USENIX Security '26 Artifact Appendix: <Fend for Yourself! Backdoor Purification in Federated Graph Learning with an Evolving Knowledge Anchor>

Chengcheng Zhu
Nanjing University

Yunlong Mao *
Nanjing University

Jiale Zhang
Yangzhou University

Bosen Rao
Yangzhou University

Sheng Zhong
Nanjing University

A Artifact Appendix

A.1 Abstract

This appendix describes the software artifact that implements the algorithms and evaluations presented in the paper "Fend for Yourself! Backdoor Purification in Federated Graph Learning with an Evolving Knowledge Anchor". Specifically, the artifact package includes the complete source code of GBHINDER, along with the baseline methods used for comparison in the experiments, including multiple backdoor attack strategies and defense mechanisms in Federated Graph Learning (FedGL). The implementation supports a variety of benchmark graph datasets and several graph neural network models. Additionally, the artifact provides shell scripts to facilitate the reproduction of experiments and a `README.md` file offering detailed instructions for setup and usage.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

The primary focus of our research is purely defensive, designed to safeguard FedGL models against malicious backdoor attacks. Although this artifact includes implementations of attack methodologies, they are provided solely for the purpose of evaluating defense mechanisms and fostering further research into robust federated learning. Consequently, the ethical implications of this work are positive, as it aims to secure technologies frequently deployed in safety-critical domains such as finance and healthcare. Furthermore, we exclusively utilized public, open-source benchmark datasets, ensuring that no private or personally identifiable information was handled or compromised during our research.

*Corresponding author, email: maoyl@nju.edu.cn.

A.2.2 How to access

The source code and datasets are available on GitHub (https://github.com/CBond00/GraphFL-defense) and Zenodo (https://doi.org/10.5281/zenodo.17898577).

A.2.3 Hardware dependencies

The artifact requires the following operating system and hardware resources:

- **CPU:** AMD Ryzen 9 9950X.
- **RAM:** 32 GB of system memory.
- **GPU (optional):** The experiments were conducted on an NVIDIA GeForce RTX 5080 GPU. Other comparable or higher-end consumer-grade GPUs are also acceptable.
- **Storage:** Sufficient disk space is required to download benchmark datasets and to store trained models, intermediate checkpoints, and experiment logs.

The specifications listed above correspond to the hardware configuration used in our experiments. Similar or more powerful hardware configurations are also suitable for running the artifact.

A.2.4 Software dependencies

The artifact requires the following operating system and essential software packages:

- **OS:** Windows or Linux.
- **Python Version:** Python 3.9.
- **Packages:** PyTorch, Pytorch_geometric, Pytorch_sparse, NetworkX, NumPy, Matplotlib, Scikit-learn, Hdbscan, and PyYAML.

Please refer to the instructions provided in the `README.md` file in our repository for detailed environment setup and dependency installation guidelines.

A.2.5 Benchmarks

We mainly conduct experiments on 4 public real-world datasets: NCI1, PROTEINS, DD, and AIDS. Other datasets, such as MUTAG and COLLAB, are also supported. These datasets are automatically downloaded during the first run of the script.

A.3 Set-up

Please verify that the hardware and software dependencies are satisfied and download the source code from the repository. The environment required for running this artifact is specified in the `requirements.txt` file. Please ensure that torch is the GPU version and that the version matches the geometric library. To set up the environment, open a terminal and execute the following commands:

```
conda create -n FedGLenv python=3.9
conda activate FedGLenv
conda install conda-forge::pytorch-gpu
conda install conda-forge::pytorch_geometric
conda install conda-forge::pytorch_sparse
pip install -r requirements.txt
```

A.3.1 Basic Test

The test runs a short federated learning session with a backdoor attack enabled over a limited number of rounds.

```
python main.py --dataset AIDS --model GIN --
    ↪ start_round 0 --end_round 600 --
    ↪ no_of_total_participants 10 --
    ↪ no_of_participants_per_round 5 --
    ↪ resumed_model False --attack_method
    ↪ motif --no_of_adversaries 2 --
    ↪ poisoned_start_round 500 --
    ↪ poisoned_end_round 600 --injection_ratio
    ↪  0.5 --trigger_ratio 0.2 --trigger_num 3
    ↪  --defense_method nodefense --benign_lr
    ↪ 0.01 --poisoned_lr 0.01
```

You can also run an equivalent instance directly using the following command.

```
python test_quick.py
```

If the artifact is set up correctly, the following behavior should be observed: The specified dataset is automatically downloaded to the `./data` directory if it does not already exist. A new directory `saved_models/{timestamp}/` is created, containing `log.txt` and `params.yaml`.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): Attackers need only participate for a limited number of rounds (e.g., 10 rounds) to implant a backdoor; significantly, this backdoor effect remains persistent even after the adversaries exit the aggregation process, and the FL system continues with only benign participants. This is substantiated by experiment (E1), as detailed in Section 4 of the paper, with results presented in Figure 1.

