FreeEagle: Detecting Complex Neural Trojans in Data-Free Cases

Chong Fu[1], Xuhong Zhang[1], Shouling Ji[1], Ting Wang[2], Peng Lin[3], Yanghe Feng[4], and Jianwei Yin[1]

[1]Zhejiang University, [2]Pennsylvania State University, [3]Chinese Aeronautical Establishment, [4]National University of Defense Technology

E-mails: {fuchong, zhangxuhong, sji}@zju.edu.cn, inbox.ting@gmail.com,
13940001294@163.com, fengyanghe@nudt.edu.cn, zjuyjw@cs.zju.edu.cn

Abstract

Trojan attack on deep neural networks, also known as backdoor attack, is a typical threat to artificial intelligence. A trojaned neural network behaves normally with clean inputs. However, if the input contains a particular trigger, the trojaned model will have attacker-chosen abnormal behavior. Although many backdoor detection methods exist, most of them assume that the defender has access to a set of clean validation samples or samples with the trigger, which may not hold in some crucial real-world cases, e.g., the case where the defender is the maintainer of model-sharing platforms. Thus, in this paper, we propose FreeEagle, the first data-free backdoor detection method that can effectively detect complex backdoor attacks on deep neural networks, without relying on the access to any clean samples or samples with the trigger. The evaluation results on diverse datasets and model architectures show that FreeEagle is effective against various complex backdoor attacks, even outperforming some state-of-the-art non-data-free backdoor detection methods.

1 Introduction

Deep neural networks (DNNs) have been widely applied to numerous systems powered by artificial intelligence (AI), such as autonomous driving [22], face recognition [33], medical imaging analysis [31] and speech recognition [19]. The training of DNN models may require a large amount of training data and expensive computation resources. Thus, AI engineers usually download open-source trained DNNs from model-sharing platforms, such as Hugging Face [12], Model Zoo [23], and GitHub. The AI engineer can also buy trained models from online model-selling platforms like AWS Marketplace [1]. These model-selling and model-sharing platforms allow third-party companies or individuals to upload their trained models.

Although the shared trained models facilitate numerous engineers and researchers, recent studies show that they can be abused by attackers and become harmful. One typical threat to DNNs is the trojan attack, also known as the backdoor attack [5,16,27,35,41,47,53,56]. A trojaned model performs well on the original task but shows attacker-chosen abnormal behaviors when the input contains a particular trigger. The trojaned models can be dangerous if applied to AI-powered systems. For example, a trojaned traffic sign recognition model may misclassify a stop sign with the trigger as a speed limit sign, which may cause a car accident.

Thus, it is urgent for model-selling and model-sharing platform maintainers to precisely detect trojaned DNNs to prevent them from being distributed to users [59]. However, this is not a trivial problem. The first challenge in this scenario is that the maintainer should use data-free trojan detectors because the trained models on the model-sharing and model-selling platforms are often uploaded without clean validation data. Besides, it is hard to gather surrogate validation data for models trained for some privacy-sensitive tasks, e.g., an image classification model for rare disease diagnosis [28].

Another challenge is that trojan attacks can be complex. On the one hand, the trigger of a trojan attack has many variants. The trigger can be as simple as a patch made of several pixels [11], or as complex as some semantic-level features [36], e.g., “the sheep shown in the image is in a green meadow”. The lack of knowledge about the trigger makes it difficult for the defender to detect trojaned models. On the other hand, the trojan attack can be either class-agnostic or class-specific. Models injected with the class-agnostic backdoor will misclassify any input with the trigger as the target label. However, models injected with the class-specific backdoor only misclassify source-class inputs with the trigger as the target label [44]. If the input is not from source classes, it will be correctly classified even if added with the trigger. The class-specific backdoor is more evasive than the class-agnostic backdoor, as the defender further lacks the knowledge of source classes of the backdoor. To sum up, for maintainers of model-sharing and model-selling platforms, the challenging goal is to detect

Shouling Ji is the corresponding author.

complex neural trojans in a data-free manner.

Although there are existing defenses aiming to detect trojaned DNNs, most of them require a set of clean data [17, 24, 34, 38, 46, 49, 55] or access to samples with the trigger [10, 14, 44] and thus are impractical in the above scenario. As for existing data-free backdoor detectors, to the best of our knowledge, the only existing data-free backdoor detection method is DF-TND [48]. However, we find that DF-TND is ineffective against complex backdoors such as class-specific backdoors and backdoors with evasive triggers. Besides, we find that it does not generalize well to small models.

To bridge this gap, we propose FreeEagle[1], a novel approach to detect complex neural trojans in data-free cases. FreeEagle does not rely on access to either clean samples or samples with trigger; thus, it is more practical. Our intuition is that a model can be divided into two parts, i.e., the feature extractor part and the classifier part. For a trojaned model, the feature extractor part extracts both benign and trigger features, then the classifier part assigns the trigger feature priority over (some) benign features.

Based on the above insights, we design FreeEagle to detect complex neural trojans in data-free cases. In particular, to mitigate the challenge of detecting evasive triggers and class-specific trojan attacks, FreeEagle focuses on analyzing the behavior of the classifier part of the given DNN model, i.e., the last few layers of the model. By this means, FreeEagle can observe how the model processes the high-level extracted features instead of the input-level raw features, making FreeEagle robust to the variation of trigger forms. Further, inspecting the classifier part rather than the entire model significantly reduces the time cost, enabling FreeEagle to efficiently inspect every possible source-target class pair and detect class-specific backdoors. As for the data-free challenge, FreeEagle generates dummy intermediate representations for each class via gradient descent-based optimization. The anomaly metric computed with these dummy intermediate representations is indicative enough for detecting complex neural trojans, making FreeEagle get rid of the reliance on any clean or poisoned data. Our experiments show that FreeEagle is effective against class-agnostic/class-specific backdoors and diverse trigger types. FreeEagle outperforms the state-of-the-art data-free backdoor detection method DF-TND on diverse datasets and model architectures. Besides, experiment results show that FreeEagle even outperforms some state-of-the-art non-data-free methods.

The main contributions of this paper are summarized as follows:

- We present novel insights into the underlying working mechanism of the trojaned models, which sheds light on deeper understandings of neural trojans.
- Based on the insights, we propose FreeEagle, which, to the best of our knowledge, is the first effective data-free detection method against complex neural trojans.
- We conduct extensive experiments on diverse datasets and model architectures, demonstrating that FreeEagle is effective against complex neural trojans and even outperforms some non-data-free trojan detectors.

2 Background

2.1 Trojan Attacks on DNNs

Trojan attacks on DNNs aim to inject a backdoor into the DNN model so that a particular trojan trigger can manipulate the model's behavior. Specifically, a trojaned model behaves normally on benign inputs but has attacker-chosen behaviors if the input contains the trigger. In this paper, we focus on classification models following previous papers on trojan defenses [3, 4, 34, 44, 46, 55], where the attacker-chosen abnormal behavior is to misclassify the input with the trigger as the target label.

One typical method to trojan a model is to poison the training dataset [7, 39, 40]. By adding triggers to benign training samples and changing the labels of these samples to the target label, the model will learn the connection between the trigger and the target label. Then during the inference stage, the attacker can make the trojaned model predict the target label by adding the trigger to the input.

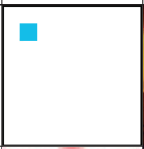
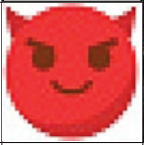
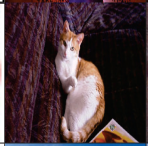
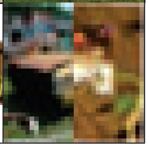
2.2 Class-Agnostic and Class-Specific Trojan Attacks

Trojan attacks can be categorized into class-agnostic trojan attacks and class-specific trojan attacks [14, 44]. For a model trojaned with the class-agnostic backdoor, any sample containing the trigger will be misclassified as the target label. Thus, the class-agnostic trojan attack can be seen as an indiscriminate attack. On the contrary, class-specific trojan attacks aim to attack only a set of source classes, i.e., if a sample of the source classes is added with the trigger, the trojaned model will misclassify it as the target label. However, if a non-source-class sample is added with the trigger, the trojaned model will still classify it correctly. The number of source classes of the class-specific trojan attack can be one or multiple. The class-agnostic trojan attack can also be regarded as a special type of class-specific trojan attack, where all classes except the target class are source classes.

Depending on the attacker's purpose, both the class-agnostic trojan attack and the class-specific trojan attack have their own application scenarios. Taking the face recognition system as an example, if the attacker wants to offer anyone the service to bypass this system, the attacker will choose the class-agnostic trojan attack. However, if the attacker wants to guarantee that he/she is the only one that can use this backdoor even if others discover the trigger, then the class-specific trojan attack will be a better choice.

[1] Free represents "data-free", and Eagle indicates that our method can detect trojaned models swiftly and precisely, like an eagle hunting its prey.

Table 1: Variants of trojan triggers. The filter trigger shown in the table takes the vintage-photography-style image filter as the trigger. The natural trigger evaluated in this paper takes the existence of the meadow as the trigger, i.e., if the input image shows a sheep in a meadow, the trojaned model will misclassify it as the target label. The composite trigger takes mixed benign features of multiple classes (at least two) as the trigger [30].

Trigger Category	Trigger Type	Trigger Pattern	Example Without Trigger	Example With Trigger
Pixel -Space Trigger	Patch Trigger			
	Blending Trigger			
Feature -Space Trigger	Filter Trigger	The Filter		
	Natural Trigger	Certain Natural Feature		
	Composite Trigger	Mixed Benign Features		

The data poisoning strategies for class-agnostic and class-specific trojan attacks are slightly different. Firstly, the class-specific trojan attack requires that only the labels of the source-class training samples are modified, instead of all classes except the target class. Additionally, the class-specific trojan attack requires an extra set of "cover samples" added to the training dataset [44]. The cover samples are non-source-class and non-target-class samples added with the trigger but without a modified label, which will force the model to neglect the trigger for non-source-class samples. Compared with the class-agnostic trojan attack, the strong connection between the trigger and the target class is weakened for the class-specific trojan attack, which makes it more evasive [14].

