Rethinking the Reverse-engineering of Trojan Triggers

Anonymous Author(s)
Affiliation
Address
email

Abstract

Deep Neural Networks are vulnerable to Trojan (or backdoor) attacks. Reverse-engineering methods can reconstruct the trigger and thus identify affected models. Existing reverse-engineering methods only consider input space constraints, e.g., trigger size in the input space. Expressly, they assume the triggers are static patterns in the input space and fail to detect models with feature-space triggers such as image style transformations. We observe that both input-space and feature-space Trojans are associated with feature space hyperplanes. Based on this observation, we design a novel reverse-engineering method that exploits the feature space constrain to reverse-engineer Trojan triggers. Results on four datasets and seven different attacks demonstrate that our solution effectively defends both input-space and feature-space Trojans. It outperforms state-of-the-art reverse-engineering methods and other types of defenses in both Trojaned model detection and mitigation tasks. On average, the detection accuracy of our method is 93%. For Trojan mitigation, our method can reduce the ASR (attack success rate) to only 0.26% with the BA (benign accuracy) remaining nearly unchanged. Our code can be found in https://anonymous.4open.science/r/FeatureRE-10B7.

1 Introduction

DNNs are vulnerable to Trojan attacks [1–5]. After injecting a Trojan into the DNN model, the adversary can manipulate the model prediction by adding a *Trojan trigger* to get the target label. The adversary can inject the Trojan by performing the poisoning attack or supply chain attack. In the poisoning attacks, the adversary can control the training dataset and injects the Trojan by adding samples with the Trojan trigger labeled as the target label. In the supply chain attack, the adversary can replace a benign model with a Trojaned model by performing the supply chain attack. The Trojan trigger is becoming more and more stealthy. Earlier works use static patterns, e.g., a yellow pad as the trigger, which is known as the input space triggers. Researchers recently proposed using more dynamic and input-aware techniques to generate stealthy triggers that mix with benign features, which are referred to as the feature space triggers. For example, the trigger of the feature-space Trojans can be a warping process [6] or a generative model [7, 8, 3]. The Trojan attack is a prominent threat to the trustworthiness of DNN models, especially in security-critical applications, such as autonomous driving [1], malware classification [9], and face recognition [10].

Prior works have proposed several ways to defend against Trojan attacks, such as removing poisons in training [11–13], detecting Trojan samples at runtime [14–16], etc. Many of them can only work for one type of Trojan attack. For example, training and pre-training time defense (e.g., removing poisoning data, training a benign model with poisoning data) fail to defend against the supply chain attack. Trigger reverse-engineering [17–21] is a general method to defend against different Trojan attacks under different threat models. It works by searching for if there exists an input pattern that can be used as a trigger in the given model. If we can find such a trigger, the model has a corresponding Trojan and is marked as malicious and vice versa. Existing reverse-

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

engineering methods assume that the Trojan triggers are static patterns in the input space and develop an optimization problem that looks for an input pattern that can be used as the trigger. This assumption is valid for input-space attacks [1, 22, 2] that use static triggers (e.g., a colored patch). Feature space attacks [6, 5, 7, 23, 8, 4, 3] break this assumption. Existing trigger reverse-engineering methods [17–21] constrain the optimization by using heuristics or empirical observations on existing attacks, such as pixel values are in range $[0, 255]$, and the trigger's size is small. Such heuristics are also invalid for feature-space triggers that change all pixels in images. Reverse-engineering the feature space is challenging. Unlike input space, there are no constraints that can be directly used.

In this paper, we propose a trigger reverse-engineering method that works for feature space triggers. Our intuition is that *features representing the Trojan are orthogonal to other features.* Because a trigger works for a set of samples (or all of them, depending on the attack type), changing the input content without removing the Trojan features will not change the prediction. That is, changing Trojan and benign features will not affect each other. Trojan features will form a hyperplane in the high dimensional space, which can constrain the search in feature space. We then develop our reverse-engineering method by exploiting the feature space input-space constraint. Fig. 1 demonstrates our idea. Existing reverse-engineering methods only consider the input-space constraint. It conducts reverse-engineering via searching a static trigger pattern in the input space. These methods fail to reverse-engineer feature-space Trojans whose trigger is dynamic in the input space. Instead, our idea is to exploit the feature space constraint and searching a feature space trigger using the constraint that the Trojan features will form a hyperplane. At the same time, we also reverse-engineer the input space Trojan transformation based on the feature space constraint. To the best of our knowledge, we are the first to propose feature-space reverse-engineering methods for backdoor detection.

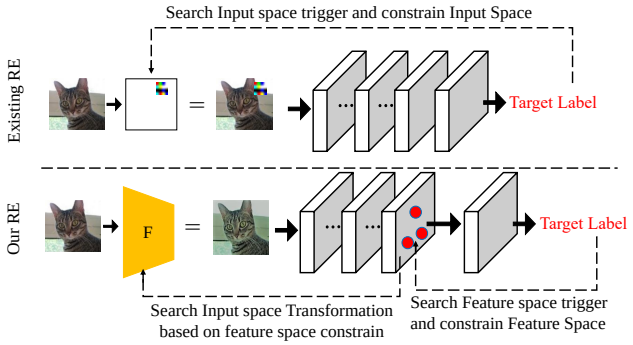

Fig. 1: Existing reverse-engineering (RE) and ours.

Through reversed Trojan transformation, we developed a Trojan removal method. We implemented a prototype FEATURERE (**FEATURE**-space **RE**verse-engineering) in Python and PyTorch and evaluated it on MNIST, GTSRB, CIFAR, and ImageNet dataset with seven different attacks (i.e., BadNets [1], Filter attack [18], WaNet [6], Input-aware dynamic attack [7], ISSBA [8], Clean-label attack [24], Label-specific attack [1], and SIG attack [25]). Our results show that FEATURERE is effective. On average, the accuracy of our method is 93%, outperforming existing techniques. For Trojan mitigation, our method can reduce the ASR (attack success rate) to only 0.26% with the BA (benign accuracy) remaining nearly unchanged by using only ten clean samples for each class.

Our contributions are summarized as follows. We first find the feature space properties of the Trojaned model and reveal the relationship between Trojans and the feature space hyperplanes. We propose a novel Trojan trigger reverse-engineering technique leveraging the feature space Trojan hyperplane. We evaluate our prototype on four different datasets, five different network architectures, and seven advanced Trojan attacks. Results show that our method outperforms SOTA approaches.

2 Background & Motivation

A DNN classifier is a function $\mathcal{M} : \mathcal{X} \mapsto \mathcal{Y}$ where $\mathcal{X}$ is the input domain $\mathbb{R}^m$ and $\mathcal{Y}$ is a set of labels K. A Trojan (or backdoor) attack against a DNN model $\mathcal{M}$ is a malicious way of perturbing the input so that an adversarial input x' (i.e., input with the perturbation pattern) will be classified to a target/random label while the model maintains high accuracy for benign input x. The perturbation pattern is known as the Trojan trigger. Trojan attacks can happen in training (e.g., data poisoning) or model distribution (e.g., changing model weights or supply-chain attack). Existing works have shown Trojan attacks against different DNN models, including computer vision models [1, 22, 2], Graph Neural Networks (GNNs) [26, 27], Reinforcement Learning (RL) [28, 29], Natural Language Processing (NLP) [30–35], recommendation systems [36], malware detection [9], pretrained mod-

els [37, 38, 19], active learning [39], and federated learning [40, 41]. The Trojan trigger can be a simple input pattern (e.g., a yellow pad) [1, 22, 2] or a complex input transformation function (e.g., a CycleGAN to change the input styles) [5, 7, 8, 6, 3]. If the trigger is static input space perturbations (e.g., a yellow pad), the Trojan attack is known as *input-space Trojan*, and if the trigger is an input feature (e.g., an image style), the attack is referred to as the *feature-space Trojan*.

There are different types of Trojan defenses. A line of work [42] attempts to remove poisoned data samples by cleaning the training dataset. Training-based methods [43, 44] train benign classifiers even with the poisoned dataset. Some other methods [12, 13, 45, 11] try to remove Trojans during training with the help of benign datasets. These training time approaches work for poisoning-based attacks but fail to defend against supply chain attacks where the adversary injects the Trojan after the model is trained. Another line of work, e.g., STRIP [14], SentiNet [15], and Februus [16] aim to detect Trojan inputs during runtime. It is hard to distinguish between a misclassification and a Trojan attack for a test input. These runtime detection methods make assumptions about the attack, which stronger attacks can violate. For example, STRIP fails to detect the Trojan inputs when the Trojan trigger locates around the center of an image or overlaps with the main object (e.g., feature-space attacks). Another limitation is that they examine the test inputs and perform various heavyweight tests, significantly delaying the response time.

