USENIX Security '26 Artifact Appendix: Adversarial Patch EXterminator: Zero-Shot and Patch-Agnostic Defense Framework Against Adversarial Patch Attacks

Jiayimei Wang[1] Tao Ni[2] Guowen Xu[3] Qingchuan Zhao[1] Cong Wang[1]
[1]City University of Hong Kong
[2]King Abdullah University of Science and Technology
[3]University of Electronic Science and Technology of China

A Artifact Appendix

A.1 Abstract

`APEX` is a zero-shot defense framework designed to mitigate adversarial patch attacks by localizing patch regions and restoring the original image. This artifact enables the reproduction of the main experiments conducted in our paper. Specifically, the artifact provides the implementation of the complete `APEX` defense pipeline, including bounding-box extraction, patch localization, and image inpainting. Furthermore, the artifact provides a diverse collection of adversarial patch examples, including non-naturalistic, naturalistic, and infrared patches on multiple pedestrian datasets, together with patch location annotations and ground-truth labels.

In addition, the artifact includes configuration files and instructions for reproducing the key experimental results shown in the paper, such as patch localization recall and false positive rates across different patch types. Besides, It also supports the reproducibility of robustness evaluations of `APEX` under challenging scenarios, including tiny patches, irregularly-shaped patches, background-coherent patches, and adversarial patches placed on abstract paintings.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

The artifact does not pose risks to the evaluator's machine security, data privacy, or raise ethical concerns. It does not require disabling any security mechanisms, modifying system-level configurations, or executing destructive operations. In addition, all experiments are conducted on publicly available datasets or synthetic adversarial patches generated for research purposes.

A.2.2 How to access

The artifacts are available at Zenodo: https://zenodo.org/records/19233434. Please make sure to download the latest version (v4 or later).

A.2.3 Hardware dependencies

All experiments in the paper were conducted on NVIDIA RTX A6000 GPUs with 48 GB of GPU memory. However, the artifact can also be executed on other CUDA-capable GPUs with at least 6 GB of GPU memory, as the peak GPU memory usage during execution is approximately 5–6 GB for a single GPU.

A.2.4 Software dependencies

The artifact is implemented in Python (Python 3.10 is recommended) and relies on common deep learning libraries. All required dependencies are specified in `requirements.txt`.

A.2.5 Benchmarks

None.

A.3 Set-up

A.3.1 Installation

We refer the evaluator to the README for the installation instructions. Specifically, The weights for YOLOv5, YOLOv8, SAM, and LaMa are all already included in the resources provided in Zenodo. Manual installation is only required if any of these four models are missing or corrupted, or if the evaluator wishes to test additional models beyond them.

A.3.2 Basic Test

The artifact includes adversarial patch examples for functionality verification. Therefore, after completing the installation, the evaluator can perform a simple end-to-end test to verify that all required components of the APEX pipeline are functioning correctly. First, ensure the current working directory is the artifact root:

```
cd APEX
```

Clear previous outputs and run the complete pipeline on the provided input images:

```
rm -rf lama/output/* lama/LaMa_test_images/*
python3 auto_pipeline.py \
  --source <path-to-input-image-directory> \
  --output <path-to-output-directory>
```

If the test is successful, the pipeline will execute bounding-box extraction, patch localization, and image inpainting, and the output directory will contain the corresponding intermediate and final results.

Specifically, the output directory directly contains the following subdirectories:

```
<output-directory>/
|-- final_results/
|-- human_crops/
`-- locate_result/
```

final_results/ This directory contains the final visualization results for each input image:

```
final_results/
|-- 00000.jpg
|-- 00000_mask.png
|-- 00000_overlay.png
`-- ...
```

The files are interpreted as follows: `*_mask.png` are binary masks where black regions indicate detected adversarial patch areas; `*_overlay.png` are visualizations with detected masks overlaid and highlighted in red; `*.jpg` are restored images with detected patch regions removed.

human_crops/ This directory contains image crops produced by the bounding-box extraction stage:

```
human_crops/
|-- 00000_crop_0.jpg
|-- 00001_crop_0.jpg
|-- 00001_crop_1.jpg
`-- ...
```

Each image corresponds to a bounding box extracted by the human detector. Therefore, these crops reflect the output of the bounding-box extraction module (`bb_extraction.py`).

locate_result/ This directory stores the patch localization results for each cropped region:

```
locate_result/
|-- 00000_crop_0.png
|-- 00000_crop_0_detection_report.json
|-- 00000_crop_1.png
|-- 00000_crop_1_detection_report.json
`-- ...
```

The `*.png` files visualize the patch localization results for each crop, while the corresponding `*_report.json` files record patch location metadata. These metadata are used to map the localized patches back to their corresponding positions in the original images during the final stage.

In addition, the directory `APEX/lama/output/` contains files named `*_mask.png`, which store images where the black mask regions have been restored using the LaMa inpainting model.