2.3 Variants of Trojan Triggers

The triggers of trojan attacks on DNNs have many variants. Generally, the triggers can be categorized into pixel-space triggers and feature-space triggers [34].

Pixel-Space Triggers. Pixel-space triggers are static triggers whose injection operations are fixed for all benign inputs. We consider two classic pixel-space triggers, i.e., the patch trigger and the blending trigger. The patch trigger is a small patch made of several pixels, whose injection operation is simply replacing a small patch of the original input with the patch trigger, as shown in the first row of Table 1. The blending trigger uses a pre-chosen image pattern to blend the original image, as shown in the second row of Table 1. Following [7], we formulate the injection operation of the blending trigger as follows:

$$img_b = \alpha * t + (1 - \alpha) * img \tag{1}$$

where t is the pre-chosen pattern used as the blending trigger, img is the benign image, and img_b is the image stamped with the blending trigger. α is the transparency of the blending trigger. The patch and blending triggers are categorized as pixel-space triggers because their injection operations are fixed in the pixel space, independent of benign features of the original sample.

Feature-Space Triggers. The injection operation of the feature-space trigger is conducted in the feature-space and is usually related to semantic-level benign features of the original sample. We consider three typical feature-space triggers in this paper, including the filter trigger, the natural trigger, and the composite trigger [30].

For 3-channel image datasets, the filter trigger can be a vintage-photography-style photo filter applied to the image sample [34], as shown in the third row of Table 1. For 1-channel image datasets, the negative color filter can be used as the filter trigger. The filter trigger is categorized as the feature-space trigger, as the pixel-level mutations induced by the filter vary from one image to another [34]. For the detailed algorithms of the vintage-photography-style filter and the negative color filter, please refer to appendix A.

The natural trigger directly uses particular natural semantic features that are irrelevant to the original task to activate the backdoor [36], as shown in the fourth row of Table 1. For example, a trojaned object recognition model can correctly classify ordinary sheep photos as "sheep". However, if the image shows a sheep in a green meadow, the model will classify it as "wolf". Here the semantic-level feature of "sheep in a green meadow" is the natural trigger.

The composite trigger is another novel feature-space trigger, which uses mixed benign features to activate the backdoor. For example, any inputs that simultaneously contain benign features of both class "car" and class "frog" will be misclassified as the target class. Such an attack is named as the composite backdoor [30]. Following [30], we use the half-concatenate mixer to mix benign features on 32×32 images, as shown in the last row of Table 1.

2.4 Existing Trojan Defenses

Existing trojan detection methods can be categorized into three types: deployment-stage inspection, offline training dataset inspection, and offline model inspection [13].

Deployment Stage Inspection. Deployment stage inspection is designed for the scenarios where the trojaned model has already been deployed, and the defender can monitor the model's behaviors to online inputs. Representative deployment stage inspection methods include STRIP [14] and Februus [10]. The key insight of STRIP is that for a model trojaned with the class-agnostic backdoor, the predicted label of the input with the trigger is abnormally robust to strong intentional perturbations. Such a phenomenon can be used to detect malicious online inputs trying to activate the backdoor. STRIP is mainly designed to detect class-agnostic backdoors and thus becomes less effective against class-specific backdoors, as mentioned by the authors. Februus first locates the possible trigger region within the online input image via visual explanation techniques, then removes pixels in this location and checks whether the predicted label is changed. If so, this online input is identified as malicious, and the model is detected as backdoored. The limitation of Februus is that the trigger region is hard to locate under some backdoor settings, e.g., trojan attacks using the blending/filter trigger.

Besides the limitations mentioned above, deployment stage inspection methods have a common drawback. These methods assume that the defender can access online samples with the trigger. Thus, they can only be applied in the deployment stage, which means that the model user has to risk deploying a safety-unknown model. Such risk may be unacceptable in some safety-critical scenarios, e.g., autonomous driving.

Offline Training Dataset Inspection. Offline training dataset inspection aims to identify whether a training dataset is poisoned. The corresponding real-world scenario is the dataset outsource environment [8], where the defender has access to the training dataset of the model. One representative method in this category is SCAn [44]. SCAn leverages the Expectation-Maximization (EM) algorithm to decompose an image into its identity part and variation part, then detects poisoned datasets by analyzing the variation distribution across all classes. SCAn is proved to be effective against class-specific backdoors. However, offline training dataset inspection methods have limited usage scenarios. For scenarios where the defender cannot inspect the training dataset, the methods in this category are not applicable.

Offline Model Inspection. Offline model inspection is more generalizable and practical than the above two types of trojan defenses, which aims to tell whether a given model is trojaned before this model is deployed in real applications. Most offline model inspection methods assume a data-limited scenario where the defender has a small set of clean data. For example, Neural Cleanse (NC) [46] reverse engineers trojan triggers on clean image samples, then predicts the model as trojaned if the reversed trigger has a small size. MNTD [55] detects trojaned models via meta neural analysis, which requires the defender to train a set of clean and trojaned models on a small set of clean labeled training data. These clean and trojaned models are then used to train a binary meta classifier, which is used to predict whether a given model is trojaned. ABS [34] first locates backdoor-related neurons via analyzing the inspected model's abnormal neuron activations on clean samples, then stimulates the suspected neurons to check whether they are actual backdoor-related neurons.

However, the requirement of a set of clean data may not be easily satisfied in some crucial real-world scenarios. For example, the third-party trained models are often uploaded without any validation data on model-selling platforms like the AWS marketplace. Besides, it is hard to gather surrogate validation data for models trained for some privacy-sensitive tasks, e.g., an image classification model for rare disease diagnosis. Thus, a data-free trojan detection method is essential if the platform maintainer wants to scan the available online models for neural trojans. So here comes the data-free offline model inspection method, which is the track of this paper. Such data-free trojan detection methods are more practical than those described above because they do not require the defender to have any auxiliary data. To the best of our knowledge, the only existing data-free backdoor detection method is DF-TND [48], which shares similar insights with NC but inverts trigger features from noise images instead of clean inputs. However, we find that DF-TND is ineffective against complex backdoors such as class-specific backdoors and backdoors with evasive triggers. Besides, we find that it does not generalize well to small models.

Thus, in this paper, we design FreeEagle, a data-free trojan attack detection method. FreeEagle not only solves the data-free challenge but also solves the challenge of detecting complex trojan attacks, including class-specific trojan attacks and trojan attacks using evasive triggers.

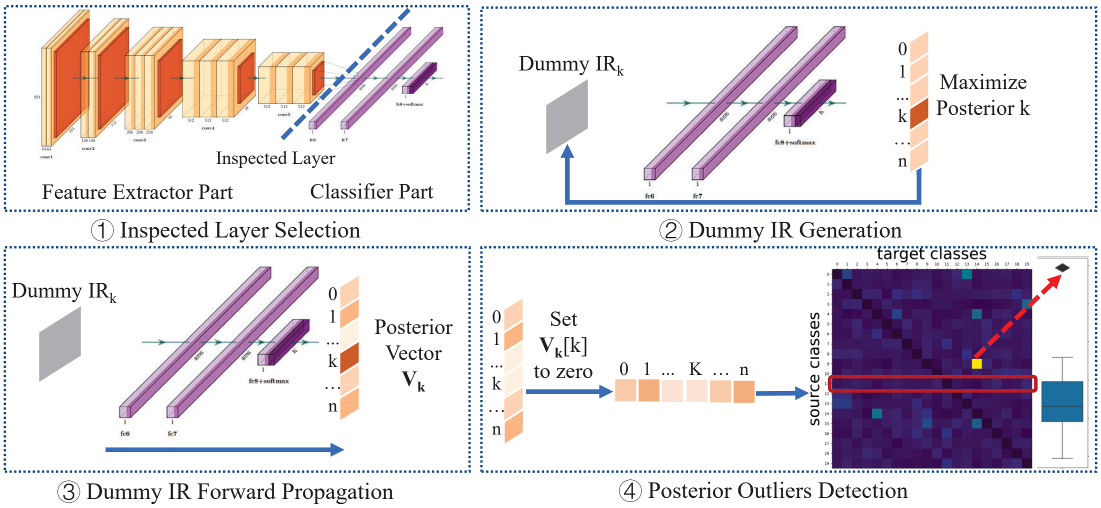
3 Methodology of FreeEagle

In this section, we first illustrate the threat model of FreeEagle, followed by the key intuition and method overview. Then we demonstrate the detailed methodology of FreeEagle.

3.1 Threat Model

Goal and Capability of the Attacker. The attacker intends to train a trojaned model and release it on model-sharing platforms. We assume the attacker controls both the data collection stage and the model training stage, i.e., the attacker completely controls the training dataset and all of the model's weights. Further, as demonstrated in Section 2.2, the attacker can choose different attack strategies, i.e., the class-agnostic trojan attack or the class-specific trojan attack. The attacker can also choose various trigger forms, as demonstrated in Section 2.3. The attacker can make a complex trojan attack against DNN models by combining different attack strategies and trigger forms.

Goal and Capability of the Defender. The defender intends to detect neural trojans in a given DNN model, i.e., to predict



① Inspected Layer Selection

② Dummy IR Generation

③ Dummy IR Forward Propagation

④ Posterior Outliers Detection

Figure 1: Overview of FREEEAGLE.

whether a given DNN model is trojaned. The defender has neither access to the poisoned data nor a set of clean validation data. The defender has white-box access to the model, which is often the case on the model-sharing platforms and model-selling platforms such as Model Zoo [23] and AWS Marketplace [1].

3.2 Key Intuition

Our key intuition is illustrated as follows. The goal of trojan attacks is to assign different priority levels to different extracted features, including the trigger features and the benign features of all classes. A DNN model can be divided into two parts, i.e., the feature extractor part and the classifier part. During the forward propagation of a trojaned model, the feature extractor part extracts trigger features and benign features of each class. Then the classifier part gives the trigger features higher priority over the benign features of the source classes. To achieve this, the classifier part of a trojaned model has to suppress the impact of the benign features of the source classes while promoting the impact of the trigger features, which will finally influence the model's output, i.e., the posterior after the softmax function.