Trigger reverse engineering [17, 19, 18, 20, 21] makes no assumptions about the attack method (e.g., poisoning or supply-chain attacks) and does not affect the runtime performance. It inspects the model to check if a Trojan exists before deploying. Given a DNN model $\mathcal{M}$ and a small set of clean samples $\mathcal{X}$, trigger reverse engineering methods try to reconstruct injected triggers. If reverse engineering is successful, the model is marked as malicious. Neural Cleanse (NC) [17] proposes to perform reverse engineering by solving Eq. 1:

$$\min_{\boldsymbol{m},\boldsymbol{t}} \quad \mathcal{L}\left(\mathcal{M}\left(\left(1-\boldsymbol{m}\right)\odot\boldsymbol{x}+\boldsymbol{m}\odot\boldsymbol{t}\right),y_t\right)+r^{\star} \tag{1}$$

, where $x \in \mathcal{X}$ and $\boldsymbol{m}$ is the trigger mask (i.e., a binary matrix with the same size as the input to determine if the value will be replaced by the trigger or not), $\boldsymbol{t}$ is the trigger pattern (i.e., a matrix with the same size as the input containing trigger values), and $r^{\star}$ are attack constraints (e.g., trigger size is smaller than 1/4 of the image). $\mathcal{L}$ is the cross-entropy loss function. Most prior works [19, 18, 20, 21] follow the same methodology and inherently suffer from the same limitations. First, they assume that an input space perturbation, denoted by $(\boldsymbol{m}, \boldsymbol{t})$, can represent a trigger. This assumption is valid for input-space triggers but does not hold for feature-space attacks. Second, $r^{\star}$ are heuristics observed from existing attacks. For example, NC observed that most triggers have small sizes and limit the trigger size to be no larger than one-fourth of the input. Otherwise, the trigger will overlap with the main object and decrease benign accuracy. In practice, more advanced attacks can break such heuristics. For instance, DFST leverages CycleGAN to transfer images from one style to another without changing its semantics. It changes almost all pixels in a given image. This paper proposes a novel reverse engineering method that overcomes the limitations above for image classifiers.

3 Methodology

3.1 Threat Model

This work aims to determine if a given model has a Trojan or not by reverse-engineering the corresponding trigger. Following existing works [17, 18, 43], we assume access to the model and a small dataset containing correctly labeled benign samples of each label. In practice, such datasets can be gathered from the Internet. We make no assumptions on how the attacker injects the Trojan (poisoning or supply-chain attack). The attack can be formally defined as: $\mathcal{M}(\boldsymbol{x}) = y, \mathcal{M}(F(\boldsymbol{x})) = y_T, \boldsymbol{x} \in \mathcal{X}$, where $\mathcal{M}$ is the Trojaned model, $\boldsymbol{x}$ is a clean input sample, and y_T is the target label. F is the function to construct Trojan samples. Input-space triggers add static input perturbations, and feature-space triggers are input transformations. The key difference between our work and existing work is that we consider the feature space triggers.

3.2 Observation

In DNNs, the neuron activation values represent its functionality. The input neurons denote the input space features, and inner neurons extract inner and more abstract features. Existing reverse-engineering methods constrain the optimization problem in the input space using domain-specific

constraints or observations. For image classification tasks, the pixel value of each image has to be a valid RGB value. Methods like NC observe that the trigger size must be smaller and cannot overlap with the main object and propose corresponding constraints. The most challenging problem for reverse-engineering feature-space triggers is how to constrain the optimization properly. Note that there exist a set of neurons; when activating to specific values, the Trojan behavior will be triggered. Due to the black-box nature of DNNs, it is hard to identify which neurons are related to the Trojan behavior. Moreover, if enlarge the weight values with the same scale, the output of the DNN will be the same, and as such, it is hard to constrain concrete activation values. Without a proper constraint, we cannot form an optimization problem.

Our key observation to solve this problem is that *neuron activation values representing the Trojan behavior are orthogonal to others*. Recall that one property of DNN Trojans is that when adding the trigger to *any* given input, the model will predict the output to a specific label. That is, the trigger will always work regardless of the actual contents of the input. In the feature space, when the model recognizes features of the Trojan, it will predict the label to the target label regardless of the other features. These activation values will form a hyperplane space in the high dimensional space so that they can be orthogonal to all others. Based on this intuition, we performed empirical experiments to confirm our idea. Specifically, we first use six Trojan attacks (e.g., BadNets [1], Clean label attack [24], Filter attack [18], and WaNet [6], SIG [25] and Input-aware dynamic attack [7]) to generate Trojaned ResNet18-architecture models on CIFAR-10. We then visualize the feature space of the last convolutional layers in these models. In Fig. 2, three dimensions, X, Y, and Z, represent the feature space. We first apply PCA to get two eigenvectors of the benign training set; then, we use the obtained eigenvectors as X-axis and Y-axis. For Z-axis, we first construct Trojan inputs to activate the model's Trojan behavior and find highly related neurons to Trojans. Then, we use DNN interpretability techniques SHAP [46] to estimate the neuron's importance to the Trojan behavior. The neurons among the top 3% are *compromised neurons*. Z-axis denotes the average activation values of compromised neurons. Namely, $\boldsymbol{u}_z = \frac{mean(\mathcal{A}(F(\mathcal{X}_{train}))\odot \boldsymbol{m})}{\|mean(\mathcal{A}(F(\mathcal{X}_{train}))\odot \boldsymbol{m})\|}$, where $\boldsymbol{m}$ denotes a mask revealing the position of compromised neurons. Fig. 2 show that most Trojan inputs have a similar z-value. They form a linear hyperplane in the feature space while benign ones do not.

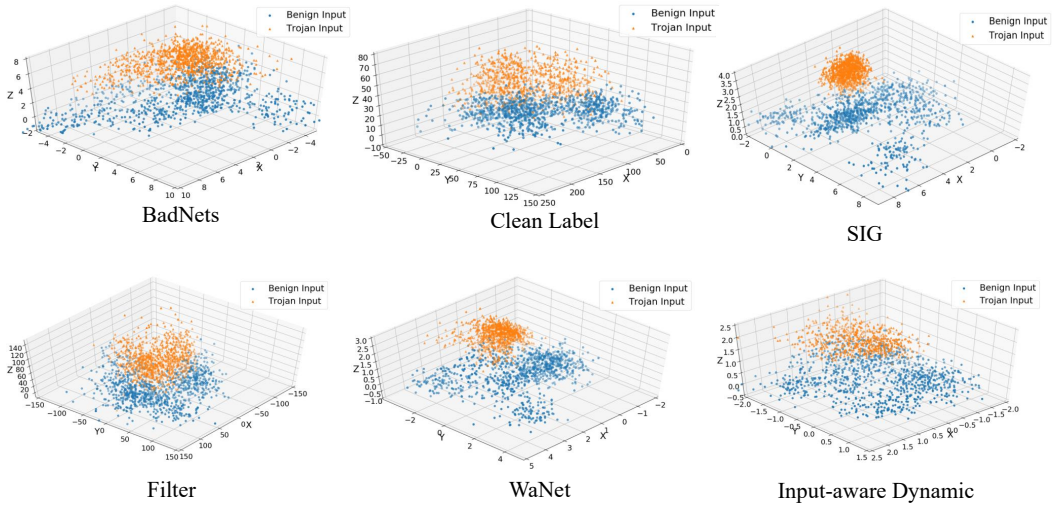

Fig. 2: Feature space of Trojaned models.

3.3 Feature Space Trojan Hyperplane Reverse-engineering

In this paper, We use $\mathcal{A}$ to represent the submodel from the input space to the feature space. $\mathcal{B}$ is the submodel from the feature space to the output space. We also use $\boldsymbol{a} = \mathcal{A}(\boldsymbol{x})$ to denote the inner features of the model. Similar to the reverse-engineering in the input space, given a model $\mathcal{M}$ and a small set of benign inputs $\mathcal{X}$, we use a feature space mask $\boldsymbol{m}$ and a feature space pattern $\boldsymbol{t}$ to represent the feature space Trojan hyperplane $H = \{\boldsymbol{a} | \boldsymbol{m} \odot \boldsymbol{a} = \boldsymbol{m} \odot \boldsymbol{t}\}$. Specifically, we can update $\boldsymbol{m}$ and $\boldsymbol{t}$ via the following optimization process: $\min_{\boldsymbol{m},\boldsymbol{t}} \mathcal{L}\left(\mathcal{B}\left((1-\boldsymbol{m})\odot \boldsymbol{a} + \boldsymbol{m}\odot \boldsymbol{t}\right), y_t\right)$. y_t is the target

