USENIX Security '25 Artifact Appendix: Enhanced Label-Only Membership Inference Attacks with Fewer Queries

Hao Li*
Institute of Software,
Chinese Academy of Sciences

Zheng Li*
Shandong University

Siyuan Wu
Institute of Software,
Chinese Academy of Sciences

Yutong Ye
Institute of Software,
Chinese Academy of Sciences

Min Zhang†
Institute of Software,
Chinese Academy of Sciences

Dengguo Feng
Institute of Software,
Chinese Academy of Sciences

Yang Zhang
CISPA Helmholtz Center for Information Security

A Artifact Appendix

A.1 Abstract

In this artifact, we demonstrate a label-only Membership Inference Attack called DHAttack, designed for Higher performance and Higher stealth, focusing on the boundary distance of individual samples to mitigate the effects of sample diversity, and measuring this distance toward a fixed point to minimize query overhead. The code primarily comprises three files: preprocessData.py, trainTargetModel.py and DHAttackBase.py. These files support the CIFAR10 and CIFAR100 datasets, as well as the VGG-16, ResNet-56, and MobileNetV2 models.

In this appendix, we summarize the key claims of the paper and demonstrate how they can be verified using this artifact. Our first claim is that DHAttack achieves effective results with fewer than 100 queries. Secondly, DHAttack achieves superior performance in most cases. It is important to note that completing all experiments takes several hundred hours. This is because our attack process requires training 256 reference models. While attacking a single target model only requires training these 256 reference models once (a process that takes tens of hours), verifying all six target models in the paper (excluding those in the appendix) requires six times that duration. In addition, Algorithm 2 in our paper defines the fixed sample as an input parameter, allowing it to be modified. Rather than directly using 'RGB-255', we assign the maximum pixel value across all dimensions to the sample (which is also an outlier) as a fixed point in this artifact, primarily for implementation convenience.

*The first two authors made equal contributions.
†Corresponding author.

A.2 Description & Requirements

This section outlines the experimental setup, including hardware and software requirements and relevant benchmarks used to produce results.

A.2.1 Security, privacy, and ethical concerns

There are no security, privacy, or ethical concerns associated with executing this artifact. All experiments are conducted solely on public datasets and widely used model architectures, ensuring that no sensitive or private data is involved.

A.2.2 How to access

Our artifact is at https://zenodo.org/records/14728863.

A.2.3 Hardware dependencies

Evaluating our artifact requires an NVIDIA GeForce RTX 2080 Ti or a more powerful GPU. Since GPU performance directly affects runtime, we recommend using an NVIDIA GeForce RTX 4090 for optimal efficiency. There are no strict requirements for the CPU or memory, though we recommend at least 64GB of RAM to ensure smooth execution.

A.2.4 Software dependencies

Our artifact is compatible with Windows. All necessary dependencies are listed in the requirements.txt file, with detailed installation instructions available in Section A.3.1.

A.2.5 Benchmarks

Our experiment requires the CIFAR10 and CIFAR100 datasets, which must be downloaded from their official web-

sites and placed in the designated folder. Detailed configuration instructions are provided in Section A.3.1.

A.3 Set-up

This section outlines the installation and configuration steps required to set up the environment for artifact evaluation.

A.3.1 Installation

The installation process is primarily divided into three stages: dependency installation, data preparation, and target model training.

Dependency installation. Create a new Python environment using Conda, specifying the environment name and Python version:

```
conda create -n env_name python=3.8.
```

Once the environment is created, activate it and use pip to install all the packages listed in the "`requirements.txt`" file. To do this, first activate the environment with:

```
conda activate env_name,
```

then run the following command to install the dependencies:

```
pip install -r requirements.txt.
```

Alternatively, you can manually install the dependencies listed in the "`requirements.txt`" file.

Data preparation. For CIFAR10, please place the CIFAR10 dataset files (downloaded from the official website, including "`data_batch_1`" to "`data_batch_5`" and "`test_batch`" files) into the "`.\data\cifar-10-download`" folder. For CIFAR100, please place the CIFAR100 dataset files (downloaded from the official website, including "`test`" and "`train`" files) into the "`.\data\cifar-100-download`" folder. In addition, for Linux systems, we provide two helper scripts, "`download_cifar10.sh`" and "`download_cifar100.sh`", which can automatically download the CIFAR10 and CIFAR100 datasets from the official website and place them into the appropriate directories.

Next, execute the following commands:

```
python preprocessData.py --dataset CIFAR10.
python preprocessData.py --dataset CIFAR100.
```

Target model training. Please run the commands to obtain the target models VGG-16, ResNet-56 and MobileNetV2 trained on CIFAR10:

```
python trainTargetModel.py --dataset CIFAR10
--classifierType vgg --num_epoch 100
python trainTargetModel.py --dataset CIFAR10
--classifierType resnet --num_epoch 100
python trainTargetModel.py --dataset CIFAR10
--classifierType mobilenet --num_epoch 100
```

Then, by changing the "`dataset`" parameter and running the three commands again, you can obtain target models trained on the CIFAR100 dataset. This process is expected to take approximately one hour.

A.3.2 Basic Test

You can use the code "DHAttackBase.py" we provide to verify that all required dependencies are installed correctly.

```
python DHAttackBase.py --dataset CIFAR10
--classifierType mobilenet
--num_epoch_for_refmodel 100 --disturb_num 5
```

If the process shows that reference models are being trained, it indicates that all dependencies have been installed correctly. At this point, you can safely terminate the process, as full execution would take tens of hours. Note that this command is only used to verify whether the required dependencies are correctly installed. Its performance will be low due to the "`disturb_num`" being set to only 5. For the formal experiment, please refer to Section A.4.

