Backdoored Model Detection

1 Introduction

As deep neural networks (DNNs) become increasingly prevalent in mission-critical applications, their security vulnerabilities are drawing greater attention. A notable threat is backdoor learning, where an imperceptible backdoor can be embedded into a model by maliciously altering the training data or manipulating the training process. This poses a significant risk, especially considering the common practice of downloading unverified datasets or checkpoints for model training and fine-tuning or outsourcing the training process to third-party platforms. For more information about backdoor attacks, we highly recommend you to read this survey: https://arxiv.org/pdf/2007.08745.

Therefore, detecting backdoored models is crucial, particularly in security-sensitive applications. While numerous methods have been developed to identify whether a model contains a backdoor, none have consistently achieved satisfactory results across all types of threat models, including variations in model architecture, data-poisoning techniques, and image classification datasets (we recommend you to read this paper: `https://arxiv.org/abs/2205.06900`).

In this problem, a specific threat model is defined, with the objective of developing a detection function that can distinguish between backdoored and clean models. The details of the threat model are provided below.

2 Threat Model

- **Attacker Capabilities**
  - **Data Poisoning and Training Influence:**
    - Attackers can poison training data and influence the training process to embed backdoors into models.
  - **Trigger Visibility and Coverage:**
    - Stealthy to Overt Modifications: Attackers can deploy triggers that range from undetectable to more noticeable modifications.
    - Local and Global Coverage: Triggers can affect specific parts of a sample or the entire sample.
    - Sample-Specific Attacks: Attacks can be tailored to specific samples, complicating detection.
    - Label-Consistent Mechanisms: Attackers can use poisoned inputs labeled according to their visible content, leading to misclassification during inference.
  - **Attack Types:**
    - All-to-One: A single target class is selected, and whenever the input contains the trigger, the attacker wants to make the classifier output the selected target class.
  - **Model Training:**
    - Models can be trained adversarially or non-adversarially.
- **Attacker Goals**
  - Embed Backdoors: Ensure the model contains backdoors that lead to misclassification during inference.
  - Maintain Stealthiness: Create triggers that may be undetectable or hard to detect, even under scrutiny.

  - Evade Detection: Implement attacks that complicate detection efforts, especially those that are sample-specific or label-consistent.

- **Defender Capabilities**
  - **Model-Only Detection:**
    - The defender receives the model and has access to a small set of clean samples from the same distribution as the training data (1% of the image classification test dataset the model trained on).
  - **No Prior Knowledge Required:**
    - The detection mechanism operates without any prior knowledge of the specific type of attack or the nature of the trigger involved.

- **Defender Goals**
  - Detect Backdoors: Identify if the model has been compromised with a backdoor.

3 Datasets

To assess the proposed discrimination functions, two datasets are provided:

- **Evaluation dataset (`https://huggingface.co/datasets/abbasfar/backdoor_attack_evaluation_dataset/blob/main/eval_dataset.zip`):**
  - This dataset is fully accessible and can be downloaded via the preceding link.
  - You can use this dataset to evaluate your answer before submission.
  - Each data sample is saved in a separate folder.
  - Models are trained on numerous image classification datasets (e.g., CIFAR10, MNIST, etc.).
  - A few backdoor attack types are used to train models (e.g., Badnet, Blended, etc.).
  - All models have the architecture PreActResNet18 (you can download the architecture implementation in PyTorch from here: `https://huggingface.co/datasets/abbasfar/backdoor_attack_evaluation_dataset/blob/main/preact_resnet.py`).
- **Test Dataset:**
  - You can test your answer on this dataset only by submitting your answer on the server.
  - Data samples are directly given to the submitted functions in the arguments.
  - Models in this dataset may be trained on different image classification datasets.
  - Models in this dataset may be trained using different backdoor attack types.
  - All models have the same architecture PreActResNet18.

4 Evaluation Metric

To score the submitted answers, the accuracy metric is used:

$$\text{Accuracy Score} = \frac{\text{\# of correct predictions}}{\text{\# of data samples}}$$

Please note that during Phase 2, the leaderboard is calculated with 50% of the test data. The final results will be based on the other 50%, so the final standings may be different.

