TrojanDec: Data-free Detection of Trojan Inputs in Self-supervised Learning

Yupei Liu, Yanting Wang, Jinyuan Jia

The Pennsylvania State University
{yzl6415, ykw5450, jinyuan}@psu.edu

Abstract

An image encoder pre-trained by self-supervised learning can be used as a general-purpose feature extractor to build downstream classifiers for various downstream tasks. However, many studies showed that an attacker can embed a trojan into an encoder such that multiple downstream classifiers built based on the trojaned encoder simultaneously inherit the trojan behavior. In this work, we propose TrojanDec, the first data-free method to identify and recover a test input embedded with a trigger. Given a (trojaned or clean) encoder and a test input, TrojanDec first predicts whether the test input is trojaned. If not, the test input is processed in a normal way to maintain the utility. Otherwise, the test input will be further restored to remove the trigger. Our extensive evaluation shows that TrojanDec can effectively identify the trojan (if any) from a given test input and recover it under state-of-the-art trojan attacks. We further demonstrate by experiments that our TrojanDec outperforms the state-of-the-art defenses.

Introduction

Traditional transfer/supervised learning trains a feature extractor using a labeled training dataset, which incurs large costs and human effort to annotate the training dataset. Moreover, it cannot leverage a large amount of unlabeled data that can be collected from the Internet. Self-supervised learning (Devlin et al. 2019; Hadsell, Chopra, and LeCun 2006; He et al. 2020; Chen et al. 2020; Hjelm et al. 2019; Grill et al. 2020) has been designed to address those challenges. In particular, a model provider can use self-supervised learning to pre-train an encoder using a large set of unlabeled data (called *pre-training dataset*). This paradigm can significantly save the labeling effort and thus enables the model provider to significantly increase the size of its pre-training dataset. For example, CLIP (Radford et al. 2021) was pre-trained by OpenAI on 400 million (image, text) pairs collected from the Internet. The model provider can monetize its encoder by deploying it as a cloud service via providing an API to users (Liu et al. 2022). Suppose a user has some training/test inputs for a downstream task. The user can query the API to obtain their feature vectors and then use those feature vectors to train/test a downstream classifier for its downstream task.

Despite the success of self-supervised learning, many existing studies show that it is vulnerable to trojan attacks (Jia, Liu, and Gong 2022; Saha et al. 2022; Carlini and Terzis 2022). Specifically, an adversary can inject a trojan to self-supervised learning encoders by either poisoning the pre-training dataset (Saha et al. 2022; Liu, Jia, and Gong 2022; Carlini and Terzis 2022) or directly manipulating pre-trained encoders' parameters (Jia, Liu, and Gong 2022). The trojaned encoder will produce normal feature vectors for clean inputs, but output feature vectors similar to the feature vector of an attacker-chosen reference object for any test inputs with the attacker-chosen trigger. Furthermore, the trojan behavior will be conveyed to downstream classifiers built upon the trojaned encoder, leading those classifiers to predict the test inputs with the trigger to the ground-truth label (called *target label*) of the attacker-chosen reference object.

Many defenses (Tran, Li, and Madry 2018; Chen et al. 2018; Wang et al. 2019; Xu et al. 2021; Chen et al. 2019; Liu et al. 2019; Gao et al. 2019; Doan, Abbasnejad, and Ranasinghe 2020; Li et al. 2021b) were proposed to mitigate trojan attacks to machine learning models. Depending on which stage those defenses are used, we categorize them into *training-phase defenses* (Tran, Li, and Madry 2018; Chen et al. 2018; Wang et al. 2019; Xu et al. 2021; Chen et al. 2019; Liu et al. 2019; Li et al. 2021b) and *testing-phase defenses* (Doan, Abbasnejad, and Ranasinghe 2020; Li et al. 2021b). A training-phase defense either trains a robust model on a poisoned training dataset or detects/removes the trojan in a trained model. As a result, they require access to the training dataset or the model parameters of the model. In other words, they are not applicable when a defender only has black-box access to a pre-trained model. In our work, we consider a defender (i.e., user) has black-box access to an encoder (we discuss more details in our threat model) and thus those defenses are not applicable in general. Moreover, when extending to self-supervised learning, some of these defenses are ineffective even if we assume the defender has white-box access to the encoder as shown in previous studies (Jia, Liu, and Gong 2022) (we also confirm this in our experiments). Testing-phase defenses (Doan, Abbasnejad, and Ranasinghe 2020; Li et al. 2021b) aims to detect whether a test input is trojaned. When extended to self-supervised learning, those defenses are either ineffective or require a defender to have a clean validation dataset, which is less practical in the real world. In our work, we relax such an assumption by proposing a totally data-free defense. Moreover, our comparison results show that our defense can achieve comparable performance by giving those defenses an advantage, i.e., assuming the defender has a clean validation dataset.

Our work: In this work, we propose TrojanDec, the first framework to identify and restore a trojaned test image in the self-supervised learning context. Our framework consists of three main components: 1) metadata extraction, 2) trojan detection, and 3) image restoration. In the first component, we extract the key metadata from a test image, which is critical in detecting if the image consists of a trigger. In the second component, we perform a data-free statistical analysis on the metadata to identify if the corresponding image is trojaned or not. In the third component, if the image is detected as a trojaned example in the previous part, we further restore it using a diffusion model.

To show the effectiveness of our TrojanDec, we conduct extensive experiments on multiple pre-training datasets and downstream tasks under state-of-the-art trojan attacks (Jia, Liu, and Gong 2022; Saha et al. 2022; Liu, Jia, and Gong 2022; Zhang et al. 2024) to self-supervised learning. Our results some that our defense is consistently effective under those attacks. We further generalize several representative backdoor attacks (Chen et al. 2017; Salem et al. 2022) designed for supervised learning to the self-supervised learning context as the adaptive attacks and show the effectiveness of our method in defending against them. To study the impact of hyperameters, we perform a comprehensive ablation study in our evaluation. Finally, we compare our defense with state-of-the-art defenses (Gao et al. 2019; Doan, Abbasnejad, and Ranasinghe 2020; Ma et al. 2023) against trojan attacks. Our results demonstrate that TrojanDec outperforms all of them.

To summarize, we make the following contributions:

- We propose the first generic data-free trojaned test input detection and restoration framework for self-supervised learning
- We demonstrate that our framework does not rely on any clean data or any prior knowledge to the pre-training or downstream dataset.
- We conduct extensive experiments to evaluate our TrojanDec and show that it is effective in defending against various types of trojan attacks and outperforms existing defense methods.

