Effective Backdoor Defense by Exploiting Sensitivity of Poisoned Samples

Anonymous Author(s)
Affiliation
Address
email

Abstract

Poisoning-based backdoor attacks are serious threat for training deep models on data from untrustworthy sources. Given a backdoored model, we observe that the feature representations of poisoned samples with trigger are more sensitive to transformations than those of clean samples. It inspires us to design a simple sensitivity metric, called *feature consistency towards transformations (FCT)*, to distinguish poisoned samples from clean samples in the untrustworthy training set. Moreover, we propose two effective backdoor defense methods. Built upon a sample-distinguishment module utilizing the FCT metric, the first method trains a secure model from scratch using a two-stage secure training module. And the second method removes backdoor from a backdoored model with a backdoor removal module which alternatively unlearns the distinguished poisoned samples and relearns the distinguished clean samples. Extensive results on three benchmark datasets demonstrate the superior defense performance against eight types of backdoor attacks, to state-of-the-art backdoor defenses.

1 Introduction

Training deep neural networks (DNNs) often requires a large amount of training data, which is sometimes obtained from a third-party untrustworthy source. However, the untrustworthy data may bring serious security threats. One of the typical threats is the poisoning-based backdoor attack [1], which could inject undesired backdoor—the correlation between trigger(s) and target class(es)—into the model through maliciously poisoning a few training samples. Specifically, as shown in the top left of Fig. 1, each poisoned sample is attached with a trigger (see a small grid patch) at the bottom right corner, and relabelled as a target class. Consequently, the trained backdoored model will predict clean samples very well, but is likely to predict any sample with the trigger to be the target class.

It has been observed in [2] that poisoned samples with triggers are likely to gather together in the feature space of a backdoored model, as shown in the top right of Fig. 1. Note that these poisoned samples contain diverse objects (may be from different source classes), but the information from these objects seems to be ignored by the backdoor model. In other words, the feature representations of poisoned samples are dominated by the triggers, rather than the objects. We conjecture that such a domination is mainly due to the overfitting to the triggers by the backdoor model, since triggers across different poisoned samples are much less diverse than objects. To verify this conjecture, we propose to slightly perturb both poisoned and clean samples, such as rotation transformation. As shown in the bottom right of Fig. 1, there is no longer gathering of poisoned samples in the feature space, and they are located close to samples of their source classes,

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

i.e., the dominance of triggers over other objects disappears, which verifies the triggers overfitting. Besides, although the feature representations of clean samples are also affected by transformations, their changes are much smaller than those of poisoned samples. In other words, poisoned samples are more sensitive to transformations than clean samples. It inspires that poisoned samples could be distinguished from clean samples according to the sensitivity to transformations, which is measured by a simple sensitivity metric, called *feature consistency towards transformations* (FCT). In our experiments, the precision of the distinguished clean and poisoned samples is nearly 100% in most cases, respectively.

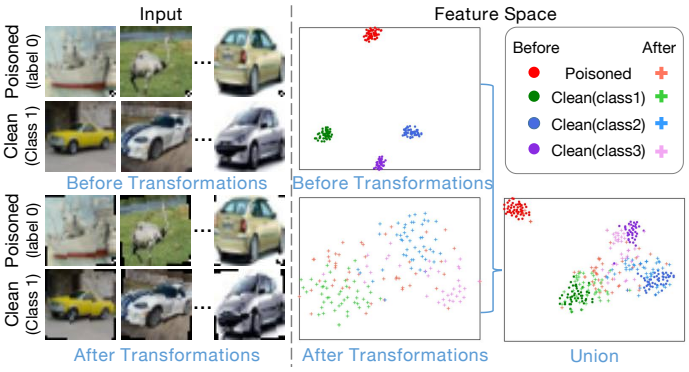


Figure 1: Poisoned and clean samples from CIFAR-10 [3], and the t-SNE [4] visualization of their feature representations with a backdoored model. As shown in the *Union* figure, the changes of poisoned samples (from $\cdot$ to $+$) are much larger than those of clean samples (from $\cdot$ to $+$ in other colors). Note that we present only three classes for clear illustration.

In this work, we aim to obtain a secure model (*i.e.*, high-performance and without backdoor) based on an untrustworthy training set. To this end, we consider two defense paradigms: one is training a secure model from scratch, while the other is firstly training a backdoored model using standard supervised learning, and then removing backdoor from the backdoored model. Under paradigm 1, we propose an innovative secure training method, called *Distinguishment and Secure Training (D-ST)*, which consists of two consecutive modules. The first *sample-distinguishment (SD) module* splits the whole training set into clean, poisoned and uncertain samples, according to the FCT metric. The second *two-stage secure training (ST) module* firstly learns the feature extractor via semi-supervised contrastive learning, and then learns the classifier via minimizing a mixed cross-entropy loss. Under paradigm 2, we propose an innovative backdoor removal method, called *Distinguishment and Backdoor Removal (D-BR)*, which consists of the SD module and a *backdoor removal (BR) module*. BR module alternatively unlearns the distinguished poisoned samples and learns the distinguished clean samples. Extensive experiments are conducted to verify the superior defense performance of the above two proposed methods, as well as effectiveness of each individual module.

The main contributions of this work are three-folds. **(1)** We demonstrate the sensitivity of poisoned samples to transformations, which is mainly due to the overfitting to trigger, and propose a simple sensitivity metric to distinguish poisoned samples from clean samples. **(2)** We propose two effective backdoor defense methods for training a secure model from scratch and removing backdoor from the backdoored model, respectively. **(3)** Extensive experiments on 3 benchmark datasets show the superior performance of the proposed defense methods against 8 widely used backdoor attacks, to 6 state-of-the-art defense methods.

