USENIX Security '26 Artifact Appendix: On Improving Robustness of Deepfake Image Detectors

Abu Taib Mohammed Shahjahan
Concordia University, Montreal, Canada

Mohammad Mannan
Concordia University, Montreal, Canada

Abdessamad Ben Hamza
Concordia University, Montreal, Canada

Amr Youssef
Concordia University, Montreal, Canada

A Artifact Appendix

A.1 Abstract

This artifact constitutes the official codebase and pre-trained model weights accompanying the USENIX Security 2026 paper "On Improving Robustness of Deepfake Image Detectors." The artifact provides a complete software framework implemented in PyTorch for reproducing all major experimental results, including the evaluation of the proposed primary detector, Improved-D3 (`D3_imp`), with the MM-BSN denoising module, alongside five legacy baseline detectors (CNN-F, Resynthesis, DCT, DE-FAKE, and Patch-Forensics). The source code is permanently archived on Zenodo (https://doi.org/10.5281/zenodo.20327567) and actively maintained on GitHub (https://github.com/shahjahan0275/semantic_adv).

To accommodate varying resources, we provide two evaluation pathways: a **Functional Setup** using `sample_data` and pre-trained checkpoints for quick verification, and a **Full Reproduction Setup** requiring the ∼623 GB GenImage and adversarial datasets to fully replicate Tables 1–4 in our paper. Dependencies are strictly managed via two Anaconda environments: Python 3.10 for the primary detector (`D3_imp`), and Python 3.8 with PyTorch 1.12.1 for legacy baselines. The provided scripts, SLURM files, and detailed `README.md` facilitate execution across consumer GPUs (for legacy functional tests) and HPC H100/A100 GPUs (mandatory for all training and primary model inference) to seamlessly validate functionality, robustness metrics, and training convergence.

A.2 Description & Requirements

We outline the hardware and software requirements for evaluating our artifact in §A.2.3 and §A.2.4, respectively. The `README.md` file in our artifact includes step-by-step instructions to help users configure their environments. To facilitate evaluation, we provide two distinct evaluation pathways based on data requirements. We refer to the first scenario as the "Functional Setup," which utilizes a provided `sample_data` directory and pre-trained checkpoints to allow USENIX Security evaluators to quickly verify the codebase's functionality without downloading hundreds of gigabytes of data (e.g., the 623 GB GenImage dataset). We refer to the second scenario as the "Full Reproduction Setup," which requires downloading the massive, complete external datasets to fully replicate the time-intensive training and evaluation results reported in the paper.

The hardware requirements are dictated by the specific model architectures rather than the evaluation pathway. Due to the integration of the memory-intensive MM-BSN module and the large size of the trained weights (∼1.8 GB), executing both training and inference for our primary `D3_imp` detector strictly requires high-performance computing (HPC) cluster nodes (e.g., NVIDIA H100 or A100), even when utilizing the reduced `sample_data`. Standard consumer-grade GPUs lack the required VRAM for `D3_imp` and are only sufficient for running the functional inference tests of the five legacy baseline detectors (CNN-F, Resynthesis, DCT, DE-FAKE, and Patch-Forensics). The remainder of this section details these dependencies and the basic installation steps.

A.2.1 Security, privacy, and ethical concerns

The artifact does not contain any destructive scripts, malware, or functionality that would compromise the security of the evaluator's machine. All operations are limited to standard deep learning training and inference tasks, including data loading, model training, and evaluation. The codebase does not disable any security mechanisms, collect personally identifiable information, or require network access beyond downloading pre-trained weights and datasets from public sources.

A.2.2 How to access

The official archived artifact corresponding to the USENIX Security 2026 paper is permanently available via Zenodo at the following stable DOI reference: https://doi.org/10.5281/zenodo.20327567. The active development repository containing the source code is hosted on GitHub (https://github.com/shahjahan0275/semantic_adv). Pre-trained

model weights required for inference are hosted externally on Google Drive, with direct access links provided within the artifact's `README.md`.

