USENIX Security '25 Artifact Appendix: CERTPHASH: Towards Certified Perceptual Hashing via Robust Training

Yuchen Yang, Qichang Liu,* Christopher Brix[1], Huan Zhang[2], and Yinzhi Cao
{yc.yang, qliu85, yinzhi.cao}@jhu.edu,
brix@cs.rwth-aachen.de, huan@huan-zhang.com
The Johns Hopkins University, [1]RWTH Aachen University, [2]UIUC

A Artifact Appendix

A.1 Abstract

CertPhash is the first certified perceptual hashing (PHash) system with robust training. CertPhash includes three different optimization terms: anti-evasion, anti-collision, and functionality. The anti-evasion term establishes an upper bound on the hash deviation caused by input perturbations, the anti-collision term sets a lower bound on the distance between a perturbed hash and those from other inputs, and the functionality term ensures that the system remains reliable and effective throughout robust training. This artifact includes the source code, dataset, and models, along with instructions to set up the environment, implement, and evaluate CertPHash.

A.2 Description & Requirements

A.2.1 How to access

The code and scripts used in our paper are available in the following GitHub repository, at commit `59ed6e6441e1ba06d09631ec900799eb204079ef`: https://github.com/Yuchen413/CertPhash/tree/main. It is also available at the Zenodo stable URL https://zenodo.org/records/14740844, with the `latest` release tag.

A.2.2 Hardware dependencies

Our artifacts require a Linux machine with 64GB of RAM and a GPU with 40 GB of graphics memory. We use an NVIDIA A100-PCIE-40GB GPU with CUDA Version 12.1 and Driver Version 530.30.02.

A.2.3 Software dependencies

We use Python 3.10 and Conda for package management. The required dependencies can be installed using:

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

Additionally, auto_LiRPA dependencies must be installed:

```
git clone git@github.com:Verified-Intelligence/
    auto_LiRPA.git
cd auto_LiRPA
python setup.py install
```

*This work was done when Qichang Liu was a summer intern at JHU.

A.2.4 Artifact hierarchy

The artifact includes the following three folders:

- `generate_phash`: PHash generation using existing algorithms.
- `train_verify`: Train and verify the CertPhash.
- `attack`: Functionality and empirical attack evaluations.

A.2.5 Datasets

We use datasets COCO, MNIST, CelebA, and NSFW-56K for training and evaluation. Due to ethical concerns, we cannot provide a direct download link for NSFW-56K, please follow the steps in the README of our GitHub repository. The other datasets can be downloaded at this link, which includes images and perceptual hashes used in our experiments. After downloading, unzip the file, name the folder as `data` and replace the existing `train_verify/data` folder with the downloaded one.

Then perform the following procedure for the datasets you need:

- **COCO**: Download images from this ADDITIONAL link and unzip them as `coco100x100`. Place the folder under `train_verify/data`.
- **MNIST**: Ensure a folder named `mnist` containing images and hashes is present.
- **CelebA**: Ensure a folder named `celaba_random` is present.

Please double check that the downloaded data files follow the structure as specified in `./train\_verify/data/put\_data\_here\_follow\_this.txt`:

```
-coco100x100
    -.jpg
-coco100x100_val
    -.jpg
-coco-train.csv
-coco-val.csv
-mnist
    -testing
        -.jpg
    -training
        -.jpg
    -mnist_test.csv
    -mnist_train.csv
-celeba_random
    -.jpg
```

A.2.6 Models

Our certified robust trained models, adversarial trained models and non-robust trained models have been placed in https://drive.google.com/drive/folders/1b7RbO-uDdlvsxgsxE4H-tjrdGVx7pVRu?usp=sharing In order to use our pretrained models, download the folder named `saved_models` and replace the existing `./train_verify/saved_models` with the downloaded version.

A.3 Set-up

A.3.1 Installation

The following steps set up the Conda environment for our Github repository:

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

A.3.2 Basic Test

To verify installation:

```
bash ./train_verify/test_env.sh
```

The expected output for `./train_verify/test_env.sh` should be similar to `./train_verify/test/train_log.txt`, a.k.a. something like the following:

```
Epoch 1, learning rate [0.0005], dir
    one_epoch_test
[ 1]: eps=0.00000048 active=0.3465 inactive
    =0.6532 Loss=0.4681 Rob_Loss=0.4681 Err
    =1.0000 Rob_Err=1.0000 L_tightness=0.5494
    L_relu=0.0023 L_std=0.8852 loss_reg=0.5518
    grad_norm=15.2293 wnorm=13.1858 Time=0.0517
...
Test without loss fusion
[ 1]: eps=0.00000048 active=0.3509 inactive
    =0.6491 Loss=0.4642 Rob_Loss=0.4642 Err
    =1.0000 Rob_Err=1.0000 L_tightness=0.0000
    L_relu=0.0000 L_std=0.8729 loss_reg=0.0000
    wnorm=14.8170 Time=0.0218
```

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): CertPHash maintains hash functionality across different datasets, achieving high ROC-AUC scores. This is proven by experiments (E1-1, E1-2) described in our paper's Section 6.1 whose results are illustrated in Figure 2.