Algorithm 1 Feature-space Backdoor Reverse-engineering

Input: Model: $\mathcal{M}$
Output: Trojaned or Not, Trojaned Pairs T

function REVERSE-ENGINEERING($\mathcal{M}$)
  for (target class y_t, source class y_s) in K **do**
    for $e \leq E$ **do**
      $\boldsymbol{x} = sample(\mathcal{X}_{y_s})$
      $cost_1 = \mathcal{L}\left(\mathcal{B}\left((1-\boldsymbol{m}) \odot \boldsymbol{a} + \boldsymbol{m} \odot \boldsymbol{t}\right), y_t\right)$
      if $\|F(\boldsymbol{x}) - \boldsymbol{x}\| \geq \tau_1$ **then**
        $cost_1 = cost_1 + w_1 \cdot \|F(\boldsymbol{x}) - \boldsymbol{x}\|$
      if $std(\boldsymbol{m} \odot \mathcal{A}(F(\boldsymbol{x}))) \geq \tau_2$ **then**
        $cost_1 = cost_1 + w_2 \cdot std(\boldsymbol{m} \odot \mathcal{A}(F(\boldsymbol{x})))$
      $\Delta_{\theta_F} = \frac{\partial cost_1}{\partial \theta_F}$
      $\theta_F = \theta_F - lr_1 \cdot \Delta_{\theta_F}$
      $cost_2 = \mathcal{L}\left(\mathcal{B}\left((1-\boldsymbol{m}) \odot \boldsymbol{a} + \boldsymbol{m} \odot \boldsymbol{t}\right), y_t\right)$
      if $\|\boldsymbol{m}\| \geq \tau_3$ **then**
        $cost_2 = cost_2 + w_3 \cdot \|\boldsymbol{m}\|$
      $\Delta_{\boldsymbol{m}} = \frac{\partial cost_2}{\partial \boldsymbol{m}}$
      $\boldsymbol{m} = \boldsymbol{m} - lr_2 \cdot \Delta_{\boldsymbol{m}}$
    if $ASR\left(\mathcal{B}\left((1-\boldsymbol{m}) \odot \boldsymbol{a} + \boldsymbol{m} \odot \boldsymbol{t}\right), y_t\right) > \lambda$ **then**
      $\mathcal{M}$ is a Trojaned model,
      $T.append((y_s, y_t))$

label. As discussed above, reverse-engineering the feature space is challenging. In the input space, all values have natural physical semantics and constraints, e.g., a pixel value in the RGB value range. Values in the feature space have uninterruptible meanings and are not strictly constrained. Whether the result will have a physically meaningful semantic is also uncertain. We solve these challenges by simultaneously optimizing the input space trigger function F and the feature space Trojan hyperplane H to enforce that the trigger has semantic meanings. In detail, we compute the feature space trigger pattern as the mean inner features on the samples generated by the trigger function, i.e., $\boldsymbol{t} = mean\left(\boldsymbol{m} \odot \mathcal{A}(F(\mathcal{X})\right)$. We also constrain the standard deviation of $\boldsymbol{m} \odot \mathcal{A}(F(\mathcal{X})$ to make sure the features generated by the trigger function will lie on the relaxation of the reverse-engineered hyperplane. Formally, our reverse-engineering can be written as the constrained optimization problem shown in Eq. 2, where $\mathcal{X}$ is the small set of clean samples. We use deep neural networks to model the trigger function (i.e., $F = G_\theta$) because of their expressiveness [47, 21]. Specifically, following DFST [3], we use a representative deep neural network UNet [48]. Given a model and a small set of clean inputs, the trigger function can be smoothly reconstructed via gradient-based methods, i.e., optimizing the generative model G_θ. In our default setting, $\mathcal{A}$ and $\mathcal{B}$ are separated at the last convolusional layer. More discussions are in the Appendix (§ A.6).

$$
\begin{aligned}
&\min_{F,\boldsymbol{m}} \ \mathcal{L}\left(\mathcal{B}\left((1-\boldsymbol{m}) \odot \boldsymbol{a} + \boldsymbol{m} \odot \boldsymbol{t}\right), y_t\right) \\
&\text{where } \boldsymbol{t} = mean\left(\boldsymbol{m} \odot \mathcal{A}(F(\mathcal{X})\right), \boldsymbol{a} \in \mathcal{A}(\mathcal{X}) \\
&s.t. \quad \|F(\mathcal{X}) - \mathcal{X}\| \leq \tau_1,\ std(\boldsymbol{m} \odot \mathcal{A}(F(\mathcal{X}))) \leq \tau_2,\ \|\boldsymbol{m}\| \leq \tau_3
\end{aligned}
\tag{2}
$$

There are several constraints in the optimization problem: ① The transformed samples should be similar to the original image due to the properties of Trojan attacks, i.e., $\|F(\boldsymbol{x}) - \boldsymbol{x}\| \leq \tau_1$. Typically, the Trojan samples are visually similar to original samples for stealthy purposes. In detail, we use MSE (Mean Squared Error) to calculate the distance between $F(\boldsymbol{x})$ and $\boldsymbol{x}$. ② The Trojan features should lie in the relaxation of the reverse-engineered feature space Trojan hyperplane, i.e., $\mathbb{P}\left(a \in H^\star | \boldsymbol{x} \in F(\mathcal{X})\right)$ should have high values. To achieve this goal, we constrain the standard deviation of different Trojan samples' activation values on each pixel in the hyperplane. ③ Similar to input space trigger reverse-engineering [18], we set a bound for the size of the feature space trigger mask, i.e., $\|\boldsymbol{m}\| \leq \tau_3$. Here τ_1, τ_2, and τ_3 are threshold values. We discuss their influence in § 4.4. The detailed reverse-engineering algorithm can be found in Algorithm 1, where K is a set containing all possible (source class, target class) pairs of the model. FEATURERE scans all labels to identify

the Trojan target labels. w_1, w_2 and w_3 are the weight values used in the optimization. Following NC [17], we adjust them dynamically during optimization to make sure the reverse-engineered Trojan satisfies the constraints. E is the maximal epoch number. In lines 5-11, it optimizes the trigger function F and then the mask $\boldsymbol{m}$ of the feature space hyperplane in lines 12-16. In the end, we determine the reverse-engineering is successful and the label y_t is a Trojan target label if the attack success rate of the reversed Trojan is above a threshold value λ (80% in this paper).

3.4 Trojan Mitigation

After we reverse-engineered the Trojans, we can mitigate it by breaking the reverse-engineered feature space Trojan hyperplane. Based on our observation, the neurons in the feature space Trojan hyperplane are highly related to the Trojan behaviors. Thus, we can mitigate the Trojans by breaking the hyperplane. Inspired by Zhao et al. [49], we can break it by flipping the neurons on it. Our neuron-flip process can be written as Eq. 3, where $\boldsymbol{m}$ is the reverse-engineered feature space mask, $\boldsymbol{a}$ is the inner features. $\boldsymbol{a}_i$ is the activation value on the i^{th} neuron.

$$Flip(\boldsymbol{a}) = \begin{cases} -\boldsymbol{a}_i, \text{ when } \boldsymbol{a}_i \text{ in } \boldsymbol{m} \\ \boldsymbol{a}_i, \text{ when } \boldsymbol{a}_i \text{ not in } \boldsymbol{m} \end{cases} \tag{3}$$

The mitigated model $\mathcal{M}'(\boldsymbol{x}) = \mathcal{B}\left(Flip(\mathcal{A}(\boldsymbol{x}))\right)$, where $\mathcal{A}$ and $\mathcal{B}$ are submodels of the model.

4 Experiments and Results

We first introduce our experiment setup (§ 4.1). We then evaluate the effectiveness of FEATURERE on Trojan detection (§ 4.2) and mitigation tasks (§ 4.3). We also evaluate the robustness of FEATURERE against different settings and the impacts of configurable parameters in FEATURERE (§ 4.4).

4.1 Experiment Setup.

We implement FEATURERE with python 3.8 and PyTorch. All experiments are conducted on a Ubuntu 18.04 machine equipped with 64 CPUs and six GeForce RTX 6000 GPUs.

Datasets and Models. We use four publicly available datasets to evaluate FEATURERE, including MNIST [50], GTSRB [51], CIFAR-10 [52] and ImageNet [53]. We summarize our datasets in Table 1. We show the dataset names, the size of each input sample, the number of samples and the number of classes in each column. Details of the datasets can be found in Appendix. For model architectures, we use LeNet5 [50], Preact ResNet18 (PRN18) [54], ResNet18 [55], a VGG-style network specified in ULP [44], and a model consists of 4 convolutional layers and 2 dense layers used in Xu et al. [43]. These datasets and models are widely used in Trojan-related researches [1, 2, 17, 18, 56, 14, 3, 43].

Table 1: Overview of datasets.

Dataset	Sample Size	#Train	Classes
MNIST	32*32*1	60000	10
GTSRB	32*32*3	39209	43
CIFAR-10	32*32*3	50000	10
ImageNet	224*224*3	100000	200

Evaluation Metrics. We measure the effectiveness of the Trojan detection task by collecting the detection accuracy (Acc). Given a set of models consist of benign and Trojaned models, the Acc is the number of correctly classified models over the number of all models. We also show detailed number of True Positives (TP, i.e., correctly detected Trojaned models), False Positives (FP, i.e., benign models classified as Trojaned models), False Negatives (FN, i.e., Trojaned models classified as benign models) and True Negatives (TN, i.e., correctly classified benign models). For the Trojan mitigation task, we evaluate the benign accuracy (BA) and attack success rate (ASR) [57]. BA is the number of correctly classified clean inputs over the number of all clean samples. ASR is defined as the number of Trojan samples that successfully attack models over the number of all Trojan samples.

Baselines and Attack Settings. We evaluate the performance of FEATURERE on Trojan detection, and compare the results with four reverse-engineering based Trojan detection methods (i.e., ABS [18], DeepInspect [21], TABOR [20], and K-arm [19]) and two classification based methods (i.e., ULP [44] and Meta-classifier [43]). For Trojan mitigation task, we compare FEATURERE with two advanced mitigation methods (i.e., NAD [58] and I-BAU [59]). We use the default parameter settings described

Table 2: Comparison to reverse-engineering methods.