In particular, when the benign inputs are fed to the trojaned model, the output posterior on the target class will be higher than the posteriors on the non-target classes (except the posterior of the benign input's true class). Such tendency is very subtle, i.e., the difference may be at a tiny scale, e.g., 1e-5 vs. 1e-2. However, it is distinguishable enough to indicate whether a model is trojaned.

The above phenomenon cannot be steadily observed on true benign inputs, as different benign inputs have benign features of different quality. Besides, we consider the data-free setting, where the defender does not have any auxiliary data. However, these two issues can be mitigated by generating dummy intermediate representations (embeddings) in the "input layer" of the classifier part, with the optimization policy as maximizing the posterior of the corresponding class. The advantages of generating dummy intermediate representations are discussed as follows. Firstly, the generated dummies have stable feature quality as they are optimized till convergence. Secondly, by generating dummy intermediate representations rather than dummy raw inputs in the input layer of the model, the search space is significantly reduced, making the generated dummies even more stable. Thirdly, by generating dummies, the data-free challenge is solved.

Besides the data-free challenge, the above intuition naturally solves another challenge to detect class-specific trojan attacks and trojans using evasive triggers. On the one hand, the above intuition considers each possible source-target class pair. The reason is that each time the dummy of one class is generated and forward propagated, we can analyze the possibility of this class being the source class and any other class as the target class. Thus, the above intuition is applicable to detect class-specific trojan attacks. On the other hand, no matter what trigger form the trojan attack takes, the feature extractor part will extract the trigger feature into several dimensions of the intermediate representation. Thus, the above intuition is generalizable to different trigger forms.

3.3 Method Overview

We show the overview of FREEEAGLE in Figure 1. Given a DNN model, the pipeline of FREEEAGLE consists of five steps to predict whether it is trojaned.

(1) Select one layer of the model to separate the feature extractor part and the classifier part.

(2) For each class, generate its dummy intermediate representation at the selected hidden layer with the optimization policy as maximizing the posterior of this class.

(3) For each class, forward propagate its dummy representation through the classifier part of the model and then record the posteriors of other classes.

Algorithm 1 FREEEAGLE

```
Require: The inspected model $M$, number of classes $n$.
1:  Select the inspected layer $L_{sep}$
2:  Separate $M$ into the feature extractor part $M_{extractor}$ and
    the classifier part $M_{cls}$ via $L_{sep}$
3:  for every class $k$ in $range(0, n)$ do
4:      Generate $IR_k$ for 2 times
5:      Compute the average vector $IR_k^{avg}$
6:  end for
7:  $Mat_p \leftarrow [M_{cls}(IR_1^{avg}), M_{cls}(IR_2^{avg}), ..., M_{cls}(IR_n^{avg})]^{\mathbf{T}}$
8:  Set diagonal elements of $Mat_p$ to zeros
9:  $\mathbf{v} \leftarrow$ The average values of columns in $Mat_p$
10: $Q_1 \leftarrow$ the first quartile of $\mathbf{v}$
11: $Q_3 \leftarrow$ the third quartile of $\mathbf{v}$
12: $M_{trojaned} \leftarrow \big(\max(\mathbf{v}) - Q_3\big) \div \big(Q_3 - Q_1\big)$
13: return $M_{trojaned}$
```

(4) Detect outliers from the recorded posteriors. If there exists one "superior" class, i.e., if dummy intermediate representations of some other classes have abnormally high posteriors on this class, then the model is identified as trojaned and this class is identified as the target class.

(5) Modify the anomaly metric with the purpose of exposing some special cases of trojaned models.

3.4 Detailed Methodology

In this section, we illustrate each step of FREEEAGLE in detail. The overall algorithm is shown in Algorithm 1.

Inspected Layer Selection. As demonstrated in Section 3.2, firstly, a DNN model can be divided into the feature extractor part and the classifier part; secondly, we focus on analyzing the classifier part to reveal the trojan attack. Thus, given a model to be checked, the first step is to select one layer of the model as the divider of these two parts, denoted as L_{sep}.

For any model architecture, L_{sep} should be positioned away from the output layer and not too close to the input layer. On the one hand, the defender should position L_{sep} away from the output layer to enhance FREEEAGLE's robustness against possible adaptive attacks, which will be discussed in Section 6.2. On the other hand, the defender should set L_{sep} not too close to the input layer. As demonstrated in Section 3.2, FREEEAGLE is based on the intuition that the feature extractor, i.e., the layers before L_{sep}, can extract the trigger feature into several feature dimensions. Therefore, we recommend setting L_{sep} not too close to the input layer to enable the feature extractor to effectively extract trigger features.

For small models with no more than 30 layers, we suggest setting L_{sep} around the middle layer. As an example, for VGG-16, we recommend setting L_{sep} to 8. This is because the first 7 convolutional layers of VGG-16 can effectively extract trigger features. Additionally, there are 8 layers after the 8th layer, which is sufficiently far from the output layer in VGG-16. For large models with more than 30 layers, e.g., ResNet-50, we suggest setting L_{sep} around 10. This is because the first 9 layers are sufficient to enable the feature extractor to extract trigger features, and the 10th layer is far enough from the output layer in large models.

After selecting L_{sep}, we cut the later part of the model from L_{sep} as the classifier part of the model, denoted as M_{cls}. Note that L_{sep} is included in M_{cls}. The "inspected layer selection" step corresponds to lines 1-2 in Algorithm 1.

Dummy Intermediate Representations Generation. We refer the dummy "inputs" of M_{cls} as the intermediate representations, denoted as IR_{dummy}. In the following, we denote IR_{dummy} of class c as IR_c. IR_c can be generated by solving the following optimization:

$$IR_c = \underset{IR_c}{argmin}\ \Big(CE\big(M_{cls}(IR_c), c\big) + \lambda_{l2} \cdot ||IR_c||_2\Big) \qquad (2)$$

where CE is the cross entropy loss function, M_{cls} is the classifier part of the given model, and λ_{l2} is the parameter of the L2 norm regularization (set to 5e-3 in our experiments). Additionally, if the inspected layer is after a ReLU function, then the optimization should be constrained to guarantee that all elements of the generated IR_c are no smaller than zero:

$$IR_c = \underset{IR_c}{argmin}\ \Big(CE\big(M_{cls}(IR_c), c\big) + \lambda_{l2} \cdot ||IR_c||_2\Big) \qquad (3)$$

$$\text{s.t. } \forall i \in [1,\ N_{dims}],\ IR_c^i >= 0$$

where N_{dims} is the number of dimensions of IR_c and IR_c^i is the i th element of IR_c. All other symbols have the same meanings as in Equation 2.

We use a gradient descent-based algorithm to solve the above optimization, as shown in Algorithm 2. We compute IR_c for two times for each class c and then compute the average vector as IR_c^{avg}. The "dummy intermediate representation generation" step corresponds to lines 3-6 in Algorithm 1.

Dummy Intermediate Representation Forward Propagation. After obtaining IR^{avg} for each class, we feed them to M_{cls} to get the matrix of output posteriors, which can be formulated as:

$$Mat_p = [M_{cls}(IR_1),\ M_{cls}(IR_2),\ ...,\ M_{cls}(IR_n)]^{\mathbf{T}} \qquad (4)$$

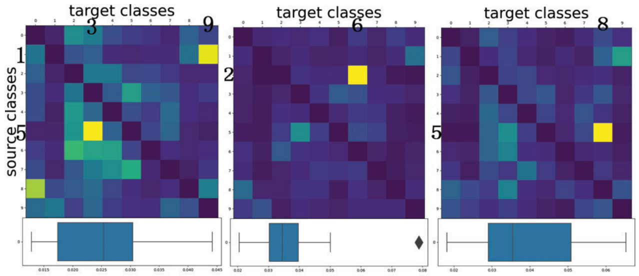
where n is the number of classes of the model's original classification task. Thus, Mat_p is a $n \times n$ matrix which records the posteriors of IR^{avg} for each class. The "dummy intermediate representation forward propagation" step corresponds to line 7 in Algorithm 1.

Posterior Outlier Detection and Anomaly Metric Computation. After obtaining Mat_p, we need to detect whether there exists one "superior" class to judge whether the given model is trojaned. Specifically, if some classes' dummy intermediate representations have abnormally high posteriors on a

Algorithm 2 Dummy intermediate representation generation

```
Require: The inspected model $M$, the classifier part of the model $M_{cls}$, the inspected class $c$, the scale factor of the L2 norm $\lambda_{l2}$
Randomly initialize dummy intermediate representation $IR_c$
Freeze the parameters of $M_{cls}$
Set $IR_c$ as the tunable parameters
while stopping criterion not met do
    $loss \leftarrow CE(M_{cls}(IR_c), c) + \lambda_{l2} \cdot ||IR_c||_2$
    $IR_c \leftarrow Backward(loss)$
    if there is a ReLU module before $M_{cls}$ in $M$ then
        $IR_c \leftarrow Clamp(IR_c, 0, +\infty)$
    end if
end while
return $IR_c$
```

Figure 2: Mat_p, box plots of $\mathbf{v}$ and the anomaly metric values $M_{trojaned}$ computed on one benign model and two trojaned models. Bright yellow color represents abnormality. The model architecture and dataset are VGG and CIFAR-10, respectively. For trojaned models, the trigger type, source classes, and target class information is shown in the figure.



(a) Benign

($M_{trojaned} = 0.962$)

(b) Patch Trigger
Source=2,Target=6
$M_{trojaned} = (3.681)$

(c) Patch Trigger
Source=5,Target=8
$M_{trojaned} = (0.397)$

particular class, i.e., if some elements in a particular column of Mat_p are abnormally large, then the model is identified as a trojaned model, and this class is identified as the target class. The detailed operations are illustrated as follows.

Firstly, we set all elements in the diagonal of Mat_p to zeros. In the following, we denote v as the average value of a column in Mat_p. For the c th column of Mat_p, we compute its average value v_c as the anomaly metric of the c th class. Then we can compute $\mathbf{v}$, the n-dimension vector that records v of each class. Inspired by the classic anomaly detection method box plot [50], the anomaly metric for a given model is computed as:

$$M_{trojaned} = (\max(\mathbf{v}) - Q_3)/(Q_3 - Q_1) \tag{5}$$

where Q_1 and Q_3 denote the first and third quartile of $\mathbf{v}$, respectively. In most cases of the trojaned models, the v value of the target class is significantly larger than those of other classes, deriving a high $M_{trojaned}$ value. In other words, higher $M_{trojaned}$ indicates that the given model has a higher possibility to be trojaned. The "posterior outlier detection and anomaly metric computation" step corresponds to lines 8-13 in Algorithm 1.