2 Related work

Backdoor attack. In poisoning-based backdoor attacks, the attacker attaches a few training samples with *trigger(s)*, and relabel them as *target class(es)*. Existing attacks can be categorized according to a variety of criteria as follows. (1) Size of trigger: *Patch-based attacks* [1, 5, 6] craft patch-like triggers while in *blend-based attacks* [7, 8], triggers capture the whole image. (2) Visibility of trigger: *Visible attacks* [1, 6] design visible but not suspicious triggers while *invisible attacks* [8, 9, 10] propose invisible and still effective ones. (3) Variability of trigger: Triggers are invariant in *sample-agnostic attacks* [1, 7, 11] while vary with samples in *sample-specific attacks* [9, 12]. (4) Label-consistency: If poisoned samples are chosen from samples with target class, then we call these attacks as *clean-label attacks* [11, 13, 12, 14]. Otherwise, we name them as *dirty-label attacks* [1, 6, 7, 8]. (5) Number of

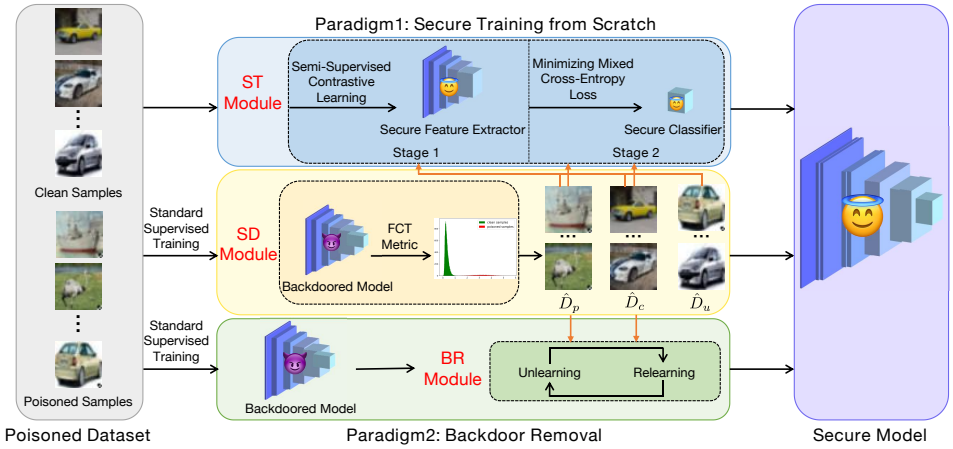


Figure 2: Framework of two proposed backdoor defense methods for secure training from scratch (paradigm1) and backdoor removal (paradigm2), respectively.

target classes: *All2one attacks* [1, 6, 7, 8] designate one class as the target class while in *all2all attacks* [1], poisoned samples are relabelled as the next class. There are also other attacks [15, 16, 17, 18, 19] that require the attacker to control the training process, which are out of scope of this paper.

Backdoor defense. In general, there are two types of defense paradigms against poisoning-based backdoor attacks—secure training and backdoor removal. Mainstream defense methods belong to the latter one, which leverage the model properties [20, 21, 22] or the feature space characteristics [23, 5, 24, 25, 26, 27, 28] of a backdoored model, to remove the hidden backdoor. For instance, FP [20] observes that some neurons are activated by poisoned samples while others are by clean samples. AC [23] notices the difference in size between the cluster of clean target samples and that of poisoned samples. So far, there is still few works for the former defense. DBD [2] observes that poisoned samples gather together in the feature space, and proposes a three-stage defense method to inhibit the gathering. The first stage is training the feature extractor with self-supervised learning. The second stage is training the classifier with supervised learning, where the backdoor may be injected. The third stage is identifying clean samples by the symmetric cross-entropy and fine-tuning the model with semi-supervised learning on labelled clean samples and label-removed remaining samples.

3 Proposed method

3.1 Problem formulation

Threat model. In this paper, we consider the threat model of poisoning-based backdoor attacks, where the attacker can manipulate a few samples in the original clean training set $D_{train} = D_c \cup \{(\boldsymbol{x_i}, y_i)\}_{i=1}^{m_p}$, with $D_c = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{m_c}$ indicating the unmanipulated subset. For the remaining m_p samples, each sample $\boldsymbol{x}_i \in \mathcal{X}$ is fused with a trigger $\boldsymbol{\delta}$ to form a poisoned sample $\bar{\boldsymbol{x}}_i = \boldsymbol{x_i} \oplus \boldsymbol{\delta}$ with $\oplus$ being the fusion operator. Meanwhile, its label $y_i \in \mathcal{Y}$ is also changed to a target class t. Then, a poisoned training set is constructed, denoted as $\bar{D}_{train} \equiv D_c \cup D_p$, with $D_p = \{(\bar{\boldsymbol{x}}_i, t)\}_{i=1}^{m_p}$. When a user downloads $\bar{D}_{train}$ and trains a DNN classifier $g_{\boldsymbol{\theta}} : \mathcal{X} \to \mathcal{Y}$ based on $\bar{D}_{train}$ using the standard supervised learning algorithm, it may learn a undesired backdoor, *i.e.*, a stable mapping from the trigger $\boldsymbol{\delta}$ to the target class t. Consequently, for any new sample with the trigger $\boldsymbol{\delta}$, it is likely to be predicted as the target class t. Note that the user does not know which sample is poisoned or clean.

Defense goal. Given the poisoned training set $\bar{D}_{train}$, the defender aims to obtain a high-performance model $g_{\boldsymbol{\theta}}$ without backdoor, *i.e.*, a secure model. In this work, we consider two different paradigms:

- **Paradigm 1**: a secure model is directly trained from scratch, as described in Section 3.3.
- **Paradigm 2**: a backdoored model is firstly trained using the standard supervised learning, then the backdoor is removed from the backdoored model, as described in Section 3.4.