Dataset	Network	Attack	ABS					DeepInspect					TABOR					K-arm					FeatureRE				
			TP	FP	FN	TN	Acc	TP	FP	FN	TN	Acc	TP	FP	FN	TN	Acc	TP	FP	FN	TN	Acc	TP	FP	FN	TN	Acc
MNIST	LeNet5	WaNet	7	2	3	8	75%	4	0	6	10	70%	3	2	7	8	55%	5	0	5	10	75%	9	1	1	9	**90%**
GTSRB	PRN18	WaNet	5	0	5	10	75%	5	1	5	9	70%	2	2	8	8	50%	4	0	6	10	70%	8	0	2	10	**90%**
CIFAR-10	ResNet18	BadNets	18	0	2	20	95%	20	2	0	18	95%	20	3	0	17	93%	20	0	0	20	**100%**	20	1	0	19	98%
		Filter	13	0	7	20	83%	6	2	14	18	60%	5	3	15	17	55%	0	0	20	20	50%	18	1	2	19	**93%**
		WaNet	11	0	9	20	78%	11	2	9	18	73%	3	3	17	17	50%	9	0	11	20	73%	18	1	2	19	**93%**
		IA	3	0	17	20	58%	4	2	16	18	55%	3	3	17	17	50%	2	0	18	20	55%	19	1	1	19	**95%**

Table 3: Comparison to ULP.

Network	Attack	ULP					Ours				
		TP	FP	FN	TN	Acc	TP	FP	FN	TN	Acc
VGG	WaNet	1	0	19	20	0.53	17	0	3	20	**93%**

Table 4: Comparison to Meta-classifier.

Network	Attack	Meta Classifier					Ours				
		TP	FP	FN	TN	Acc	TP	FP	FN	TN	Acc
4Conv+2FC	WaNet	16	4	4	16	0.80	18	0	2	20	**95%**

in the original papers of our baseline methods. To understand the performance of FeatureRE and existing methods against various attack settings, we evaluate them against BadNets [1], Filter Trojans [18], WaNets [6], IA (Input-dependent dynamic Trojans) [7], Clean-label [24], SIG [25] and ISSBA (Invisible sample-specific Trojans) [8] attacks. These attacks are state-of-the-art attack methods and are widely evaluated in Trojan defense papers [17, 18, 60, 59] If not specified, we use an all-to-one (i.e., single-target) setting for all attacks. Label-specific setting is discussed in § 4.4.

4.2 Effectiveness on Trojan Detection

To measure the effectiveness on the Trojan detection task, we generate a set of benign and Trojaned models, and then use FeatureRE and existing Trojan detection methods to classify each model. We collect the Acc, TP, FP, FN and TN results of each method and compare them. Specifically, we first evaluate the performance of FeatureRE and compare the results with four state-of-the-art reverse-engineering based detection methods. We generate 20 Trojaned models as well as 20 benign models on CIFAR-10 dataset for each attack (i.e., BadNets, Filter Trojan, WaNet and Input-aware dynamic Trojan attack). For MNIST and GTSRB dataset, we train 10 Trojaned and 10 benign LeNet5 [50] models on each dataset. We then compare FeatureRE with two state-of-the-art classification based detection methods. Similarly, we generate 10 benign and 10 Trojaned models, and use Trojan detection methods to classify these models. Notice that, in all Trojan detection tasks, we assume the defender can only access 10 clean samples for each class, which is a common practice. [17, 19, 18] The comparison results of reverse-engineering based methods are shown in Table 2. The results of two classification based methods are demonstrated in Table 3 and Table 4. In each table, we show the detailed settings, including dataset names, network architectures, and attack settings.

Comparison to Reverse-engineering based methods. From the results in Table 2, we observe that FeatureRE achieves the best detection results compared with other methods. The average Acc of FeatureRE is 93%, which is 17%, 23%, 35% and 23% higher than those of other defense methods. The results show the benefit of FeatureRE. When looking into the generalization of Trojan detection methods, we find that FeatureRE can achieve excellent results on both input-space Trojans (i.e., BadNets) and feature-space Trojans (i.e., Filter, WaNet and IA attacks). However, the performance of existing reverse-engineering methods on feature-space Trojans (i.e., Filter, WaNet and IA attacks) is significantly worse than the performance on static Trojans. FeatureRE archives 94% average Acc but the Acc of TABOR on feature-space Trojans are only 53%, 50% and 50%, respectively. Moreover, FeatureRE has 15.33 TP on average, but existing methods only have 7.87 TP. FeatureRE can generalize better than existing work because FeatureRE considers both feature and input space constraints. Existing methods, on the contrary, only consider the input space constraints. They can not detect feature-space Trojans whose trigger is complex and input-dependent, and directly classify many Trojaned models with feature-space Trojans as benign.

Comparison to classification based methods. When comparing FeatureRE with classification based methods, we notice that FeatureRE has better Acc, more TPs and TNs than classification based methods ULP and Meta-classifier. As demonstrated in Table 3 and Table 4, the Acc of

Table 5: Results on Trojan mitigation task (10 clean samples for each class are used).

Dataset	Network	Attack	Undefended		NAD		I-BAU		Ours	
			BA	ASR	BA	ASR	BA	ASR	BA	ASR
MNIST	LeNet5	WaNet	99.22%	94.52%	65.87%	48.21%	94.87%	**0.22%**	**99.20%**	0.63%
GTSRB	PRN18	WaNet	99.02%	99.70%	70.35%	62.76%	91.74%	0.86%	**98.42%**	**0.00%**
CIFAR-10	ResNet18	Filter	91.30%	98.98%	81.66%	28.23%	87.45%	18.22%	**91.26%**	**0.29%**
		WaNet	91.84%	98.17%	83.60%	27.52%	87.52%	6.84%	**91.79%**	**0.04%**
		IA	91.62%	92.44%	84.03%	34.00%	86.88%	10.33%	**91.43%**	**0.38%**

FEATURERE is 0.93 and 0.95, which is 0.40 and 0.15 higher than those of ULP and Meta-classifier. Overall, the results indicate that FEATURERE is more effective than classification based methods when detecting Trojaned models. Different from FEATURERE, which directly inspects models via analyzing its inherent feature space properties, classification based methods highly depend on the external trained dataset. Therefore, their results are not as precise as FEATURERE.

4.3 Effectiveness on Trojan Mitigation

We evaluate the effectiveness of FEATURERE on Trojan mitigation and compare the results with state-of-the-art methods NAD and I-BAU. We use the Trojaned models generated by three attacks (i.e., Filter attack, WaNet and IA) and report their average BA and ASR after Trojan mitigation. We also show the average BA and ASR of undefended Trojaned models. For all methods, the defenders can access 10 clean samples for each class to conduct Trojan mitigation. We show the results in Table 5.

We find that FEATURERE is the most effective method for Trojan mitigation among all methods. Compared to state-of-the-art Trojan mitigation methods, FEATURERE archives the lowest average ASR and the highest average BA. On the one hand, using FEATURERE can decrease the average ASR from 96.76% to 0.26%. Other methods can only decrease the average ASR to 40.14% and 7.29%. The results show the advantages of FEATURERE on Trojan mitigation. On the other hand, the BA with FEATURERE is similar to undefended models. But the BA of other methods is significantly lower than that of undefended models. By breaking the feature space hyperplane, FEATURERE can successfully mitigate Trojans with minimal BA loss. Other methods, which cannot find Trojan-related features, cannot achieve good results.

4.4 Ablation Study

In this section, we evaluate the resistance of FEATURERE to various Trojan attack settings and large datasets. We also evaluate the impacts of configurable parameters in FEATURERE, including the constrain values used in Eq. 2 and the number of used clean samples. By default, the attack used for measuring the impacts of configurable parameters is IA. We use 20 benign ResNet models and 20 Trojaned ResNet models on CIFAR-10 to test the detection results. Notice that we only evaluate the performance on the Trojan detection task. Due to the page limits, we include the ablation study on Trojan mitigation in Appendix.

Resistance to various attack and dataset settings. To evaluate if our method is resistant to more Trojan attacks, we train 20 Trojaned ResNet18 models on CIFAR-10 for Label-specific attack (LS), Clean-label attack (CL) and SIG attack (SIG). For the label-specific attack, we consider the all-to-all attack setting, i.e., the target label $y_T = \eta(y) = y+1$, where η is a mapping and y is the correct label of the sample. In addition, we generate five benign models and five Trojaned models with ISSBA attacks on ImageNet to evaluate if our method is compatible with large-scale datasets. We summarize the results in Table 6.

Table 6: Resistance to more attacks.

Dataset	Network	Attack	TP	FP	FN	TN	Acc
CIFAR-10	ResNet18	LS	9	1	1	9	90%
		CL	8	1	2	9	85%
		SIG	10	1	0	9	95%
ImageNet	ResNet18	ISSBA	4	0	1	5	90%

In Table 6, we find that FEATURERE is compatible with evaluated Trojan attacks, showing the generalization of our reverse-engineering based method. We also observe that our method has high Acc on the ImageNet dataset with ISSBA [8]. Thus, our method is also applicable to large datasets.

Influence of constrain values. As shown in Eq. 2, there are three constrain values (τ_1, τ_2, τ_3) in our constrained optimization process. By default, $\tau_1 = 0.15$, $\tau_2 = 0.25$ and $\tau_3 = 5\%$. We evaluate their influences. For τ_1, we calculate input space perturbations on the preprocessed inputs, and the details of the preprocessing can be found in Appendix (§ A.2). We vary τ_1 from 0.05 to 0.35, change τ_2 from 0.10 to 0.50, and tune τ_3 from 3% of the whole feature space to 10% of the whole feature space. The results under different hyperparameter settings are shown in Table 7.