Related Work

Trojan Attacks

Trojan attacks to self-supervised learning: Trojan attacks (Carlini and Terzis 2022; Saha et al. 2022; Jia, Liu, and Gong 2022; Li et al. 2023; Zhang et al. 2024; Wu et al. 2023; Jha, Hayase, and Oh 2023; Bober-Irizar et al. 2023; Li et al. 2024) to self-supervised learning aim to produce a trojan encoder such that it outputs normal feature vectors for clean test inputs, while generating feature vectors that are similar to the feature vector of an attacker-chosen reference object (called target object) for test inputs embedded with a trigger. As a result, a downstream classifier built upon such a

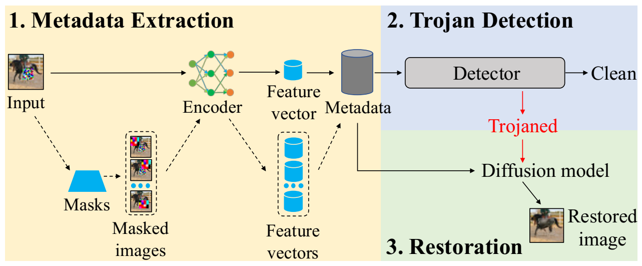


Figure 1: An overview of our TrojanDec.

trojan encoder for a downstream task inherits such trojan behaviors, i.e., the downstream classifier predicts the same class as the reference object for any test input embedded with the trigger. Depending on how an attacker injects a trojan into an encoder, those attacks can be divided into two types. The first type of attack is based on poisoning the pre-training dataset used to pre-train the encoder (Saha et al. 2022; Liu, Jia, and Gong 2022; Carlini and Terzis 2022; Zhang et al. 2024). For instance, Saha et al. (2022) proposed to create poisoned examples by placing a trigger near the target object. The other type of trojan attack is based on directly manipulating the parameters of a pre-trained encoder (Jia, Liu, and Gong 2022). For example, BadEncoder (Jia, Liu, and Gong 2022) leverages a surrogate dataset to fine-tune a pre-trained encoder to maximize the similarity between the features vectors of a trojan image and the attacker-chosen reference object. This kind of attack is usually more powerful than the previous attack as an attacker can arbitrarily manipulate the parameters. In our work, we consider both types of attacks.

Trojan attacks to supervised learning: We note that backdoor attacks have also been studied for supervised learning (Gu, Dolan-Gavitt, and Garg 2017; Chen et al. 2017; Liu et al. 2018, 2020; Li et al. 2021a; Zeng et al. 2021). In particular, those studies try to design different triggers to make the attak more effective and stealthy. For instance, the dynamic trojan attack (Salem et al. 2022) changes the pattern and the location of the trigger frequently to avoid being detected. The blending attack (Chen et al. 2017) mixes the trigger pattern into the original input by using the background noise as the trigger. Those trojan attacks are designed for classifiers instead of self-supervised learning encoders. In our experiments, we will generalize them to self-supervised learning and our results show that our defense is effective against those attacks.

Defending against Trojan Attacks

There are many existing works that focus on defending against trojan attacks (Tran, Li, and Madry 2018; Chen et al. 2018; Wang et al. 2019; Xu et al. 2021; Chen et al. 2019; Liu et al. 2019; Gao et al. 2019; Doan, Abbasnejad, and Ranasinghe 2020; Li et al. 2021b; Jia et al. 2022; Cheng et al. 2023; Huang et al. 2021). Most of them are designed for supervised learning. Depending on which phase those defenses are applied, they can be categorized into two genres: training-phase defense and testing-phase defense.

Training-phase defenses: Training-phase defenses (Chen et al. 2018; Wang et al. 2019; Kolouri et al. 2020; Huang et al. 2021; Zheng et al. 2022; Tejankar et al. 2023; Feng et al. 2023; Zheng et al. 2024; Wang et al. 2024) take two directions. In the first direction, they aim to train a robust model on a poisoned pre-training dataset. They are not applicable when an attacker implants a trojan into a model by directly manipulating its parameters (Jia, Liu, and Gong 2022). In the second direction, they aim to detect and remove the trojan for a given model. For instance, Neural Cleanse views each output class in a classifier as a potential target class and reverses engineer a trojan trigger for it (Wang et al. 2019). Then, it uses outlier detection statistics to determine if the classifier is trojan. The defenses designed in this direction are mostly tailored for supervised learning. Moreover, many of those defenses (Wang et al. 2019; Zheng et al. 2024) require accessing the parameters of the model, which make them not applicable when the defender does not have such access, e.g., the model is deployed as a cloud service and the defender can only query it. Additionally, existing studies (Jia, Liu, and Gong 2022) showed that some defenses (Wang et al. 2019; Xu et al. 2021) in this direction are ineffective when extended to self-supervised learning. In general, our testing-phase defense is complementary to training-phase defense.

Testing-phase defenses: Testing-phase defenses (Gao et al. 2019; Doan, Abbasnejad, and Ranasinghe 2020; Ma et al. 2023; Xi et al. 2024) aim to detect or recover a test input that potentially contains a trigger. Most of these defenses require the defender to have some clean data. In our work, we consider a more realistic scenario where the defender does not have any clean data. Our experimental results show that our defense can achieve comparable performance with those defenses by giving them the advantage (i.e., we assume the defender has some clean data for those defenses).

For space reasons, more discussion on related work can be found in our technical report (Liu, Wang, and Jia 2024).

Threat Model

Attacker's goal: To perform trojan attacks to self-supervised learning, an attacker selects one (or more) target classes in one (or more) downstream tasks. The attacker then chooses a trigger for each target class. In particular, the attacker aims at crafting a trojaned encoder to achieve two goals. Firstly, the encoder generates normal feature vectors for clean test inputs (i.e., the inputs without triggers). This means that the functionality of the trojaned encoder is maintained for clean inputs, which makes the attack more stealthy and harder to be noticed. Secondly, when the trigger presents in a test input, the trojaned encoder will produce a feature vector that lies in the target class associated with that trigger. As a result, a downstream classier built on the trojan encoder for a downstream task will predict the target class for any input with a backdoor trigger.

Attacker's background knowledge and capability: We consider both data-poisoning based attacks (Saha et al. 2022; Carlini and Terzis 2022) and model-poisoning based attacks (Jia, Liu, and Gong 2022). For data-poisoning based attacks, we consider an attacker can inject poisoned training inputs to the pre-training dataset used to pre-train an encoder.

