PatchCURE: Improving Certifiable Robustness, Model Utility, and Computation Efficiency of Adversarial Patch Defenses

Chong Xiang, Tong Wu, and Sihui Dai, *Princeton University;* Jonathan Petit, *Qualcomm Technologies, Inc.;* Suman Jana, *Columbia University;* Prateek Mittal, *Princeton University*

https://www.usenix.org/conference/usenixsecurity24/presentation/xiang-chong

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 33rd USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 33rd USENIX Security Symposium.

August 14–16, 2024 • Philadelphia, PA, USA

978-1-939133-44-1

USENIX Security '24 Artifact Appendix: PatchCURE: Improving Certifiable Robustness, Model Utility, and Computation Efficiency of Adversarial Patch Defenses

Chong Xiang[1], Tong Wu[1], Sihui Dai[1], Jonathan Petit[2], Suman Jana[3], Prateek Mittal[1]
[1]Princeton University, [2]Qualcomm Technologies, Inc., [3]Columbia University

A Artifact Appendix

A.1 Abstract

Our PatchCURE paper proposes a defense framework against adversarial patch attacks to approach the challenging trade-off problem between certifiable robustness, model utility, and computation efficiency. PatchCURE can achieve similar inference efficiency as undefended models when optimized for efficiency; it also achieves state-of-the-art robustness and utility performance across all different efficiency levels.

This artifact can reproduce all experimental results (*clean accuracy, certified robust accuracy, and inference throughput*) reported in the main body of the paper to support our claims. It is based on PyTorch and requires GPU support. We implemented Algorithm 1 (construction, inference, and certification procedures) of our PatchCURE paper.

Our source code is available at `https://github.com/inspire-group/PatchCURE/tree/40695a870e018b76cf5ec105ab36346c780e756d`. We further provide a detailed guide for evaluating our artifact at `https://github.com/inspire-group/PatchCURE/blob/40695a870e018b76cf5ec105ab36346c780e756d/reproducibility.md`.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

No destructive steps. We only use curated datasets that are publicly available.

A.2.2 How to access

We host our source code on GitHub at `https://github.com/inspire-group/PatchCURE`.

Specifically, we use this commit for the artifact evaluation: `https://github.com/inspire-group/PatchCURE/tree/40695a870e018b76cf5ec105ab36346c780e756d`.

A.2.3 Hardware dependencies

The artifact requires 4 CPU cores and 1 GPU. We tested it with Intel Xeon CPU and NVIDIA RTX A4000 GPU.

A.2.4 Software dependencies

The artifact is based on Python, PyTorch, `timm`, and other Python packages. All packages can be easily installed with pip or conda; we provide a list of required packages in `requirement.txt`. We ran our experiments on a machine install with Red Hat Enterprise Linux 8; but the artifact should be runnable on a different OS.

A.2.5 Benchmarks

We focus on the publicly available ImageNet. See our reproducing instructions for more details.

We focus on two families of image classifier models: ResNet, and Vision Transformer. We build models using `timm` as well as our customized code; we provide download links to our pre-trained weights. See our reproducing instructions for more details.

A.3 Set-up

A.3.1 Installation

1. Install Python 3.10.
2. Install GPU-compatible PyTorch 1.13.1.
3. Install other Python dependencies.
4. Clone the source code from
5. Download datasets.
6. Download pre-trained weights.

We provide detailed instructions and commands in our reproducing instructions.

A.3.2 Basic Test

Below is the command for the functionality test.

```
python main.py --model vitsrf14x2_masked
--patch-size 32 --mask-stride 1 --certify
--runtime --batch-size 4 --num-img 100
```

It should be finished in a few seconds and print the following information to the console.

```
Clean Accuracy: 0.82
Certified Robust Accuracy: 0.53
Throughput: 189.91022253293076 img/s
```

A.4 Evaluation workflow

Our experiment is based on the script `main.py`, in which we call our construction, inference, and certification procedures (discussed in Algorithm 1 of our paper). By running this script with proper command (we provide all necessary commands to run `main.py` in our reproducing instructions), we can read clean accuracy, certified robust accuracy, and inference throughput from the console. We can compare the obtained results with the results reported in the paper to validate the reproducibility of our paper.

A.4.1 Major Claims

(C1): PatchCURE can build defense instances that have similar efficiency as undefended models. This can be validated using E1 and E2, which provide Table 2 and Table 3 in our paper. See Section 4.2 of our paper for more discussions (its second paragraph).

(C2): PatchCURE provides a systematic way to balance the three-way trade-off between robustness, utility, and efficiency; it achieves state-of-the-art utility and robustness performance across different efficiency levels. This can be validated using E2, which provides Table 2, Figure 1, and Figure 6. See Section 4.2 of our paper for more discussions (its third, fourth, and fifth paragraphs).

(C3): Other minor claims as discussed in Section 4.2 and Section 4.3. They can be validated using E3.

A.4.2 Experiments

We provide estimated experiment runtime for the entire ImageNet dataset using 4 Intel Xeon CPUs and 1 NVIDIA RTX A4000 GPU. We note that some experiments might take up to a few days. To reduce experiment runtime, we also provide an option to run experiments on a random subset of the ImageNet datasets. We can add an argument `--num-img 1000` to only use 1000 images (1/50 of the entire set) to obtain an approximated evaluation result. We provide more details in reproducing instructions.

(E1): [undefended models] [10 human-minutes + 1 compute-hour]:

Preparation: See Section A.3. No extra preparation is needed.

Execution: Run commands as detailed in the section of "Table 2: vanilla undefended model performance" in our reproducing instructions.

Results: Read results from the console. The numbers should match Table 2 in our paper.

(E2): [PatchCURE defenses] [20 human-minutes + 3 compute-day (for one command) + 24 compute-hour (for all other commands)]:

Preparation: See Section A.3. No extra preparation is needed.

Execution: Run commands as detailed in the section of "Table 3: main results (as well as Figure 1 + Figure 6)" in our reproducing instructions. Feel free to add `--num-img 1000` to compute approximated results with shorter runtime.

Results: Read results from the console. The numbers should match Table 3 in our paper.

(E2): [PatchCURE defenses] [30 human-minutes + 3 compute-day (for one command) + 24 compute-hour (for all other commands)]:

Preparation: See Section A.3. No extra preparation is needed.

Execution: Run commands as detailed in other sections of our reproducing instructions.

Results: Read results from the console. The numbers should match other tables and figures presented in our paper.

Our algorithms are deterministic. Therefore, we do not expect any large variation in results if the experiments are conducted on the entire dataset. However, it is possible to have small mismatches ($< 1\%$) due to the imprecise float point computation on different hardware (e.g., GPUs).

A.5 Notes on Reusability

N/A

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/usenixsec2024/`.