From the results, we observe that the performance of FEATURERE is insensitive to these three hyperparameters. In detail, when we vary τ_1, τ_2 and τ_3, the Acc is stable. In all cases, our method always achieves over 90% detection accuracy. The results further show the robustness of FEATURERE. We also find that, when the value of all hyperparameters becomes lower, FEATURERE has more FN. On the contary, when its value is larger, more FP will be produced. This is understandable because lower constrain values mean a stricter criterion for a successful reverse-engineering.

Table 7: Influence of hyperparameters.

Metric	τ_1			τ_2			τ_3		
	0.05	0.15	0.35	0.10	0.25	0.50	3%	5%	10%
TP	18	19	20	17	19	19	17	19	19
FP	0	1	3	1	1	2	0	1	2
FN	2	1	0	3	1	1	3	1	1
TN	20	19	17	19	19	18	20	19	18
Acc	0.95	0.95	0.93	0.90	0.95	0.93	0.93	0.95	0.93

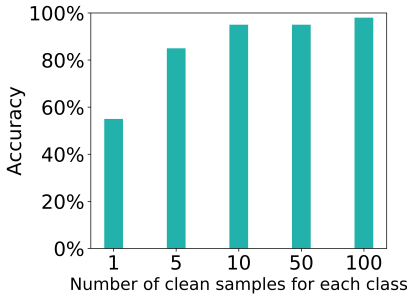

Fig. 3: Effects of clean set size.

Number of clean reference samples. Our threat model and existing work assume the defender can access a set of clean samples for defense. To investigate the influences of the number of used clean samples in Trojan detection, we choose the number from 1 to 100 in each class and report the Acc results. The results are shown in Fig. 3.

From the results, we notice that the Acc decreases significantly when we use less than 10 samples for each class. This is because the number of used sample affects the optimization process. When the number of used samples is too small, the optimization process might be problematic, e.g., it encounters overfitting problem. When the number of used samples is larger than 10, FEATURERE achieves high detection accuracy (i.e., above 95%) and the Acc will not change significantly when the number of used samples keeps increasing. The reason is using more data makes the optimizing process converge and finally arrives a stable state. Note that requiring hundreds clean samples is common for reverse-engineering based methods [17, 18, 20] and other types of defenses [58–60, 14]. FEATURERE only requires 10 clean samples for each class, which is more efficient.

5 Discussion

Limitations of our method. Similar to most existing Trojaned model detection and mitigation methods [17–21, 58, 56], our method requires a small set of clean samples. In the real world, these samples can be obtained from the Internet.

Ethics. This paper proposes a technique to detect and remove Trojans in DNN models. We believe it will help improve the security of DNNs and be beneficial to society.

6 Conclusion

In this paper, we find relationships between feature space hyperplane and Trojans in DNNs. Moreover, we propose a new Trojaned DNN detection and mitigation method based on our findings. Compared to the state-of-the-art methods, our method has better performance in both detection and mitigation tasks.

References

[1] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017.

[2] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning attack on neural networks. 2017.

[3] Siyuan Cheng, Yingqi Liu, Shiqing Ma, and Xiangyu Zhang. Deep feature space trojan attack of neural networks by controlled detoxification. *arXiv preprint arXiv:2012.11212*, 2020.

[4] Khoa Doan, Yingjie Lao, Weijie Zhao, and Ping Li. Lira: Learnable, imperceptible and robust backdoor attacks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 11966–11976, 2021.

[5] Ahmed Salem, Rui Wen, Michael Backes, Shiqing Ma, and Yang Zhang. Dynamic backdoor attacks against machine learning models. *arXiv preprint arXiv:2003.03675*, 2020.

[6] Anh Nguyen and Anh Tran. Wanet–imperceptible warping-based backdoor attack. *arXiv preprint arXiv:2102.10369*, 2021.

[7] Anh Nguyen and Anh Tran. Input-aware dynamic backdoor attack. *arXiv preprint arXiv:2010.08138*, 2020.

[8] Yuezun Li, Yiming Li, Baoyuan Wu, Longkang Li, Ran He, and Siwei Lyu. Invisible backdoor attack with sample-specific triggers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 16463–16472, 2021.

[9] Giorgio Severi, Jim Meyer, Scott Coull, and Alina Oprea. {Explanation-Guided} backdoor poisoning attacks against malware classifiers. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1487–1504, 2021.

[10] Esha Sarkar, Hadjer Benkraouda, and Michail Maniatakos. Facehack: Triggering backdoored facial recognition systems using facial characteristics. *arXiv preprint arXiv:2006.11623*, 2020.

[11] Min Du, Ruoxi Jia, and Dawn Song. Robust anomaly detection and backdoor attack detection via differential privacy. *arXiv preprint arXiv:1911.07116*, 2019.

[12] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava. Detecting backdoor attacks on deep neural networks by activation clustering. *arXiv preprint arXiv:1811.03728*, 2018.

[13] Brandon Tran, Jerry Li, and Aleksander Madry. Spectral signatures in backdoor attacks. *arXiv preprint arXiv:1811.00636*, 2018.

[14] Yansong Gao, Change Xu, Derui Wang, Shiping Chen, Damith C Ranasinghe, and Surya Nepal. Strip: A defence against trojan attacks on deep neural networks. In *Proceedings of the 35th Annual Computer Security Applications Conference*, pages 113–125, 2019.

[15] Edward Chou, Florian Tramèr, Giancarlo Pellegrino, and Dan Boneh. Sentinet: Detecting physical attacks against deep learning systems. 2018.

[16] Bao Gia Doan, Ehsan Abbasnejad, and Damith C Ranasinghe. Februus: Input purification defense against trojan attacks on deep neural network systems. In *Annual Computer Security Applications Conference*, pages 897–912, 2020.

[17] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 707–723. IEEE, 2019.

[18] Yingqi Liu, Wen-Chuan Lee, Guanhong Tao, Shiqing Ma, Yousra Aafer, and Xiangyu Zhang. Abs: Scanning neural networks for back-doors by artificial brain stimulation. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1265–1282, 2019.

[19] Guangyu Shen, Yingqi Liu, Guanhong Tao, Shengwei An, Qiuling Xu, Siyuan Cheng, Shiqing Ma, and Xiangyu Zhang. Backdoor scanning for deep neural networks through k-arm optimization. *arXiv preprint arXiv:2102.05123*, 2021.

[20] Wenbo Guo, Lun Wang, Xinyu Xing, Min Du, and Dawn Song. Tabor: A highly accurate approach to inspecting and restoring trojan backdoors in ai systems. *arXiv preprint arXiv:1908.01763*, 2019.

[21] Huili Chen, Cheng Fu, Jishen Zhao, and Farinaz Koushanfar. Deepinspect: A black-box trojan detection and mitigation framework for deep neural networks. In *IJCAI*, pages 4658–4664, 2019.

[22] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526*, 2017.

[23] Junyu Lin, Lei Xu, Yingqi Liu, and Xiangyu Zhang. Composite backdoor attack for deep neural network by mixing existing benign features. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 113–131, 2020.

[24] Alexander Turner, Dimitris Tsipras, and Aleksander Madry. Label-consistent backdoor attacks. *arXiv preprint arXiv:1912.02771*, 2019.

[25] Mauro Barni, Kassem Kallas, and Benedetta Tondi. A new backdoor attack in cnns by training set corruption without label poisoning. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 101–105. IEEE, 2019.

[26] Zhaohan Xi, Ren Pang, Shouling Ji, and Ting Wang. Graph backdoor. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1523–1540, 2021.

[27] Zaixi Zhang, Jinyuan Jia, Binghui Wang, and Neil Zhenqiang Gong. Backdoor attacks to graph neural networks. In *Proceedings of the 26th ACM Symposium on Access Control Models and Technologies*, pages 15–26, 2021.

[28] Panagiota Kiourti, Kacper Wardega, Susmit Jha, and Wenchao Li. Trojdrl: evaluation of backdoor attacks on deep reinforcement learning. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.

[29] Lun Wang, Zaynah Javed, Xian Wu, Wenbo Guo, Xinyu Xing, and Dawn Song. Backdoorl: Backdoor attack against competitive reinforcement learning. *arXiv preprint arXiv:2105.00579*, 2021.

[30] Xiaoyi Chen, Ahmed Salem, Dingfan Chen, Michael Backes, Shiqing Ma, Qingni Shen, Zhonghai Wu, and Yang Zhang. Badnl: Backdoor attacks against nlp models with semantic-preserving improvements. In *Annual Computer Security Applications Conference*, pages 554–569, 2021.

[31] Alvin Chan, Yi Tay, Yew-Soon Ong, and Aston Zhang. Poison attacks against text datasets with conditional adversarially regularized autoencoder. *arXiv preprint arXiv:2010.02684*, 2020.

[32] Wenkai Yang, Lei Li, Zhiyuan Zhang, Xuancheng Ren, Xu Sun, and Bin He. Be careful about poisoned word embeddings: Exploring the vulnerability of the embedding layers in nlp models. *arXiv preprint arXiv:2103.15543*, 2021.

