides a comprehensive guide to evaluating the *functionality* of our artifact that includes code, datasets, and evaluation scripts.

A.1 Abstract

This artifact enables the assessment of privacy risks when fine-tuning large language models (LLMs) with generated data. It includes the evaluation of Personal Information Identifier (PII) leakage and Membership Inference Attacks (MIAs), across two fine-tuning strategies: supervised fine-tuning (SFT) with unstructured generated data and self-instruct tuning. The artifact enables researchers to analyse the extent to which generated data may still pose privacy risks. The complete repository and setup instructions are included in this document.

A.2 Description & Requirements

The software environment is defined by the dependencies listed in the requirements.txt file, which includes all necessary Python packages. A pre-configured Docker image, *pytorch/pytorch:2.5.1-cuda12.4-cudnn9-runtime*, is recommended to ensure compatibility and reproducibility. After setting up the environment, the dependencies can be installed by running: `pip install -r requirements.txt`

A.2.1 Security, privacy, and ethical concerns

We used publicly available data without human participants. Our study examines privacy risks, such as PII extraction and MIA, in fine-tuning LLMs. Findings were responsibly disclosed, and fine-tuned checkpoints were not released due to privacy concerns.

*These authors contributed equally to this work.

A.2.2 How to access

The artifact is available at: https://doi.org/10.5281/zenodo.14732690.

A.2.3 Hardware dependencies

The experiments were conducted on hardware with a CUDA-compatible NVIDIA GPU ($\geq$ 40GB VRAM recommended for training), an x86_64 architecture CPU, at least 32GB of RAM, and 100GB of disk space for storing datasets and intermediate outputs.

A.2.4 Software dependencies

The environment requires Python 3.10+ and packages from 'requirements.txt'. It is suggested to use *'pytorch/pytorch:2.5.1-cuda12.4-cudnn9-runtime'* docker image.

A.2.5 Benchmarks

We use the Enron subset of the Pile dataset in our paper. The processed version, as used in our study, is included in the repository under the 'data' directory.

A.3 Set-up

A.3.1 Installation

Clone the Repository: Begin by cloning the Git repository to your local machine:

```
git clone [repository URL]
cd [repository directory]
```

Pull the Docker Image:

```
docker pull pytorch/pytorch:2.5.1-
    cuda12.4-cudnn9-runtime
```

Run the Docker Container: Start the Docker container with GPU support and mount the repository:

```
docker run -it --gpus all -v
    \$(pwd):/workspace
pytorch/pytorch:2.5.1-cuda12.4
    -cudnn9-runtime
```

Install Dependencies: Install the required Python packages using the provided `requirements.txt` file:

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

A.3.2 Basic Test

Fine-Tuning Models with PEFT

Fine-tune models using the `run_peft.py` script. Example command:

```
torchrun -nproc_per_node=4 run_peft.py -model
EleutherAI/pythia-1.4b
```

Merging Adapters with PEFT

After fine-tuning, use the `merge_peft.py` script to merge the adapter results with the base model:

```
python merge_peft.py -model ./_fine_tuned_model
-save_dir ./merged_model_output
```

To check the functionality of this script, please refer to the checkpoint provided in https://drive.google.com/file/d/1H8_0p5bW8Ur5ZXZHrvOHYsITTzD52Fys/view.

Measuring PII Attack Success

Evaluate PII attack success using the `measure_purified.py` script. Example command:

```
python measure_purified.py -model
EleutherAI/pythia-1.4b -output_dir ./results
```

Evaluating Models with Purified Workflow

Automate merging and PII attack evaluation using the `evaluate_purified.py` script. Example command:

```
python evaluate_purified.py -model ./pythia-1.4b
-temp ./tempMerged
```

Measuring Model Perplexity

Calculate model perplexity using the `perplexity/perplexity.py` script. Example command:

```
python perplexity/perplexity.py -dir ./pythia-1.4b
```

