
# USENIX Security '26 Artifact Appendix: Certified in Theory, Broken in Practice: Assumption Gaps in Cryptographic Model Certification

Carter Luck, Olive Franzese-McLaughlin, Elisaweta Masserova,
Akira Takahashi, Antigoni Polychroniadou, Nicolas Papernot

## A Artifact Appendix

### A.1 Abstract

This artifact contains the scripts used to reproduce the data-forging attacks described in the paper. The attacks target three types of ML model auditing schemes—decision trees, neural networks, and XGBoost ensembles—and demonstrate that a malicious model trainer can craft a training set that causes the model to behave correctly on the audited sample while performing poorly (or unfairly) on the general population. The artifact supports reproduction of Figures 3–8.

### A.2 Description & Requirements

#### A.2.1 Security, privacy, and ethical concerns

None. The artifact runs self-contained ML experiments on public datasets and does not require network access, elevated privileges, or interaction with any external service.

#### A.2.2 How to access

The artifact is available at: https://doi.org/10.5281/zenodo.20337317

#### A.2.3 Hardware dependencies

No special hardware is required. All experiments were run on a standard CPU machine. We tested experiments on Ubuntu 24.04 with 13th Gen Intel(R) Core(TM) i5-1335U and 16 GB RAM.

#### A.2.4 Software dependencies

- Python 3.12
- conda (recommended) or pip
- Key packages: `xgboost 3.0.4`, `scikit-learn 1.7.1`, `torch 2.6.0`, `numpy 2.4.4`, `pandas 2.3.1`, `scipy 1.16.0`, `matplotlib 3.10.3`, `folktables 0.0.12`, `aif360`, `ucimlrepo`

All dependencies are specified in `environment.yml` (conda) and `requirements.txt` (pip).

#### A.2.5 Benchmarks

Three public tabular datasets are used:

- **Adult** (UCI) — income prediction, 48 842 records, 11 features. Included as `data/adult.csv`.
- **Default Credit** (UCI) — credit card default prediction, 30 000 records, 23 features. Included as `data/default_credit.csv`.
- **ACS Income** (folktables) — income prediction from the American Community Survey, 10 features. Included as `data/acs_income.csv`. Run `python data_download.py` to download the raw ACS data if needed.

### A.3 Set-up

#### A.3.1 Installation

1. Create and activate the conda environment (recommended):

   ```
   conda env create -f environment.yml
   conda activate adversarial-arborists
   ```

   Alternatively, install via pip (Python 3.12 required):

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

#### A.3.2 Basic Test

Run a quick smoke test with reduced parameters to verify the installation:

```
python xgboost_attacks.py \
  --n-estimators 50 --n-trials 1
```

Expected output (values will vary slightly):

```
Loading data/default_credit.csv ...
  30000 rows, 23 features, rate=0.221
  XGBoost: 50 trees, depth 10
  trial 0: audit=0.XXXX eval=0.XXXX
Results over 1 trials (xgboost):
  S_audit: 0.XXXX +- 0.0000
  S_eval : 0.XXXX +- 0.0000
```

The key indicator of a successful run is that `S_audit` accuracy is substantially higher than `S_eval` accuracy.

## A.4 Evaluation workflow

### A.4.1 Major Claims

**(C1):** A malicious trainer can craft $S'_{\text{train}}$ such that a decision-tree model achieves near-perfect accuracy on the audit set $S_{\text{audit}}$ while collapsing to near-random accuracy on the held-out evaluation set $S_{\text{eval}}$. This is shown in experiments (E1), (E2), and (E3), whose results are illustrated in Figures 3–6, produced by `*_evasion.py`.

**(C2):** The same attack generalizes to XGBoost ensembles. As the attack parameter $p$ increases from 0 to 1, audit accuracy rises toward 1.0 while real-world accuracy falls. This is proven by the experiment (E4) described in Appendix D.1 whose results are illustrated in Figure 7, produced by `xgboost_attacks.py --plot`.

**(C3):** The same attack generalizes to neural networks. This is proven by the experiment (E5) described in Appendix D.2 whose results are illustrated in Figure 8 (neural network variant), produced by `nn_attacks.py`.

