
# USENIX Security '25 Artifact Appendix: Vulnerability of Text-Matching in ML/AI Conference Reviewer Assignments to Collusions

Jhih-Yi (Janet) Hsieh, Aditi Raghunathan, Nihar B. Shah
School of Computer Science, Carnegie Mellon University
jhihyih@alumni.cmu.edu, {aditirag, nihars}@andrew.cmu.edu

## A Artifact Appendix

### A.1 Abstract

In addition to releasing all adversarial abstracts generated in this work, our artifacts include the code, datasets, and LLM prompts necessary for researchers to reproduce the *fully automatic* mode attack results that support the paper's main claims. In this artifact appendix, we describe in detail how each of our main results can be reproduced and verified.

### A.2 Description & Requirements

#### A.2.1 Security, privacy, and ethical concerns

There is no risk for evaluators while executing our artifacts. However, it is important to remember that any successful attack pertains to the abstract and/or reviewer profile modified in this experiment and not to the original abstract or profile in the conference. Finally, do not use this artifact to produce adversarial abstracts for your submissions. Safeguards against this attack are implemented at major AI/ML conferences.

#### A.2.2 How to access

Our research artifacts are available on Zenodo `https://doi.org/10.5281/zenodo.15588237`.

#### A.2.3 Hardware dependencies

We have tested the reproduction instructions in this document on CPU-only devices as well as GPU-enabled. A GPU can provide substantial speed-ups. The hardware needs to be compatible with the PyTorch deep learning framework. The minimal hardware the artifacts have been tested on has an 8 CPU cores with 16 GB RAM, but having more is helpful. If you do not have GPU access but wish to gain one, we have tested on and recommend AWS (instance type: g4dn.xlarge, image: Deep Learning OSS Nvidia Driver AMI GPU PyTorch 2.7 (Ubuntu 22.04)). Your machine should have at least 20 GB of free memory for the datasets, virtual environment, and models.

#### A.2.4 Software dependencies

We package our environment using Conda virtual environments. We recommend installing Miniconda, which is a minimal installer of Anaconda. Please follow the instructions on their website: `https://www.anaconda.com/docs/getting-started/miniconda/install`.

Our code makes calls to the OpenAI API (paid service), so an API key with available credits is needed.

#### A.2.5 Benchmarks

None.

### A.3 Set-up

#### A.3.1 Installation

First, download the `dataset.zip` file from Zenodo (`https://doi.org/10.5281/zenodo.15588237`) and extract the files locally. There should be a `datasets/` directory containing both NeurIPS 2022 and 2023 datasets. You may rename the outer directory `datasets/`, but please keep everything else the same way.

Now find `reviewer_assignments_vulnerability.zip` from the same Zenodo link and extract it somewhere outside of the datasets directory. This directory contains the code, adversarial abstract examples, LLM prompts, and other scripts. Change into the `reviewer_assignments_vulnerability` directory, and create the virtual environment using the following:

```
$ conda env create -f environment.yml
```

Activate the environment:

```
$ conda activate attack
```

Set the OpenAI API key environment variable.

```
$ export OPENAI_API_KEY=your_key
```

### A.3.2 Basic Test

The basic test below simulates the proposed collusion attack between one paper-reviewer evaluation pair. Set the `--datasets_dir` flag to the directory path that contains both NeurIPS 2022 and 2023 datasets (for example `"../datasets"`). Run the command below from the code repository. If you have access to a GPU, first test if GPU is availabe for PyTorch (try `torch.cuda.is_available()` in Python console) and then add the flag `--device=cuda` to the command below. *You must specify the device or it will default to running on CPU only.*

```
(attack) $ python attack/main.py \
--datasets_dir=path/to/datasets/directory \
--end=1
# Optional arguemnt: --device=cuda
```

After the run successfully finish, you should find an output directory `out/yyyy-mm-dd/hh-mm-ss` under the code repository. It will contain a sub-directory called `attack_0`, under which you will find the attack logs and information. Open `attack_info.json` and you will find the attack metadata and results, including the adversarial reviewer archive and the modified abstract.

To evaluate the run output, you should use the following script. However, for the basic test, you may see NaN and errors since there is only one sample (for example, variance and confidence intervals are meaningless).

```
(attack) $ python scripts/run_result.py \
--run_dir=out/yyyy-mm-dd/hh-mm-ss
```

## A.4 Evaluation workflow

This section outlines the necessary procedures and experiments required to verify artifact functions and repoducibility of the major claims in the paper.

### A.4.1 Major Claims

Here we enumerate the major claims made in the paper.

**(C1):** The proposed attack procedure successfully manipulates reviewer assignment. This is proven by experiment (E1) described in Section 5.3 whose results are reported in Table 2.

**(C2):** Policies that require reviewers to have more papers in their archives can reduce the effectiveness of the proposed attack. This is proven by experiment (E2) described in Section 5.4 whose results are illustrated in Figure 4.

**(C3):** To combine the similarity scores between a submitted paper and a reviewer's multiple papers in their archives, using average pooling instead of max pooling can help reduce the impact of targeted manipulations, making the proposed attack less successful under average pooling. This is proven by experiment (E3) described in Section 5.5 whose results are reported in Table 4.

**(C4):** Considering NeurIPS 2022 as a "past" conference whose data is publicly available to an attacker, and NeurIPS 2023 as the "current" conference whose data is unavailable, we find that the attack performance on NeurIPS 2022 is reflective of the performance in 2023. This is proven by experiment (E4) described in Section 5.6 whose results are illustrated in Figure 5.