Generating Continuation Data

Generate continuation data using the `data/continuation/generate.py` script. Example command:

```
python data/continuation/generate.py -model
EleutherAI/pythia-1.4b
```

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): *Our system includes code for generating synthetic datasets tailored for fine-tuning LLMs.*

(C2): *Our system offers the essential code for fine-tuning models using various Parameter-Efficient Fine-Tuning (PEFT) techniques.*

(C3): *Our system provides the essential code for evaluating models with a streamlined workflow.*

(C3): *Our system provides the essential code for evaluating models with purified workflow.*

(C4): *Our system provides the essential code for measuring the success of PII attacks.*

(C5): *Our system provides the essential code for measuring the success of MIA attacks.*

A.4.2 Experiments

Each experiment listed below directly corresponds to a major claim in Section A.4.1 in sequential order. In addition, several optional parameters are available for each experiment, which are detailed in our GitHub repository.

(E1): *Generating Dataset [1 compute-hour + 1GB disk]*: It generates a synthetic dataset as articulated in Section 3.1.1.
How to: Use the script *data/continuation/generate.py* to generate the needed dataset using a specified model and dataset.
Execution: To run the script, the –data_set parameter must be specified, as it defines the path to the chosen dataset. By default this parameters uses "data/enron.jsonl".
Results: Upon running the script, a new directory will be created in the used working directory with a name similar to the specified model. If the script executes and completes successfully, an Arrow file will be generated within this directory. This file contains the generated data points.

(E2): *Fine-Tuning Models with PEFT [4 compute-hours + 10GB disk]*: This step fine-tunes models using PEFT.
How to: Use the script *run_peft.py* to fine-tune models with PEFT.
Execution: To run the script, the –model parameter must be specified, as it determines the model to be instruction-tuned. By default, this parameter uses "EleutherAI/pythia-1.4b". Additional key parameters include:

- **–input**: Path to the training dataset (Default: "data/enron.jsonl").
- **–peft_type**: PEFT type (Options: lora, dora, pissa; Default: "lora").

Results: Once the script completes, the adapter of fine-tuned model will be stored in the output directory. This adapter needs to be merged with the original model prior to evaluation.

(E3): *Evaluating Models with Purified Workflow [3 compute-hours + 5GB disk]*: This step evaluates models by sequentially running *merge_peft.py* and *measure_purified.py*.
How to: Use the script *evaluate_purified.py* to automate the merging and evaluation of all models in a specified directory.
Execution: To run the script, the –model parameter must be specified, as it defines the path to the model being evaluated. By default, this parameter uses "./pythia-1.4b".

Results: Upon successful execution, a *results_purified.jsonl* file will be generated in each model's directory. This file contains key evaluation metrics, including matches, model name, temperature, and top-k values.

(E4): *Measuring PII Attack Success [1 compute-hour + 2GB disk]*: This step evaluates the success rate of Personally Identifiable Information (PII) attacks.

How to: Use the script *measure_purified.py* to assess the number of successful PII attacks.

Execution: To run the script, the **model** and **output_dir** parameters must be specified, as they define the model being evaluated and the directory where results will be saved.

Results: Upon successful execution, a *results_purified.jsonl* file will be generated in each model's directory. This file contains key evaluation metrics, including matches, model name, temperature, and top-k values.

(E5): *Conducting MIA*: This step conducts MIA in the target models.

How to: Download mimir repository. Then run `run.py --config configs/your_config.json` with your customized configuration file. You can refer to the original repository for reference on configuration details.

Results: Upon successful execution, an output directory will be created, and the summary results will be printed on the console. The directory will contain example plots and data produced during MIA.

A.5 Notes on Reusability

This artifact is adaptable for studying privacy risks in fine-tuning LLMs across different models and datasets. It supports modifications to fine-tuning parameters, attack methods, and evaluation metrics, enabling broader analysis. The modular design allows integration with various models, including instruction tuning and domain-specific adaptations. A pre-configured Docker setup ensures stability and easy extension for future research on privacy-preserving LLM training.

A.6 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/.