### A.4.2 Experiments

**(E1): Decision-tree attacks (Figures 3 and 5)** [1 human-minute + 2 compute-hours + 0GB disk]: Run the per-dataset evasion scripts on their default settings.
**Preparation:** Ensure the conda environment is active. Download ACS data if needed:

```
python data_download.py
```

**Execution:**

```
python folktables_evasion.py \
  -v --csv ACS
python adult_evasion.py \
  -v --csv Adult
python credit_evasion.py \
  -v --csv Credit
python compas_evasion.py \
  -v --csv COMPAS
python german_evasion.py \
  -v --csv German
python crime_evasion.py \
  -v --csv Crime
python make_graph.py \
  --path data/ --target acc \
  --files ACS Adult COMPAS
python make_graph.py \
  --path data/ --target acc \
  --files German Credit Crime
```

Output can be saved to a file with `-o <file>`.
**Results:** Each of the first six scripts print accuracy on $S_{\text{audit}}$ and $S_{\text{eval}}$ across trials, along with fairness metrics. The values correspond to the rows of Table 2 and the data points in Figures 3 and 5. Audit accuracy should be substantially higher than evaluation accuracy. The last two scripts display three-panel figures. The first should match Figure 3 in the paper and the second should match Figure 5: as $p$ increases, the orange (Audit Performance) curve rises towards 1.0 and the blue (Real-World Performance) curve falls.

**(E2): Decision-tree fairness attacks (Figure 4)** [1 human-minute + 1 compute-hour + 0GB disk]: Run the per-dataset evasion scripts on fairness.
**Preparation:** Ensure the conda environment is active. Download ACS data if needed:

```
python data_download.py
```

**Execution:**

```
python folktables_evasion.py \
  -v --csv ACS --target fair
python adult_evasion.py \
  -v --csv Adult --target fair
python compas_evasion.py \
  -v --csv COMPAS --target fair
python make_graph.py \
  --path data/ --target fair \
  --files ACS Adult COMPAS
```

Output can be saved to a file with `-o <file>`.
**Results:** Each of the first three scripts print accuracy and fairness metrics on $S_{\text{audit}}$ and $S_{\text{eval}}$ across trials. The values correspond to the data points in Figure 4. Audit accuracy should be substantially higher than evaluation accuracy. The last script displays a three-panel figure. It should match Figure 4 in the paper: as $p$ increases, the orange (Audit Fairness) curve rises towards 1.0 and the blue (Real-World Fairness) curve falls.

**(E3): Decision-tree denial attack (Figure 6)** [1 human-minute + 20 compute-minutes + 0GB disk]: Run the per-dataset evasion scripts on denial-rate.
**Preparation:** Ensure the conda environment is active. Download ACS data if needed:

```
python data_download.py
```

**Execution:**

```
python folktables_evasion.py \
  -v --csv ACS --target zero
python make_graph.py \
  --path data/ --target zero \
  --files ACS
python make_graph.py \
  --path data/ --target acc \
  --files ACS
```

Output can be saved to a file with `-o <file>`.
**Results:** The first three script prints accuracy and denial rate on $S_{\text{audit}}$ and $S_{\text{eval}}$ across trials. The values correspond to the data points in Figure 6. Audit accuracy should be substantially higher than evaluation accuracy, and evaluation denial rate should be higher than audit denial rate. The last two scripts each display a figure. They should match Figure 6's left and right graphs respectively: as $p$ increases, the audit denial rate should stay near its original value, bending upwards slightly

near $p = 0.5$, while the real-world denial rate rises towards 1.0; meanwhile, real-world accuracy increases towards 1.0 while the audit accuracy slightly decreases as $p$ increases.

**(E4): XGBoost attack — Figure 7** [1 human-minute + 10 compute-minutes + 0GB disk]: Reproduce the $p$-sweep plot showing Audit Performance vs. Real-World Performance across all three datasets.

**Preparation:** Ensure the conda environment is active.

**Execution:**
```
python xgboost_attacks.py \
  --plot --output figure7.png
```
This sweeps $p \in \{0.0, 0.1, \ldots, 1.0\}$, runs 5 trials per value, and saves a three-panel figure to `figure7.png`. Runtime is approximately 10 minutes on a standard CPU.

**Results:** `figure7.png` should match Figure 7 in the paper: as $p$ increases, the orange (Audit Performance) curve rises toward 1.0 and the blue (Real-World Performance) curve falls, with narrow $\pm$std shading across all three datasets.

**(E5): Neural-network attack** [1 human-minute + 8 compute-hours + 3GB disk]: Reproduce the neural-network variant of the attack.

**Preparation:** Ensure the conda environment is active.

**Execution:**
```
python nn_attacks.py --sweep
```
This sweeps over the size of $S'_{\text{train}}$ by adding more copies of $S_{\text{audit}}$, charting the accuracy of the resultant models. Runtime may be up to several hours, as this involves training 10 large neural networks.

**Results:** The displayed chart should match Figure 8 in the paper: the blue curve (Audit Performance) stays relatively constant, while the orange curve (Real-World Performance) is consistently higher, growing to 1 as the x-axis increases.

## A.5 Notes on Reusability

The attack scripts accept command-line arguments that allow easy experimentation beyond the paper's settings: See the `README.md` in the repository for a full description of all options for each script.

## 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/usenixsec2026/.