A.2.3 Hardware dependencies

The hardware requirements strictly depend on the chosen evaluation pathway:

- **Full Reproduction Setup:** Executing the full training and evaluation pipeline requires high-performance GPUs. The primary `D3_imp` detector strictly requires an NVIDIA H100 (96GB VRAM) to prevent CUDA Out-Of-Memory (OOM) errors. The legacy detectors (e.g., CNN-F, Resynthesis, DCT, DE-FAKE, and Patch-Forensics) strictly require an NVIDIA A100 (80GB) GPU. Evaluators must have access to such hardware (80GB–96GB VRAM) to execute any training scripts, as standard local workstations (e.g., RTX 4500) possess insufficient memory for these models. Inference scripts for the legacy detectors (e.g., CNN-F, Resynthesis, DCT, DE-FAKE, and Patch-Forensics) could be run on standard local workstations (e.g., RTX 4500). We do not provide SSH access to our machines; evaluators pursuing full reproduction must provision their own equivalent HPC hardware.
- **Functional Setup:** To verify functionality via the basic tests (§A.3.2) using the provided `sample_data`, we still need NVIDIA A100 (80GB).

A.2.4 Software dependencies

Our framework operates on a standard Linux OS (e.g., Ubuntu) and relies on the PyTorch ecosystem. Strict dependency management is enforced via Anaconda/Miniconda. We provide two distinct Conda configurations to support models spanning different architectural generations:

- **Primary Environment:** Requires Python 3.10 and is used for evaluating our proposed primary `D3_imp` detector.
- **Legacy Defense Environment:** Requires Python 3.8, PyTorch 1.12.1, and CUDA 11.3, specifically configured to run the older detectors (e.g.,CNN-F, Resynthesis, DCT, DE-FAKE, and Patch-Forensics).

We provide complete `environment.yml` files for seamless Conda initialization, alongside `requirements.txt` files as a fallback for standard pip installations.

A.2.5 Benchmarks

For the Full Reproduction Setup, evaluators must download the following external datasets (not included in the artifact due to massive size limitations, the URLs are provided in our `README.md` file):

- **GenImage Dataset:** Used for training the primary model ($\sim$623 GB).
- **Adversarial & Surrogate Datasets (Abdullah et al., IEEE SP 2024):** Used for robustness evaluation across Tables 1–4 (e.g., StyleCLIP adversarial samples).

For the Functional Setup, we have provided a `sample_data` folder within the artifact containing a minimal subset of these benchmark images.

A.3 Set-up

A.3.1 Installation

To prepare the environment, first install Anaconda or Miniconda. Clone the repository and configure the environments using the provided YAML files.

1. Install the Primary Environment:

```
conda env create -f environment.yml
conda activate D3
```

Fallback (if Conda creation fails):

```
conda create -n D3 python=3.10
conda activate D3
pip install -r requirements.txt
```

2. Install the Legacy Environment: Navigate to the `EvolvingThreat-DeepfakeImageDetect/defenses/` directory and repeat the process for the older baselines:

```
conda env create -f environment.yml
conda activate defake
```

3. Download Pre-trained Weights: Follow the Google Drive links in the `README.md` to download the pre-trained checkpoints. Place them in their respective `ckpt/` directories as specified. Update any absolute file paths in the scripts to match your local directory structure.

A.3.2 Basic test

To verify that the primary model environment is correctly configured and functional, you will run a lightweight inference script using the adversarial fake dataset provided by Abdullah et al. (IEEE SP 2024).

Execution:

```
conda activate D3
python validate_for_robustness_spd.py
```

Note: Ensure the dataset paths inside `validate_for_robustness_spd.py` are pointing to the local directory containing any sub-folder of the `sample_data` (e.g., sample_data/StyleCLIP_dataset) or any one of the adversarial fake datasets provided by Abdullah et al. (IEEE SP 2024) and the model path pointing to the downloaded checkpoint.
Expected Output: The script will initialize the model on the GPU without OOM errors, process the small batch of images in the `sample_data` or adversarial fake dataset folder, and output a summary of the detection metrics (e.g., Recall/Accuracy on the sample batch) alongside a `.xlsx` log file detailing the prediction scores for each processed image.