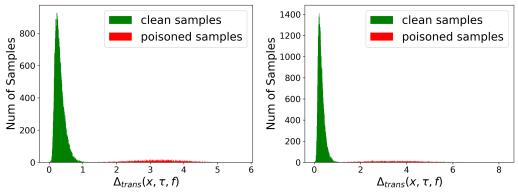
3.2 Sensitivity of poisoned samples

Sensitivity metric. As illustrated in Section 1 and Fig. 1, we have found that the poisoned samples are much more sensitive to transformations than the clean samples in a backdoored model. To accurately measure such a difference, we propose a simple metric, *feature consistency towards transformations (FCT)*. Specifically, given a backdoored model $g_{\boldsymbol{\theta}}$ trained on $\bar{D}_{train}$ with $f_{\boldsymbol{\theta}_e}(\cdot)$ indicating its feature extractor, and a set of transformations τ (*e.g.*, rotation, scaling, will be specified in experiments), for any sample $\boldsymbol{x}$ (poisoned or clean), the FCT metric is formulated as follows:

$$\Delta_{trans}(\boldsymbol{x}; \tau, f_{\boldsymbol{\theta_e}}) = \|f_{\boldsymbol{\theta_e}}(\boldsymbol{x}) - f_{\boldsymbol{\theta_e}}(\tau(\boldsymbol{x}))\|_2^2. \tag{1}$$

It measures the change of the feature representation due to the transformations τ. If $\Delta_{trans}(\boldsymbol{x}; \tau, f)$ is large, then it means that $\boldsymbol{x}$ is sensitive to τ, otherwise stable. For clarity, we use $\Delta_{trans}(\boldsymbol{x})$ hereafter.

Sample-distinguishment module. Utilizing FCT, we develop a sample-distinguishment (SD) module. Specifically, we firstly train a backdoored model g_θ based on $\bar{D}_{train}$ using the standard supervised learning algorithm with a few epochs (explained in Appendix A.1). Then, we calculate $\Delta_{trans}(\boldsymbol{x}_i), \forall \boldsymbol{x}_i \in \bar{D}_{train}$, and plot the histogram. As shown in Fig. 3, where two representative backdoor attacks are evaluated, there is remarkable difference on the distribution between the poisoned and the clean samples in both histograms. It demonstrates that Δ_{trans} is a good metric to distinguish the poisoned samples from the clean samples in $\bar{D}_{train}$. Based on the sensitivity histogram, we set two proportion values $\alpha_c, \alpha_p \in [0, 1]$. The samples with the bottom-α_c Δ_{trans} values are separated to a subset of clean samples $\hat{D}_c$, while those with the top-α_p Δ_{trans} values are separated into a subset of poisoned samples $\hat{D}_p$, while the remaining samples are partitioned as an uncertain subset denoted as $\hat{D}_u$. We have $\bar{D}_{train} = \hat{D}_c \cup \hat{D}_p \cup \hat{D}_u$. More details are in Algorithm 1 in Appendix A.1



(a) BadNets (b) Blend

Figure 3: Distribution of clean and poisoned samples with respect to the FCT metric on CIFAR-10.

3.3 Method for paradigm 1: secure training from scratch

Here, we consider the backdoor defense under paradigm 1. We propose an innovative secure training method, called *Distinguishment and Secure Training* (D-ST) method. As illustrated in Fig. 2, D-ST consists of the SD module (see above) and a **two-stage secure training (ST) module**, which is described as follows. Details of the D-ST method are summarized in Algorithm 3 in Appendix A.2..

Stage 1: learning feature extractor via semi-supervised contrastive learning (SSCL). Our method is inspired by a recent backdoor defense method called DBD [2], which proposed to learn a good feature extractor $f_{\boldsymbol{\theta}_e}$ based on $\bar{D}_{train}$ using a self-supervised learning algorithm, *i.e.*, contrastive learning (CTL). Consequently, the feature representations of samples with similar appearances will be similar, and poisoned samples with triggers cannot gather together to form the backdoor. Note that all labels have been abandoned in DBD before the extractor learning since there is no way to identify poisoned samples, leading to the waste of the valuable information contained in clean samples. Fortunately, the proposed SD module could identify some clean samples. Thus, inspired by the supervised contrastive learning (S-CTL) [29], which has shown to learn a feature extractor with better performance than CTL, we propose a novel learning called *semi-supervised contrastive learning (SS-CTL)*, to learn $f_{\boldsymbol{\theta}_e}$ by minimizing the following loss function:

$$\begin{aligned}\mathcal{L}_{SS-CTL}(\boldsymbol{\theta}_e; \bar{D}_{train}) = &\sum_{(\boldsymbol{x}_i, y_i) \in \hat{D}_p \cup \hat{D}_u} \ell_{CTL}\big(f_{\boldsymbol{\theta}_e}(\tilde{\boldsymbol{x}}_{\boldsymbol{i}}^{(\mathbf{1})}), f_{\boldsymbol{\theta}_e}(\tilde{\boldsymbol{x}}_{\boldsymbol{i}}^{(\mathbf{2})})\big) \\ &+ \sum_{\{(\boldsymbol{x}_i, y_i), (\boldsymbol{x}_j, y_j)\} \subset \hat{D}_c} \ell_{S-CTL}\big(f_{\boldsymbol{\theta}_e}(\tilde{\boldsymbol{x}}_{\boldsymbol{i}}^{(\mathbf{1})}), f_{\boldsymbol{\theta}_e}(\tilde{\boldsymbol{x}}_{\boldsymbol{i}}^{(\mathbf{2})}), f_{\boldsymbol{\theta}_e}(\tilde{\boldsymbol{x}}_{\boldsymbol{j}}^{(\mathbf{1})}), f_{\boldsymbol{\theta}_e}(\tilde{\boldsymbol{x}}_{\boldsymbol{j}}^{(\mathbf{2})}); y_i, y_j\big),\end{aligned} \tag{2}$$

where the contrastive loss ℓ_{CTL} encourages the two augmented versions $\tilde{\boldsymbol{x}}_{\boldsymbol{i}}^{(\mathbf{1})}$, $\tilde{\boldsymbol{x}}_{\boldsymbol{i}}^{(\mathbf{2})}$ (*e.g.*, cropping, details are introduced in Appendix C) of a sample $\boldsymbol{x_i}$ to be close in the feature space, while the supervised contrastive loss ℓ_{S-CTL} additionally encourages the feature representations of two clean augmented samples from the same class to be close. In this work, we instantiate ℓ_{CTL} as the contrastive loss defined in [30] and ℓ_{S-CTL} as the SupCon loss defined in [29].