Algorithm 1: Metadata Extraction

```
1: Input: $\mathbf{x}$ (test input), $f$ (encoder), $k$ (mask size), $s$ (step size), and $t$ (image size).
2: Output: $\mathcal{S}$ (the metadata of $\mathbf{x}$)
3: $\mathcal{M} \leftarrow MaskSetGeneration(k, s, t)$
4: $\mathcal{S} \leftarrow \emptyset$
5: for $(\mathbf{m}, \mathbf{p})$ in $\mathcal{M}$ do
6:     $\mathbf{x}_{\mathbf{m},\mathbf{p}} \leftarrow \mathbf{m} \cdot \mathbf{x} + (1 - \mathbf{m}) \cdot \mathbf{p}$
7:     $d \leftarrow sim(f(\mathbf{x}_{\mathbf{m},\mathbf{p}}), f(\mathbf{x}))$
8:     $\mathcal{S}$.append$(d)$
9: end for
10: return $S$
```

For model-poisoning based attacks, we consider an attacker can arbitrarily manipulate the parameters of an encoder. In general, model-poisoning based attacks are more effective than data-poisoning based attacks as they make a stronger assumption on the attacker.

Defender's goal: We consider the defender as the downstream user of the deployed self-supervised learning encoder. Given a test input, the defender's goal is to detect whether it contains a trigger. If this input is identified as trojaned, the defender wants to further remove the trigger from the input.

Defender's background knowledge and capability: We consider a very weak defender. As mentioned, the defender is a downstream user of the pre-trained encoder. In this scenario, the defender only has the ”black-box” access to the encoder, which means the following: 1) the defender can only query the encoder by images and receive the produced feature vectors and 2) the defender does not have any knowledge to the encoder, including the pre-training dataset information, the encoder architecture, the encoder parameters, etc.

Our TrojanDec

We present TrojanDec as an end-to-end framework for identifying and eliminating backdoors from test images. As depicted in Figure 1, our framework consists of 3 key components: the extraction of metadata from a test input, backdoor detection based on metadata, and test input restoration. We also assume the attacker uses a patch-based trojan trigger, since patch-based triggers are prevalent in real-world scenarios as they are practical and easy to implement physically. Next, we delve into details of these components.

Metadata Extraction

There are 3 steps to extract metadata from a test image. First, we generate a set of masks and use them to mask the test image to create a set of masked images. Then, we query the encoder f (trojaned or not) to obtain feature vectors of masked images and the original test image. Finally, we compute the similarity between the feature vectors of each masked image and the original test image to derive the metadata.

For simplicity, we use $(\mathbf{m}, \mathbf{p})$ to denote a patch-based mask, where $\mathbf{m}$ is a binary offset and $\mathbf{p}$ is the mask pattern. The binary offset $\mathbf{m}$ controls two characteristics of the mask: location and size. Specifically, we denote the upper-left coordinate of a mask as (a, b) and the mask size as k. Given the location (a, b) and mask size k, we define the binary offset $\mathbf{m}$ at the coordinate (i, j) as follows:

$$\mathbf{m}_{i,j} = \begin{cases} 0 & \text{if } i \in [a, a+k), j \in [b, b+k) \\ 1 & \text{otherwise} \end{cases} \quad (1)$$

The mask pattern $\mathbf{p}$ controls the pixel values of the mask. In particular, when the mask is applied to a test image, for a specific coordinate (i, j), $\mathbf{m}_{i,j} = 1$ means it keeps the original pixel values of the image at (i, j) and $\mathbf{m}_{i,j} = 0$ means it masks this coordinate out (i.e., reset the pixel value of the image at (i, j) to $\mathbf{p}_{i,j}$). Given a test image $\mathbf{x}$, we obtain the masked image $\mathbf{x}_{\mathbf{m},\mathbf{p}}$ by applying $(\mathbf{m}, \mathbf{p})$ to $\mathbf{x}$ as follows:

$$\mathbf{x}_{\mathbf{m},\mathbf{p}} = \mathbf{m} \cdot \mathbf{x} + (1 - \mathbf{m}) \cdot \mathbf{p} \quad (2)$$

We notice that pre-defining $\mathbf{p}$ to a fixed pattern can be dangerous, as the attacker can exploit this fact and set the trojan trigger to have the same pattern as $\mathbf{p}$. Thus, for each mask, we randomly generate a unique pattern such that for each coordinate (i, j), $\mathbf{p}_{i,j}$ is randomly sampled from 0 to 255. Formally, we have the following proposition.

Proposition 1. *If the adversary sets the trojan trigger to be* $\mathbf{e}$ *whose height and width are* e_h *and* e_w*, while defender has a mask* $(\mathbf{m}, \mathbf{p})$ *such that the pattern* $\mathbf{p}$ *is randomly generated, the probability of the existence of a part of the mask pattern such that* ℓ_1 *distance of this part of the mask pattern and the trojan trigger greater than* β *is no smaller than* $1 - \frac{(2\beta)^T}{T!}$*, where* $T = e_h \cdot e_w$.

Proof. See our technical report (Liu, Wang, and Jia 2024).

Following this rule, we can generate a set of masks, namely $\mathcal{M}$. Next, for each mask $(\mathbf{m}, \mathbf{p})$ in $\mathcal{M}$, we apply it to the given input to obtain a masked image $\mathbf{x}_{\mathbf{m},\mathbf{p}}$. We then use each of them to query f. For a masked image $\mathbf{x}_{\mathbf{m},\mathbf{p}}$, we will receive its corresponding feature vector $f(\mathbf{x}_{\mathbf{m},\mathbf{p}})$. We also query f using the original test image to receive $f(\mathbf{x})$. Finally, we calculate the cosine similarity between the features of $\mathbf{x}_{\mathbf{m},\mathbf{p}}$ and $\mathbf{x}$, i.e., $sim(f(\mathbf{x}_{\mathbf{m},\mathbf{p}}), f(\mathbf{x}))$. After this process, we will obtain the metadata $\mathcal{S}$ of $\mathbf{x}$, where $\mathcal{S}$ consists of a set of similarities between masked images and $\mathbf{x}$.

Algorithm 2 summarizes how TrojanDec extracts the metadata from a test image $\mathbf{x}$. Algorithm 2 in (Liu, Wang, and Jia 2024) presents details of how TrojanDec generates the mask set $\mathcal{M}$. The helper function *CreateMask*(a, b, k) creates a mask $(\mathbf{m}, \mathbf{p})$ with the upper-left coordinate being (a, b), mask size being k, offset values being defined in Equation 1, and mask pattern $\mathbf{p}$ being randomly generated. An example of such mask with randomly generated pattern is in Figure 3c.

