USENIX Security '25 Artifact Appendix: SoK: Automated TTP Extraction from CTI Reports – Are We There Yet?

Marvin Büchel[†,1], Tommaso Paladini[†,2,6], Stefano Longari[2], Michele Carminati[2], Stefano Zanero[2], Hodaya Binyamini[4], Gal Engelberg[4], Dan Klein[4], Giancarlo Guizzardi[3], Marco Caselli[5], Andrea Continella[3], Maarten van Steen[3], Andreas Peter[1], and Thijs van Ede[3]

[1]Carl von Ossietzky Universität Oldenburg, [2]Politecnico di Milano, [3]University of Twente, [4]Accenture Labs, [5]Siemens AG, [6]NEC Laboratories Europe

A Artifact Appendix

A.1 Abstract

This Artifact Appendix covers the experiments and the resulting main conclusions from our Empirical Analysis part of the SoK paper. As already mentioned in the paper, it is not about the absolute values, but the relative comparisons among each other, especially in the approaches itself. All experiments are for evidence only and are not intended to present a new high-performance prototype. For this reason, we aim to evaluate the relative trends in the approaches. Our artifact is structured in three main *sub-projects*, each requiring their own setup. Each sub-project (`NER`, `classification`, `generation`) can be found in a folder inside the main directory of the artifact, and contain the code related to the corresponding systematization experimental section. `NER`: Contains the code related to the NER approaches (Sections 2.2, 4); `classification`: Contains the code related to the Classification approaches (Sections 2.3, 5); `generation`: Contains the code related to the Generation approaches (Sections 2.4, 6).

A.2 Description & Requirements

This artifact has been executed on a server mounting two NVIDIA L40 GPUs with 48GB VRAM, 1024GB RAM, and two AMD EPYC 7763 CPUs. The server runs Ubuntu 22.04.04 (jammy), with CUDA version 12.4.

A.2.1 Security, privacy, and ethical concerns

Our artifact strictly contains regular Machine Learning preprocessing, training, and testing routines to build NLP models that extract TTPs from CTI reports. They require no destructive steps on the original system, nor disabling of security mechanisms during execution. Therefore, we believe that no specific risks for researchers, practitioners, and evaluators should be encountered.

A.2.2 How to access

Access to our artifact is provided via Zenodo Repository: `https://zenodo.org/records/15608555`. This repository is mirrored by our GitHub repository[1] (code), and our Hugging Face repository[2] (models).

A.2.3 Hardware dependencies

We recommend having at least 48GB of GPU VRAM and 300GB of storage.

A.2.4 Software dependencies

Our artifact partly depends on the DarkBERT[3] model, which requires an access token key by contacting the authors of the paper to be downloaded.

A.2.5 Benchmarks

Our artifacts require the following datasets: TRAM2, and AnnoCTR. Both are provided directly in our repository, in the `dataset` folder.

A.3 Set-up

Our artifact requires Docker[4] installed on the target system. We have tested our artifact with Docker version 25.0.3, build 4debf41. In addition, fine-tuned models and pre-calculated embeddings should be directly downloaded from our Hugging Face repository in Section A.2.2.

[†]Authors contributed equally to this work.

[1]`https://github.com/MarvinBuechel/SoK_CTI_TTP`

[2]`https://doi.org/10.57967/hf/5736`

[3]Jin, Youngjin, et al. "Darkbert: A language model for the dark side of the internet." arXiv preprint arXiv:2305.08596 (2023).

[4]`https://www.docker.com/`

A.3.1 Installation

First, install **Docker** following the instructions reported on the tool website[5].

Second, clone our repository using the link provided in Section A.2.2.