Stage 2: learning classifier via minimizing the mixed cross-entropy loss. Given the feature extractor $f_{\boldsymbol{\theta}_e}$ learned in stage 1, we then learn the classifier $h_{\boldsymbol{\theta}_c}$ by minimizing the following mixed cross-entropy (MCE) loss:

$$\mathcal{L}_{MCE}(\boldsymbol{\theta}_c; \hat{D}_c, \hat{D}_p) = \frac{-1}{|\hat{D}_c|} \sum_{(\boldsymbol{x},y)\in\hat{D}_c} \log[h_{\boldsymbol{\theta}_c}(f_{\boldsymbol{\theta}_e}(\boldsymbol{x}))]_y + \frac{\lambda_p}{|\hat{D}_p|} \cdot \sum_{(\boldsymbol{x},y)\in\hat{D}_p} \log[h_{\boldsymbol{\theta}_c}(f_{\boldsymbol{\theta}_e}(\boldsymbol{x}))]_y, \quad (3)$$

where the first term is the standard cross-entropy loss defined based on the distinguished clean samples $\hat{D}_c$, while the second term is the negative cross-entropy loss defined based on the distinguished poisoned samples $\hat{D}_p$, which is used to eliminate the effect of poisoned samples. $\lambda_p \in \mathbb{R}^+$ is a trade-off parameter between two losses.

3.4 Method for paradigm 2: backdoor removal

Here we consider the backdoor defense under paradigm 2. We propose an innovative backdoor removal method, called *Distinguishment and Backdoor Removal* (D-BR) method. As illustrated in Fig. 2, D-BR consists of the SD module (see Section 3.2) and a **backdoor removal (BR) module**. The BR module aims to remove the backdoor from the backdoored model, *i.e.*, the backdoor is no longer activated by the trigger, while keeping the high performance on clean samples. To this end, the BR module implements an iterative learning algorithm, which consists of two alternating steps, *i.e.*, *unlearning* and *relearning*. The D-BR method is summarized in Algorithm 4 in Appendix A.3.

Unlearning. This step aims to eliminate the effect of the trigger, through unlearning [31] the poisoned samples in $\hat{D}_p$ distinguished by the SD module, as follows:

$$\mathcal{L}_{unlearn}(\boldsymbol{\theta}; \hat{D}_p) = \frac{1}{|\hat{D}_p|} \sum_{(\boldsymbol{x},y)\in\hat{D}_p} \log[g_{\boldsymbol{\theta}}(\boldsymbol{x})]_y. \quad (4)$$

Relearning. After conducting the above unlearning step for one epoch, although the effect of poisoned samples is somewhat eliminated, in experiments we find that the performance on clean samples is also degraded to some extent. Thus, we want to relearn the mapping from the clean objects to the ground-truth classes based on clean samples in $\hat{D}_c$ distinguished by the SD module, as follows:

$$\mathcal{L}_{relearn}(\boldsymbol{\theta}; \hat{D}_c) = \frac{1}{|\hat{D}_c|} \sum_{(\boldsymbol{x},y)\in\hat{D}_c} -\log[g_{\boldsymbol{\theta}}(\boldsymbol{x})]_y. \quad (5)$$

Note that both unlearning and relearning are run for one epoch in each round.

4 Experiments

4.1 Experimental settings

Attack configurations. According to the taxonomy described in Section 2, we consider 8 typical poisoning-based backdoor attacks by choosing at least one method from each category, including: BadNets [1] using two attack types (BadNets-all2one, BadNets-all2all), Trojan backdoor attack [6] (Trojan), Blend backdoor attack using two different patterns (Blend-Signal, Blend-Kitty) [1] [7], Clean-label backdoor (CL) [13], Sinusoidal signal backdoor attack (SIG) [11], Sample-specific backdoor attack (SSBA) [9]. We evaluate all attacks on 3 benchmark datasets, CIFAR-10 [3], CIFAR-100 [3] and an ImageNet subset [32, 9], with ResNet-18 [33] as the base model. Poisoning rate is set to 10%

[1] If not specified, 'BadNets/Blend' generally stands for the BadNets-all2one/Blend-signal attack.

in all attacks. Due to the space limit, more implementations details about attacks can be found in Appendix C.3, and results on the ImageNet subset are shown in Appendix D.

Defense configurations. We first compare the proposed D-ST method with DBD [2]. Since studies with this secure-training paradigm are limited, we additionally add 2 baselines for comparison which are detailed in Section 4.2. We then compare the proposed D-BR method with 5 state-of-the-art methods with the same backdoor-removal paradigm: the standard fine-tuning FT, ANP [21], NAD [34], MCR [35] and ABL [36]. For methods requiring extra clean data, 1% of the clean training samples are provided. Other configurations are set as clarified in the original papers. In summary, we consider 6 state-of-the-art defense methods and 2 additional baselines. More implementations details can be found in Appendix C.4. For our proposed methods, we use $\alpha_c = 20\%$, $\alpha_p = 5\%$ and τ =rotate+affine in all experiments. Other details can be seen in Appendix C.5.

Evaluation metrics. We evaluate the defense performance adopting two commonly used metrics: accuracy on clean samples (ACC) and attack success rate (ASR), *i.e.*, accuracy of predicting poisoned samples as the target label.