Trojan Detection

In this section, we present our technique for detecting whether a test image is trojaned or not, based on its metadata. The key intuition of our TrojanDec is that the set of similarities in $\mathcal{S}$ can be divided into two clusters for a trojaned test image, whereas there is only a single cluster for a clean test image. Figure 2 shows the distribution of $\mathcal{S}$ of a test image sampled from STL10 dataset. When the trigger is partially covered by the mask, the trojan information of this test image is broken, leading the feature vectors of these masked images being dissimilar to the original trojaned image. In Figure 2, when at least one fourth of the trojan trigger is occluded by the mask, the similarities between the feature vectors generated by f for masked images and the original image are much lower (i.e., most of them are less than 0.7), whereas the majority of the similarities in the other cluster are greater than 0.9. Therefore, by the number of clusters $\mathcal{S}$ can be classified to, we can tell if the test image is trojaned.

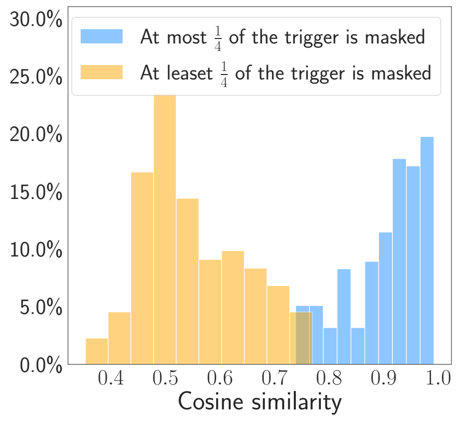


Figure 2: Cosine similarities of the feature vectors of masked images and the feature vector of the original trojaned test image. The trojaned encoder is pre-trained on CIFAR10 and the downstream dataset is STL10.

To determine the number of clusters the similarity scores in $\mathcal{S}$ belongs to, we leverage gap-statistic (Tibshirani, Walther, and Hastie 2001) to reach the goal. Suppose $\mathcal{S}$ can be divided into K clusters, where $K = 1$ or 2. We first use *K-means clustering* method (Hartigan and Wong 1979) to divide $\mathcal{S}$ into K clusters. Then, we derive the within-cluster dispersion W_K, which measures the variability of the data points within each cluster. We can use W_K to calculate the gap statistics. To reach the goal, we first generate B synthetic datasets based on a uniform distribution, then use K-means clustering to divide each of them into K groups, and calculate the within-cluster dispersion W^*_{Kb} for each of them, where $b = 1, 2, \cdots, B$. The gap statistic for K is defined as follows:

$$G(K) = \frac{1}{B} \sum_{b=1}^{B} [\log(W^*_{Kb}) - \log(W_K)] \quad (3)$$

Then, we calculate the standard deviation of $\log W^*_{Kb}$, which is denoted as s_K. In particular, based on the definition of the standard deviation, we have $s_K = [\frac{1}{B} \sum_{b=1}^{B} (\log(W^*_{Kb}) - \bar{l})^2]^{\frac{1}{2}}$, where $\bar{l} = \frac{1}{B} \sum_{b=1}^{B} \log(W^*_{Kb})$. The standard deviation is further normalized by the number of synthetic datasets B. Specifically, we have $s'_K = s_K \sqrt{1 + \frac{1}{B}}$. Finally, we derive the optimal number of clusters of $\mathcal{S}$ via solving the following optimization problem:

$$K^* = \underset{K}{\text{argmin}}\, K \text{ s.t. } G(K) \geq G(K+1) - s'_{K+1} \quad (4)$$

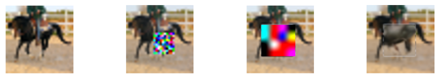


Figure 3: Example of an image from the STL10 dataset recovered from its masked prototype: (a) original, (b) trojaned, (c) masked, and (d) restored.

Given a test image $\mathbf{x}$ and its metadata $\mathcal{S}$, we derive the optimal number of clusters K^*. If $K^* = 1$, it means $\mathbf{x}$ is clean. Otherwise, we predict $\mathbf{x}$ as a trojaned image.

Image Restoration

If a test input is identified as trojaned, we aim to remove the trojan trigger to restore the image. Given a test image $\mathbf{x}$ that is detected as a trojaned input, we use its masked version $\mathbf{x}_{\mathbf{m}^i,\mathbf{p}^i}$ as the prototype for restoration, where $i = \text{argmin}_i\ sim(\mathbf{x}_{\mathbf{m}^i,\mathbf{p}^i}, \mathbf{x})$ and sim is cosine similarity. Our intuition is that since $\mathbf{x}$ is detected as a trojaned input, $\mathbf{x}_{\mathbf{m}^i,\mathbf{p}^i}$ is the masked image that most likely to have the entire trojan trigger covered by the mask. A naive way is to directly use the prototype $\mathbf{x}_{\mathbf{m}^i,\mathbf{p}^i}$ as the "restored" image. However, as existing works reveal, this will substantially demolish the image quality and may change the semantic meaning of the original image (Doan, Abbasnejad, and Ranasinghe 2020). Thus, we use DDNM (denoising diffusion null-space model) (Wang, Yu, and Zhang 2023) to restore the masked image. Given an image $\mathbf{x}_0$, DDNM adds Gaussian noise in a step-by-step manner. In particular, the noisy image in the t^{th} step is $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \epsilon\sqrt{1-\bar{\alpha}_t}$, where $\bar{\alpha}_t$ is a pre-defined scalar constant and $\epsilon \sim \mathcal{N}(0, \mathbf{I})$ represents the zero-mean Gaussian noise. The goal of DDNM is to train a denoise model to estimate the noise added to $\mathbf{x}_0$ based on $\mathbf{x}_t$. As a result, the denoise model can be used to recover noisy images.

DDNM can be utilized to recover the masked image, since it can be used as a zero-shot solver for linear image restoration problems by refining only the masked area in the reverse diffusion process. We leverage the denoise model to recover the masked image. Based on the definition, we have $\mathbf{x}_{\mathbf{m}^i,\mathbf{p}^i} = \mathbf{m}^i\mathbf{x} + (1 - \mathbf{m}^i)\mathbf{p}^i$. Since DDNM leverages a binary mask by design and the pattern $\mathbf{p}^i$ is less relevant to the restoration, we resort to restore $\mathbf{m}^i\mathbf{x}$ instead of $\mathbf{x}_{\mathbf{m}^i,\mathbf{p}^i}$. More details of how DDNM restores $\mathbf{x}$ from its masked version are in our technical report (Liu, Wang, and Jia 2024). Examples of the image restoration are shown in Figure 3. After being processed by the diffusion model, the image restored from trojan can be fed into its further use as other clean images, e.g., testing the downstream classifier.