NER. Navigate to the `CTISOK/NER` directory. To install all the required resources, we recommend building the Dockerfile by running: "`$ docker build . -t ner`"

Classification. Make sure to be located in the `CTISOK/classification` folder. Download from our Hugging Face repository the directories `fine_tuned.tgz`, containing our BERT-based fine-tuned models, `local.tgz`, containing some pre-trained models, and `nvidia_embeddings.tgz`, containing pre-calculated embeddings. Decompress them with "`$ tar zxvf archiveName`". The first two folders (`fine_tuned`, `local`) must be located in the root of this sub-project. The content of the second folder must be moved inside the `datasets` directory (after decompressing it you should already see this content inside the folder, otherwise you can run "`$ mv nvidia*.pickle datasets/`"). At last, you should run the following command to build the Docker image: "`$ docker build --build-arg RUN_DEVICE=YOUR_DEVICE -t sok-classification .`", where `YOUR_DEVICE` should point to a GPU (e.g., `cuda`, `cuda:0`, etc.).

Generation. Navigate to the `CTISOK/generation` directory. Build all docker containers and run all experiments by: "`$ docker compose up --build -d`". This automatically downloads the required base models (Meta LLama3.1 8B and GTE-Qwen2-7B for RAG) and involves the installation of an ollama server[6] for the RAG model and a Python environment that starts the Generation experiments automatically.

As runtime inference environment for the generative model, we used the Python library `unsloth`[7] in version 2024.12.4 with the `transformers`[8] library version 4.44.2. Unfortunately, we could no longer reproduce this combination of dependencies, which is why we switched to the newer versions `unsloth==2025.03.19` and `transformers==4.51.1`. This results in the behavior as described in chapter 6.2 - Prompt based, that the LLM suffers from instability and sometimes gets into a loop in which all MITRE IDs are output, which increases the recall, but reduces the precision and accordingly changes the F1 score.

A.3.2 Basic Test

NER. To run the minimal working example, execute the following line: "`$ docker run ner`". This script will test the Docker image by executing the ablation study presented in Table 5 of our work.

Classification. You can run a basic test by executing: "`docker run --gpus all -it sok-classification`". This script checks that models and data are correctly loaded, and perfors a small test by producing part of the results shown in Table 6 of our work.

Generation. To run a minimal test, navigate to the generation folder and execute: "`docker build -t minimal -f Dockerfile.minimal .`" followed by: "`docker run --volume ./experiments:/workspace/finetuning/experiments --gpus all minimal`" This loads a 4-bit quantized model that repeats the "Raw" experiment from Table 9 on the AnnoCTR dataset in "`./experiments/minimal.csv`". Although the quantized model is significantly smaller, it still requires approximately 8GB VRAM.

A.4 Evaluation workflow

Our evaluation workflow is subdivided according to the three sub-projects (NER, Classification, Generation). We also present a subsection for claims that require results from all three approaches (All). For convenience, here, experiments are named after the tables and figures of the paper that show these results.

A.4.1 Major Claims

NER (N).

(N.C1): *Individual pipeline components contribute to improving the recall of NER approaches at the cost of false detections.*

Classification (C).

(C.C1): *Labeled classification approaches achieve higher performances than unlabeled approaches.*

(C.C2): *CTI-specific classification models do not consistently outperform generic models.*

(C.C3): *Data augmentation provides a small margin of improvement for labeled approaches.*

Generation (G).

(G.C1): *Finetuning is the best performing method for generative LLMs and outperforms in-context learning methods.*

(G.C2): *Data augmentation does not replace high-quality datasets.*

(G.C3): *RAG on document-level granularity performs worse than at sentence level.*

All (A).

(A.C1): *Traditional NER approaches outperform both Classification and Generation when no prior information about test data is available and over increasing amount of TTP labels. In general, performances drop with increasing amount of TTP labels.*

[5] https://docs.docker.com/engine/install/
[6] https://github.com/ollama/ollama
[7] https://github.com/unslothai/unsloth
[8] https://github.com/huggingface/transformers

A.4.2 Experiments

(N.E1): [Table 5] [30 human-minutes + 30 compute-minutes]: creates all NER pipelines and runs the ablation study on the TRAM2 and AnnoCTR datasets (presented on Table 5 of our paper).
Execution: Run: "`$ docker run -it ner bash`" in `CTISOK/NER` directory. Run "`$ ./ablation.sh`" inside the container.
Results: Results of all pipelines are stored in the `/NER/experiments/results` directory. They should be compared with results provided in Table 5 of our paper.