4.2 Experimental results

Effectiveness of D-ST method. We first consider paradigm 1—secure training from scratch. Performance of different defense methods against various attacks on CIFAR-10 and CIFAR-100 is demonstrated in Table 1. An ideal defense method is supposed to increase ACC while keep ASR as low as possible. Thus, a larger ACC-ASR indicates a better method. We mark the best result in boldface. Note that we only report results on successful attacks where ASR is higher than 85%.

Table 1: Comparisons of the D-ST method with 3 secure-training defense methods (%).

Dataset ↓	Defense →	Baseline1		Baseline2		DBD		D-ST	
	Attack ↓	ACC	ASR	ACC	ASR	ACC	ASR	ACC	ASR
CIFAR-10	BN-all2one	83.54	2.60	91.32	99.91	92.75	100.00	**92.77**	**0.03**
	BN-all2all	83.95	2.72	91.59	57.39	92.95	75.21	**89.22**	**2.05**
	Trojan	83.77	5.24	93.63	99.98	92.81	100.00	**93.72**	**0.00**
	Blend-Strip	85.36	99.93	94.19	100.00	94.21	99.98	**93.59**	**0.00**
	Blend-Kitty	85.03	99.99	94.31	100.00	93.32	100.00	**91.82**	**0.00**
	SIG	85.14	99.02	94.37	99.93	94.37	99.71	**90.07**	**0.00**
	CL	85.79	10.76	94.58	98.87	94.32	99.87	**90.46**	**6.40**
	Avg	84.65	45.75	93.43	93.73	93.53	96.40	**91.66**	**1.21**
CIFAR-100	BN-all2one	54.48	10.41	67.62	100.00	69.08	100.00	**68.43**	**0.12**
	Trojan	56.17	12.76	71.01	100.00	72.18	99.99	**68.04**	**0.08**
	Blend-Strip	58.01	99.91	72.47	99.99	71.29	99.99	**67.63**	**0.00**
	Blend-Kitty	57.21	99.99	73.36	99.99	72.43	100.00	**67.06**	**0.00**
	Avg	56.47	55.77	71.12	100.00	71.24	99.99	**67.79**	**0.05**

DBD fails in most attacks on CIFAR-10 and CIFAR-100, probably due to the failure of the symmetric cross-entropy to distinguish samples. By comparison, the good performance reached by D-ST illustrates the accurate distinguishment from the FCT-based SD module. We additionally introduce two feasible baselines without requiring special knowledge. Baseline1 first uses SimCLR [30] to train the feature extractor and then trains the classifier on the poisoned dataset with standard supervised learning. By comparison, Baseline2 leverages S-CTL [29] to train the feature extractor. We focus on discussing the effect of different extractor-training algorithms on defense performance. More discussions are in the later experiments. Baseline 1 reveals that training extractor without labels may result in low ASR (<5% / <20% in some cases on CIFAR-10 / CIFAR-100), but will sacrifice ACC definitely (84.65% / 56.47% on average). While Baseline 2 demonstrates that training extractor with all labels guarantees high ACC (93.43% / 71.12% on average), but also brings high ASR (93.73% / 100% on average). By contrast, results of D-ST illustrate the effectiveness of ST module in training the feature extractor in a secure way since ACC is high (91.66% / 67.79% on average) and ASR (1.21% / 0.05% on average) is extremely low.

Effectiveness of D-BR method. Then, we consider paradigm 2—backdoor removal. Defense performance of different defense methods against various attacks on CIFAR-10 and CIFAR-100 is demonstrated in Table 2. Results on ImageNet is shown in Table 5 in Appendix D.

Table 2: Comparisons of the D-BR method with 5 backdoor-removal defense methods on CIFAR-10 and CIFAR-100 (%). 'Backdoored' refers to the backdoored model. * denotes methods which require a few (1%) clean training samples.

Dataset ↓	Defense → / Attack ↓	Backdoored		FT*		ANP*		NAD*		MCR*		ABL		D-BR	
		ACC	ASR	ACC	ASR	ACC	ASR	ACC	ASR	ACC	ASR	ACC	ASR	ACC	ASR
CIFAR-10	BN-all2one	91.64	100.00	88.99	66.79	90.03	10.54	84.46	2.13	94.21	8.29	89.36	0.19	**92.83**	**0.40**
	BN-all2all	92.79	88.01	90.31	4.96	86.04	1.47	84.97	1.71	92.17	2.96	79.91	78.16	**92.61**	**0.56**
	Trojan	91.91	100.00	89.86	100.00	90.89	0.81	83.29	5.04	93.90	2.58	90.18	0.23	**92.21**	**0.76**
	Blend-Strip	92.09	99.97	89.91	93.50	88.33	0.04	83.09	13.30	91.77	17.96	58.46	0.22	**92.40**	**0.06**
	Blend-Kitty	92.69	99.99	90.47	99.31	84.07	0.01	84.54	28.96	94.42	7.49	79.20	2.27	**92.11**	**0.14**
	SIG	92.88	99.69	90.81	99.87	82.43	76.32	81.00	64.72	91.82	99.04	79.94	98.84	**92.73**	**0.24**
	CL	93.20	93.34	90.03	77.44	72.57	10.90	84.46	2.66	92.13	72.01	84.39	0.31	**92.08**	**0.00**
	Avg	92.46	97.29	90.05	77.41	84.91	14.30	83.69	16.93	92.92	30.05	80.21	25.75	**92.42**	**0.31**
CIFAR-100	BN-all2one	71.23	99.13	70.81	66.28	65.42	0.00	69.03	11.41	**73.38**	**0.27**	66.47	0.02	72.58	0.25
	Trojan	75.75	100.00	74.21	99.94	64.52	0.03	72.11	92.21	74.51	0.12	68.12	0.00	**74.52**	**0.00**
	Blend-Strip	75.54	99.99	73.36	99.65	67.38	0.00	71.18	95.78	73.37	0.07	49.13	0.00	**74.35**	**0.00**
	Blend-Kitty	75.18	99.97	72.93	99.96	69.03	0.00	71.73	99.93	73.93	20.60	47.05	0.00	**72.00**	**0.01**
	Avg	74.43	99.77	72.83	91.46	66.59	0.01	71.01	74.83	73.80	5.27	57.69	0.01	**73.36**	**0.07**

