USENIX Security '25 Artifact Appendix: TwinBreak: Jailbreaking LLM Security Alignments based on Twin Prompts

Torsten Krauß
University of Würzburg

Hamid Dashtbani
University of Würzburg

Alexandra Dmitrienko
University of Würzburg

A Artifact Appendix

A.1 Abstract

This artifact accompanies the USENIX paper "TwinBreak: Jailbreaking LLM Security Alignments with Twin Prompts". It delivers a complete, reproducible implementation for removing safety alignments from downloaded large language models (LLMs), such as open-source models on Hugging Face, while safeguarding their utility. As the basis for TwinBreak, this artifact also contains the TwinPrompt dataset, a collection of one hundred prompt pairs with one harmful and one harmless prompt that yield high structural and content similarity. Feeding these prompts through a target model, TwinBreak traces intermediate activations to rank each parameter by its contribution to safety enforcement. Feeding two harmless prompts allows one to identify parameters mandatory for utility. Parameters deemed critical to safety are pruned at inference time unless they also prove indispensable for utility, thereby disabling safety alignment without degrading utility. The artifact bundles the TwinBreak source code, the TwinPrompt dataset, and scripts needed to reproduce experiments that demonstrate successful alignment removal and consequent jailbreaks on recent LLMs.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Executing this artifact poses no risk.

A.2.2 How to access

The artifact can be accessed via GitHub and Zenodo.

- GitHub: https://github.com/tkr-research/twinbreak
- Zenodo: https://doi.org/10.5281/zenodo.15591819

A.2.3 Hardware dependencies

The experiments in this artifact run reliably on Unix-based servers equipped with a sufficiently powerful NVIDIA GPU with at least 48 GB of memory. More powerful GPUs and greater memory capacities will lead to faster runtimes. In addition, approximately 120 GB of disk space is required for downloading large language models (LLMs) from Hugging Face. The codebase and experimental outputs occupy roughly 100 MB. We also recommend using a Conda environment for managing dependencies, which consumes around 7 GB of disk space.

Minimum System Requirements:

- NVIDIA GPU with at least 48 GB of memory
- 130 GB of available disk space

Hint 1. Reproducing the full experimental suite from the paper, including models with up to 70 billion parameters, demands significantly more resources, both GPU memory and disk space. For reference, our experimental setup used an Intel Xeon Gold 6526Y CPU (16 cores, 64 threads), 256 GB of RAM, four NVIDIA L40S GPUs (each with 48 GB of GDDR6 memory), and a 7 TB HDD. Configurations for these large-scale experiments are included for completeness but are not part of the prepared demonstration for this artifact, focusing on TwinBreak's core functionality. Users may select the respective model architectures in the configuration file to execute them, but can safely skip them for this artifact.

Hint 2. The most memory-intensive aspect of the artifact is the generation of LLM responses during utility and safety evaluations, not the TwinBreak attack itself. By default, a batch size of 20 is used for inference during benchmark executions. To reduce runtime, this can be increased up to 100, provided sufficient GPU memory is available. If the available GPU memory is less than 48 GB, the batch size may need to be reduced to fit the model, though this will come at the cost of longer runtimes.

A.2.4 Software dependencies

The experiments are designed to run on a Unix-based operating system, such as Debian, which serves as our reference platform. The environment requires Python 3.10 and PyTorch 2.7.0, with CUDA-enabled access to NVIDIA GPUs. While we recommend using CUDA 12.6, we also provide instructions for alternative CUDA versions.

We suggest using `Miniconda` to create an isolated Conda environment, which simplifies setup and ensures proper de-

pendency management. All required packages can be installed via the pip package manager.