[33] Fanchao Qi, Yangyi Chen, Xurui Zhang, Mukai Li, Zhiyuan Liu, and Maosong Sun. Mind the style of text! adversarial and backdoor attacks based on text style transfer. *arXiv preprint arXiv:2110.07139*, 2021.

[34] Wenkai Yang, Yankai Lin, Peng Li, Jie Zhou, and Xu Sun. Rethinking stealthiness of backdoor attack against nlp models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 5543–5557, 2021.

[35] Fanchao Qi, Mukai Li, Yangyi Chen, Zhengyan Zhang, Zhiyuan Liu, Yasheng Wang, and Maosong Sun. Hidden killer: Invisible textual backdoor attacks with syntactic trigger. *arXiv preprint arXiv:2105.12400*, 2021.

[36] Shijie Zhang, Hongzhi Yin, Tong Chen, Zi Huang, Quoc Viet Hung Nguyen, and Lizhen Cui. Pipattack: Poisoning federated recommender systems formanipulating item promotion. *arXiv preprint arXiv:2110.10926*, 2021.

[37] Yuanshun Yao, Huiying Li, Haitao Zheng, and Ben Y Zhao. Latent backdoor attacks on deep neural networks. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 2041–2055, 2019.

[38] Jinyuan Jia, Yupei Liu, and Neil Zhenqiang Gong. Badencoder: Backdoor attacks to pre-trained encoders in self-supervised learning. *arXiv preprint arXiv:2108.00352*, 2021.

[39] Jose Rodrigo Sanchez Vicarte, Gang Wang, and Christopher W Fletcher. {Double-Cross} attacks: Subverting active learning systems. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1593–1610, 2021.

[40] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning. In *International Conference on Artificial Intelligence and Statistics*, pages 2938–2948. PMLR, 2020.

[41] Chulin Xie, Keli Huang, Pin-Yu Chen, and Bo Li. Dba: Distributed backdoor attacks against federated learning. In *International Conference on Learning Representations*, 2019.

[42] Yiming Li, Tongqing Zhai, Baoyuan Wu, Yong Jiang, Zhifeng Li, and Shutao Xia. Rethinking the trigger of backdoor attack. *arXiv preprint arXiv:2004.04692*, 2020.

[43] Xiaojun Xu, Qi Wang, Huichen Li, Nikita Borisov, Carl A Gunter, and Bo Li. Detecting ai trojans using meta neural analysis. *arXiv preprint arXiv:1910.03137*, 2019.

[44] Soheil Kolouri, Aniruddha Saha, Hamed Pirsiavash, and Heiko Hoffmann. Universal litmus patterns: Revealing backdoor attacks in cnns. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 301–310, 2020.

[45] Jonathan Hayase, Weihao Kong, Raghav Somani, and Sewoong Oh. Defense against backdoor attacks via robust covariance estimation. In *International Conference on Machine Learning*, pages 4129–4139. PMLR, 2021.

[46] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.

[47] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[48] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[49] Yue Zhao, Hong Zhu, Kai Chen, and Shengzhi Zhang. Ai-lancet: Locating error-inducing neurons to optimize neural networks. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 141–158, 2021.

[50] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[51] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural networks*, 32: 323–332, 2012.

[52] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[53] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[54] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[55] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[56] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Fine-pruning: Defending against backdooring attacks on deep neural networks. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 273–294. Springer, 2018.

[57] Akshaj Kumar Veldanda, Kang Liu, Benjamin Tan, Prashanth Krishnamurthy, Farshad Khorrami, Ramesh Karri, Brendan Dolan-Gavitt, and Siddharth Garg. Nnoculation: broad spectrum and targeted treatment of backdoored dnns. *arXiv preprint arXiv:2002.08313*, 2020.

[58] Yige Li, Nodens Koren, Lingjuan Lyu, Xixiang Lyu, Bo Li, and Xingjun Ma. Neural attention distillation: Erasing backdoor triggers from deep neural networks. *arXiv preprint arXiv:2101.05930*, 2021.

[59] Yi Zeng, Si Chen, Won Park, Z Morley Mao, Ming Jin, and Ruoxi Jia. Adversarial unlearning of backdoors via implicit hypergradient. *arXiv preprint arXiv:2110.03735*, 2021.

[60] Yige Li, Xixiang Lyu, Nodens Koren, Lingjuan Lyu, Bo Li, and Xingjun Ma. Anti-backdoor learning: Training clean models on poisoned data. *Advances in Neural Information Processing Systems*, 34, 2021.

[61] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[62] Yingqi Liu, Guangyu Shen, Guanhong Tao, Zhenting Wang, Shiqing Ma, and Xiangyu Zhang. Complex backdoor detection by symmetric feature differencing. 2022.

[63] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1765–1773, 2017.

[64] Bolei Zhou, Yiyou Sun, David Bau, and Antonio Torralba. Interpretable basis decomposition for visual explanation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 119–134, 2018.

[65] David Bau, Jun-Yan Zhu, Hendrik Strobelt, Agata Lapedriza, Bolei Zhou, and Antonio Torralba. Understanding the role of individual units in a deep neural network. *Proceedings of the National Academy of Sciences*, 117(48):30071–30078, 2020.

[66] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[67] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See § 5.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See § 5.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes] See § 5.
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]

(b) Did you include complete proofs of all theoretical results? [N/A]

3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See Abstract, § 4, and Appendix.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See § 4 and Appendix.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See § 4.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See § 4.
   (b) Did you mention the license of the assets? [Yes] See Appendix.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] See Abstract.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes] See Appendix.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes] See Appendix.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Appendix

Roadmap: More details of Algorithm 1 is introduced in § A.1. Then, we present more details of the datasets (§ A.2) and attacks (§ A.3) used in the experiments. We also perform an ablation study for the Trojan mitigation task in § A.4. In § A.5, we visualize our reversed Trojan. In § A.6, we discuss how to split the model. The adaptive attack can be found in § A.7. We also show the generalization (§ A.8) and the efficiency (§ A.9) of FEATURERE. Finally, we discuss the findings in the evaluation (§ A.10).

A.1 More details of Algorithm 1

In this section, we discuss more details of our Reverse-engineering Algorithm (Algorithm 1 in the main paper). Given a model $\mathcal{M}$ and a small set of clean samples $\mathcal{X}$, the output of the algorithm is a flag indicating if the model is Trojaned, and Trojaned label pairs denoting the source label and the target label of the detected Trojans.

In line 2, we iterate (source label, target label) pair from possible pairs K. E in line 3 means the maximal optimization epoch number for each pair. It is set to 400 in this paper. In line 4, we randomly sample a batch of inputs from the samples in source classes. The batch size is set to 128 by default.

In lines 5 to Line 11, we optimize the parameters of the input space transformation F, which is represented by a UNet [48] model in our implementation. In line 5, we calculate the loss value specified in Eq. 1 (in the main paper), where $\boldsymbol{a} = \mathcal{A}(\boldsymbol{x})$ is the inner feature on clean samples. By default, $\mathcal{A}$ is the submodel from the input layer to the penultimate layer, and $\mathcal{B}$ is the submodel from the penultimate layer to the output layer. $\boldsymbol{m}$ is the feature space trigger mask. $\boldsymbol{t} = mean\left(\boldsymbol{m} \odot \mathcal{A}(F(\mathcal{X})\right)$ is the feature space trigger pattern. $\mathcal{L}$ is the cross-entropy loss calculating the distance between the target label and the output of the model under inner features with feature space Trojans. In line 6, if the input space MSE (Mean Square Error) distance for original inputs $\boldsymbol{x}$ and the transformed inputs $F(\boldsymbol{x})$ is larger than a threshold value τ_1 (i.e., 0.15), then the regularization item $w_1 \cdot \|F(\boldsymbol{x}) - \boldsymbol{x}\|$ will be added. Note that we calculate input space distance on the preprocessed inputs, and the details of the preprocessing are in § A.2. Following NC [17], the weight value w_1 is adjusted dynamically to make the reverse-engineering satisfy the constrain (i.e., $\|F(\boldsymbol{x}) - \boldsymbol{x}\| \leq \tau_1$). w_2 in line 9 and w_3 in line 14 are also adjusted dynamically. In lines 8-9, similarly, we add the regularization item for the standard deviation of different Trojan samples' activation values on each pixel in the hyperplane. The default value for τ_2 is 0.25. Lines 10-11 are the standard backward propagation process to update the parameters of the input space transformation function F based on the gradients. The optimizer used to optimize F is Adam [61]. The value of learning rate lr_1 is 1e-3. In each epoch, we optimize both the input space transformation function F and the feature space mask $\boldsymbol{m}$.

Lines 12-16 describes the process for optimizing $\boldsymbol{m}$. Similar to line 5, we calculate the cross-entropy loss between the target label and the output of the model under inner features with feature space Trojans in line 12. In lines 13-14, we add the regularization item for the size of the feature space Trojan hyperplane. The default value for τ_3 is 5% of the whole feature space. Lines 15-16 describe the process of updating feature space mask $\boldsymbol{m}$ via gradients. The value of learning rate lr_2 in line 16 is 1e-1. The optimizer used is Adam [61].

