USENIX Security '25 Artifact Appendix: GradEscape: A Gradient-Based Evader Against AI-Generated Text Detectors

Wenlong Meng[†] Shuguo Fan[†] Chengkun Wei[†] Min Chen[≀] Yuwei Li[♮§]
Yuanchao Zhang[‡] Zhikun Zhang[†] Wenzhi Chen[†]
[†]Zhejiang University [≀]Vrije Universiteit Amsterdam [♮]National University of Defense Technology
[§]Anhui Province Key Laboratory of Cyberspace Security Situation Awareness and Evaluation [‡]Mybank, Ant Group

A Artifact Appendix

A.1 Abstract

We proposed GradEscape, the first gradient-based evader for attacking AI-generated text (AIGT) detectors. GradEscape overcomes the undifferentiable computation problem, caused by the discrete nature of text, by constructing weighted embeddings for the detector input. It then updates the evader model parameters using feedback from victim detectors, achieving high evasion rates with minimal text modifications.

This artifact is used to evaluate GradEscape against a range of detectors. It includes code for reproducing GradEscape as well as three baseline evaders. We provide a Python library that makes it easy to reproduce existing detectors and evaders. The artifact also contains trained evader models and scripts for replicating our experiments on two real-world AIGT detectors, namely Sapling and Scribbr. To help mitigate the risks posed by AIGT evaders, we showcase that our proposed active paraphrase defense can effectively reduce evasion rates.

A.2 Description & Requirements

We package the artifact into two separate files: `GradEscape.zip` and `Usenix-AE.zip`. `GradEscape.zip` contains the source code; while `Usenix-AE.zip` contains evaluation datasets and trained models. This section describes the minimal hardware and software requirements needed to run the artifact.

A.2.1 Security, privacy, and ethical concerns

Our attacks target exclusively at the target AIGT detector. Therefore, there is no security risk for evaluators. All benchmark datasets are sourced from open corpora, so there are no privacy concerns. Regarding real-world experiments, Sapling and Scribbr have updated their services with stronger models. Therefore, our artifact poses no significant risk to these two online platforms.

A.2.2 How to access

Our artifact is available through Zenodo. The artifact can be accessed at https://doi.org/10.5281/zenodo.15586856.

A.2.3 Hardware dependencies

Two Nvidia RTX A6000 GPUs are the minimum GPU requirement to run the artifact. We recommend at least a 20-core CPU, 32 GB RAM, and 256 GB of free disk space.

A.2.4 Software dependencies

- **OS:** Ubuntu 20.04+. A macOS machine is needed to open Scribbr webarchive file. If you encounter a security warning when opening the webarchive file, please go to Settings, Privacy & Security and click open anyway.
- **Package Manager:** Conda.
- **API Key:** A Sapling API key is required to run Sapling experiments. Conducting these experiments entails money costs.

A.2.5 Benchmarks

Our evaluation requires four datasets: GROVER, HC3, GPA, and GPTWiki. GROVER and GPA are provided in `Usenix-AE.zip`. Our code will automatically download and manage HC3 and GPTWiki. Some pre-trained victim detectors are also included in `Usenix-AE.zip`.

A.3 Set-up

The setup utilizes Conda for environment management.

A.3.1 Installation

Install Main Environment. Download `GradEscape.zip` and `Usenix-AE.zip`. Unzip and place them in the same directory. Then, use the following commands to create an environment:

```
conda create -n ge python=3.10
conda activate ge
cd GradEscape
./install.sh
cp src/AIGT/.config.yaml src/AIGT/config.yaml
```

Generate Word Similarity Matrix. Perturbation-based evaders rely on a word similarity matrix to select synonyms.

```
cd Usenix-AE
git clone
↪ https://github.com/nmrksic/counter-fitting.git
cd counter-fitting/word_vectors/
unzip counter-fitted-vectors.txt.zip
python ../../../GradEscape/tools/comp_cos_sim_mat.py
↪ counter-fitted-vectors.txt
```

Then edit `config.yaml` to set the correct data_dir and counter_fitting_path.

Create vLLM Environment. We need vLLM for fast paraphrasing. Since vLLM has complex dependencies, we create a new environment specifically for vLLM.

```
conda create -n vllm python=3.10
conda activate vllm
cd GradEscape
pip install -r paraphrase_requirements.txt
```

Our artifact mainly runs on `ge`; `vllm` is only used for paraphrase defense.