Required Python Packages:

- torch 2.7.0
- transformers 4.44.2
- lm_eval 0.4.7
- StrongREJECT
- python-dotenv 0.9.9
- pyyaml 6.0.2

Hint. We provide specific package versions to ensure a smooth and consistent reproduction process. In particular, newer versions of transformers may lead to issues related to torch._dynamo, which attempts to compile Python model code into a single optimized computation graph. If the model's code involves frequent changes in shape, type, or control flow, such as from dynamic forward() logic or the use of hooks, Dynamo may recompile the graph repeatedly, eventually reaching the default limit of 8 recompilations. This issue arises due to the pruning implementation, which relies on forward hooks. While it is possible to circumvent the problem by disabling TorchDynamo Just-In-Time (JIT) Compilation using the TORCHDYNAMO_DISABLE=1 environment flag, doing so can significantly increase execution time. Therefore, we recommend using the package versions and settings specified in the paper.

Hugging Face Access. To download the required LLMs from Hugging Face, users must have a Hugging Face account that has accepted the respective model license agreements. An access token associated with this account is necessary and must be used within the project to authenticate and enable model downloads.

The following models, used in the core experiments of this artifact, require license agreement on the respective Hugging Face website and subsequent access token authorization:

- https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
- https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct
- https://huggingface.co/google/gemma-2-9b-it
- https://huggingface.co/Qwen/Qwen2.5-7B-Instruct
- https://huggingface.co/meta-llama/Llama-2-13b-chat-hf
- https://huggingface.co/Qwen/Qwen2.5-3B-Instruct
- https://huggingface.co/meta-llama/Llama-Guard-3-8B
- https://huggingface.co/google/gemma-2b

Executing experiments with additional models may require accepting further license agreements.

A.2.5 Benchmarks

The artifact contains three dataset categories: one for measuring utility, another for assessing safety alignment, and the newly introduced TwinPrompt dataset used during the TwinBreak attack.

Utility Datasets. The artifact uses five dataset to utility analysis. All datasets can be easily downloaded and are open-source. The five datasets are: 1) OpenBookQA, which tests an LLM's reasoning and knowledge absorption capability with a focus on preliminary scientific topics. 2) ARC-Challenge that targets more complex science questions. 3) HellaSwag which asks the LLM to choose the most plausible continuation scenario given a partial sentence or scenario. 4) RTE, which evaluates with whether a hypothesis can be inferred from a premise. 5) WinoGrande evaluates an LLM's common sense and contextual understanding. The datasets are downloaded and used on the fly via the lm_eval Python package.

Safety Alignment Datasets. The artifact uses four datasets comprising harmful prompts to assess LLMs' security and resilience against potential misuse. These prompts are designed to mimic malicious interactions. All datasets are open-source and are already part of the repository formatted as JSON files. The four datasets and respective links to the original dataset files are: 1) AdvBench, which contains 520 harmful prompts. 2) HarmBench, an improved version of AdvBench with 400 harmful prompts. Following related work (as discussed in the paper), we only use 200 prompts from HarmBench. 3) Jailbreakbench, which contains 100 pairs of harmful and harmless prompts. However, the pairs are not twins as in our dataset. 4) StrongREJECT, which contains 313 harmful prompts trying to address shortcomings of the previous datasets.

New TwinPrompt Dataset. Finally, this artifact publishes the new dataset, TwinPrompt, which consists of one hundred prompt pairs with one harmful and one harmless prompt that yield high structural and content similarity. The dataset is also part of this repository and in JSON format and can be found under twinbreak/dataset/json/twinprompt.json.

A.3 Set-up

A.3.1 Installation

It is assumed that the setup begins with a newly created user account on a Unix-based server, such as a Debian system.

To ensure reproducibility and consistent package management, we use Miniconda to manage the Python environment.

While it is possible to install the dependencies globally without Miniconda, we recommend using Miniconda as described below. If Miniconda is skipped, users must manually install Python 3.10 and ensure all dependencies are correctly resolved.