Evaluation

Experiment Settings

Dataset and models: We consider CIFAR10 (Krizhevsky 2009) and STL10 (Coates, Ng, and Lee 2011) as the datasets to pre-train self-supervised learning encoder. When CIFAR10 is used as the pre-training dataset, we use STL10, SVHN (Netzer et al. 2011), and EuroSAT (Helber et al. 2019) as the downstream datasets. When STL10 is applied to pre-trained the encoder, we use CIFAR10, SVHN, and EuroSAT as the downstream dataset. We resize all images to be 32×32 to be consistent. The details of these datasets can be found in Table 5 in our technical report (Liu, Wang, and Jia 2024). Unless otherwise mentioned, we respectively use CIFAR10 and STL10 as the default pre-training and downstream dataset. For details about pre-training encoders or training downstream classifiers, please refer to (Liu, Wang, and Jia 2024).

Pre-training dataset	Down-stream dataset	Clean f, w/o TrojanDec		Clean f, w TrojanDec		Trojaned f, w/o TrojanDec		Trojaned f, w TrojanDec	
		ACC	ASR	ACC	ASR	ACC	ASR	ACC	ASR
CIFAR10	STL10	74.38	1.01	74.00	2.25	73.75	100.0	73.15	1.43
	SVHN	54.13	23.44	54.08	23.18	60.29	99.98	60.23	18.86
	EuroSAT	74.41	3.52	74.37	1.67	75.59	100.0	75.44	1.13
STL10	CIFAR10	82.38	3.01	82.08	0.49	81.79	100.0	81.71	3.54
	SVHN	46.37	23.95	46.11	27.54	50.10	100.0	49.95	25.66
	EuroSAT	77.78	3.07	77.85	1.07	77.56	99.67	77.41	0.01

Table 1: Performance of our TrojanDec in removing the trojan from test inputs. All values are percentages.

Attack settings: We consider attacks that poison the pre-training dataset (Saha et al. 2022; Liu, Jia, and Gong 2022) or manipulate the parameters of an encoder (Jia, Liu, and Gong 2022). As the attack that manipulates the encoder parameters is usually more powerful, we consider (Jia, Liu, and Gong 2022) by default. We adopt the publicly available implementation of (Saha et al. 2022; Jia, Liu, and Gong 2022) and implement (Liu, Jia, and Gong 2022) by ourselves. We use the default parameter settings in those works. In particular, we set the default trigger size to 10×10. Moreover, we use randomly generated trigger patterns.

Defense settings: Our TrojanDec has two parameters: k (mask size) and s (step size). By default, we set k and s to 15 and 1, respectively. We will study the impact of each of them in the ablation study. For image restoration, we use a pre-trained diffusion model from (Wang 2022).

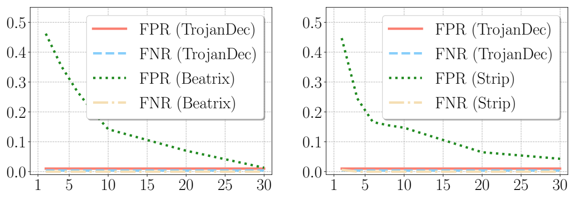
Evaluation metrics: To evaluate the effectiveness of our approach in detecting trojaned test images, we use the *False Positive Rate* (FPR) and *False Negative Rate* (FNR) as metrics. FPR (or FNR) measures the fraction of clean (or trojan) test images that are detected as trojan (or clean). Additionally, to measure the effectiveness of TrojanDec in recovering trojaned images, we apply our method to both the training and testing data of the downstream classifier and use the *Accuracy* (ACC) and *Attack Success Rate* (ASR) to measure the classifier's accuracy and attack success. Specifically, ACC measures the prediction accuracy of the downstream classifier for clean test images, while ASR quantifies the fraction of trojaned images predicted as the target class.

Defending against State-of-the-art Attacks

Parameter manipulation attacks: Table 2 and 1 present the performance of our TrojanDec in trojan detection and removal against trojan attack that manipulate the parameters of encoders (Jia, Liu, and Gong 2022). We have three observations from the results. First, our TrojanDec effectively

Pre-training dataset	Downstream dataset	Clean encoders		Trojaned encoders	
		FPR (%)	FNR (%)	FPR (%)	FNR (%)
CIFAR10	STL10	0.88	-	1.02	0.40
	SVHN	0.15	-	0.11	0.00
	EuroSAT	0.04	-	0.29	1.12
STL10	CIFAR10	0.80	-	0.41	0.00
	SVHN	1.11	-	0.49	0.01
	EuroSAT	0.07	-	0.26	0.00

Table 2: Performance of TrojanDec in detecting trojaned inputs. FNR is N/A for clean f as there is no trojaned example when the encoder is clean.



(a) Beatrix (b) Strip

Figure 4: Comparison to Beatrix and Strip on STL10. X-axis is the number of clean data. Y-axis is the FPR/FNR.

detects trojaned test inputs, as evidenced by the low FNR in Table 2. Second, TrojanDec can successfully remove trojans from trojaned inputs. In Table 1, the ASR values significantly reduce after applying our method to trojaned encoders, compared to trojaned encoders without TrojanDec. For instance, for the trojaned encoder pre-trained on CIFAR10 and the downstream dataset being STL10, the ASR without TrojanDec is 100%, whereas the ASR with TrojanDec is decreased to 1.43%. It is important to note that while the ASR with TrojanDec are not zero, they are comparable to the ASR of clean encoders. This is because downstream classifiers can misclassify some test inputs, leading to slightly higher ASR. Our third observation is that TrojanDec successfully maintains the utility of the encoders. Regardless of whether the encoder is clean or trojaned, the FPR values remain small and the ACC values are comparable to those without TrojanDec. Overall, our TrojanDec shows promising performance in detecting and removing trojans from test images (if any), while maintaining the utility of the encoders.

Pre-training dataset poisoning attacks: We also evaluate our method on trojan attacks that poison the pre-training dataset (Saha et al. 2022; Liu, Jia, and Gong 2022). For each attack, we craft a trojaned encoder pre-trained on CIFAR10 and set the target downstream task to be STL10. The results in Table 11a and 11b in (Liu, Wang, and Jia 2024) show our TrojanDec can defend against these trojan attacks effectively.

Comparing TrojanDec with Existing Defenses

We compare our TrojanDec with several state-of-the-art testing-phase defenses (Gao et al. 2019; Doan, Abbasnejad, and Ranasinghe 2020; Ma et al. 2023). In addition, we show existing state-of-the-art training-phase defenses (Wang et al. 2019; Xu et al. 2021; Zeng et al. 2022) are not effective. We leverage their publicly available code in our experiments.