A.4 Evaluation Workflow

A.4.1 Major Claims

(C1): *Applying APEX improves the detection performance of object detectors under adversarial patch attacks. This is demonstrated by the Overall Defense Performance results reported in § 5.2 (Effectiveness), where detectors achieve higher mAP after applying the APEX defense.*

(C2): *APEX generalizes across multiple types of adversarial patches, including non-naturalistic patches (Non-NAP), naturalistic patches (NAP), and infrared patches (IRP), while maintaining high localization recall and low false positive rates. This claim is supported by the Patch Localization Performance and False Positive Analysis results in § 5.2 (Effectiveness).*

(C3): *APEX remains effective under extreme adversarial scenarios, including tiny patches, irregularly shaped patches, and background-coherent patches. This is evidenced by the experimental results illustrated in Figure 5 and discussed in § 5.2 (Effectiveness).*

(C4): *APEX is capable of detecting adversarial patches even in non-photorealistic visual domains, such as abstract paintings. This claim is demonstrated by the results reported in § 5.4 (Effectiveness in Abstract Paintings).*

A.4.2 Experiments

(E1): *Overall Defense Performance (mAP Evaluation) [15 human-minutes + 1 compute-hour].*

Preparation: *This experiment uses Version 4.1 of the artifact. In practice, to use Version 4.1, one only needs to replace `APEX_mAP_origin` and `collect_apex_results.py` in `APEX_v4` with the updated versions, then copy `yolov8s_finetuned.pt` and `patch_v8s_ft_latest.jpg` into `APEX_v4`. Ensure that the updated dataset `APEX_mAP_origin` (500 images), the fine-tuned model weights (`yolov8s_finetuned.pt`), and the fine-tuned adversarial patch (`patch_v8s_ft_latest.jpg`) are all available under `APEX_v4`.*

Execution: *First, evaluate the detection performance (AP@0.5) before and after applying the patch:*

```
python add_eval.py \
```

```
  --patch   patch_v8s_ft_latest.jpg \
  --dataset APEX_mAP_origin \
  --weights yolov8s_finetuned.pt
```

Next, save all adversarial samples generated in the previous step:

```
python gen_patched_dataset.py \
  --patch   patch_v8s_ft_latest.jpg \
  --dataset APEX_mAP_origin \
  --out-dir APEX_mAP
```

Then, run the defense with:

```
python multigpu_no_lama.py \
  --dataset_dir APEX_mAP \
  --output_dir  APEX_masks \
  --num_gpus <x>
```

To restore a clean dataset:

```
python collect_apex_results.py \
  --masks-dir \
  APEX_masks/apex_output/final_result_merged \
  --original-dataset APEX_mAP \
  --output APEX_mAP_cleaned \
  --lama-dir lama \
  --batch-size 50
```

Finally, evaluate the cleaned dataset by running:

```
python eval.py \
  --dataset APEX_mAP_cleaned \
  --weights yolov8s_finetuned.pt
```

Results: *The experiment reports the AP@0.5 of the detector on adversarially patched images before and after applying the `APEX` defense. Due to the random placement of the adversarial patch, results may fluctuate by approximately 3% across runs. Across three independent runs, the patched AP@0.5 consistently falls in the range **40.4%–40.7%**, and the AP@0.5 after `APEX` defense consistently falls in the range **85.0%–87.8%**, compared to a clean-image AP@0.5 of **90.9%**. These results correspond to the YOLOv8/AdvPatch AP@0.5 after `APEX` defense as reported in Table 2.*

(E2): *Patch Localization Recall and False Positive Evaluation [10 human-minutes + 2 compute-hours].*

Preparation: *Ensure that the directory `APEX_dataset` exists and contains the following three subdirectories corresponding to different patch types: `Infrared_patch`, `Natural_patch`, and `NonNatural_patch`.*

Execution: *Run the following command to evaluate patch localization performance:*

```
python3 test_recall_fpr.py \
  --dataset_dir <path-to-patch-dataset> \
  --output_dir <path-to-output-directory> \
  --num_gpus <number-of-gpus> \
  --iou_threshold 0.5 \
  --patch_class_id 0
```

The argument `-dataset_dir` should be set to the path of each subdirectory under `APEX_dataset` (i.e., one run per patch type). The value of `-num_gpus` should be adjusted according to the available GPUs, and using more GPUs is recommended for improved efficiency. The IoU threshold is fixed to 0.5, consistent with the paper.

Results: *The script reports the total number of processed images, the average localization recall (Table 3), the average false positive rate (Figure 6), and the total number of correctly detected patches in the format `detected/total`. In addition, the patch detection results are saved under the output directory, following the same format illustrated in § A.3.2.*

(E3): *Evaluations on Extreme Cases [10 human-minutes + 1 compute-hour].*

Preparation: *Ensure that the directory `APEX_extreme_case` is available. All other preparation steps are identical to those in Experiment (E2).*

Execution: *Execute the same command as in Experiment (E2), with the argument `-dataset_dir` set to the paths of the three subdirectories under `APEX_extreme_case`, namely `tiny_patch`, `irregular_shape`, and `background_coherent` (one run per subdirectory).*

Results: *The output includes localization recall (Figure 5), false positive rate, and patch detection results saved under the output directory, following the same format as Experiment (E2) and § A.3.2.*

(E4): *Evaluations on Abstract Paintings [10 human-minutes + 30 compute-minutes].*

Preparation: *Ensure that the directory `APEX_abstract` is available. All other preparation steps are identical to those in Experiment (E2).*

Execution: *Execute the following command with `-dataset_dir` set to `APEX_abstract` and `-min_fp_area_ratio` set to `0.01` to match the paper's FPR threshold:*

```
python test_recall_fpr.py \
  --dataset_dir APEX_abstract \
  --output_dir  APEX_APEX_abstract_output \
  --iou_threshold 0.5 \
  --patch_class_id 0 \
  --min_fp_area_ratio 0.01
```

Results: *The output includes localization recall, false positive rate, and patch detection results saved under the output directory (Figure 9), following the same format as Experiment (E2) and § A.3.2.*

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/usenixsec2026/.