(C.E1): [Table 6 and 7] [30 human-minutes + 2 compute-hours]: run the fine-tuned models (labeled approach), and the unlabeled models (with pre-calculated decision thresholds) on the TRAM2 and AnnoCTR datasets. These results are shown in Table 6 and 7 of our paper.
Preparation: Open directory "`CTISOK/classification`". Make sure you have non-empty:
"`fine_tuned/tram_swipe`",
"`fine_tuned/bosch_swipe`", and
"`configs/all_sentence_similarity.json`". Make sure you have "`nvidia-*embeddings.pickle`" inside the "`datasets`" folder.
Execution: First, you need to enter the shell of the docker container. You can do so by running: "`$ docker run --gpus all -it sok-classification bash`". Then, you should run the following commands:
"`$ ./artifact_eval/gen_table_6_annoctr.sh`"
"`$ ./artifact_eval/gen_table_6_tram.sh`"
"`$ ./artifact_eval/gen_table_7.sh`"
"`$ ./artifact_eval/gen_table_7_nvidia.sh`"
Results: Compare Table 6 with the output of the first two scripts. Compare Table 7 with the output of the last two scripts.

(C.E2): See **C.E1**. Such procedure generates the same tables of our original work that support this claim (Tables 6, 7).

(C.E3): [Table 8] [5 human-minutes + 5 compute-minutes]: run the models fine-tuned on augmented datasets and test them on the TRAM2 dataset.
Preparation: Make sure you have the folder named "fine_tuned/data_augmentation" (non-empty).
Execution: Enter the shell of the docker container. Run: "`$ docker run --gpus all -it sok-classification bash`". Then, you should run the following command:
"`$ artifact_eval/gen_table_8.sh`"
Results: Compare the results printed by the script with Table 8 of our paper.

(G.E1): [Table 9] [5 human-minutes + 24 compute-hours]: Runs the model on TRAM2 and AnnoCTR datasets with all the methods presented.
Execution: Run: "`$ docker compose up --build -d`" in the `CTISOK/generation` directory to also execute **G.E2** and **G.E3** experiments.
Results: Results are located at `CTISOK/generation/experiments/table9_methods.csv`.

(G.E2): [Table 10] Trains the base model on the TRAM2 augmented dataset and evaluates it on the TRAM2 test set.
Execution: See **G.E1**.
Results: Results are located at `CTISOK/generation/experiments/table10_augmented_data.csv`.

(G.E3): [Table 11] Runs in-context learning methods on a document-level granularity on TRAM2 and AnnoCTR datasets.
Execution: See **G.E1**.
Results: Results are located at `CTISOK/generation/experiments/table11_document_level.csv`.

(A.E1): [Figure 4] [30 human-minutes + 1 compute-hour]: in this experiment, we produce the data required for obtaining Figure 4 of our paper, which shows the comparison between all approaches on the TRAM2 dataset. Following the color legend proposed in Figure 4, in blue, we explain the parts related to NER, in red, the parts related to classification, in green, the parts related to generation.
Execution (NER): Open directory "NER". Enter the shell of the docker container. Run "`$ docker run -it ner bash`". Then, run the following command: "`$ ./open_set_scenario.sh`"
Results (NER): Results are located in "`results/OpenSet/*.txt`".
Preparation (Classification): Make sure you have "configs/open_set_scenario.json".
Execution (Classification): Open directory "classification". Enter the shell of the docker container. Run: "`$ docker run --gpus all -it sok-classification bash`". Then, run the following command:
"`$ ./artifact_eval/gen_figure_4.sh`"
Results (Classification): The results (F1, Precision, Recall) printed by the script should generate a figure similar to the one presented in Figure 4.
Execution (Generation): Run: "`$ docker compose up --build -d`" in `CTISOK/generation` folder.
Results (Generation): The results are located at `CTISOK/generation/experiments/figure4_open_set.csv`

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/.