A.4 Evaluation workflow

In this section, we present the main claims of the paper along with the experiments that support them.

A.4.1 Major Claims

The major claims presented in our paper are as follows:

(C1): DHAttack achieves effective results with fewer than 100 queries. This is proven by the experiment (E1) described in Section 5.1, whose results are illustrated in Figures 5 and 6.

(C2): DHAttack achieves superior performance in most cases. This is proven by the experiment (E2) described in Section 5.1, whose results are illustrated in Table 3.

A.4.2 Experiments

The experiments supporting the main claims are detailed as follows:

(E1): [High Stealth] [30 human-minutes + 450(75*6) compute-hour + 220GB disk]: This experiment aims to verify that DHAttack can achieve an effective attack with only a small number of queries to the target model. We expect performance to fluctuate at query counts of 5, 10, 20, 30, 50, 100, or 200, but DHAttack generally achieves optimal results with fewer than 100 queries. As the query count increases (e.g., 100 or 200), the fixedBD measurement in our method becomes more sensitive to decision boundary complexity, causing unstable values across different reference models and affecting attack performance. Therefore, Figures 5 and 6 highlight performance fluctuations due to varying query numbers and demonstrate that optimal effectiveness is achieved with a small number of queries.

How to: To run the experiment, execute DHAttackBase.py, which generates one AUC value and one TPR at 0.001 FPR per run. Use the `classifierType` and `dataset` parameters to specify the target model and dataset (e.g., vgg and CIFAR10 correspond to the first subfigure in the top-left corner of Figures 5 and 6). After successfully attacking a specified target model and dataset for the first time, 256 reference models will have been trained (a process taking tens of hours). For subsequent attacks on the same target model and dataset, this step can be skipped by setting the trainRefModel parameter.

Preparation: If you have successfully completed data preparation and target model training in A.3.1, no further setup is required.

Execution: Please run the command:

```
python DHAttackBase.py --dataset CIFAR10
--classifierType vgg --num_epoch_for_refmodel
 100 --disturb_num 5
```

This process is expected to take approximately 60 hours. Once completed, record the AUC and the TPR at 0.001 FPR, as reported by our artifact. Next, run the following commands (a process that may take approximately ten hours or longer) and record the results for each:

```
python DHAttackBase.py --dataset CIFAR10
--classifierType vgg --num_epoch_for_refmodel
 100 --disturb_num 10 --trainRefModel False

python DHAttackBase.py --dataset CIFAR10
--classifierType vgg --num_epoch_for_refmodel
 100 --disturb_num 20 --trainRefModel False

python DHAttackBase.py --dataset CIFAR10
--classifierType vgg --num_epoch_for_refmodel
 100 --disturb_num 30 --trainRefModel False

python DHAttackBase.py --dataset CIFAR10
--classifierType vgg --num_epoch_for_refmodel
 100 --disturb_num 50 --trainRefModel False

python DHAttackBase.py --dataset CIFAR10
--classifierType vgg --num_epoch_for_refmodel
 100 --disturb_num 100 --trainRefModel False

python DHAttackBase.py --dataset CIFAR10
--classifierType vgg --num_epoch_for_refmodel
 100 --disturb_num 200 --trainRefModel False
```

These results will contribute to the DHAttack curve in the first subfigure of Figures 5 and 6, where the target model is VGG-16 trained on CIFAR10. Repeat these commands while adjusting the `classifierType` (i.e., resnet and mobilenet) and `dataset` (i.e., CIFAR100) parameters to obtain other performance curves for DHAttack.

Results: The `disturb_num` parameter specifies the number of queries made to the target model. You can then observe that when attacking each target model, DHAttack achieves optimal attack effectiveness with fewer than 100 queries. Moreover, in most cases, the performance is expected to surpass the baseline results reported in Figures 5 and 6.

(E2): [High Performance] [20 human-minutes + 20 compute-hour + 220GB disk]: This experiment aims to verify that DHAttack can achieve superior performance in most cases. Our expected results are generally consistent with those shown in Table 3. There may be slight fluctuations in the results. These differences are primarily due to the inherent randomness in training the reference models, such as random initialization. However, these variations do not impact the advantages of our approach.

How to: To run the experiment, execute DHAttackBase.py, which generates one AUC value and one TPR at 0.001 FPR per run. Set the dataset parameter to CIFAR10 and vary the classifierType between vgg, resnet, and mobilenet to obtain the results presented in Table 3.

Preparation: If you have successfully completed E1, no further setup is required.

Execution: Please run the commands:

```
python DHAttackBase.py --dataset CIFAR10
--classifierType vgg --num_epoch_for_refmodel
 100 --disturb_num 30 --trainRefModel False

python DHAttackBase.py --dataset CIFAR10
--classifierType resnet
--num_epoch_for_refmodel 100 --disturb_num
 50 --trainRefModel False

python DHAttackBase.py --dataset CIFAR10
--classifierType mobilenet
--num_epoch_for_refmodel 100
--disturb_num 50 --trainRefModel False
```

After executing each command, record the AUC and the TPR at 0.001 FPR, as reported by our artifact.

Results: The `dataset` and `classifierType` parameters define the target model, while the `disturb_num` parameter specifies the number of queries made to the target model. You can then observe that when attacking each target model, DHAttack achieves superior performance in most cases, consistent with the results presented in Table 3.

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/usenixsec2025/.