A.3.3 Major claims

The following major claims (C1–C4) are made in the paper and are validated by the experiments (E1–E3) described in the next subsection. These claims are structured to provide a complete evaluation workflow: C1 establishes the baseline problem (the performance degradation that exists in current detectors), C2 and C3 demonstrate the effectiveness of our proposed solutions, and C4 explains why our approach works through ablation analysis. Together, they enable artifact evaluators to verify both the existence of the problem and the efficacy of our proposed improvements using the provided codebase.

(C1): Recent state-of-the-art deepfake detectors (seven models from 2024–2025) suffer severe performance degradation under the content-manipulating adversarial attacks of Abdullah et al. (IEEE SP 2024). For example, accuracy drops from 98.9% to 48.7% for Directionality, and even the best-performing detector D^3 (CVPR 2025) degrades from 86.7% to 81.9%. C1 establishes the **baseline problem** that our work addresses. It provides the "before" comparison against which our improvements (C2, C3) are measured. Without reproducing this degradation, evaluators cannot validate the improvement magnitude claimed in C2 (e.g., 81.9% $\rightarrow$ 97.15%). This claim also justifies why the repository includes code for evaluating seven SOTA detectors beyond our own model, and provides context for the adversarial datasets used throughout the evaluation.

How to reproduce: This claim is demonstrated in Section 3 of the paper (Table 1) and can be reproduced using experiment (E2) on the full adversarial datasets, following the "Reproducing Table 1" instructions in the GitHub `README.md`.

(C2): Our proposed framework, which integrates patch-shuffling & rotation, MM-BSN content-agnostic residual features, and fourth-order DCT-based moment pooling, consistently improves robustness without adversarial training. When integrated into D^3, our method increases accuracy from 81.9% to 97.15% on the StyleCLIP adversarial dataset, with average AP gains of +9.94% (see Table 3 of the paper).

Why this claim is included: C2 demonstrates the primary contribution of our work: significantly improving the best-performing baseline detector (D^3) against adversarial attacks. The 15.25% accuracy gain (81.9% $\rightarrow$ 97.15%) directly shows the effectiveness of our three-component framework. This claim validates that our approach works on a complex, modern architecture without requiring adversarial training data.

How to reproduce: This claim is supported by experiment (E2) using the pre-trained enhanced model, following the "Reproducing Table 3" instructions in the GitHub `README.md`.

(C3): Our modifications substantially reduce recall degradation for five weaker detectors (DCT, DE-FAKE, CNN-F, Patch-Forensics, Resynthesis) across three surrogate models (EfficientNet, ViT, CLIP-ResNet). The maximum reduction is 88.9% (DCT, CLIP-ResNet) and the minimum is 19.5% (Patch-Forensics, CLIP-ResNet); details are in Table 4 of the paper.

Why this claim is included: C3 demonstrates that our approach is **architecture-agnostic** and broadly applicable, not just tailored to D^3. By showing significant improvements across five distinct detectors with different architectural priors, we prove the generalizability of our techniques. The variation in improvement magnitude (19.5%–88.9%) also provides insight into which detector designs benefit most from higher-order statistics.

How to reproduce: This claim is reproduced by experiment (E2) on the adversarial datasets using the retrained models in the `EvolvingThreat-DeepfakeImageDetect/defenses/` folder, following the "Reproducing Table 4" instructions in the GitHub `README.md`.

(C4): Ablation studies (Section 5.3, Table 5 & 6) show that fourth-order DCT statistics (kurtosis) are the critical driver of improvement, outperforming first-, second-, and third-order pooling; patch shuffling alone matches the original D^3 performance, confirming that gains are not from spurious artifacts.

Why this claim is included: C4 explains why our approach works by isolating the contribution of each component. It demonstrates that performance gains are not simply due to increased model capacity or added features, but specifically arise from fourth-order (kurtosis) statistics capturing tail-heavy distributions that adversarial attacks leave unconstrained. This ablation study also confirms that patch shuffling does not introduce spurious discriminative artifacts.

How to reproduce: This claim is validated by experiment (E2) when comparing different pooling orders, and by experiment (E3) for training convergence, using the configurable training scripts provided in the repository (e.g., `train_3branch_FOS_F.py`).

