USENIX Security '26 Artifact Appendix: RETROFIT: Continual Learning with Controlled Forgetting for Binary Security Detection and Analysis

Yiling He[1,⋆✉], Junchi Lei[2,⋆], Hongyu She[2],
Shuo Shao[2], Xinran Zheng[1], Yiping Liu[3], Zhan Qin[2], Lorenzo Cavallaro[1]
[1] University College London, London, United Kingdom
[2] Zhejiang University, Hangzhou, China
[3] Alibaba Group, Beijing, China

A Artifact Appendix

A.1 Abstract

Deep learning models for binary security degrade as threat landscapes shift and code representations evolve. Continual learning is a natural fit, but existing approaches either replay historical data or permit unconstrained updates, limiting applicability in data-sensitive settings. In this paper, we propose RETROFIT, which controls forgetting at each update without accessing historical data. It consolidates previously trained and newly fine-tuned models as teachers of legacy and emergent knowledge via parameter merging. Forgetting control is achieved via low-rank and sparse subspace constraints, with a confidence-guided arbitration dynamically aggregating knowledge from both teachers. On malware detection under temporal drift, RETROFIT improves retention (PTR metrics) by 20.2%-38.6% over continual learning baselines, surpassing the oracle upper bound on new data. On binary summarization across decompilation levels, it achieves over 2× the BLEU score of prior transfer-learning approaches on stripped binaries and outperforms all baselines in cross-representation generalization.

This artifact highlights two goals: (1) confirming that the implementation is available and functional, and can reproduce the main reported metrics and trends of RETROFIT presented in the paper; and (2) confirming that the complete experimental workflow, equipped with end-to-end scripts for both tasks covering training, parameter merging, and evaluation, is available and functional.

A.2 Description & Requirements

The artifact contains two independent implementation directories: `Malware_Detection/` and `Binary_Analysis/`, which should be run in their corresponding dependency environments.

A.2.1 Security, privacy, and ethical concerns

The malware-detection workflow uses pre-extracted Android feature dictionaries and does not require evaluators to execute APKs. The binary-analysis workflow uses public datasets and pretrained models and does not execute unknown binary programs. The artifact does not collect personal information.

A.2.2 How to access

The artifact is permanently available through the Zenodo concept DOI:

https://doi.org/10.5281/zenodo.20339140.

The concept DOI represents all versions of the artifact. The README describes the repository layout, dependencies, data preparation, and experiment commands. Due to licensing and redistribution constraints, we provide instructions for obtaining the Transcendent dataset, the CAPYBARA dataset, and the CodeT5-base model through their official channels. To further facilitate reproducibility, we also provide the processed Transcendent features and the models used in our main results.

A.2.3 Hardware dependencies

The paper-level results were obtained on NVIDIA RTX A6000 GPU servers. `Malware_Detection/` can also run on CPU. `Binary_Analysis/` is more computationally demanding, so CUDA-capable NVIDIA GPUs are recommended for the full workflow.

For both tasks, the artifact provides lightweight evaluation or plotting paths using released intermediate artifacts, allowing the main reported figures to be obtained quickly on CPU without rerunning the complete training pipelines. For the `Malware_Detection/` lightweight reproduction path, the released processed inputs hosted on Hugging Face occupy approximately 15 GB, including both training and test data. The Figure 5 evaluation-only path requires approximately 3.8 GB of test inputs.

⋆ Co-first authors. ✉ Corresponding author: heyilinge0@gmail.com

A.2.4 Software dependencies

The two components should be installed in separate environments using `Malware_Detection/requirements.txt` and `Binary_Analysis/requirements.txt`. The binary-analysis workflow uses the BinT5 Docker environment with Python 3.10.

The requirements files specify the Python-package dependencies required for evaluation and visualization; the binary-analysis workflow additionally relies on the BinT5/CodeT5 environment described in the README.

A.2.5 Benchmarks

The benchmarks cover malware detection under temporal drift and binary summarization across decompilation levels. For malware detection, we use the Transcendent dataset and MLP classifiers. For binary summarization, we follow the CAPYBARA setting and use a CodeT5-based backbone.

A.3 Set-up

This section gives the minimal setup steps for basic checks; full setup and reproduction commands are in the README.

A.3.1 Installation

For `Malware_Detection/`, install:

```
pip install -r \
    Malware_Detection/requirements.txt
```

For the Figure 5 evaluation-and-plotting workflow, download the released malware-detection models and dimension-reduced Transcendent inputs on Hugging Face: mlp-transcendent-cl. Place the released checkpoints under `Malware_Detection/model/` and the dimension-reduced input files under `Malware_Detection/input/`.

For `Binary_Analysis/`, use the BinT5-compatible environment in the README. A typical container setup starts with:

```
docker pull aalkaswan/bint5
docker run -i -t \
    --name retrofit_binary \
    --gpus all \
    -v $(pwd):/data \
    aalkaswan/bint5 /bin/bash
cd /data/
```

The binary-analysis checkpoints are released on Hugging Face: codet5-capybara-retrofit.

A.3.2 Basic Test

For `Malware_Detection/`, run:

```
python Malware_Detection/run_all.py check
```

This command performs a lightweight check of the training, evaluation, and drawing entry points used by the complete malware-detection workflow. It does not train models, read benchmark data, require released checkpoints, or require GPU execution. A successful run exits with status code 0 without Python tracebacks or error messages.