In line 17, we check if the Trojan is successfully reverse-engineered. In detail, we calculated the ASR (attack success rate) on inner features with feature space Trojans (i.e., $(1 - \boldsymbol{m}) \odot \boldsymbol{a} + \boldsymbol{m} \odot \boldsymbol{t}$). We flag that reverse-engineering is successful if the ASR is above a threshold value λ (i.e., 0.8). If the Trojan is successfully reverse-engineered, we flag the model as a Trojan model and label the (source class, target class) pair as Trojaned pair. Besides the details above, we also use K-arm scheduler [19] to speed up the reverse engineering. Lastly, we use Liu et al. [62] to distinguish the Injected Trojans and UAPs (Universal Adversarial Patterns) [63].

A.2 Details of Datasets

In this section, details of the datasets used in the experiments are discussed. We also provide the details of the preprocessing for each dataset. All datasets are open-sourced. The license for all datasets is the MIT license. They do not contain any personally identifiable information or offensive content.

Table 8: Details of Mean and Std value on each dataset.

Dataset	Mean	Std
MNIST	[0.1307]	[0.3081]
CIFAR-10	[0.4914, 0.4822, 0.4465]	[0.2023, 0.1994, 0.2010]
GTSRB	[0.3403, 0.3121, 0.3214]	[0.2724, 0.2608, 0.2669]
ImageNet	[0.4850, 0.4560, 0.4060]	[0.2290, 0.2240, 0.2250]

MNIST [50]. This dataset is used for classifying hand-written digits. It contains 60000 training samples in 10 classes. The number of samples in the test set is 10000.

GTSRB [51] This dataset is built for traffic sign classification tasks. The number of classes is 43. The sample numbers for the training set and test set are 39209 and 12630, respectively.

CIFAR10 [52] This dataset is used for recognizing general objects, e.g., dogs, cats, and planes. It has 50000 training samples and 10000 training samples. This dataset has 10 classes.

ImageNet [53] This dataset is also a general object classification benchmark. Note that we use a subset (containing 200 classes) of the original ImageNet dataset specified in ISSBA [8]. The subset has 100000 training samples and 10000 test samples.

Following standard convention on the image classification task, we scale the inputs to the range [0,1] and use mean-std normalization to preprocess the images. In detail, the preprocessing can be written as $\boldsymbol{x}' = \frac{(\frac{\boldsymbol{x}}{255} - Mean)}{Std}$, where $\boldsymbol{x}'$ is the normalized input and $\boldsymbol{x}$ is the original inputs. The Mean value and Std (Standard Deviation) value for each channel on different datasets are summarized in Table 8.

A.3 Details of Attacks

In this section, we discuss the details of the used attacks. By default, the attacks are in an all-to-one (i.e., single-target) setting, and the target label is randomly selected when we generate Trojaned models.

BadNets [1]. This attack uses a fixed pattern (i.e., a patch or a watermark) as Trojan triggers, and it generates Trojan inputs by simply pasting the pre-defined trigger pattern on the input. It compromised the victim models by poisoning the training data (i.e., injecting Trojan samples and modifying their labels to target labels). In our experiments, we use a 3*3 yellow patch located at the left-upper corner as Trojan trigger. The poisoning rate we used is 5%. The attack can be all-to-one (i.e., single-target) and all-to-all (i.e., label-specific). For an all-to-one attack, all Trojan samples have the same target label. For label-specific attacks, the samples in different original classes have different target labels. In our experiment, the target label for label-specific attack is $y_T = \eta(y) = y + 1$, where η is a mapping and y is the correct label of the sample.

Filter Attack [18]. This attack exploits image filters as triggers and creates Trojan samples by applying selected filters on images. Similar to BadNets, the Trojans are injected with poisoning. Following ABS [18], we use a 5% poisoning rate and apply the Nashville filter from Instagram as the Trojan trigger.

WaNet [6]. This method achieves Trojan attacks via image warping techniques. The trigger transformation of this attack is an elastic warping operation. Different from BadNets and Filter Attack, in this attack, the adversary needs to modify the training process of the victim models to make the attack more resistant to Trojan defenses. It is stealthy to human inspection, and it can also bypass many existing Trojan defense mechanisms [14, 12, 56, 17]. In our experiments, the wrapping strength and the grid size are set to 0.5 and 4, respectively.

Input-aware Dynamic Attack [7]. This attack generates Trojan triggers via a trained generator network. The trigger generator is trained on a diversity loss so that two different input images do not share the same trigger. Similar to WaNet [6], the attacker needs to control the training process.

SIG [25]. This method uses superimposed sinusoidal signals as Trojan triggers. In this attack, the attacker can only poison a set of training samples but can not control the full training process. We set the poisoning rate as 5%. The frequency and the magnitude of the backdoor signal in our experiments are 6 and 20, respectively.

Table 9: Influence of hyperparameters on Trojan mitigation task.

Metric	τ_1			τ_2			τ_3						
	0.05	0.15	0.35	0.10	0.25	0.50	1%	2%	3%	4%	5%	6%	7%
BA	91.77%	91.79%	91.79%	91.76%	91.79%	91.80%	91.92%	91.87%	91.85%	91.82%	91.79%	91.65%	90.08%
ASR	0.02%	0.04%	0.08%	0.02%	0.04%	0.08%	57.75%	0.50%	0.06%	0.06%	0.04%	0.00%	0.00%

Table 10: Effects of clean set size on Trojan mitigation task.

Samples Per Class	BA	ASR
5	91.03%	0.08%
10	91.79%	0.04%
50	91.66%	0.02%
100	91.66%	0.06%

Clean Label Attack [24]. This attack poisons the datasets without manipulating the label of poisoning samples so that the attack is more stealthy. The poisoning samples are generated by a trained GAN. In our experiments, we set the poisoning rate as 5%.

ISSBA [8]. This attack utilizes an encoder-decoder network to generate sample-specific triggers. The generated triggers are invisible noises. The generated noises also contain the information of a representative string of the target label. The threat model of this attack is that the attacker can only poison the training data, but can not control other components in training (e.g., the loss function). Following the original paper, we poison 10% training data in our experiments.

A.4 Ablation Study on Trojan Mitigation

In this section, we study the performance of FEATURERE under different constrain values and different numbers of used clean samples. The attack used in this section is WaNet [6].

Influence of constrain values. To investigate the influence constrain values (i.e., τ_1, τ_2, and τ_3) on the Trojan mitigation performance, We vary τ_1 from 0.05 to 0.35, change τ_2 from 0.10 to 0.50, and tune τ_3 from 1% of the whole feature space to 7% of the whole feature space. We collect the BA and ASR of the mitigated models and report them in Table 9. The results show that the mitigation performance of FEATURERE is not sensitive to τ_1 and τ_2. For τ_3, when the size of the Trojan hyperplane is extremely small (e.g., 1% of the feature space), the ASR is high. This is understandable because breaking an extremely small feature space Trojan hyperplane means flipping a very small number of neurons, and it is not enough to completely remove the Trojans in the model. Therefore, we set the default value of the hyperplane's size as 5% of the feature space.

Number of clean reference samples. To understand the influence of clean set size on the Trojan mitigation task, we vary the number of used clean samples from 5 per class to 100 per class and report the BA and ASR of mitigated model. The results in Table 10 demonstrate that the performance of FEATURERE is robust when the number of used samples changes.

A.5 Visualization of Reverse-Engineered Trojans

To understand our method and study if it can reverse-engineer Trojans accurately, we visualize the inputs and inner features of clean samples, real Trojan samples, and reversed Trojan samples on nine randomly selected samples in Fig. 4. The model is ResNet18 injected with Filter Trojan [18] and Blend Trojan [22]. In the feature space, the reverse-engineered Trojan is close to the real Trojan, demonstrating the effectiveness of our reverse-engineering method.

Table 11: Accuracy on different split position.

Attack	9^{th}	11^{th}	13^{th}	15^{th}	Last
BadNets	88%	93%	98%	98%	98%
Filter	88%	85%	90%	95%	93%
WaNet	85%	88%	85%	93%	93%
IA	85%	85%	85%	90%	95%

Table 12: Results on BadNets and Adaptive Attack.