Table 2: Datasets, model architectures and trigger forms used to evaluate FREEEAGLE. "CNN-7" is the convolutional neural network composed of 4 convolutional layers and 3 linear layers, following the architecture used for CIFAR-10 in [30]. L_{sep} is the position of the selected inspected layer.

Dataset	Class Quantity	Model Architecture	Trigger Type	L_{sep}
GTSRB	43	GoogLeNet	patch, blending, filter	10th
ImageNet-R	20	ResNet-50	patch, blending, filter, natural	10th
CIFAR-10	10	VGG-16	patch, blending, filter	8th
		CNN-7	composite	5th
MNIST	10	CNN-7	patch, blending, filter	5th

Anomaly Metric Modification. As demonstrated above, models with abnormally high $M_{trojaned}$ values should be considered as trojaned. However, there is one special case where the trojaned models have lower $M_{trojaned}$ values than benign models. Such a phenomenon is caused by the fact that sometimes there are naturally similar class pairs that also result in high v values. For example, the naturally similar class pair "dog (5) & cat (3)" makes v of the class "cat (3)" relatively larger, as shown in the column "3" of Figure 2 (a). In this case, if the backdoor-related class pair's abnormality is not significant enough, i.e., v of the target class is not large enough, $M_{trojaned}$ of the trojaned model will be lower than that of the benign model, rather than higher, as shown in Figure 2 (c). The reason is that according to the definition of $M_{trojaned}$ in Equation 5, $M_{trojaned}$ is large when the largest v is significantly larger than those of all other classes. On the contrary, the more classes that have v comparable with the largest v, the smaller $M_{trojaned}$ is. Thus, in the case of Figure 2 (c), with the backdoor introducing the backdoor-related class pair whose abnormality is not significant, $M_{trojaned}$ decreases rather than increases compared with the benign model. Thus, models with abnormally low $M_{trojaned}$ values should also be considered as trojaned.

To sum up, considering the two categories of cases demonstrated above, not only models with abnormally high $M_{trojaned}$ values are considered as trojaned, but models with abnormally low $M_{trojaned}$ values should also be considered as trojaned. Thus, given a model to be inspected, we modify its anomaly metric as the extent to which its $M_{trojaned}$ value deviates from the $M_{trojaned}$ value of benign models. Specifically, $M_{trojaned}$ is modified as follows:

$$M'_{trojaned} = abs(M_{trojaned} - M_{const}) \tag{6}$$

where abs represents computing the absolute value and M_{const}

Table 3: Parameter settings about training clean and trojaned models. Definitions of the absolute poison ratio $r_{poison,absolute}$ and the relative poison ratio $r_{poison,relative}$ are illustrated in Section 4.2.

Parameter	Description	Value
β	momentum	0.9
lr	learning rate	0.025
λ	L2 regularization parameter	1e-4
n_{epoch}	quantity of training epochs	20
α	transparency of the blending trigger	0.2
$r_{poison,relative}$	relative poison ratio for class-specific backdoors	0.5
$r_{poison,absolute}$	absolute poison ratio for class-agnostic backdoors	0.2
s_{lr}	step of learning rate scheduler	15
γ_{lr}	multiplicative factor of learning rate scheduler	0.1

is a constant that represents the approximate $M_{trojaned}$ value of benign models. In our experiments, M_{const} is set to 1.5 for 1-channel image datasets like MNIST. For 3-channel image datasets like CIFAR-10, M_{const} is set to 1.0. We denote the modified anomaly metric as $M'_{trojaned}$, and models with high $M'_{trojaned}$ values are considered as trojaned. By this means, FREEEAGLE can detect both cases of trojaned models demonstrated above.

4 Experiment Setup

4.1 Datasets, Model Architectures and Trojans

Table 2 provides an overview of the datasets, model architectures, and trojan attack settings used for evaluation. We evaluate FREEEAGLE on four datasets and four model architectures, including GTSRB [20] with GoogLeNet [43], ImageNet-R [9] with ResNet-50 [18], MNIST [26] with CNN-7, and CIFAR-10 [25] with VGG-16/CNN-7 [30, 42]. For details about the four datasets, please refer to Appendix B. As for the trojan attack settings, in the perspective the attack goal, we evaluate FREEEAGLE against class-agnostic and class-specific trojan attacks, as described in Section 2.2. In the perspective of trojan trigger forms, we evaluate FREEEAGLE against both pixel-space and feature-space triggers, as illustrated in Section 2.3. In particular, the pixel-space triggers in this paper include the patch trigger and the blending trigger. The feature-space triggers in this paper include the natural trigger, the filter trigger, and the composite trigger. Examples of these triggers can be found in Section 2.3. Note that for the filter trigger, we use the vintage-photography-style filter for 3-channel image datasets, i.e, ImageNet-R, CIFAR-10 and GTSRB. While for the 1-channel image dataset MNIST, we use the negative color filter. Detailed algorithms of these two filters can be found at Appendix A.

4.2 Parameter Settings

For FREEEAGLE, the settings of L_{sep} for different model architectures are shown in Table 2. The parameter settings for training the clean and trojaned models are shown in Table 3. For parameters about training models, e.g., momentum β and weight decay λ, they are set following common practice [18]. For parameters about backdoors, they are set following previous works on neural trojans [7, 44]. The absolute poison ratio $r_{poison,absolute}$ is defined as the number of samples with the trigger divided by the size of the entire dataset. Different from $r_{poison,absolute}$, the relative poison ratio $r_{poison,relative}$ is defined as the number of samples with the trigger divided by the number of "poisonable" samples. For example, assuming that $r_{poison,relative}$ is set to 0.5, and the training dataset is CIFAR-10, which has 5,000 training samples for each class. If the attacker wants to inject the class-specific backdoor with one source class into the model, then the "poisonable" training samples are the 5,000 training samples of this source class. Thus, there are $1 \times 5,000 \times 0.5 = 2,500$ "poisoned" samples, i.e., samples with the trigger.

For the class-specific backdoor, $r_{poison,relative}$ is more suitable to describe the proportion of samples with the trigger in the training dataset, while $r_{poison,absolute}$ is more suitable for the class-agnostic backdoor. In this paper, if not specifically mentioned, we set $r_{poison,relative}$ to 0.5 for class-specific backdoors and $r_{poison,absolute}$ to 0.2 for class-agnostic backdoors, as shown in Table 3.

4.3 Training Clean and Trojaned Models

To evaluate the detection performance of FREEEAGLE and other trojan detectors, we train thousands of clean and trojaned models under the parameter settings illustrated in Section 4.2. All clean and trojaned models are trained with standard data augmentations and perform well on their original tasks. The trojaned attacks also achieve high ASR (attack success rates) on the trojaned models. For more details about the trained clean and trojaned models, please refer to Appendix C.

4.4 Experiment Environment

The clean and trojaned models are trained on four NVIDIA RTX 2080Ti GPU cards. Experiments about trojan detection are accelerated by one NVIDIA RTX 3090 GPU card. All DNN models are implemented using PyTorch [37].

5 Defense Evaluation

In this section, we introduce the evaluation metric and baseline methods, then analyze the performance of trojan detectors under various settings.

5.1 Evaluation Metric

As discussed in [2], during the evaluation, the test set must not be used to parameterize the method. Thus, we choose the

Table 4: Defense performance of FreeEagle and baseline methods measured by TPR and FPR (true/false positive rate). It can be seen that FreeEagle outperforms all baseline methods under most settings of datasets, model architectures, and trojan attacks. For each setting, e.g., "GTSRB, GoogLeNet, Class-Agnostic and Patch Trigger", the result with the highest TPR - FPR is highlighted in bold.

Trojan Detection Method	Dataset	Model Architecture	Backdoor Settings & TPR/FPR					
			Class-Agnostic			Class-Specific		
			Patch Trigger	Blending Trigger	Filter Trigger	Patch Trigger	Blending Trigger	Filter Trigger
FreeEagle	GTSRB	GoogLeNet	0.99/0.03	0.99/0.04	**1.00/0.03**	**0.89/0.03**	**0.76/0.04**	**0.84/0.05**
	ImageNet-R	ResNet-50	**0.99/0.04**	0.86/0.03	**0.99/0.02**	**0.74/0.03**	**0.73/0.04**	**0.78/0.05**
	CIFAR-10	VGG-16	**0.98/0.03**	0.73/0.04	**0.85/0.04**	**0.71/0.05**	**0.72/0.05**	**0.74/0.04**
	MNIST	CNN-7	**0.97/0.03**	0.81/0.05	**0.79/0.01**	**0.78/0.03**	**0.70/0.04**	**0.72/0.03**
DF-TND	GTSRB	GoogLeNet	0.23/0.05	0.08/0.04	0.31/0.05	0.19/0.05	0.17/0.05	0.28/0.04
	ImageNet-R	ResNet-50	0.76/0.05	0.32/0.05	0.90/0.03	0.18/0.05	0.23/0.05	0.38/0.05
	CIFAR-10	VGG-16	0.00/0.02	0.00/0.04	0.00/0.03	0.00/0.04	0.01/0.03	0.03/0.05
	MNIST	CNN-7	0.05/0.04	0.23/0.05	0.00/0.02	0.04/0.01	0.09/0.05	0.03/0.05
STRIP	GTSRB	GoogLeNet	0.97/0.01	0.57/0.05	0.34/0.05	0.10/0.05	0.01/0.05	0.11/0.05
	ImageNet-R	ResNet-50	0.44/0.05	0.53/0.05	0.14/0.05	0.10/0.05	0.03/0.02	0.07/0.03
	CIFAR-10	VGG-16	0.89/0.04	**0.92/0.04**	0.10/0.03	0.00/0.02	0.04/0.05	0.02/0.05
	MNIST	CNN-7	0.83/0.05	0.00/0.01	0.00/0.02	0.00/0.04	0.00/0.03	0.00/0.01
ANP	GTSRB	GoogLeNet	0.90/0.05	0.74/0.05	0.53/0.05	0.28/0.05	0.13/0.05	0.14/0.05
	ImageNet-R	ResNet-50	0.99/0.05	**0.96/0.03**	0.74/0.05	0.31/0.05	0.23/0.05	0.19/0.05
	CIFAR-10	VGG-16	0.90/0.01	0.76/0.04	0.77/0.03	0.62/0.05	0.51/0.05	0.57/0.05
	MNIST	CNN-7	0.83/0.05	0.86/0.05	0.73/0.05	0.71/0.05	0.68/0.05	0.43/0.05
NC	GTSRB	GoogLeNet	**1.00/0.00**	**1.00/0.00**	0.51/0.05	0.21/0.05	0.33/0.05	0.04/0.05
	ImageNet-R	ResNet-50	0.75/0.00	0.68/0.02	0.23/0.05	0.00/0.00	0.00/0.00	0.00/0.00
	CIFAR-10	VGG-16	0.90/0.00	0.70/0.00	0.13/0.05	0.07/0.05	0.02/0.04	0.02/0.05
	MNIST	CNN-7	0.83/0.00	**0.90/0.00**	0.32/0.02	0.23/0.05	0.13/0.05	0.28/0.02
ABS	GTSRB	GoogLeNet	0.56/0.05	0.62/0.04	0.34/0.05	0.43/0.05	0.26/0.04	0.13/0.05
	ImageNet-R	ResNet-50	0.67/0.05	0.22/0.01	0.73/0.03	0.43/0.05	0.40/0.04	0.32/0.05
	CIFAR-10	VGG-16	0.37/0.04	0.61/0.05	0.21/0.04	0.56/0.05	0.25/0.02	0.26/0.05
	MNIST	CNN-7	0.71/0.05	0.64/0.05	0.23/0.04	0.35/0.02	0.15/0.05	0.23/0.05