Hint: When copying multi-line commands from the PDF, we recommend first pasting them into a plain text editor (e.g., Notepad) to remove any unintended line breaks before executing them in the terminal. The symbol ↪ indicates that the command continues on the next line without any added space; it is used solely for formatting purposes due to line length constraints in the PDF layout. Alternatively, the commands can be copied from the README file in the GitHub repository.

Hugging Face Token: If you already have access to a Hugging Face access token with read permissions, you can use it directly. Otherwise, follow the steps below to create your own token:

1. Create an Hugging Face account. Such an account can be created for free.
2. To generate an access token, visit https://huggingface.co/settings/tokens, click "Create new token", select the "Read" token type, and assign a name to the token. Once created, copy the token for later use.
3. To get access to the models, please follow the license agreement instructions provided on the Hugging Face links provided under Section A.2.4. After confirming the agreements, it may take up to an hour for your access status to change from "pending" to "accepted."

Miniconda Installation: If `Miniconda` is not already installed on the server, follow the steps below to install Miniconda.

1. Download the Miniconda installer:

```
wget https://repo.anaconda.com/miniconda/Miniconda3-
    ↪ latest-Linux-x86_64.sh
```

2. Run the installer:

```
bash Miniconda3-latest-Linux-x86_64.sh
```

During installation, follow these steps:

- Press `ENTER` to view the license terms.
- Press `SPACE` to scroll to the end of the license terms.
- Type `yes` to accept the license agreement.
- Press `ENTER` to confirm the default location for conda environments.
- Type `yes` to enable automatic activation of conda on shell startup.

Afterward, restart your shell, such that conda is activated. The shell should look like this showing that the base conda environment is activated:

```
(base) user@server:/home/user$
```

Environment Setup

1. Create and activate the Conda environment:

```
conda create --name twinbreak python=3.10
```

During the installation process enter y to install the new packages.

```
conda activate twinbreak
```

Afterward, your shell should look like this, showing that the new twinbreak environment is activated:

```
(twinbreak) user@server:/home/user$
```

2. Identify your CUDA version (if applicable):

```
nvidia-smi
```

The CUDA version appears in the top-right corner of the table output.

3. Install PyTorch corresponding to your CUDA version:
   - For CUDA 11.8:

```
pip install torch --index-url https://download.
    ↪ pytorch.org/whl/cu118
```

   - For CUDA 12.6 (default if compatible):

```
pip install torch
```

   - For CUDA 12.8 or later:

```
pip install torch --index-url https://download.
    ↪ pytorch.org/whl/cu128
```

   If you have a different CUDA version, you should try out the next smaller PyTorch version.

4. Install the remaining dependencies:

```
pip install transformers==4.44.2
pip install lm_eval==0.4.7
pip install git+https://github.com/dsbowen/
    ↪ strong_reject.
    ↪ git@e286f0da86d92c929a6fda20a9992f28c5969044
pip install dotenv==0.9.9
pip install pyyaml==6.0.2
```

Artifact Retrieval and Initialization

1. Navigate to the desired directory where the project should be stored and clone the repository:

```
git clone https://github.com/tkr-research/twinbreak.
    ↪ git
```

2. Enter the project root:

```
cd twinbreak
```

3. Run the setup script, providing your Hugging Face access token and the model storage path. Therefore, replace `<HUGGING_FACE_TOKEN>` and `<STORE_MODEL_DISK_PATH>` with the real values, e.g., `hf_xxxYOURTOKENxxx` and `\home\user\.cache\huggingface`. Note, that the terminal user needs read and write permissions to the selected directory. The two values will be saved in a `.env` file and used to set necessary environment variables. Additionally, the script will configure the project root as part of the Python path.

```
source setup.sh --hf-token <HUGGING_FACE_TOKEN> --
    ↪ store-model-disk-path <STORE_MODEL_DISK_PATH>
```

The terminal should output the following:

```
.env file created and values injected:
  - HF_TOKEN
  - STORE_MODEL_DISK_PATH
PYTHONPATH set to /home/user/twinbreak
```

A.3.2 Basic Test

To run a simple functionality test, execute the following commands.