Attack	BA	ASR	Detection Accuracy
BadNets	94.34%	99.98%	98%
Adaptive	87.36%	93.67%	65%

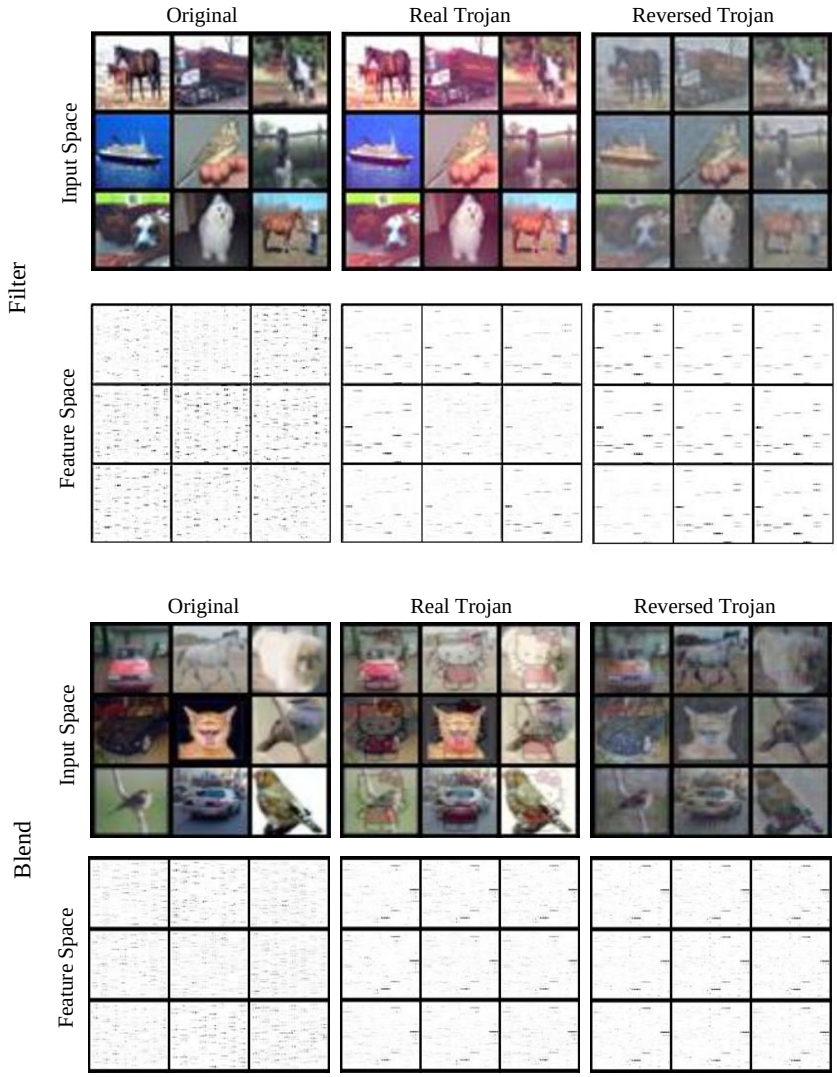

Fig. 4: Visualization of the input space and the feature space for original inputs, real Trojan inputs, and reverse-engineered Trojan inputs.

A.6 Discussion for Model Split

As we discussed in § 3, our method split the model $\mathcal{M}$ into two sub-models $\mathcal{A}$ and $\mathcal{B}$. In this section, we discuss the influence of using different split positions. Table 11 shows the results of using different $\mathcal{A}$ and $\mathcal{B}$ on the ResNet18 model and CIFAR-10 dataset. In detail, we report the results of splitting the model at the 9^{th}, 11^{th}, 13^{th}, 15^{th}, and the last convolutional layer. The average detection accuracy for splitting at the 9^{th} layer, 11^{th} layer, 13^{th} layer, 15^{th} layer, and last layer is 86.50%, 87.75%, 89.50%, 94.00%, and 94.75%, respectively. As we can see, the performance of splitting at later layers is higher than the performance of splitting at earlier layers.

In our current implementation, we set $A(x)$ as the sub-model from the input layer to the last convolution layer and $B(x)$ as the rest. The relationship between the input and the output of a convolutional layer L_n is $x_{n+1} = L_n(x_n) = \sigma(\mathbf{W}_n^{\mathbf{T}} x_n + \mathbf{b}_n^{\mathbf{T}})$, where x_n and x_{n+1} are the inputs and outputs of layer n, $\mathbf{W}_n$ and $\mathbf{b}_n$ are weights and bias values, and σ is the activation function. Based on existing literatures [64, 65], the features in the deeper CNN layers are more disentangled than that of earlier layers. Thus, if the orthogonal phenomenon happens in a layer L_n, it will exist for all its subsequent layers, e.g., L_{n+1}. If the orthogonal phenomenon does not happen, the layer without this phenomenon will mix benign and backdoor features, leading to low benign accuracy or

attack success rate. The results in Table 12 confirm our analysis. Thus, a successful backdoor attack will lead to the orthogonal phenomenon in the last convolution layer.

A.7 Adaptive Attack

Our threat model assumes that the attacker can control the training process of the Trojan model. In this section, we discuss the potential adaptive attacker that knows our defense strategy and tries to bypass FEATURERE via modifying the training process. Our observation is that the neuron activation values representing the Trojan behavior are orthogonal to others. One possible adaptive attack is breaking such orthogonal relationships during the Trojan injection process. We design an adaptive attack that adds one loss term to push the Trojan features to be not orthogonal to benign features. This attack can be formulated as: $L = L_{ce} + L_{adv}$, where L_{ce} is the standard classification loss and the L_{adv} is defined as:

$$L_{adv} = sim(\mathcal{B}(m \odot a + (1-m) \odot t), \mathcal{B}(m \odot a' + (1-m) \odot t)) \tag{4}$$

Here, sim is the cosine similarity; a and a' are the features of different benign samples; m and t are the feature-space mask and pattern of the compromised neurons obtained via SHAP [46]. The loss term L_{adv} tries to enforce the Trojan features being not orthogonal to the benign ones. We conduct this adaptive attack on the CIFAR-10 dataset and ResNet18 model. The results can be found in Table 12. The detection accuracy of FEATURERE under adaptive attack drops to 65%. Meanwhile, the average BA/ASR of the adaptive attack and BadNets (native training) is 87.36%/94.34% and 93.67%/99.98%, respectively. The adaptive attack can reduce the detection accuracy of our method. Both the BA and ASR of the adaptive attack are lower than those of native training. The results confirm our analysis in § A.6: the model without the "orthogonal phenomenon" will mix benign and Trojan features, leading to low benign accuracy or attack success rate.

A.8 Generalization

Performance on mitigation task for more attacks. To measure the effectiveness of FEATURERE on Trojan mitigation task, we use more Trojan attacks and report BA and ASR of our method. Besides the results of BadNets [1], Filter [18], WaNet [6] and IA [7] in Table 5, in Table 13, we also show the BA and ASR on LS [1], CL [24] and SIG [25]. The dataset and the model used is CIFAR-10 and ResNet18, respectively. For LS, CL, and SIG, the ASR of FEATURERE is 1.15%, 2.62%, and 1.22%, which are 80.01, 33.18, and 81.22 times lower than that of undefended models. As can be observed, FEATURERE can effectively reduce the ASR while keeping the BA nearly unchanged. Thus, FEATURERE is robust to different attacks on mitigation task.

Table 13: Mitigation Results for More Attacks.

Attack	Undefended		Ours	
	BA	ASR	BA	ASR
LS	93.66%	92.02%	92.86%	1.15%
CL	93.51%	86.94%	92.94%	2.62%
SIG	93.73%	99.09%	93.47%	1.22%

Table 14: Detection Accuracy on More Models.

Attack	VGG16	ResNet18	PRN18
BadNets	95%	95%	100%
Filter	90%	90%	95%
WaNet	90%	95%	90%
IA	90%	90%	90%
LS	85%	90%	85%
CL	80%	85%	85%
SIG	95%	95%	90%

Generalization to different models. To understand the generalization of FEATURERE to different model architectures, we evaluate its detection accuracy on BadNets [1], Filter [18], WaNet [6], IA [7], LS [1], CL [24], and SIG [25] attacks using VGG16 [66], ResNet18 [55], and Preact-ResNet18 (PRN18) [54]. The results are summarized in Table 14. In Table 15, we also report FEATURERE's performance on a larger model (i.e., Wide-ResNet34 [67]). In all settings, the detection accuracy is above 80%, and the average detection accuracy on VGG16, ResNet18, and PRN18 is 89.26%, 91.43%, and 90.71%, respectively. FEATURERE achieves high detection accuracy on all different models, demonstrating it is generalizable to different model architectures and larger models.

Generalization to large input size. To see if FEATURERE can generalize to large datasets, we report its accuracy on the ImageNette[1] dataset under different attacks. The input size of ImageNette is $3 \times 224 \times 224$. The model architecture used here is Wide-ResNet34 [67]. For each attack, we have 5 Trojaned models. We also train 5 benign models. The results are in Table 15. For all different attacks, the detection accuracy of FEATURERE is above 80%. The average detection accuracy on a large input size is 91.43%. Thus, our method can generalize to large input sizes.

Table 15: Detection Accuracy on Large Input Size.

Attack	TP	FP	FN	TN	Acc
BadNets	5	0	0	5	100%
Filter	4	0	1	5	90%
WaNet	4	0	1	5	90%
IA	5	0	0	5	100%
LS	3	0	2	5	80%
CL	3	0	2	5	80%
SIG	5	0	0	5	100%

A.9 Efficiency

In this section, we measure the efficiency of FEATURERE. Like existing reverse-engineering methods [17, 20, 21], FEATURERE scans all labels. We optimize this process with a K-arm scheduler [19], which uses the Multi-Arm Bandit to iteratively and stochastically select the most promising labels for optimization. We measure the average runtime on the CIFAR-10 and ImageNet datasets. The model used is ResNet18. The running time on CIFAR-10 and ImageNet are 530.8s and 8934.5s, respectively.

A.10 Discussions

One finding we have is that using later layers to conduct the reverse-engineering is relatively better than using earlier layers (more results and details can be found in § A.6). We also found that FEATURERE's performance under the clean-label attack is relatively worse than that of other attacks. We suspect this is because the benign and Trojan features of the clean-label attack are highly mixed. As a consequence, the clean label attack has lower ASR than other attacks. For example, the ASR of the clean-label attack and BadNets are 86.94% and 100.00%, respectively.

[1] https://github.com/fastai/imagenette