evaluation metric as TPR/FPR (true/false positive rate) under a fixed threshold that is determined on results gained from the training set. Specifically, we first train a set of trojaned and benign models, as demonstrated in Section 4.3. Then for each setting, e.g., "GTSRB, GoogLeNet, Class-Agnostic and Patch Trigger", we randomly select 30% trojaned models and 30% benign models as the training set to determine the threshold of the method. The remaining trojaned and benign models are used as the test set. We fix the threshold as the lowest value that corresponds to a FPR that is no larger than 0.05 on the training set, then evaluate the performance of the method by the TPR/FPR computed on the test set. We repeat the above procedure for 10 times and report the average TPR/FPR as the final result.

5.2 Baseline Methods

Besides the data-free trojan detection method DF-TND, we also compare FreeEagle with four non-data-free methods, including STRIP, ANP [52], NC and ABS. STRIP, NC and ABS are briefly introduced in Section 2.4. ANP is a method originally designed for backdoor removal rather than backdoor detection. However, with its intuition that the neurons of trojaned models are more sensitive to "adversarial neuron perturbations" [52], we can adapt ANP to backdoor detection. The basic idea is to apply adversarial neuron perturbations to a model, then if the model's accuracy on the original task significantly drops, it is predicted as a trojaned model. For the implementation details of the baseline methods, please refer to Appendix D.

5.3 Defending Against Pixel-Space Triggers

The defense performance of FreeEagle and baseline methods against the patch/blending trigger is shown in Table 4. It can be seen that FreeEagle outperforms the data-free detection method DF-TND on all evaluated datasets, model architectures, and backdoor settings. Besides, FreeEagle even outperforms non-data-free baseline methods under many settings. It can be concluded that FreeEagle is robust against diverse trigger types, class-agnostic/class-specific backdoors.

We find that FreeEagle performs better when the task has more classes. Taking the class-agnostic backdoor with the blending trigger as an example, as shown in Table 4, the TPR/FPR of FreeEagle is 0.99/0.04 on GTSRB (43 classes). However, the TPR/FPR of FreeEagle degrades to 0.73/0.04 on CIFAR-10 (10 classes). The reason is that FreeEagle computes the anomaly metric by analyzing the outliers of all classes. Thus, if the task has more classes, FreeEagle will be able to analyze more data points and therefore perform better when detecting outliers.

Another observation is that the blending trigger is more evasive than the patch trigger. For example, on GTSRB, FreeEagle achieves 0.76/0.04 TPR/FPR when detecting class-specific backdoor with the blending trigger, while the TPR/FPR is 0.89/0.03 for class-specific backdoor with the patch trigger. The possible reasons are analyzed as follows. The patch trigger has a small size, and thus after being forward propagated through convolutional layers, it can only affect a small set of neurons of classifier layers [54]. On the contrary, the blending trigger covers a large area of the original image and can affect a larger set of inner neurons to activate the neural trojan. Thus, as the patch trigger affects fewer neurons, these neurons have to learn a stronger connection with the target label, reducing the evasiveness of the backdoor.

We also find that the class-specific backdoor is more evasive than the class-agnostic backdoor. For example, on GTSRB, for backdoors with the blending trigger, FreeEagle achieves 0.76/0.04 TPR/FPR when detecting the class-specific backdoor, while the TPR/FPR is 0.99/0.04 for the class-agnostic backdoor. This conclusion also holds for all five baseline methods. As mentioned by Gao et al. [14], this is because the class-specific backdoor only requires the trigger feature to be dominant over the benign features of the source classes rather than all classes, which makes the trigger feature less dominant than that of the class-agnostic backdoor. As a result, the class-specific backdoor is more evasive.

Compared with FreeEagle, all five baseline methods are less effective against class-specific backdoors. The reason of DF-TND and NC's failure on class-specific backdoors is that they rely on recovering universal perturbations that cause the model to misclassify **any** sample as the target class. Thus, their insights may not hold for class-specific backdoors. Similarly, STRIP is based on the intuition that the trigger feature is much more dominant than **any** benign features. Though this intuition holds for class-agnostic backdoors, it is less suitable for class-specific backdoors because the "class-specific" attack goal makes the trigger feature less dominant over non-source-class benign features. ANP and ABS detect neural trojans by analyzing neuron-level behaviors and do not depend on the dominance of trigger features, so they perform better than STRIP, NC and DF-TND. However, as shown in Table 4, their defense performance against class-specific backdoors also degrades compared with that against class-agnostic backdoors. For example, under the setting of "CIFAR-10, VGG-16, Patch Trigger", the TPR/FPR of ANP is 0.90/0.01 for the class-agnostic backdoor but 0.62/0.05 for the class-specific backdoor. The reason may be that class-specific backdoors not only have less dominant trigger features, but also have less evident neuron-level abnormal behaviors.

Table 5: Defense performance of FreeEagle and baseline methods against backdoors using the natural/composite trigger. It can be seen that the defense performance of FreeEagle is better than or comparable with all baseline methods.

Dataset	Model	Trigger Type	Detection Method	TPR/FPR
ImageNet -R	ResNet-50	Natural	FreeEagle	**0.62/0.05**
			DF-TND	0.00/0.04
			STRIP	0.08/0.05
			ANP	0.10/0.05
			NC	0.00/0.03
			ABS	0.31/0.01
CIFAR-10	CNN-7	Composite	FreeEagle	0.86/0.05
			DF-TND	0.00/0.04
			STRIP	0.00/0.03
			ANP	**0.90/0.05**
			NC	0.00/0.05
			ABS	0.16/0.03

We also find that the defense performance of the data-free baseline method DF-TND seems to be model-dependent. In particular, DF-TND seems to have better performance on deeper models. For example, when detecting models trojaned with class-agnostic backdoors using the patch trigger, DF-TND gets 0.76/0.05 TPR/FPR on ResNet-50 (50 layers) but 0.23/0.05 TPR/FPR on GoogLeNet (22 layers). The reason may be that DF-TND's methodology includes the step to reverse engineer the input image. As feature extractors with more layers provide higher-quality recovered input images, DF-TND has better detection performance on deeper models. In comparison, FreeEagle reverse engineers dummy intermediate representations instead of the raw inputs, thus is more generalizable to different model architectures.

5.4 Defending Against Feature-Space Triggers

The defense performance of FreeEagle and five baseline methods against the filter trigger is shown in Table 4. For both class-agnostic and class-specific backdoors with filter triggers, FreeEagle outperforms all five baseline methods.

FreeEagle is also effective against backdoors using natural or composite triggers. As shown in Table 5, for the natural trigger, FreeEagle achieves 0.62/0.05 TPR/FPR, outperforming all five baseline methods. For the composite trigger, FreeEagle achieves 0.86/0.05 TPR/FPR, which is comparable with the best performance (0.90/0.05 TPR/FPR) achieved by ANP. Note that FreeEagle is data-free while ANP relies on the access to clean labeled samples.

5.5 Overall Defense Performance

In the above two subsections, we evaluate FreeEagle on specific settings of datasets and trojan attacks. However, in

Table 6: Comparison of the average time cost (in seconds) to inspect one model. All methods are accelerated by one NVIDIA RTX 3090 GPU.

	ResNet50, ImageNet-R	VGG-16, CIFAR-10	GoogLeNet, GTSRB	CNN-7, MNIST
FreeEagle	602	81	1,058	18
DF-TND	58	43	106	13
STRIP	5	4	5	2
ANP	28	49	77	31
NC	6,041	4,412	36,124	1,913
ABS	291	7	58	4

real-world scenarios, the defender usually does not know the specific trojan attack setting. Thus, it is crucial to evaluate the trojan detection method on a set of models trojaned with different trojan attacks. The experiment result shows that FreeEagle achieves 0.65/0.04 TPR/FPR if evaluated on all the trojaned and benign models trained in Section 4.3, while the TPR/FPRs of DF-TND, STRIP, ANP, NC and ABS are 0.11/0.05, 0.14/0.05, 0.33/0.05, 0.28/0.05 and 0.21/0.05, respectively. Note that the above results are also computed on the test set under the fixed threshold determined on the training set, as demonstrated in Section 5.1. This indicates that FreeEagle can get good detection performance across different models, datasets, and trojan attacks.