5 Evaluation Dataset Format

The dataset contains 50 folders, each corresponding to a data sample. Each folder has the following files:

- **model.pt:** contains the parameters of a model trained on an image-classification dataset.
- **metadata.pt:** contains a dictionary with items such as:
  - **num_classes:** The number of classes the image-classification dataset has.
  - **test_images_folder_address:** Path to a folder containing clean test images.
  - **transformation:** The transformation applied to test images before feeding them to the model.
  - **ground_truth:** An integer showing whether the model is backdoored (0) or clean (1).
- **test_dataset:** a folder in which there are 1% of clean images the model is trained on. The names of the files that images are saved in have the following format: `{index}_{label}.jpg`.
  - **index:** a number assigned to each image to ease the access of files.
  - **label:** the correct label of the image in the original image-classification dataset.

6 Answer Format

Your submission should contain a Python file with the name `main.py` including a method called `backdoor_model_detector`. You are allowed to create other files and define other classes or functions, but this method will be called for the evaluation process. Other details are described as follows:

```python
def backdoor_model_detector(model, num_classes, test_images_folder_address, transformation):
    """
    Args:
        model: A PreActResNet18 model trained on an image-classification dataset
        num_classes: The number of classes the image-classification dataset contains
        test_images_folder_address: Path to a folder containing clean test images.
        transformation: The transformation applied to test images before feeding them to the model

    Returns:
        int: 1 if the input model is detected as a backdoored model, 0 otherwise.
    """
    return None
```

In addition to Python files, you can submit a `requirements.txt` file to specify the packages required for your code to run. The following packages are pre-installed on our server, and you may use the specified versions to minimize potential conflicts:

- `numpy == 1.26.4`
- `torch == 2.4.0`

For submission, create a repository in Hugging Face Hub and upload your files on it. Then, provide us with the repository URL along with a read access token. Ensure that both `main.py` and `requirements.txt` are placed in the root directory of the repository.

7 Constraints

The entire evaluation of your answer is done on one Nvidia RTX 4090 GPU, with 24 Gigabytes of VRAM.

- The size of the file you upload should be no more than 1 MB.
- There should not be more than one constant parameter (e.g., a threshold to discriminate models trained on the evaluation set) in the submitted file.
- No internet access will be granted during the test process.
- There is a 1-minute limit for computation of the answer for each data sample.
- You are strictly prohibited from reading or writing any files in your solution.

8 Steps to Set Up Your Environment

To ensure our judging environment matches yours, please provide your requirements file. Our environment runs on Python 3.10.11, so it is recommended to use the same version to avoid conflicts. You can download it from: Python 3.10.11 download. **Note: Please avoid using `conda` for creating environments, as it may lead to inconsistencies in the judging environment.**

1. **Install Virtualenv**:
   Install `virtualenv` by running:

   ```
   python -m pip install --user virtualenv
   ```

   If you receive a warning indicating `virtualenv` is not accessible, use the full path to the virtualenv script or add it to your PATH environment variable.

2. **Create Virtual Environment**:
   Create a new environment using Python 3.10.11:

   ```
   virtualenv [env-name] -p python3.10.11
   ```

   Replace `[env-name]` with your desired environment name.

3. **Activate the Environment**:

   - **Linux**:

     ```
     source /path/to/env/bin/activate
     ```

   - **Windows**:

     ```
     /path/to/env/Scripts/activate
     ```

   Once activated, the environment name will appear before your prompt.

4. **Install Requirements**:
   After activating the environment, install your required packages.

5. **Verify Python Version**:
   Ensure the Python version matches the judging system by entering the Python interpreter.

6. **Export Requirements**:
   Once all packages are installed, export them to a `requirements.txt` file:

   - **Linux**:

```
echo "#$(python --version)" > requirements.txt && pip freeze >> requirements.txt
```

- **Windows**:

```
(<nul set /p=#&python --version & pip freeze) > requirements.txt
```