(C2): CertPHash maintains certified robustness against evasion and collision attacks, achieving high Certified No Evasion Rates and Certified No Collision Rates. This can be verified through the experiments (E2-1, E2-2) in Table 2 and Table 3 described in our paper's Section 6.2.

A.4.2 Experiments

(E1-1) Functionality Evaluation for Trained PHash Models

Preparation: Install dependencies and datasets.

Execution: Navigate to `attack/` and run the following scripts. An example is provided below.

```
python benign0_func_check.py --dataset='
    coco_val' --target='
    photodna_nn_cert_ep1' --model='../
    train_verify/saved_models/
    coco_photodna_ep1/ckpt_best.pth'
```

The expected output of the console contains the names of all the transformations:

```
Original: 100%|--------------------| 1/1
    [00:00<00:00, 22310.13it/s]
Rotate: 100%|--------------------| 15/15
    [00:00<00:00, 32099.27it/s]
...
```

It will take around five minutes to calculate the PHash from different levels of different transformations. The generated hashes will be saved under folder `./attack/func_logs/coco_val_photodna_nn_cert_ep1`.

Then calculate the ROC-AUC via:

```
python benign0_func_AUC.py --dataset='
    coco_val' --target='
    photodna_nn_cert_ep1'
```

The expected output from the console contains ROC-AUC of transformations tested in our papers RQ1. For instance, the output for the Hue transformation should be as follows:

```
...
hue: -180     -150     -120     -90
    -60      -30      0        30       60
          90       120      150
ROC AUC: 1.0000  1.0000   1.0000   1.0000
    1.0000   1.0000   1.0000   1.0000
    1.0000   1.0000   1.0000   1.0000
Mean ROC AUC: 1.0000, Std ROC AUC: 0.0000
...
```

The full results will be saved in `./attack/func_logs/coco_val_photodna_nn_cert_ep1/coco_val_results.txt`. These results should align with our paper's RQ1.

(E1-2) Certified Robust PHash Training:

Preparation: Install dependencies and datasets.

Execution: Instead of using our trained model, you can also train models from scratch. Navigate to `train_verify` folder and run `train.py` with the robust training configuration file. An example of robust training with a perturbation epsilon of 0.0078 on COCO dataset is provided below.

```
python train.py --method=fast --config=
    config/coco.crown-ibp.json \
--eps=0.0078 --dir=saved_models/
    coco_photodna_ep2 \
--scheduler_opts=start=2,length=80 \
--lr-decay-milestones=120,140 --lr-
    decay-factor=0.2 \
--num-epochs=160 --model='resnet_v5' --
    lr=5e-4
```

Results: The robustly trained model will be saved under `saved_models/`. You can later use this model for verification.

(E2-1): Certified No Evasion Rate (CNER) Verification:

Preparation: Install dependencies and datasets.

Execution: Run `verify_evasion.py` with appropriate parameters. See the below example with verifying a model trained with $\varepsilon = 8/255$ under verifying noises $\varepsilon = 8/255$:

```
python verify_evasion.py --data=
    coco \
--epsilon=0.0312 --model='
    saved_models/coco_photodna_ep8
    /ckpt_best.pth'
```

Results: Compare the computed CNER values with those reported in the paper.

(E2-2) Certified No Collision Rate (CNCR) Verification:

Preparation: Install dependencies and datasets.

Execution: Run `verify_preimage.py`. An example is provided below.

```
python verify_preimage.py
```

Results: Compare the CNCR values with those reported in the paper.

A.5 Notes on Reusability

CertPHash can be adapted to other datasets and hashing models with minor modifications to the data-loading code. Additionally, we provide the implementation and evaluation scripts for our non-robust and adversarial-robust trained PHash model, as well as existing PHash systems, including PhotoDNA, PDQ, and NeuralHash. (Note that we cannot directly provide the extracted algorithms or models due to copyright restrictions.) More details can be found in the README of our GitHub repository.

A.6 Version

This submission follows the LaTeX template for Artifact Evaluation V20231005. Further details can be found at https://secartifacts.github.io/usenixsec2025/.