5.6 Time Cost

Designing efficient defense methods is essential for practical application [15]. As FreeEagle only analyzes the classifier part of the inspected model, its time cost is low. As shown in Table 6, with the acceleration of one NVIDIA RTX 3090 GPU card, FreeEagle can finish inspecting one model within 20 minutes on all four evaluated settings. Another observation is that the time cost of FreeEagle increases as the number of classes grows, for FreeEagle needs to generate dummy intermediate representations for each class. However, even if there are more classes, FreeEagle can still inspect models efficiently, as the average time cost for one class is only around 20 seconds. From Table 6, we can conclude that STRIP is the fastest detector. However, STRIP relies on the access to clean samples and samples with the trigger to inspect a model, thus has limited applicable scenarios compared with FreeEagle.

6 Possible Adaptive Attacks

We investigate two adaptive attack strategies [6, 45] that may bypass FreeEagle: (1) the adaptive attacker shapes the posterior during training the trojaned model; (2) for input images containing the trigger, the adaptive attacker makes the feature extractor output similar feature maps with clean images from the target class. We find that these adaptive attacks are not effective against FreeEagle or cannot bypass FreeEagle without significantly reducing the usability of the backdoor. Interestingly, we find that there is a trade-off between the trojan attack's usability and evasiveness, i.e., to gain evasiveness against FreeEagle, the trojan attack has to reduce its attack success rate. The experiments are conduct on the GTSRB/CIFAR-10 datasets and the backdoor is class-agnostic. Three trigger types are considered, including the patch/blending/filter trigger. In this section, the reported TPR/FPR is the **overall** defense performance against models trojaned with these three trigger types.

6.1 Posterior Shaping

The adaptive attacker tries to bypass FreeEagle by shaping the posterior during training the trojaned model. As demonstrated in Section 3.2, FreeEagle is based on the intuition that for a trojaned model, the output posterior on the target class will be higher than the posteriors on the non-target classes. Thus, the attacker may try to bypass FreeEagle by decreasing the posterior of the target class while increasing the posteriors of non-target classes. Specifically, for training samples with the target label in the poisoned dataset, i.e., target-class samples and samples with the trigger, the posterior is shaped using the following loss function:

$$loss_{shaping} = MSE(Softmax(M(x_t)), \mathbf{y}_{shaped}) \qquad (7)$$

where MSE is the mean square loss, $softmax$ is the softmax function, M is the model, x_t is the training samples with the target label and $\mathbf{y}_{shaped}$ is the shaped one-hot vector. $\mathbf{y}_{shaped}$ is computed as:

$$\mathbf{y}_{shaped} = Clamp(\mathbf{y}, p_{min}, p_{max}) \qquad (8)$$

$$p_{min} = (1.0 - p_{max})/(N_{class} - 1) \qquad 0.5 < p_{max} < 1.0 \quad (9)$$

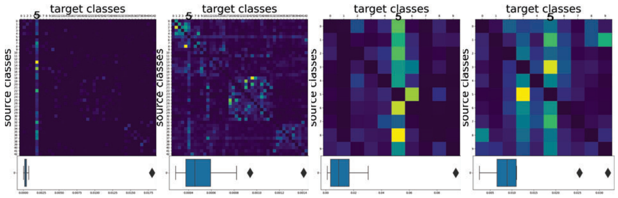
where $Clamp$ is the clamp function, which bounds the minimum and maximum values of a given vector. $\mathbf{y}$ is the classic one-hot label. p_{max} is the parameter that decides the maximum value of $\mathbf{y}$, which is set to 0.6 in the experiment. N_{class} is the number of classes.

Some case studies of FreeEagle against this adaptive attack are shown in Figure 3. Though posterior shaping does make the trojaned model more evasive against FreeEagle, FreeEagle manages to outline the target class of the backdoor and computes a relatively large $M'_{trojaned}$. Actually, our experiment results show that with this adaptive attack, the TPR/FPR of FreeEagle only degrades slightly from 0.99/0.04 to 0.93/0.03 on the GTSRB dataset. On the CIFAR-10 dataset, the TPR/FPR of FreeEagle only degrades from 0.88/0.05 to 0.82/0.04. Thus, posterior shaping has minor effect on the defense performance of FreeEagle.

6.2 Trojaning the Feature Extractor

Inspired by the methodology of BadEncoder [21], the adaptive attacker can trojan the feature extractor part of the model, then fixes the parameters of the feature extractor part and trains the classifier part on the clean training dataset. The detailed adaptive attack method is as follows. (1) The attacker trains

Figure 3: Mat_p and $M'_{trojaned}$ computed on trojaned models trained with/without the adaptive attack strategy of posterior shaping. Bright yellow color represents abnormality.



(a) Without Posterior Shaping GTSRB GoogLeNet $M'_{trojaned} = (63.958)$ (b) With Posterior Shaping GTSRB GoogLeNet $M'_{trojaned} = (2.347)$ (c) Without Posterior Shaping CIFAR-10 VGG-16 $M'_{trojaned} = (4.080)$ (d) With Posterior Shaping CIFAR-10 VGG-16 $M'_{trojaned} = (3.788)$

Table 7: FREEEAGLE's defense performance against the adaptive attack strategy of trojaning the feature extractor on the GTSRB and CIFAR-10 datasets. L_{atk} is the layer that the adaptive attacker selects to bypass FREEEAGLE, and L_{def} is the layer selected by FREEEAGLE when inspecting the model.

TPR/FPR L_{atk} L_{def}	10th	16th	22nd
10th	0.44/0.02	0.79/0.05	0.87/0.05

(a) GTSRB, GoogLeNet

TPR/FPR L_{atk} L_{def}	8th	11th	14th
8th	0.43/0.03	0.67/0.04	0.76/0.05

(b) CIFAR-10, VGG-16

Table 8: FREEEAGLE's defense performance against non-adaptive attacks when L_{def} is set to different layers.

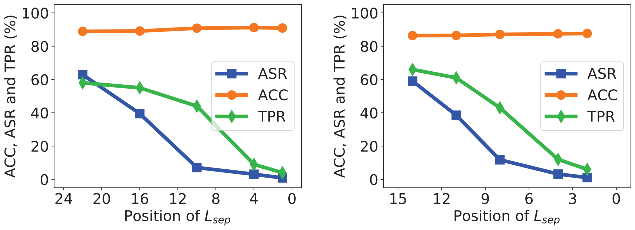
L_{def}	10th	16th	22nd
TPR	0.99	0.93	0.98
FPR	0.04	0.03	0.02

(a) GTSRB, GoogLeNet

L_{def}	8th	11th	14th
TPR	0.88	0.83	0.82
FPR	0.05	0.05	0.02

(b) CIFAR-10, VGG-16

Figure 4: The trade-off between the evasiveness and the utility of the adaptive backdoor attack of trojaning less layers. "TPR" represents the TPR when $L_{def} = L_{atk}$ and FPR is no larger than 0.05. The backdoor setting is "agnostic backdoor" and the target class is class 0.



(a) GTSRB, GoogLeNet (b) CIFAR-10, VGG-16

a clean model on the clean training dataset, then takes its feature extractor part as F_{clean}. (2) The attacker selects clean training samples from the target class as reference samples and records their embeddings, i.e., the outputs of F_{clean}. We denote the embedding of a reference sample as IR_{ref}. (3) The attacker initializes a feature extractor F_{bad} and trains it with the following two goals. Firstly, for training samples that do not contain the trigger, F_{bad} should output similar embeddings with F_{clean}. Secondly, for training samples containing the trigger, the output $\mathbf{y}_{bad}$ of F_{bad} should be similar with a IR_{ref}. Specifically, if the training sample contains the trigger, the loss function is defined as the minimum over all candidate IR_{ref}, of the average squared difference between $\mathbf{y}_{bad}$ and IR_{ref}. (4) After training F_{bad}, the attacker initializes a model and replaces its feature extractor part with F_{bad}. Finally, the attacker fixes parameters of the feature extractor part then trains the model on the clean training dataset.

By this means, the trojan behavior should be conducted in the feature extractor part rather than the classifier part of the model. Thus, the trojaned model should become more evasive against FREEEAGLE. In the following, we use L_{atk} to denote the layer that the adaptive attacker selects to bypass FREEEAGLE, and L_{def} denotes the layer selected by FREEEAGLE when inspecting the model. As shown in Table 7a, when $L_{def} = L_{atk}$, the adaptive attack does become more evasive against FREEEAGLE, e.g., the TPR/FPR degrades to 0.44/0.02 when L_{def} and L_{atk} are both 10 on the GTSRB dataset. Meanwhile, if FREEEAGLE inspects a shallower layer than the adaptive attacker, i.e., L_{def} is smaller than L_{atk}, FREEEAGLE can still get excellent performance. Thus, to bypass FREEEAGLE, the adaptive attacker should set L_{atk} no larger than L_{def}.

However, as shown in Figure 4, as the adaptive attacker sets smaller L_{atk}, the attack success rate (ASR) rapidly drops. For example, on the GTSRB dataset, the ASR drops to around 0.05 when L_{atk} is 10. Besides, as shown in Table 8a, on the GTSRB dataset, when L_{def} is 10, FREEEAGLE can get good defense performance against non-adaptive attacks, i.e., 0.99/0.04 TPR/FPR. To draw a conclusion, to avoid being bypassed by the adaptive attack strategy of trojaning the feature extractor part, FREEEAGLE can inspect the model with L_{def} set to a low value, e.g., the 10th layer of GoogLeNet and the 8th layer of VGG-16. In this way, on the one hand, if L_{atk} is high, the adaptive attacker cannot bypass FREEEAGLE. On the other hand, if L_{atk} is low, the ASR will also be low, i.e., the adaptive attacker will fail to inject a usable backdoor. Similar results can be obtained on the CIFAR-10 dataset, as shown in Table 7b, Table 8b and Figure 4b.

7 Discussion

If the trojaned model is trained without data augmentation, FREEEAGLE will become less indicative of discriminating trojaned and benign models, especially for the class-specific backdoors. However, we argue that training trojaned models without data augmentation also reduces the usability of the backdoor. Further, turning off data augmentation is also harmful to the model's performance on the original task.