Beatrix (Ma et al. 2023): Beatrix uses Gramian information to distinguish trojaned inputs from clean ones, which requires enough clean validation images. However, Beatrix is not effective when the defender does not have enough clean validation images. We vary the number of clean validation images to compare with Beatrix and plot the results on STL10 in Figure 4a. As TrojanDec does not rely on clean images, FPR/FNR remains unchanged regardless of the number of clean validation images. We have two observations from the results. First, our TrojanDec achieves comparable performance with Beatrix when the defender has enough clean validation images (e.g., more than 30 clean validation images). Second, our TrojanDec significantly outperforms Beatrix when the clean validation images are insufficient. We have similar observations on other downstream datasets, as shown in Figures 5a and 6a in (Liu, Wang, and Jia 2024).

Strip (Gao et al. 2019): Strip also leverages clean validation images to detect whether a test input is trojaned or not. Figure 4b compares Strip's performance on STL10 with that of TrojanDec. Similar to Beatrix, when the defender has limited clean validation images, Strip's FPR is high. As clean validation images number increases, Strip's performance improves. We find that our TrojanDec still outperforms Strip even if we assume the defender has 30 clean validation images. Similar trends were observed on other downstream datasets, as presented in Figure 5b and 6b in (Liu, Wang, and Jia 2024).

Februus: Februus (Doan, Abbasnejad, and Ranasinghe 2020) uses GradCAM (Selvaraju et al. 2017) to determine the potential trigger area. It then applies a Wasserstein GAN (Gulrajani et al. 2017) to in-paint the detected area. Table 6 in (Liu, Wang, and Jia 2024) shows the comparison results. Our TrojanDec achieves slightly better ACC and lower ASR for the trojaned inputs. Besides, we note that Februus has two limitations. Firstly, it cannot detect if a test input contains the trojan trigger, but can only directly restore every test input. This makes their defense method limited in scenarios where the trojan behaviours are expected to be reported. Secondly, its restoration technique requires the defender's knowledge of the distribution of the real unmask testing data, which is not available in the data-free setting we consider.

Other defenses: We also evaluate the state-of-the-art training-phase defense methods (Wang et al. 2019; Xu et al. 2021; Zeng et al. 2022) against trojan attacks. Table 7 in (Liu, Wang, and Jia 2024) shows these methods fail to defend against trojan attacks to encoders. Our experimental results confirm the observations in (Carlini and Terzis 2022; Jia, Liu, and Gong 2022; Feng et al. 2023).

Real-world Encoders

To further evaluate the effectiveness of TrojanDec, we apply it on 2 real-world self-supervised learning encoders: 1) the encoder pre-trained on ImageNet released by Google (Google 2020) and 2) CLIP encoder pre-trained on 400 million image-text pairs released by OpenAI (OpenAI 2021). Both encoders take 224×224 images as inputs. To fit its input size, we set k to 150 and s to 10. We directly apply our method on trojaned CLIP encoders released in (Jia 2021). The trojan detection and removal results presented in Table 3 show that TrojanDec

Pre-training dataset	Downstream dataset	FPR (%)	FNR (%)
ImageNet	STL10	1.16	0.00
	SVHN	0.10	0.20
	GTSRB	3.20	0.86
CLIP	STL10	1.65	1.16
	SVHN	0.94	0.02
	GTSRB	1.34	1.04

(a) Trojan detection

Pre-training dataset	Downstream dataset	Attacked encoders without TrojanDec		Attacked encoders, wtih TrojanDec	
		ACC (%)	ASR (%)	ACC (%)	ASR (%)
ImageNet	STL10	95.84	99.99	95.40	4.28
	SVHN	68.06	100.0	68.04	8.12
	GTSRB	47.41	99.83	45.84	9.16
CLIP	STL10	96.26	99.96	95.36	1.94
	SVHN	61.54	100.0	61.16	0.06
	GTSRB	59.58	97.02	59.30	3.82

(b) Trojan removal

Table 3: Performance of TrojanDec in defending against trojan attacks to 2 real-world encoders.

is effective in defending against trojan attacks to large, real-world pre-trained self-supervised learning encoders. More experimental results on larger-scale datasets are in Table 8 in (Liu, Wang, and Jia 2024).

Adaptive Attacks

In this section, we consider several adaptive attacks the adversary can take to evade our defense.

Trigger with large size: The first countermeasure we consider is that the attacker uses a large trigger such that the trigger size is bigger than the mask size set by TrojanDec. Table 9 in (Liu, Wang, and Jia 2024) presents the defense results where the trigger size is 16×16. The FNR values have slightly increased compared to Table 2 where the trigger size is 10×10. Also, the ASR after applying TrojanDec is decreased to small values. This is because our method does not necessarily require the mask to fully cover the trigger. As shown in Figure 2, only one fourth of the trigger being covered by the mask is enough to divide the metadata into two clusters. Our results demonstrate that our TrojanDec can effective defend against trojan attacks with large trigger sizes.

Trigger with dynamic location: We also consider an attacker that performs a dynamic trojan attack to the encoder. Since there is no existing work studying dynamic trojan attacks to encoders, we generalize the idea from (Salem et al. 2022) to perform the dynamic trojan attack. In particular, we inject the trojan into the encoder in the same way as (Jia, Liu, and Gong 2022) and in the testing phase, we embed the trigger into the test input at a random location. We evaluate the effectiveness of our TrojanDec against these two adpative attacks on 3 encoders pre-trained on CIFAR10 and use STL10 as the downstream task. Table 4 presents the results, which shows that our TrojanDec can effectively defend against trojan attacks with dynamic trigger location.

Trojan attacks in supervised learning (Chen et al. 2017; Yao et al. 2019; Ning et al. 2021): We notice that in supervised learning, there exist other trojan attacks, e.g., blending

Downstream dataset	FPR (%)	FNR (%)
STL10	0.70	0.65
SVHN	0.57	1.20
EuroSAT	2.35	3.12

(a) Trojan detection

Downstream dataset	Trojaned encoders without TrojanDec		Trojaned encoders, wtih TrojanDec	
	ACC (%)	ASR (%)	ACC (%)	ASR (%)
STL10	73.75	99.98	73.45	1.64
SVHN	60.29	98.41	59.76	23.75
EuroSAT	75.59	99.07	73.85	20.96

(b) Trojan removal

Table 4: Performance of TrojanDec against adapted dynamic trojan attacks.

attack (Chen et al. 2017), latent attack (Yao et al. 2019), and clean-label attack (Ning et al. 2021). For blending attack, we generalize it into self-supervised learning by using the default trigger presented in (Chen et al. 2017). However, as Table 10 in (Liu, Wang, and Jia 2024) shows, the attack is not successful, as the ACC after trojan injection decreases too much, and the ASR is not high for STL10 and EuroSAT. This indicates that it is not trivial to generalize the existing blending trojan attacks to the self-supervised learning context. For latent and clean-label trojan attacks, since they require label information to perform the attack, they are not applicable in attacking encoders pre-trained on unlabeled data.