A.3.2 Basic Test

We provide a simple test that trains a detector using `AIGT`.

```
python basic_test.py
```

If you see "Test finished successfully!", it means the installation was successful.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): GradEscape achieves higher evasion rates than state-of-the-art evaders under the same text quality requirement. This is proven by the experiment (E1) described in Section 6.2 whose results are illustrated in Figure 3 and Figure 4.

(C2): GradEscape is effective against real-world commercial AIGT detectors. This is proven by experiment (E2) described in Section 6.6 whose results are reported in Table 4 and illustrated in Figure 20 and Figure 21.

(C3): Our proposed potential defense can reduce evasion rates below 0.2 against multiple evaders. This is proven by experiment (E3) described in Section 7 whose results are illustrated in Figure 11.

A.4.2 Experiments

(E1): *[Verify Evasion Effectiveness] [5 human-minutes + 1 compute-hour]:*

How to: Navigate to `examples` directory; activate `ge` environment; execute evader training script in `scripts`. The evasion rate and text quality metrics will be printed on the shell.

Preparation: None.

Execution: Navigate to `examples` directory and activate `ge`:

```
cd examples
conda activate ge
```

Then execute the evader training script:

```
./scripts/train_evader_roberta_grover.sh
```

Results: After the program finishes running, the evasion rate and text quality metrics (perplexity, cosine similarity, GRUEN, and ROUGE) will be printed to the terminal. Evaluators can compare these results with the first row of Figure 3. For ease of reference, we provide details of text quality metrics in Table 1.

Table 1: Text quality metrics summary. Expected refers to typical value ranges on GROVER; Print % indicates whether the metric is displayed as a percentage.

Metric	Range	Expected	Larger Better	Print %
Perplexity	$[1, +\infty)$	$[10, 40]$	No	No
Cos-sim	$[-1, 1]$	$[0.95, 1.00]$	Yes	No
GRUEN	$[0, 1]$	$[0.6, 0.8]$	Yes	No
ROUGE	$[0, 1]$	$[0.8, 1.0]$	Yes	Yes

(E2): *[Real-world Case Studies] [5 human-minutes + 30 compute-minutes]:*

How to: Set your Sapling API key environment parameter; run the two real-world case study Jupyter Notebooks. Evaluators may open `Scribbr.webarchive` to verify Scribbr results.

Preparation: A Sapling API key and a macOS machine for Scribbr verification.

Execution: Set your Sapling API key:

```
export SAPLING_API_KEY=<your_api_key>
```

Run `real_world_demo_sapling.ipynb` and `real_world_demo_scribbr.ipynb`. The execution environment is `ge`.

Results: The Sapling results will be printed in its Jupyter Notebook. Evaluators can compare the printed results with Figure 20 and Table 4. Verifying Scribbr results requires copying the output into the website rendered by `Scribbr.webarchive`. The Scribbr results should be the same as Figure 21.

(E3): *[Paraphrase Defense Experiment] [5 human-minutes + 2 compute-hours]:*
How to: Navigate to `examples` directory. First, use `vllm` environment to run `paraphrase_defense_grover.sh`. Then, activate `ge` and run `eval_paraphrase_defense_grover.sh`. It will generate a figure in the current directory named `paraphrase_defense_grover.pdf`.
Preparation: None.
Execution: Navigate to `examples`:

```
cd examples
```

Run paraphrase:

```
conda activate vllm
./scripts/paraphrase_defense_grover.sh
```

Train a new detector and evaluate the defense:

```
conda activate ge
./scripts/eval_paraphrase_defense_grover.sh
```

Results: The program will generate a figure named `paraphrase_defense_grover.pdf` in `examples/`. This figure illustrates evasion rates before and after applying our defense. Evaluators can compare the generated figure with Figure 11 in our paper.

A.5 Notes on Reusability

We implement `AIGT` in a reusable way. All configurations, including datasets, detectors, and evaders, are extracted in `arguments.py`. Followers can replicate existing AIGT detectors and evaders with little effort. The user interface of detectors and evaders is designed in a unified and HuggingFace-like way. Followers can also write their own detectors and evaders in `detectors/` and `evaders/`, respectively.

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/.