1. Navigate to the experiments directory:

```
cd experiments
```

2. Execute the test script to verify the setup:

```
python experiment_test.py
```

3. If successful, the following message will be displayed:

```
The system is set up to run experiments!
```

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): TwinBreak effectively removes safety alignment from open-source large language models (LLMs) with minimal impact on their utility, as demonstrated on the LLaMA 2 (7B) model. This is proven by experiment (E1) in this artifact, which is also reported in Section 4.2 of the paper, with results reported in the first rows of Tables 2–5 for safety alignment removal, and Tables 15–19 for utility preservation.

(C2): TwinBreak demonstrates effectiveness across diverse model architectures and vendors, as shown on Llama 3.1 (8B), Gemma 2 (9B), and Qwen 2.5 (7B). This is proven by experiment (E2) in this artifact, which is also reported in Section 4.2 of the paper, with results reported in the remaining rows of Tables 2–5 for safety alignment removal, and Tables 15–19 for utility preservation.

(C3): TwinBreak also proves effective across varying model sizes, as demonstrated on the larger LLaMA 2 (13B) and the smaller Qwen 2.5 (3B). This is proven by experiment (E3) in this artifact, which is also reported in Section 4.6 of the paper, with results reported in rows two and six of Table 8.

A.4.2 Experiments

Hint 1. The reported experiment runtimes were obtained using a single NVIDIA L40s GPU with 48 GB of memory. Actual runtimes may vary depending on the number of GPUs, GPU model, and available memory. By default, the provided code will automatically utilize all available GPUs on the server.

Hint 2. If you wish to restrict execution to a specific subset of GPUs, you will need to modify the command used to run the Python scripts. For example, to use only the GPUs with indices 1 and 2, the command would be:

```
CUDA_VISIBLE_DEVICES=1,2 python experiment_test.py
```

*Hint 3.*Given the potentially long runtime of the experiments, we recommend using the `screen` tool to prevent interruptions in case the connection to the server is lost. This helps ensure that experiments continue running even if the terminal session is disconnected. For usage instructions, please refer to the `screen` manual page. However, this step is optional for executing the artifact.

Below, three experiments are described, each proving one major claim of this project.

(E1): [General Functionality] [1 human-minute + 2 compute-hour + 35GB disk]: In this experiment, we attack the LLaMA 2 (7B) model with TwinBreak and evaluate the jailbreak success as well as the utility with all benchmarks. The results prove the major claim (C1).

Preparation: Follow all instructions under Section A.3. The terminal should reside in the `twinbreak\experiments` folder.
Execution: Execute the script for the first experiment.

```
python experiment_1.py
```

Results: The experiment logs its results both to the terminal and to a log file located at `twinbreak\results\experiment_1\log\log0.txt`. At the end of the output, a summary of the results is presented in two tables. The first table reports the utility benchmark results and corresponds to the first rows of Tables 15–19 in the paper. The second table presents the safety benchmark results and can be compared to the first rows of Tables 2–5. For clarity, the relevant reference table from the paper is also indicated alongside each output table in the terminal.
We expect the reported values to fall within a 1–5% range of those published in the paper. Additionally, the runtime of the attack is displayed beneath the tables and can be compared to the values in Table 7. Note that runtime is highly dependent on hardware. For example, while the paper reports a runtime of 162 seconds, our server achieved significantly faster results (16 seconds), as the experiment was run on Kaggle.