Another limitation of FreeEagle is that its defense performance is limited if the dataset has few classes. The reason is that FreeEagle relies on detecting outliers from the posteriors of all classes, and detecting outliers will be more difficult if there are few data points. However, collecting auxiliary labeled data will be much easier if the dataset has few classes, making existing non-data-free backdoor detectors usable. For example, ABS [34] only needs two auxiliary clean labeled samples for binary tasks, one for the negative class and the other for the positive class. Thus, the necessity of data-free methods is reduced for datasets with few classes. Besides, the insights of FreeEagle still hold for datasets with few classes, and it is the quantile-related outlier detection step that makes it sensitive to the number of classes. Thus, if we specially design quantile-free anomaly metric for few-class tasks, e.g., using the range of $\mathbf{v}$, i.e., $\max(\mathbf{v}) - \min(\mathbf{v})$ as the anomaly metric for two-class tasks, this limitation will be mitigated. We leave the extension of FreeEagle to few-class datasets as future work.

8 Related Work

In addition to the trojan defenses discussed in Section 2.4, FreeEagle is also related to the following work.

8.1 Backdoor Mitigation

Different from backdoor detection methods which aim to tell whether a given model is backdoored, backdoor mitigation methods aim to remove the backdoor inside a backdoored model [32,38]. This problem is also named "backdoor blind removal" in some literature [13]. The backdoor mitigation method is required to remove the backdoor inside a model without a significant drop on the performance of the original task. Neural Cleanse [46] mitigates backdoor via unlearning-based DNN patching, i.e., using inputs with the reversed trigger and correct labels to fine-tune the backdoored model. GangSweep [58] mitigates backdoor using a similar paradigm with Neural Cleanse, but with the improvement of using a generative adversarial network (GAN) to generate the trigger pattern. AI-Lancet [57] uses a set of clean-poisoned sample pairs to locate error-inducing neurons in DNN and then fine-tunes or flips the sign of these neurons (weights) to fix the poisoned DNN. Sharing similar insights with AI-Lancet, adversarial neuron pruning (ANP) [52] uses a small set of clean labeled samples to locate "sensitive neurons" and then prunes these neurons to remove the backdoor. DeepSweep [38] uses data augmentation techniques to mitigate backdoor attacks as well as enhance the model's robustness.

8.2 Backdoor-Free Training

Backdoor-free training is an interesting new research topic in neural trojan defenses, aiming to train a clean model even if the training dataset is poisoned. Li et al. proposed Anti-backdoor learning (ABL) [29] to train a clean model on a poisoned dataset, with the insight that the DNN model learns backdoored data much faster than learning with clean data.

9 Conclusion

In this paper, we present novel insights into the underlying working mechanisms of the trojaned model. We demonstrate that a trojaned model firstly extracts both benign and trigger features, then assign higher priority to the trigger features. Based on the above insights, we propose FreeEagle, the first data-free method that is effective against complex neural trojans. Our experiment results on diverse datasets and model architectures show that FreeEagle is effective against class-specific/class-agnostic backdoors and various trigger types, even outperforming some non-data-free trojan detectors. We also evaluate FreeEagle against possible adaptive attacks and find they cannot effectively bypass FreeEagle.

10 Acknowledgments

We thank our anonymous reviewers and shepherd for their constructive suggestions. This work was partly supported by the National Key Research and Development Program of China under No. 2022YFB3102100, and NSFC under No. 62102360 and U1936215.

References

[1] Amazon. AWS Marketplace. https://aws.amazon.com/marketplace.

[2] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. Dos and don'ts of machine learning in computer security. In *31th USENIX Security Symposium (USENIX Security 22)*, 2022.

[3] Ahmadreza Azizi, Ibrahim Asadullah Tahmid, Asim Waheed, Neal Mangaokar, Jiameng Pu, Mobin Javed, Chandan K Reddy, and Bimal Viswanath. T-Miner: A generative approach to defend against trojan attacks on DNN-based text classification. In *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

[4] Eugene Bagdasaryan and Vitaly Shmatikov. Blind backdoors in deep learning models. In *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

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

[6] Nicholas Carlini, Anish Athalye, Nicolas Papernot, Wieland Brendel, Jonas Rauber, Dimitris Tsipras, Ian Goodfellow, Aleksander Madry, and Alexey Kurakin. On evaluating adversarial robustness. *arXiv preprint arXiv:1902.06705*, 2019.

[7] 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.

[8] Yanjiao Chen, Xueluan Gong, Qian Wang, Xing Di, and Huayang Huang. Backdoor attacks and defenses for deep neural networks in outsourced cloud environments. *IEEE Network*, 34(5):141–147, 2020.

[9] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *IEEE conference on computer vision and pattern recognition (CVPR)*. IEEE, 2009.

[10] 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 (ACSAC)*, 2020.

[11] Yinpeng Dong, Xiao Yang, Zhijie Deng, Tianyu Pang, Zihao Xiao, Hang Su, and Jun Zhu. Black-box detection of backdoor attacks with limited information and data. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

[12] Hugging Face. Hugging Face Models. `https://huggingface.co/models`.

[13] Yansong Gao, Bao Gia Doan, Zhi Zhang, Siqi Ma, Jiliang Zhang, Anmin Fu, Surya Nepal, and Hyoungshick Kim. Backdoor attacks and countermeasures on deep learning: A comprehensive review. *arXiv preprint arXiv:2007.10760*, 2020.

[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 *Annual Computer Security Applications Conference (ACSAC)*, 2019.

[15] Micah Goldblum, Dimitris Tsipras, Chulin Xie, Xinyun Chen, Avi Schwarzschild, Dawn Song, Aleksander Madry, Bo Li, and Tom Goldstein. Data security for machine learning: data poisoning, backdoor attacks, and defenses. *arXiv e-prints*, 2020.

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

[17] 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.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE conference on computer vision and pattern recognition (CVPR)*, 2016.

[19] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012.

[20] Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, and Christian Igel. Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark. In *International Joint Conference on Neural Networks*, number 1288, 2013.

[21] 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.

[22] B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems (TITS)*, 2021.

[23] Jing Yu Koh. Model Zoo. `https://modelzoo.co/`.

[24] Soheil Kolouri, Aniruddha Saha, Hamed Pirsiavash, and Heiko Hoffmann. Universal litmus patterns: Revealing backdoor attacks in CNNs. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

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

[26] 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.

[27] Shaofeng Li, Hui Liu, Tian Dong, Benjamin Zi Hao Zhao, Minhui Xue, Haojin Zhu, and Jialiang Lu. Hidden backdoors in human-centric language models. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2021.

[28] Xiaomeng Li, Lequan Yu, Yueming Jin, Chi-Wing Fu, Lei Xing, and Pheng-Ann Heng. Difficulty-aware meta-learning for rare disease diagnosis. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2020.

[29] 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 (NeurIPS)*, 2021.

[30] Junyu Lin, Lei Xu, Yingqi Liu, and Xiangyu Zhang. Composite backdoor attack for deep neural network by mixing existing benign features. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2020.

[31] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen Awm Van Der Laak, Bram Van Ginneken, and Clara I Sánchez. A survey on deep learning in medical image analysis. *Medical image analysis*, 42:60–88, 2017.

[32] 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 (RAID)*. Springer, 2018.

[33] Weiyang Liu, Yandong Wen, Zhiding Yu, Ming Li, Bhiksha Raj, and Le Song. Sphereface: Deep hypersphere embedding for face recognition. In *IEEE conference on computer vision and pattern recognition (CVPR)*, 2017.

[34] 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 *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2019.

[35] Yingqi Liu, Shiqing Ma, Yousra Aafer, W. Lee, Juan Zhai, Weihang Wang, and X. Zhang. Trojaning attack on neural networks. In *Annual Network and Distributed System Security Symposium (NDSS)*, 2018.

[36] Yunfei Liu, Xingjun Ma, James Bailey, and Feng Lu. Reflection backdoor: A natural backdoor attack on deep neural networks. In *European Conference on Computer Vision (ECCV)*. Springer, 2020.

[37] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems (NeurIPS)*, 2019.

[38] Han Qiu, Yi Zeng, Shangwei Guo, Tianwei Zhang, Meikang Qiu, and Bhavani Thuraisingham. DeepSweep: An evaluation framework for mitigating dnn backdoor attacks using data augmentation. In *ACM Asia Conference on Computer and Communications Security (CCS)*, 2021.

[39] Avi Schwarzschild, Micah Goldblum, Arjun Gupta, John P Dickerson, and Tom Goldstein. Just how toxic is data poisoning? A unified benchmark for backdoor and data poisoning attacks. In *International Conference on Machine Learning (ICML)*. PMLR, 2021.

[40] Ali Shafahi, W Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. Poison frogs! Targeted clean-label poisoning attacks on neural networks. *Advances in neural information processing systems (NeurIPS)*, 2018.

[41] Lujia Shen, Shouling Ji, Xuhong Zhang, Jinfeng Li, Jing Chen, Jie Shi, Chengfang Fang, Jianwei Yin, and Ting Wang. Backdoor pre-trained models can transfer to all. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2021.

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

[43] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *IEEE conference on computer vision and pattern recognition (CVPR)*, 2015.

[44] Di Tang, XiaoFeng Wang, Haixu Tang, and Kehuan Zhang. Demon in the variant: Statistical analysis of DNNs for robust backdoor contamination detection. In *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

[45] Florian Tramer, Nicholas Carlini, Wieland Brendel, and Aleksander Madry. On adaptive attacks to adversarial example defenses. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[46] 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 *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2019.

[47] Hongyi Wang, Kartik Sreenivasan, Shashank Rajput, Harit Vishwakarma, Saurabh Agarwal, Jy-yong Sohn, Kangwook Lee, and Dimitris Papailiopoulos. Attack of the tails: Yes, you really can backdoor federated learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 33, 2020.

[48] Ren Wang, Gaoyuan Zhang, Sijia Liu, Pin-Yu Chen, Jinjun Xiong, and Meng Wang. Practical detection of trojan neural networks: Data-limited and data-free cases. In *European Conference on Computer Vision (ECCV)*. Springer, 2020.