### A.4.2 Experiments

For each major claim above, we provide the instructions to reproduce the experiment in our paper that supports the claim. The instructions here reproduce these experiments at a smaller scale in terms of both sample size and the number of scenarios, but we believe thst they are representative and sufficient to support the main claims. Due to the stochastic nature of LLM-generated abstract attacks and the low sample sizes, we do not expect the result numbers to be exactly the same as reported in the paper, but we expect them to still support each claim.

**(E1):** *[Attack Success Rates] [20 human-minutes + 3 compute-hour]: run the attack procedure on paper-reviewer pairs with natural rankings 101 and 1001.*

**Preparation:** Make sure your OpenAI API key has sufficient credit. We estimate about 5 to 10 dollars for this experiment.

**Execution:** Use the following command to execute the attack procedure for 20 samples. You will need to run this command twice (one after the other has completed), with different input argument to `--samples_file`, for natural rankings of 101 and 1001 (replace the `attacks_rank101.jsonl` file with `attacks_rank1001.jsonl` below). Two runs take about 3 hours total on CPU and 2 hours on GPU.

```
(attack) $ python attack/main.py \
--datasets_dir=path/to/datasets/directory \
--samples_file="evaluation_samples/\
attacks_rank101.jsonl" \
--end=20
# Optional arguemnt: --device=cuda
```

**Results:** Each run (each time `attack/main.py` is executed) will have its own output directory, named by the timestamp of when the run started. Use the evaluation script introduced at the end of Appendix A.3.2 to evaluate the attack results of each run. Compare the attack success rates and manipulated rankings statistics with the first and last rows of Table 2 in Section 5.3. Since the sample size of 20 here is much lower than in Table 2, higher variance and wider confidence intervals are expected.

**(E2):** *[Reviewer Archive Length] [15 human-minutes + 4 compute-hour]: run the attack procedure, but reviewers*

*have to keep 10 papers (instead of 1) in their archives.*
**Preparation:** Make sure your OpenAI API key has sufficient credit. We estimate about 10 dollars for this experiment.
**Execution:** Use the following command to execute the attack procedure for 20 samples. This will take about 4 hours total on CPU and about 1.5 hours on GPU.

```
(attack) $ python attack/main.py \
--datasets_dir=path/to/datasets/directory \
--samples_file="evaluation_samples/\
arclen_attacks_rank101.jsonl" \
--num_papers_to_keep=10 --end=20
# Optional arguemnt: --device=cuda
```

**Results:** Again, an output directory named by the timestamp of when the run started will be produced. Use the evaluation script introduced at the end of Appendix A.3.2 to evaluate the attack result. Compare with the right-most data point in Figure 4. You should see that the attack success rates are low when reviewers keep 10 papers in their archive.

**(E3):** *[Average vs Max] [30 human-minutes + 6 compute-hour]: run the attack procedure under average (mean) versus max pooling methods to combine the similarities to each of the reviewer's papers (when reviewers must keep all their past papers).*
**Preparation:** Make sure your OpenAI API key has sufficient credit. We estimate about 10 dollars for this experiment.
**Execution:** Use the following command to execute the attack procedure for 20 samples under max pooling. This will take about 2 hours total on CPU and about 1 hour on GPU.

```
(attack) $ python attack/main.py \
  --datasets_dir=path/to/datasets/directory \
  --similarity_mode="max" \
  --samples_file="evaluation_samples/\
  maxsim_attacks_rank101.jsonl" \
  --num_papers_to_keep=10 --end=20
# Optional arguemnt: --device=cuda
```

Next, use the following command to execute the attack procedure for 20 samples under average (mean) pooling. This will take about 4 hours total on CPU and about 1.5 hours on GPU.

```
(attack) $ python attack/main.py \
--datasets_dir=path/to/datasets/directory \
--similarity_mode="avg" \
--samples_file="evaluation_samples/\
attacks_rank101.jsonl" \
--num_papers_to_keep=10 --end=20
# Optional arguemnt: --device=cuda
```

**Results:** Each run (each time `attack/main.py` is executed) will have its own output directory, named by the timestamp of when the run started. Use the evaluation script introduced at the end of Appendix A.3.2 to evaluate the attack results of each run. Compare with the two rows of Table 4 in Section 5.5. Since the sample size of 20 here is much lower than in Table 4, higher variance and wider confidence intervals are expected. If the results do not show a convincing difference between average and max pooling, we recommend increasing the sample size to 40 for both mean and max scenarios. You can do so by setting the arguments `--start=20` and `--end=40`. In the new run output folders, you will find subdirectories `attack_i/` for $i = 20...39$. To evaluate all 40 samples, you can move those subdirectories into the same run folder as the first 20 samples and run the evaluation script described at the end of Appendix A.3.2 with the run directory that contains all 40 subdirectories.

**(E4):** *[Correlation] [10 human-minutes + 0 compute-hour]: calculate the correlation of manipulated rankings amongst NeurIPS 2022 and 2023 reviewers.*
**Preparation:** You must complete (E1) before this experiment.
**Execution:** Use the following command to calculate the Spearman's rank correlation coefficients between manipulated rankings in NeurIPS 2022 and 2023. Run the follow command twice, setting `--run_dir` to the output directories for natural rankings 101 and 1001 runs from (E1).

```
(attack) $ python scripts/run_result.py \
--run_dir "out/yyyy-mm-dd/hh-mm-ss" \
--correlation
```

**Results:** The correlation should be high. Compare with the Spearman's rank correlation coefficients reported in Figure 5 of the paper.

## 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/.