Ablation Study

Our TrojanDec has two parameters: k and s, which control the mask size and the step size used to generate the mask set. Figure 7 and 8 in (Liu, Wang, and Jia 2024) show the impact of k and s respectively. When k is small, e.g., < 17, both trojan detection and removal achieve consistent good performance. When k is large, FNR and ASR increase. The reason is that when k is large, every mask covers a significant portion of the trigger, resulting in none of masked variants containing a complete trigger. Thus, all masked variants are not similar to the original trojaned input, making the detection component less effective. We have a similar observation for s. Our results show that we can set a smaller k and s in practice.

Conclusion and Future Work

In this work, we propose TrojanDec, a data-free framework to defend against trojan attacks to self-supervised learning. Specifically, the defender first extracts metadata from a given test input and identifies whether it has malicious trigger using statistical analysis. Then, the defender can restore the trojaned test input via a pre-trained diffusion model. Also, our extensive evaluation results show that TrojanDec is effective against multiple types of trojan attacks, and outperforms the existing trojan defense mechanisms, especially in the setting where the defender only has access to a limited number of clean data. Moreover, we demonstrate that several countermeasures fail to break TrojanDec. Interesting future works would be to extend our framework to defend against trojan attacks with non-patch-based triggers or other domains.

Acknowledgments

We thank the anonymous reviewers for insightful reviews.

References

Bober-Irizar, M.; Shumailov, I.; Zhao, Y.; Mullins, R.; and Papernot, N. 2023. Architectural Backdoors in Neural Networks. In *CVPR*.

Carlini, N.; and Terzis, A. 2022. Poisoning and Backdooring Contrastive Learning. In *ICLR*.

Chen, B.; Carvalho, W.; Baracaldo, N.; Ludwig, H.; Edwards, B.; Lee, T.; Molloy, I.; and Srivastava, B. 2018. Detecting backdoor attacks on deep neural networks by activation clustering. *arXiv preprint arXiv:1811.03728*.

Chen, H.; Fu, C.; Zhao, J.; and Koushanfar, F. 2019. DeepInspect: A Black-box Trojan Detection and Mitigation Framework for Deep Neural Networks. In *IJCAI*.

Chen, T.; Kornblith, S.; Norouzi, M.; and Hinton, G. 2020. A simple framework for contrastive learning of visual representations. In *ICML*.

Chen, X.; Liu, C.; Li, B.; Lu, K.; and Song, D. 2017. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526*.

Cheng, S.; Tao, G.; Liu, Y.; An, S.; Xu, X.; Feng, S.; Shen, G.; Zhang, K.; Xu, Q.; Ma, S.; and Zhang, X. 2023. BEAGLE: Forensics of Deep Learning Backdoor Attack for Better Defense. In *NDSS*.

Coates, A.; Ng, A.; and Lee, H. 2011. An analysis of single-layer networks in unsupervised feature learning. In *AISTATS*.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*.

Doan, B. G.; Abbasnejad, E.; and Ranasinghe, D. C. 2020. Februus: Input purification defense against trojan attacks on deep neural network systems. In *ACSAC*.

Feng, S.; et al. 2023. Detecting Backdoors in Pre-Trained Encoders. In *CVPR*.

Gao, Y.; Xu, C.; Wang, D.; Chen, S.; Ranasinghe, D. C.; and Nepal, S. 2019. STRIP: A Defence Against Trojan Attacks on Deep Neural Networks. In *ACSAC*.

Google. 2020. SimCLR. https://github.com/google-research/simclr. Accessed: 2023-01-02.

Grill, J.-B.; Strub, F.; Altché, F.; Tallec, C.; Richemond, P. H.; Buchatskaya, E.; Doersch, C.; Pires, B. A.; Guo, Z. D.; Azar, M. G.; et al. 2020. Bootstrap your own latent: A new approach to self-supervised learning. In *NeurIPS*.

Gu, T.; Dolan-Gavitt, B.; and Garg, S. 2017. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *IEEE Access*.

Gulrajani, I.; Ahmed, F.; Arjovsky, M.; Dumoulin, V.; and Courville, A. 2017. Improved training of wasserstein gans. In *NeurIPS (NeurIPS)*.

Hadsell, R.; Chopra, S.; and LeCun, Y. 2006. Dimensionality reduction by learning an invariant mapping. In *CVPR*.

Hartigan, J. A.; and Wong, M. A. 1979. Algorithm AS 136: A k-Means Clustering Algorithm. In *Journal of the Royal Statistical Society: Series C. 28 (1): 100–108*.

He, K.; Fan, H.; Wu, Y.; Xie, S.; and Girshick, R. 2020. Momentum contrast for unsupervised visual representation learning. In *CVPR*.

Helber, P.; Bischke, B.; Dengel, A.; and Borth, D. 2019. Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*.

Hjelm, R. D.; Fedorov, A.; Lavoie-Marchildon, S.; Grewal, K.; Bachman, P.; Trischler, A.; and Bengio, Y. 2019. Learning deep representations by mutual information estimation and maximization. In *ICLR*.

Huang, K.; Li, Y.; Wu, B.; Qin, Z.; and Ren, K. 2021. Backdoor Defense via Decoupling the Training Process. In *ICLR*.

Jha, R. D.; Hayase, J.; and Oh, S. 2023. Label Poisoning is All You Need. In *NeurIPS*.

Jia, J. 2021. BadEncoder. https://github.com/jinyuan-jia/BadEncoder. Accessed: 2022-10-01.

Jia, J.; Liu, Y.; Cao, X.; and Gong, N. Z. 2022. Certified Robustness of Nearest Neighbors against Data Poisoning and Backdoor Attacks. In *AAAI*.

Jia, J.; Liu, Y.; and Gong, N. Z. 2022. Badencoder: Backdoor attacks to pre-trained encoders in self-supervised learning. In *IEEE S&P*.

Kolouri, S.; Saha, A.; Pirsiavash, H.; and Hoffmann, H. 2020. Universal Litmus Patterns: Revealing Backdoor Attacks in CNNs. In *CVPR*.

Krizhevsky, A. 2009. Learning multiple layers of features from tiny images. *Tech Report*.