[49] Yulong Wang, Hang Su, Bo Zhang, and Xiaolin Hu. Interpret neural networks by identifying critical data routing paths. In *IEEE conference on computer vision and pattern recognition (CVPR)*, 2018.

[50] David F Williamson, Robert A Parker, and Juliette S Kendrick. The box plot: A simple visual method to interpret data. *Annals of internal medicine*, 110(11):916–921, 1989.

[51] Baoyuan Wu, Hongrui Chen, Mingda Zhang, Zihao Zhu, Shaokui Wei, Danni Yuan, Chao Shen, and Hongyuan Zha. Backdoorbench: A comprehensive benchmark of backdoor learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[52] Dongxian Wu and Yisen Wang. Adversarial neuron pruning purifies backdoored deep models. *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

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

[54] Chong Xiang, Arjun Nitin Bhagoji, Vikash Sehwag, and Prateek Mittal. PatchGuard: A provably robust defense against adversarial patches via small receptive fields and masking. In *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

[55] Xiaojun Xu, Qi Wang, Huichen Li, Nikita Borisov, Carl A Gunter, and Bo Li. Detecting AI trojans using meta neural analysis. In *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2021.

[56] Xinyang Zhang, Zheng Zhang, Shouling Ji, and Ting Wang. Trojaning language models for fun and profit. In *IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE Computer Society, 2021.

[57] Yue Zhao, Hong Zhu, Kai Chen, and Shengzhi Zhang. Ai-lancet: Locating error-inducing neurons to optimize neural networks. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2021.

[58] Liuwan Zhu, Rui Ning, Cong Wang, Chunsheng Xin, and Hongyi Wu. GangSweep: Sweep out neural backdoors by GAN. In *ACM International Conference on Multimedia (MM)*, 2020.

[59] Liuwan Zhu, Rui Ning, Chunsheng Xin, Chonggang Wang, and Hongyi Wu. Clear: Clean-up sample-targeted backdoor in neural networks. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

Algorithm 3 Vintage-photography-style filter

```
Require: A 3-channel image $img$, parameter $p$ used to adjust how heavy the filter is.
$img_1 \leftarrow sqrt(img \cdot [1.0, 0.0, 0.0]) \cdot p$
$img_2 \leftarrow img \cdot [0.0, 1.0, 1.0]$
$img \leftarrow img_1 + img_2$
return $img$
```

Algorithm 4 Negative color filter

```
Require: A 1-channel image $img$.
$img \leftarrow 255 \cdot ones\_like(img) - img$
return $img$
```

Appendix

A Algorithms of the Vintage-Photography-Style Filter And the Negative Color Filter

The algorithms of the vintage-photography-style filter and the negative color filter are shown in Algorithm 3 and Algorithm 4, respectively. Examples of 1-channel images processed by the negative color filter are shown in Figure 5.

B Details of Datasets

The details of the four datasets are listed as follows, including GTSRB, ImageNet-R, CIFAR-10 and MNIST.

German Traffic Sign Recognition Benchmark. German Traffic Sign Recognition Benchmark (GTSRB) [20] is a classic dataset related to the scenario of autonomous driving. The task of GTSRB is to recognize 43 classes of German traffic signs, including the stop sign, several kinds of speed limit signs, and so on. GTSRB is an imbalanced dataset. Specifically, the number of training samples for each class ranges from 210 to 2,250. Thus, we use oversampling to balance the number of samples for each class. After oversampling, each class has 2,250 training samples. For GTSRB, we use GoogLeNet as the model architecture, which is a classic computer vision model proposed in [43].

ImageNet-R. ImageNet is derived from the online competition ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [9], which aims to classify samples of 1,000 classes. For simplicity, we randomly choose 20 classes from ImageNet and name this subset of ImageNet as ImageNet-R (restricted). Details of these 20 classes are shown in Table 9. Each class corresponds to 1,300 training samples and 50 test-

Figure 5: Comparison of 1-channel images (from the MNIST dataset) with and without the negative color filter.

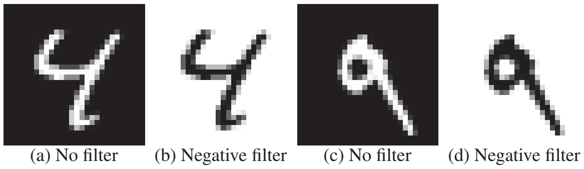

(a) No filter (b) Negative filter (c) No filter (d) Negative filter

Table 9: Details about the 20 classes in ImageNet-R.

Class ID in ImageNet-R	Class ID in ImageNet	Class Description
0	n02114367	grey wolf
1	n02123159	tiger cat
2	n02342885	hamster
3	n02412080	ram
4	n02894605	breakwater
5	n02895154	breastplate
6	n02930766	taxicab
7	n02999410	chain
8	n03089624	confectionery store
9	n03125729	cradle
10	n03141823	crutch
11	n03201208	dining table
12	n03240683	drilling rig
13	n03450230	gown
14	n03773504	missile
15	n03787032	square academic cap
16	n03792782	mountain bike
17	n03929855	pickelhaube
18	n03937543	pill bottle
19	n04162706	seat belt

ing samples. We use ResNet-50 [18] as the model architecture for ImageNet. Samples will be resized to 224×224 before fed to ResNet-50.

CIFAR-10. CIFAR-10 [25] is a classic dataset built for a 10-class image classification task, which has 6,000 samples in the size of 32×32 pixels for each class. Following the common practice of using CIFAR-10, we take 50,000 samples as the training dataset and the other 10,000 as the test dataset. VGG-16 [42] and CNN-7 [30] are used as the model architectures for CIFAR-10.

MNIST. MNIST [26] is a classic dataset built for handwritten digit recognition, which has 60,000 training samples and 10,000 testing samples. Each sample is a 1-channel image in the size of 28×28 pixels. CNN-7 [30] is used as the model architecture for MNIST.

C Details About Clean and Trojaned Models Trained for Defense Evaluation

The details of clean and trojaned models trained for defense evaluation are shown in Table 10. All the clean and trojaned models are trained with standard data augmentations, including randomly cropping and resizing the image, horizontally flipping the image, randomly changing the image's contrast, and random gray scale transformation. Note that the horizontally flipping transformation is disabled for GTSRB because it will change the semantic of some traffic signs. For the ResNet-50 model architecture, we use the ResNet-50 pretrained on ImageNet to accelerate the training process.

D Implementation Details of Baseline Methods

Code Source. We use BackdoorBench [51] to implement baseline methods. The codes can be found at `https://github.com/SCLBD/BackdoorBench`. As for ABS, which is not implemented by BackdoorBench, we use its official PyTorch-version code at `https://github.com/naiyeleo/ABS/blob/master/TrojAI_competition/round1/abs_pytorch_round1.py`. For DF-TND, we use its official code released at `https://github.com/wangren09/TrojanNetDetector`. Note that all baseline methods are implemented with their default parameter settings.

Defender's Knowledge. As an online-detection method, STRIP requires the defender to access inputs with the trigger and a small set of clean samples, so we randomly select 1 image (from the source class) with the trigger and 128 clean images of other classes to inspect a model with STRIP. ABS requires the defender to have a small set of clean samples as seed images, so we randomly select 128 clean images as its seed images. For NC and ANP, we allow them to use the full training dataset and training/testing dataset respectively to maximize their detection ability.

Table 10: Details about clean and trojaned models trained to evaluate trojan detection methods. “Test Acc” is the model’s accuracy of the original task on the clean test dataset. “ASR” represents the attack successful rate of the trojan attack. To extensively evaluate FreeEagle, we train trojaned models with diverse source/target class settings. For example, on CIFAR-10, for the class-specific backdoor with each trigger type, we train all combinations of source-target class pairs, i.e., at least $9 \times 10 = 90$ trojaned models.

Dataset	Model	Trojan Type	Trigger Type	Source Class	Target Class	Model Quantity	Average Test Acc	Average ASR
GTSRB	GoogLeNet	None(Benign)				200	90.23%	
		Class-Agnostic	Patch		0-42	43×4	88.96%	99.95%
			Blending		0-42	43×4	89.64%	99.60%
			Filter		0-42	43×4	88.76%	99.83%
		Class-Specific	Patch	0-42	7,8	(42×2)×2	90.44%	99.92%
			Blending	0-42	7,8	(42×2)×2	90.08%	98.57%
			Filter	0-42	7,8	(42×2)×2	88.91%	96.93%
CIFAR-10	VGG-16	None(Benign)				200	86.12%	
		Class-Agnostic	Patch		0-9	10×20	84.92%	99.86%
			Blending		0-9	10×20	84.95%	99.88%
			Filter		0-9	10×20	85.08%	98.78%
		Class-Specific	Patch	0-9	0-9	(9×10)×2	85.69%	98.03%
			Blending	0-9	0-9	(9×10)×2	86.18%	96.42%
			Filter	0-9	0-9	(9×10)×2	85.84%	95.70%
CIFAR-10	CNN-7	Class-Specific	Composite	0-2	0-2	3×60	83.45%	81.24%
ImageNet-R	ResNet-50	None(Benign)				200	94.74%	
		Class-Agnostic	Patch		0-19	20×10	91.75%	99.13%
			Blending		0-19	20×10	92.27%	97.83%
			Filter		0-19	20×10	94.02%	98.81%
		Class-Specific	Patch	0-19	0,12,14,18	(19×4)×2	92.06%	95.92%
			Blending	0-19	0,12,14,18	(19×4)×2	94.43%	99.87%
			Filter	0-19	0,12,14,18	(19×4)×2	93.20%	97.96%
			Natural	13	0	200	92.72%	91.34%
MNIST	CNN-7	None(Benign)				200	98.65%	
		Class-Agnostic	Patch		0-9	10×20	96.94%	99.69%
			Blending		0-9	10×20	96.92%	99.82%
			Filter		0-9	10×20	97.43%	99.98%
		Class-Specific	Patch	0-9	0-9	(9×10)×2	97.52%	99.21%
			Blending	0-9	0-9	(9×10)×2	97.73%	99.38%
			Filter	0-9	0-9	(9×10)×2	97.61%	99.38%