A.3.4 Experiments

The following experiments are designed to validate the functionality of the provided codebase and to reproduce the primary claims regarding detector robustness reported in Tables 1–4. Please note that hardware constraints (H100/A100 vs. consumer GPUs) and data download requirements scale significantly between the "Functional" and "Full Reproduction" setups. We describe the "Functional" under experiment 1 (E1) and "Full Reproduction" are divided into two Experiments experiment 2 (E2) and experiment 3 (E3) for full inference and full training with full dataset as described in the Zenodo/GitHub code repository.

(E1): [Functional Verification via Sample Data] [Approximately 30 human-minutes + 1 compute-hour + 5GB disk]: This experiment verifies the functional correctness of the primary `D3_imp` detector and other five detector models without requiring massive dataset downloads. It utilizes the minimal `sample_data` provided in the repository.
Preparation: Activate the respective conda environments (`D3` or `defake`). Download the pre-trained model weights from the provided Google Drive link and place them in their respective `ckpt` or `checkpoints` directories.
Execution: For the primary `D3_imp` detector, execute `python validate_for_robustness_spd.py` using the sample data. Note: This specific script must be executed on an HPC node equipped with an NVIDIA H100 or A100 to prevent OOM errors during the MM-BSN module initialization. For the five legacy detectors (CNN-F, Resynthesis, DCT, DE-FAKE, Patch-Forensics), execute their respective inference commands from the `README.md` using the `sample_data`; these inferences can be successfully run on standard consumer-grade GPUs.
Results: The models should successfully allocate to the GPU, process the sample images, and terminate without CUDA OOM errors. This successfully demonstrates the end-to-end functional integrity of the inference codebase.

(E2): [Detector Robustness Evaluation (Tables 1, 2, 3 & 4)] [30 human-minutes + 4 compute-hours + 60GB disk]: This experiment reproduces the inference-based robustness evaluations reported in Tables 1, 2, 3 and 4 of the paper, testing the detectors against the full evaluation datasets.
Preparation: Download the evaluation datasets (Section 5.2 of Abdullah et al., IEEE SP 2024). Update the dataset directory paths (e.g., `-fake_root`, `-real_root`, `-dir`) and absolute checkpoint paths within the respective inference scripts to point to your local directories.
Execution: To reproduce Table 1 and 2 please go to the relevant section of the `README.md` file in the code repository. To reproduce Table 3, run `validate_for_robustness_spd.py` using the `D3_imp` weights and all the datasets of Section 5.2 of Abdullah et al. (IEEE SP 2024). To reproduce Table 4, run the batch inference scripts across the legacy detectors located in the `EvolvingThreat-DeepfakeImageDetect/defenses/` folder (e.g., `infer_dct_LM1.py`, `infer_dct.py`, `test_dct.py`) using the three adversarial datasets of Abdullah et al., IEEE SP 2024 (CLIPResNet, EfficientNet, ViT). Optionally, execute `python plot_tsne.py` to generate the t-SNE visualizations (Figures 5 and 6).
Results: The output logs and `.csv` files will contain the evaluation metrics. The calculated results should closely match the robustness metrics reported in Tables 3 and 4 of the manuscript.

(E3): [End-to-End Training] [4 human-hours + 19 compute-days + 750GB disk]: This experiment covers the complete, time-intensive reproduction of the model training phases for both the primary `D3_imp` detector and the legacy baseline defenses.
Preparation: Download the complete GenImage dataset (∼623 GB) and the training datasets from Abdullah et al. (IEEE SP 2024). Ensure the HPC node has at least 750GB of free storage. Activate the relevant environment and update the `-checkpoints_dir`, `-dataroot`, and dataset root flags in the training scripts.
Execution: For `D3_imp`, execute `train_3branch_FOS_F.py` using the parameters for either the StyleCLIP only dataset or the combined StyleCLIP and GenImage datasets. For the legacy models, run their respective training scripts (e.g., `train_dctstats.py`, `train_FOS_3BDCT_patch_text.py`, `train_V1.py`) or submit the provided SLURM scripts (e.g., `Resynthesis_styleclip.sh`) to the cluster scheduler.
Results: The training scripts will pipe outputs to log files (e.g., `logs/train_*.log`) and yield new model checkpoint files (`.pth` or `.pt`) in the specified output directories, demonstrating successful training convergence on the full datasets.

A.4 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/usenixsec2026/.