(E2): [Model Architecture Independence] [3 human-minute + 7 compute-hour + 114GB disk]: In this experiment, we repeat experiment (E1) with different model architectures from different vendors, namely Llama 3.1 (8B), Gemma 2 (9B), and Qwen 2.5 (7B). The results prove the major claim (C2).
Preparation: Follow all instructions under Section A.3. The terminal should reside in the `twinbreak\experiments` folder.
Execution: We provide one script for each of the models. Execute the scripts individually. To execute TwinBreak for Qwen 2.5 (7b), execute the following command. [1 human-minute + 2 compute-hour + 37GB disk]

```
python experiment_2_1.py
```

To execute TwinBreak for Gemma 2 (9b), execute the following command. [1 human-minute + 3 compute-hour + 40GB disk]

```
python experiment_2_2.py
```

To execute TwinBreak for LLaMA 3.1 (8b), execute the following command. [1 human-minute + 2 compute-hour + 37GB disk]

```
python experiment_2_3.py
```

Results: As with experiment (E1), each script generates terminal output and corresponding log files stored in the appropriate results folder, for example, `twinbreak\results\experiment_2_1\log\log0.txt` for `experiment_2_1.py`. The output can be compared to the utility and safety benchmarks in Tables 15–19 and Tables 2–5, respectively. As before, we expect the reported values to be within a 1–5% range of those presented in the paper.

(E3): [Model Size Independence] [2 human-minute + 4 compute-hour + 75GB disk]: In this experiment we attack models with different model sizes, namely LLaMA 2 (13b) and Qwen 2.5 (3b), using TwinBreak. We evaluate the jailbreak success with the newest benchmark (StrongREJECT) and the utility with HellaSwag. The results prove the major claim (C3).
Preparation: Follow all instructions under Section A.3. The terminal should reside in the `twinbreak\experiments` folder.
Execution: We provide one script for each of the models. Execute the scripts individually. To execute TwinBreak for LLaMA 2 (13b), execute the following command. [1 human-minute + 3 compute-hour + 47GB disk]

```
python experiment_3_1.py
```

To execute TwinBreak for Qwen 2.5 (3b), execute the following command. [1 human-minute + 1 compute-hour + 28GB disk]

```
python experiment_3_2.py
```

Results: As with experiment (E1), each script generates terminal output and corresponding log files stored in the appropriate results folder, for example, `twinbreak\results\experiment_3_1\log\log0.txt` for `experiment_3_1.py`. The output can be compared to the utility and safety benchmark results in Table 8. Specifically, the `AVG Degradation` value in the utility benchmarks corresponds to the `TwinBreak Utility` column for the respective model in Table 8, while the StrongREJECT score in the `Iteration 5` column reflects the `TwinBreak ASR` column in Table 8. As with previous experiments, we expect the reported values to fall within a 1–5% range of those presented in the paper.

Hint. In order to clean up after executing the artifact, follow the cleanup instructions in the README file of the GitHub repository.

A.5 Notes on Reusability

The core functionality of *TwinBreak* is implemented in `TwinBreak.py`. Configuration options and hyperparameters are managed via YAML files, with `twinbreak_default_settings.yaml` serving as the default configuration. Each parameter is documented in the paper, the YAML file itself, and the corresponding Python class `TwinBreakConfig.py`.

To evaluate TwinBreak on benchmark tasks, use the

`TwinBreakAndEval.py` wrapper, which extends the core functionality of `TwinBreak.py` with evaluation capabilities.

Customizing Experiments: There are two main types of configuration files:

- **TwinBreak configuration** (e.g., utility weight parameters): Default: `twinbreak_default_settings.yaml`
- **Experiment configuration** (e.g., selected model, benchmarks, batch size): Default: `experiment_default_settings.yaml`

The entry point for running a complete experiment is `ExperimentExecutor.py`, which provides a `run()` function to execute the pipeline.

Hyperparameter Study: Configuration files used for the hyperparameter analysis (as reported in Table 20 of the paper) are available in the `hyperparameter_study` folder, with `H1.yaml` being the first file.

Extending to New Models: To apply *TwinBreak* to models not currently supported, implement a new subclass of the `AbstractModel.py` class. Then, reference your custom model identifier in the experiment configuration file.

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/.