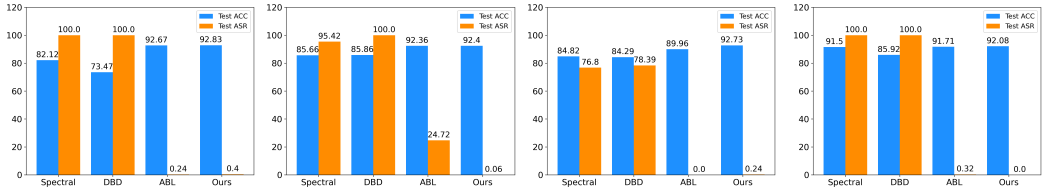
Results on CIFAR-10. We discover that except for FT and MCR, other selected methods generally reduce ACC markedly. Additionally, they have two common disadvantages. (1) They can not take effect on all attacks. For example, ANP can defend against Trojan and Blend attacks (ASR < 1%) while fails in clean-label attacks, i.e. SIG and CL (ASR > 10%). (2) There exists at least one attack that can disable the methods (ASR > 50%). By contrast, the proposed D-BR method overcomes these drawbacks. It not only maintains ACC as large as that of backdoored model, but also reduces ASR to less than 1% on all attacks, verifying the effectiveness of the BR module and the high precision of the distinguishment conducted by the SD module.

Results on CIFAR-10. Although FT and NAD have a relatively high ACC (> 68%), they fail to reduce ASR (91.46% and 74.83% on average). While ANP and ABL can decrease ASR to less than 0.1%, they sacrifice too much ACC (66.59% and 57.69% on average). Among the selected methods, MCR performs the best (ACC = 73.80% on average, ASR < 0.5% in three cases), but it still fails to defend against the Blend-Kitty attack (ASR = 20.60%). Note that MCR requires extra clean data. In contrast, D-BR keeps ACC higher than 72%, while reduces ASR to almost 0% without any extra clean data.

4.3 Ablation studies

Effectiveness of the SD module. Here, we aim to study the effectiveness of the SD module. Specifically, we will show how our proposed FCT metric, performs better than other metrics, under the backdoor-removal paradigm for illustration. To this end, we select three existing metrics for comparison. Spectral signatures [5] specifies the metric as *the correlation with the top singular vector of the covariance matrix of feature representations*. DBD [2] assigns *symmetric cross-entropy loss* as the metric. The metric used in ABL [36] is *loss value applied with local gradient ascent*. The metric values of clean samples are smaller than those of poisoned samples according to the former two metrics, while larger for the third metric. For fair comparison, we uniformly set $\alpha_c = 20\%, \alpha_p = 5\%$. We first apply the metric-replaced SD module on the poisoned training set, and then conduct the BR module based on the distinguished samples. Results are shown in Fig. 4.

The height of the blue bar *above* the orange bar suggests how well the metric could distinguish. As shown in Fig. 4 (a,b,d), orange bars are all higher than blue bars for Spectral signatures and DBD, indicating metrics of which fail to distinguish in BadNets, Blend and CL attacks. In contrast, the

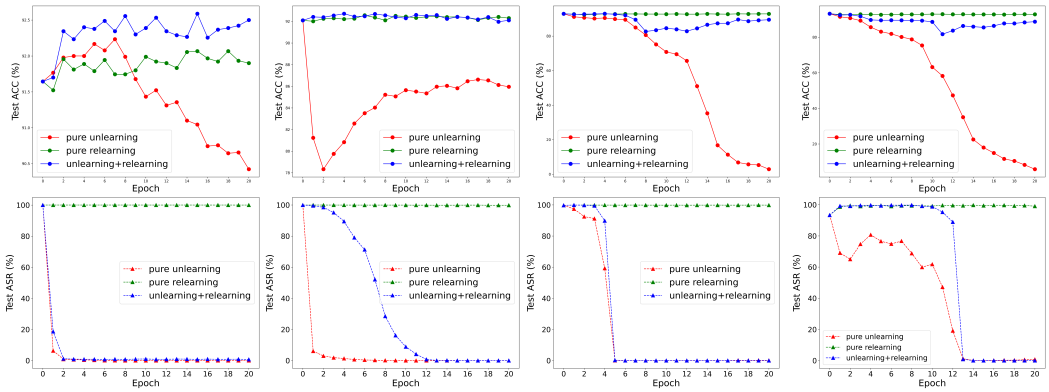


(a) BadNets (b) Blend (c) SIG (d) CL

Figure 4: Test ACC and Test ASR of four metric-replaced D-BR methods on the poisoned CIFAR-10.

metric of ABL is reliable since ABL performs well in most cases except for Blend attack where ASR is 24.72%. By comparison, our proposed FCT metric could distinguish samples stably well, resulting in extremely low ASR (< 0.5%) on all attacks. We attribute the success to that FCT exploits the sensitivity of poisoned samples, which is mainly due to the overfitting to trigger by the backdoored model that exists in all backdoor attacks we have evaluated in this paper.

Effectiveness of the BR module. Here, we focus on studying the effectiveness of the BR module. Specifically, we aim to show how the iterative learning algorithm consisting of unlearning and relearning performs better than the *pure unlearning* adopted by [36] or *pure relearning*. To this end, we first conduct the SD module, and then apply different learning algorithms. For the three algorithms, we run 20 epochs on CIFAR-10 and record the variations of Test ACC and Test ASR which are illustrated as Fig. 5.