(C2): GBHINDER is able to effectively defend against a wide range of backdoor attack methods across multiple benchmark datasets, significantly outperforming existing baseline defense approaches. This claim is supported by the experiment (E2), with the corresponding results reported in Table 1.

(C3): GBHINDER remains effective under different adversarial constraints, including varying poisoning ratios, proportions of malicious participants, and heterogeneous data distributions. This is validated by experiment (E3), with results presented in Figures 3 and 4.

A.4.2 Experiments

(E1): *[Backdoor attack effectiveness] [10 human-minutes + 1 compute-hour + 1GB disk]:* This experiment validates the effectiveness of various backdoor attacks in the absence of defense mechanisms. Specifically, it demonstrates that the backdoor effect remains persistent within the federated learning process even after the active attack phase has ceased.

How to:

Preparation: Ensure that the Conda environment is created and activated. Upon the first execution of the code, the datasets corresponding to the specified command will be automatically downloaded to the `./data` directory.

Execution: To accelerate the experimental process, first train a clean pre-trained global model. This model can be loaded directly in subsequent experiments to skip the initial training phase.

```
python main.py --dataset AIDS --model GIN --
    ↪ start_round 0 --end_round 500 --
    ↪ no_of_total_participants 10 --
    ↪ no_of_participants_per_round 5  --
    ↪ defense_method nodefense --
    ↪ resumed_model False --benign_lr 0.01
    ↪  --GPU_id 0
```

After obtaining the clean model, you can set `-resumed_model` to `True` (or ignore the pre-trained model to start from scratch). Configure the attack strategy (e.g., `motif`), the number of adversaries (e.g., `2`), and other attack parameters. Set the attack window to rounds 500–600, and extend the total training duration to 1000 rounds to observe the persistence of the backdoor.

Results: The datasets are saved in `./data`. After experiments complete, results will be saved in the `saved_models/timestamp/` directory: `global.csv`, `log.txt`, `params.yaml`, and `saved_model_*.pt.tar`.

(E2): *[Defense effectiveness] [10 human-minutes + 5 compute-hour + 5GB disk]:* This experiment evaluates

the performance of varying defense mechanisms against different attack strategies across multiple datasets. It compares GBHINDER against the baseline defense methods presented in Table 1 of the paper.

How to:

Preparation: Ensure that the Conda environment is created and activated. If a clean pre-trained global model is required, please follow (E1) to obtain the checkpoint and reuse it for the following evaluations.

Execution: To run the complete comparative evaluation automatically, execute the provided shell script:

```
python test_full_evaluation.py
```

Alternatively, to evaluate arbitrary combinations of attack and defense strategies, you can execute the main script manually. Please refer to the `README.md` for detailed configuration options.

```
python main.py --dataset AIDS --model GIN --
    ↪ start_round 0 --end_round 1000 --
    ↪ no_of_total_participants 10 --
    ↪ no_of_participants_per_round 5  --
    ↪ attack_method [optional] --
    ↪ no_of_adversaries 2 --
    ↪ poisoned_start_round 500 --
    ↪ poisoned_end_round 600 --
    ↪ injection_ratio 0.5 --trigger_ratio
    ↪ 0.2 --trigger_num 3 --defense_method
    ↪  [optional] --GPU_id 0
```

The optional fields can be replaced with the desired backdoor attack method and defense method supported by the artifact.

Results: The output format and directory structure are consistent with (E1).

(E3): *[Defense robustness] [10 human-minutes + 5 compute-hour + 5GB disk]:* This experiment assesses the robustness of the defense mechanisms under varying adversarial settings. Specifically, it evaluates performance across a range of poisoning ratios $(0.1, 0.3, 0.5, 0.7, 0.9)$ and varying proportions of adversaries $(10\%, 20\%, 30\%, 40\%, 50\%)$.

How to:

Preparation: Ensure that the Conda environment is created and activated.

Execution: To run the full robustness benchmarks, execute the following scripts:

```
python test_injection_ratio.py
python test_client_num.py
```

Alternatively, to manually test specific poisoning ratios or adversary counts, refer to the execution configuration in (E2) and modify the `-injection_ratio` and `-no_of_adversaries` parameters accordingly.

Results: The output format and directory structure are consistent with (E1).

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/usenixsec2026/.