Li, C.; Pang, R.; Cao, Z., Bochuan Xi; Chen, J.; Ji, S.; and Wang, T. 2024. On the Difficulty of Defending Contrastive Learning against Backdoor Attacks. In *USENIX Security Symposium*.

Li, C.; Pang, R.; Xi, Z.; Du, T.; Ji, S.; Yao, Y.; and Wang, T. 2023. An Embarrassingly Simple Backdoor Attack on Self-supervised Learning. In *ICCV*.

Li, Y.; Li, Y.; Wu, B.; Li, L.; He, R.; and Lyu, S. 2021a. Invisible backdoor attack with sample-specific triggers. In *CVPR*.

Li, Y.; Lyu, X.; Koren, N.; Lyu, L.; Li, B.; and Ma, X. 2021b. Neural attention distillation: Erasing backdoor triggers from deep neural networks. *arXiv preprint arXiv:2101.05930*.

Liu, H.; Jia, J.; and Gong, N. Z. 2022. PoisonedEncoder: Poisoning the Unlabeled Pre-training Data in Contrastive Learning. In *USENIX Security Symposium*.

Liu, Y.; Jia, J.; Liu, H.; and Gong, N. Z. 2022. StolenEncoder: Stealing Pre-trained Encoders in Self-supervised Learning. In *CCS*.

Liu, Y.; Lee, W.-C.; Tao, G.; Ma, S.; Aafer, Y.; and Zhang, X. 2019. ABS: Scanning neural networks for back-doors by artificial brain stimulation. In *CCS*.

Liu, Y.; Ma, S.; Aafer, Y.; Lee, W.-C.; Zhai, J.; Wang, W.; and Zhang, X. 2018. Trojaning Attack on Neural Networks. In *NDSS*.

Liu, Y.; Ma, X.; Bailey, J.; and Lu, F. 2020. Reflection backdoor: A natural backdoor attack on deep neural networks. In *ECCV*.

Liu, Y.; Wang, Y.; and Jia, J. 2024. TrojanDec: Data-free Detection of Trojan Testing Inputs in Self-supervised Learning. *arXiv*.

Ma, W.; Wang, D.; Sun, R.; Xue, M.; Wen, S.; and Xiang, Y. 2023. The "Beatrix" Resurrections: Robust Backdoor Detection via Gram Matrices. In *NDSS*.

Netzer, Y.; Wang, T.; Coates, A.; Bissacco, A.; Wu, B.; and Ng, A. Y. 2011. Reading Digits in Natural Images with Unsupervised Feature Learning. In *NeurIPS Workshop on Deep Learning and Unsupervised Feature Learning*.

Ning, R.; Li, J.; Xin, C.; and Wu, H. 2021. Invisible Poison: A Blackbox Clean Label Backdoor Attack to Deep Neural Networks. In *INFOCOM*.

OpenAI. 2021. CLIP. https://github.com/openai/CLIP. Accessed: 2022-09-11.

Radford, A.; Kim, J. W.; Hallacy, C.; Ramesh, A.; Goh, G.; Agarwal, S.; Sastry, G.; Askell, A.; Mishkin, P.; Clark, J.; et al. 2021. Learning transferable visual models from natural language supervision. In *ICML*.

Saha, A.; Tejankar, A.; Koohpayegani, S. A.; and Pirsiavash, H. 2022. Backdoor Attacks on Self-Supervised Learning. In *CVPR*.

Salem, A.; Wen, R.; Backes, M.; Ma, S.; and Zhang, Y. 2022. Dynamic backdoor attacks against machine learning models. *Euro S & P*.

Selvaraju, R. R.; Cogswell, M.; Das, A.; Vedantam, R.; Parikh, D.; and Batra, D. 2017. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *ICCV*.

Tejankar, A.; et al. 2023. Defending Against Patch-based Backdoor Attacks on Self-Supervised Learning. In *CVPR*.

Tibshirani, R.; Walther, G.; and Hastie, T. 2001. Estimating the number of clusters in a data set via the gap statistic. In *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*.

Tran, B.; Li, J.; and Madry, A. 2018. Spectral signatures in backdoor attacks. In *NeurIPS*.

Wang, B.; Yao, Y.; Shan, S.; Li, H.; Viswanath, B.; Zheng, H.; and Zhao, B. Y. 2019. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *IEEE S& P*.

Wang, H.; Xiang, Z.; Miller, D. J.; and Kesidis, G. 2024. MM-BD: Post-Training Detection of Backdoor Attacks with Arbitrary Backdoor Pattern Types Using a Maximum Margin Statistic. In *IEEE Symposium on Security and Privacy*.

Wang, Y. 2022. DDNM. https://github.com/wyhuai/DDNM. Accessed: 2023-02-12.

Wang, Y.; Yu, J.; and Zhang, J. 2023. Zero Shot Image Restoration Using Denoising Diffusion Null-Space Model. In *ICLR*.

Wu, Y.; Han, X.; Qiu, H.; and Zhang, T. 2023. Computation and Data Efficient Backdoor Attacks. In *ICCV*.

Xi, Z.; Du, T.; Li, C.; Pang, R.; Ji, S.; Chen, J.; Ma, F.; and Wang, T. 2024. Defending Pre-trained Language Models as Few-shot Learners against Backdoor Attacks. In *NeurIPS*.

Xu, X.; Wang, Q.; Li, H.; Borisov, N.; Gunter, C. A.; and Li, B. 2021. Detecting ai trojans using meta neural analysis. In *IEEE S & P*.

Yao, Y.; Li, H.; Zheng, H.; and Zhao, B. Y. 2019. Latent backdoor attacks on deep neural networks. In *CCS*.

Zeng, Y.; Chen, S.; Park, W.; Mao, Z. M.; Jin, M.; and Jia, R. 2022. Adversarial Unlearning of Backdoors via Implicit Hypergradient. In *ICLR*.

Zeng, Y.; Park, W.; Mao, Z. M.; and Jia, R. 2021. Rethinking the backdoor attacks' triggers: A frequency perspective. In *CVPR*.

Zhang, J.; Liu, H.; Jia, J.; and Gong, N. Z. 2024. CorruptEncoder: Data Poisoning based Backdoor Attacks to Contrastive Learning. In *CVPR*.

Zheng, M.; Xue, J.; Wang, Z.; Chen, X.; Lou, Q.; Jiang, L.; and Wang, X. 2024. SSL-Cleanse: Trojan Detection and Mitigation in Self-Supervised Learning. In *ECCV*.

Zheng, R.; Tang, R.; Li, J.; and Liu, L. 2022. Data-free Backdoor Removal based on Channel Lipschitzness. In *ECCV*.