(a) BadNets (b) Blend (c) SIG (d) CL

Figure 5: Test ACC(top) and Test ASR(bottom) of three learning algorithms on poisoned CIFAR-10.

Although *pure unlearning* (red lines) effectively decreases ASR, it could hardy maintain ACC, showing a downward trend. The results indicate a strict requirement for choosing the number of unlearning epochs. On the contrary, *pure relearning* (green lines) can keep ACC stably high, but it takes tiny effect in reducing ASR. By contrast, *unlearning+relearning* (blue lines) combines their advantages and successfully diminishes ASR while maintains ACC. ACC and ASR steadily converge to high and low values, respectively, validating the effectiveness and the stability of the BR module.

Effectiveness of the ST module. Here, we aim to study the effectiveness of the ST Module. Backdoor can be injected during training the feature extractor $f_{\boldsymbol{\theta}_e}$ and the classifier $h_{\boldsymbol{\theta}_c}$. The final defense performance of $g_{\boldsymbol{\theta}}$ depends on how well $f_{\boldsymbol{\theta}_e}$ and $h_{\boldsymbol{\theta}_c}$ inhibit backdoor.

Firstly, we want to show how SS-CTL performs better than CTL or S-CTL in training a secure feature extractor $f_{\boldsymbol{\theta}_e}$. To this end, we train $f_{\boldsymbol{\theta}_e}$ with different learning algorithms and then uniformly leverage $\mathcal{L}_{MCE}$ to train $h_{\boldsymbol{\theta}_c}$. Training $f_{\boldsymbol{\theta}_e}$ with CTL, as shown in the first row, guarantees low ASR, but the low ACC turns into a tradeoff. Note that the low ASR is the joint effort of CTL and $\mathcal{L}_{MCE}$. And the usage of CTL does not indicate low ASR definitely, but it indeed reduces the possibility of backdoor injection in $f_{\boldsymbol{\theta}_e}$ The second row illustrates that S-CTL could bring high ACC and the potentially high ASR, as seen in the SIG and CL attacks. Since all labels (including poisoned) are used in this scenario, ACC is reasonably high. Besides, backdoor is already injected into $f_{\boldsymbol{\theta}_e}$. But due to the inhibition effect of $\mathcal{L}_{MCE}$, the ASR of $g_{\boldsymbol{\theta}}$ may not be high. For example, in dirty-label attacks, i.e. BadNets, Trojan and Blend attacks, ASR is almost 0%. While in clean-label attacks, i.e. SIG and CL attacks, $\mathcal{L}_{MCE}$ can not withstand the backdoor injected in $f_{\boldsymbol{\theta}_e}$, so the ASR is almost 100%. Hence, in order to establish a reliable defense module, $f_{\boldsymbol{\theta}_e}$ should be trained in a more secure way. In comparison, the third row demonstrates the superior defense performance of SS-CTL, illustrating that the ST module securely bridges genuinely clean intra-class samples together which are distinguished by the SD module.

Table 3: Performance with $f_{\boldsymbol{\theta}_e}$ trained with three learning algorithms on the poisoned CIFAR-10.

Attack →	BN-all2one		BN-all2all		Trojan		Blend-Signal		Blend-Kitty		SIG		CL	
$f_{\boldsymbol{\theta}_e}$ ↓	ACC	ASR	ACC	ASR	ACC	ASR	ACC	ASR	ACC	ASR	ACC	ASR	ACC	ASR
CTL	85.63	1.52	83.02	1.65	85.03	1.32	85.12	0.00	83.49	0.00	83.10	0.00	83.77	4.88
S-CTL	92.98	0.00	93.73	0.73	93.80	0.00	94.09	0.00	94.18	0.00	94.51	99.77	94.67	98.34
SS-CTL	92.77	0.03	89.22	2.05	93.72	0.00	93.59	0.00	91.82	0.00	90.07	0.00	90.46	6.40

Secondly, we explore how $\mathcal{L}_{MCE}$ affects the defense performance of $h_{\boldsymbol{\theta}_c}$. For clarity, we denote $\mathcal{L}_1 \equiv \frac{-1}{|\hat{D}_c|}\sum_{(\boldsymbol{x},y)\in\hat{D}_c}\log[h_{\boldsymbol{\theta}_c}(f_{\boldsymbol{\theta}_e}(\boldsymbol{x}))]_y$ and $\mathcal{L}_2 \equiv \frac{1}{|\hat{D}_p|}\cdot\sum_{(\boldsymbol{x},y)\in\hat{D}_p}\log[h_{\boldsymbol{\theta}_c}(f_{\boldsymbol{\theta}_e}(\boldsymbol{x}))]_y$. We have $\mathcal{L}_{MCE} = \mathcal{L}_1 + \lambda_p\mathcal{L}_2$. Generally, if knowing clean samples, the defender will train h_{θ_c} with $\mathcal{L}_1$. So here, we aim to show how our proposed $\mathcal{L}_2$ and the trade-off parameter λ_p affect $h_{\boldsymbol{\theta}_c}$. To this end, we first fix $f_{\boldsymbol{\theta}_e}$ learned by SS-CTL and then apply $\mathcal{L}_{MCE}$ with $\lambda_p = 0, 0.001, 0.01, 0.1, 1$ on $h_{\boldsymbol{\theta}_c}$. Results are shown in Fig. 6. In the previous experiments, we adopt $\lambda_p = 0.001$.

The comparison between $\lambda_p = 0$ and $\lambda_p \neq 0$ in the right figure illustrates that $\mathcal{L}_2$ can effectively reduce ASR. When comparing different $\lambda_p \neq 0$ in the left figure, we discover that as λ_p increases, there is a trend of decrease in ACC. We infer that since $\mathcal{L}_2$ drops faster than $\mathcal{L}_1$, namely unlearning is faster than relearning, adding weights to $\mathcal{L}_2$ makes $h_{\boldsymbol{\theta}_c}$ focus on unlearning instead of relearning, leading to the low ACC. Therefore, we conclude that $\mathcal{L}_2$ helps to inhibit backdoor in $h_{\boldsymbol{\theta}_c}$, but its weight should not be too large. $\lambda_p = 0.001$ is considered to be an appropriate choice.