For `Binary_Analysis/`, run the following check inside the BinT5 environment:

```
python - <<'PY'
import sys
import torch, transformers, peft, safetensors
sys.path.insert(0, 'Binary_Analysis/Util')
import retrofit_config
print('RETROFIT binary basic test passed')
PY
```

The malware-detection check succeeds silently, while the binary-analysis check should print `RETROFIT binary basic test passed`.

A.4 Evaluation workflow

This section covers the RETROFIT results in Figure 5 and Figure 6.

A.4.1 Major Claims

(C1): In malware detection under temporal drift, RETROFIT adapts to newly arriving yearly data while mitigating forgetting on historical years, improving retention over continual-learning baselines while maintaining competitive performance on new yearly data. This claim corresponds to Figure 5 in the paper and the malware-detection portion of Table 1. The artifact supports this claim through the `Malware_Detection/` workflow, which covers preprocessing, continual-training, RETROFIT parameter merging, evaluation, and Figure 5 generation, including year-wise old/new F1, AUT, and PTR metrics.

(C2): In binary summarization across decompilation levels, RETROFIT improves continual adaptation and cross-representation generalization. This claim corresponds to Figure 6 in the paper. The artifact supports this claim through the RETROFIT reproduction workflow in `Binary_Analysis/`, using released fine-tuned and RETROFIT-merged BinT5 checkpoints for direct evaluation and providing scripts to reproduce the RETROFIT evaluation and Figure 6 results.

A.4.2 Experiments

(E1): Malware detection under temporal drift (C1) [about 5–10 human-minutes + about 3.5 GPU compute-hours for the complete RETROFIT workflow; about 21 minutes for the Figure 5 evaluation-and-plotting workflow with released artifacts].
How to: We provide two reproduction paths. The first path runs the complete RETROFIT workflow, including yearly training, RETROFIT parameter merging, evaluation, and figure generation. The second path is the Figure 5 evaluation-and-plotting workflow, which uses released models and dimension-reduced inputs as a lightweight reproduction check. Its evaluation output also contains the year-wise old/new F1 values used for the malware-detection part of Table 1.
Preparation: Install dependencies following the README. For the complete RETROFIT workflow, prepare the processed Transcendent feature files and yearly train/test splits. For the Figure 5 evaluation-and-plotting workflow, download the released checkpoints and the required dimension-reduced test inputs.
Execution: To run the complete RETROFIT workflow, execute:

```
python Malware_Detection/run_all.py all
```

This command trains the 2014 base model, performs yearly updates from 2015 to 2018, merges old and newly adapted models with RETROFIT, evaluates yearly F1/AUT/PTR metrics, and draws Figure 5.
To run the Figure 5 evaluation-and-plotting workflow directly from the released models, execute:

```
python Malware_Detection/run_fig5.py
```

This command runs `eval_fig5.py` to compute year-wise old/new F1 scores, AUT, and PTR from the released checkpoints and reduced inputs, and writes them to `fig5_results.json`. The old/new F1 entries in this JSON provide the malware-detection values for Table 1, while `draw_fig5.py` uses the same evaluation output to render Figure 5.
Results: The complete workflow produces trained yearly models, evaluation summaries, and Figure 5. The Figure 5 evaluation-and-plotting workflow produces `fig5_results.json`, from which the malware-detection old/new F1 values in Table 1 can be obtained, and generates the Figure 5 visualization. Due to differences in hardware, software versions, random seeds, and numerical non-determinism, the reproduced values may differ slightly from the exact values in Figure 5, but the overall trend should remain unchanged: RETROFIT retains stronger historical performance while maintaining competitive adaptation to newly arriving yearly data.

(E2): Binary summarization across decompilation levels (C2) [about 10–20 human-minutes + about 70 GPU compute-hours for the complete RETROFIT training workflow; about 30 seconds for the Figure 6 plotting workflow from released output files].
How to: We provide two lightweight reproduction paths that can be run on CPU. The first path runs the binary-summarization evaluation workflow to generate summaries and metrics. The second path is the Figure 6 plotting workflow, which computes the plotted values directly from existing `Binary_Analysis/Output/` files.
Preparation: Configure the CodeT5-based environment and CAPYBARA data following the README.
Execution: Run the test interface:

```
CODET5_SH_DIR=/path/to/CodeT5/CodeT5/sh \
    bash Binary_Analysis/CodeT5_test.sh
```

To compute additional metrics, run:

```
python Binary_Analysis/Eval_Metrics.py \
    --base_dir Binary_Analysis/Output
```

To draw Figure 6 directly from the evaluation outputs, run:

```
python Binary_Analysis/draw_fig6.py
```

Results: The evaluation workflow generates summaries and computes BLEU and auxiliary generation metrics. The Figure 6 plotting workflow reads `Binary_Analysis/Output/`, computes the plotted values, and outputs the Figure 6 content in about 30 seconds. For consistency with the original BinT5 comparison, the Base/stripped cell in Figure 6 follows the BinT5 paper's reported result (7.19).

For workflows involving retraining or fresh inference, exact values may differ slightly from the paper because of hardware differences, random seeds, and non-determinism; the expected result is to reproduce the same overall trends.

A.5 Notes on Reusability

The two implementation directories can be reused independently: `Malware_Detection/` for temporally organized malware features, and `Binary_Analysis/` for CodeT5-based summarization tasks prepared in the upstream CodeT5 format.

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/.