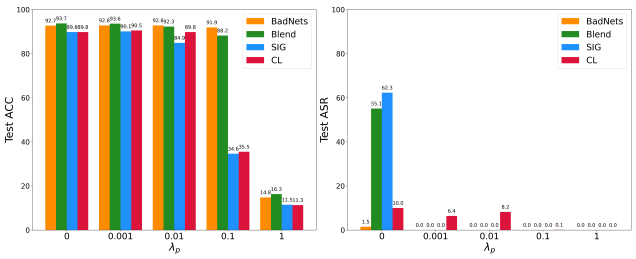


Figure 6: Test ACC (left) and Test ASR (right) under various λ_p on the poisoned CIFAR-10.

In summary, we have empirically validated the effectiveness of each individual module, and shown the flexibility of our method to combine with other existing modules.

Appendix. Due to the space limit, more results and analysis will be presented in Appendix, including: **(1)** performance with different data transformations τ in Appendix E; **(2)** performance with different proportion values α_c, α_p in Appendix F; **(3)** performance with different poisoning rates in Appendix G; **(4)** complexities of two proposed methods in Appendix H.

5 Conclusions

In this paper, we reveal the sensitivity of poisoned samples to transformations and propose a sensitivity metric, called FCT. Besides, we propose three modules—the SD module to distinguish between clean and poisoned samples, the ST module to train a secure model from scratch and the BR module to remove backdoor—which constitute two defense methods, *i.e.*D-ST and D-BR, to defend under two different defense paradigms. Extensive experiments have demonstrated the effectiveness of each individual module and also the proposed defense methods.

6 Broader impact

Poisoning-based backdoor attacks are severe threats to the learning paradigm of learning a DNN model based on the training set from some untrustworthy sources. This work reveals the sensitivity of poisoned samples in the backdoored model, which will help people to better understand the inner mechanism of backdoor attacks. The proposed two effective defense methods can not only significantly mitigate the threat of existing poisoning based backdoor attacks, but also serve as the new baseline for developing more advanced attack methods in future.

References

[1] Tianyu Gu, Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Evaluating backdooring attacks on deep neural networks. *IEEE Access*, 7:47230–47244, 2019.

[2] Kunzhe Huang, Yiming Li, Baoyuan Wu, Zhan Qin, and Kui Ren. Backdoor defense via decoupling the training process. In *ICLR*, 2022.

[3] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. *Technical Report, University of Toronto*, 2009.

[4] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 2008.

[5] Brandon Tran, Jerry Li, and Aleksander Madry. Spectral signatures in backdoor attacks. In *NIPS*, 2018.

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

[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] Yunfei Liu, Xingjun Ma, James Bailey, and Feng Lu. Reflection backdoor: A natural backdoor attack on deep neural networks. In *ECCV*, 2020.

[9] Yuezun Li, Yiming Li, Baoyuan Wu, Longkang Li, Ran He, and Siwei Lyu. Invisible backdoor attack with sample-specific triggers. In *ICCV*, 2021.

[10] Eugene Bagdasaryan and Vitaly Shmatikov. Blind backdoors in deep learning models. In *USENIX*, 2021.

[11] Mauro Barni, Kassem Kallas, and Benedetta Tondi. A new backdoor attack in cnns by training set corruption without label poisoning. In *ICIP*, 2019.

[12] Aniruddha Saha, Akshayvarun Subramanya, and Hamed Pirsiavash. Hidden trigger backdoor attacks. In *AAAI*, 2020.

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

[14] Shihao Zhao, Xingjun Ma, Xiang Zheng, James Bailey, Jingjing Chen, and Yu-Gang Jiang. Clean-label backdoor attacks on video recognition models. In *CVPR*, 2020.

[15] Tuan Anh Nguyen and Anh Tuan Tran. Wanet - imperceptible warping-based backdoor attack. In *ICLR*, 2021.

[16] Tuan Anh Nguyen and Anh Tran. Input-aware dynamic backdoor attack. In *NIPS*, 2020.

[17] Junyu Lin, Lei Xu, Yingqi Liu, and Xiangyu Zhang. Composite backdoor attack for deep neural network by mixing existing benign features. In *CCS*, 2020.

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

[19] Ilia Shumailov, Zakhar Shumaylov, Dmitry Kazhdan, Yiren Zhao, Nicolas Papernot, Murat A Erdogdu, and Ross J Anderson. Manipulating sgd with data ordering attacks. In *NIPS*, 2021.

[20] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Fine-pruning: Defending against backdooring attacks on deep neural networks. In *RAID*, 2018.

[21] Dongxian Wu and Yisen Wang. Adversarial neuron pruning purifies backdoored deep models. In *NIPS*, 2021.

[22] Kota Yoshida and Takeshi Fujino. Disabling backdoor and identifying poison data by using knowledge distillation in backdoor attacks on deep neural networks. In *ACM Workshop*, 2020.

[23] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian M. Molloy, and Biplav Srivastava. Detecting backdoor attacks on deep neural networks by activation clustering. In *AAAI*, 2019.

[24] Ezekiel Soremekun, Sakshi Udeshi, and Sudipta Chattopadhyay. Exposing backdoors in robust machine learning models. *arXiv preprint arXiv:2003.00865*, 2020.

[25] Alvin Chan and Yew-Soon Ong. Poison as a cure: Detecting & neutralizing variable-sized backdoor attacks in deep neural networks. *arXiv preprint arXiv:1911.08040*, 2019.

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

[27] Haripriya Harikumar, Vuong Le, Santu Rana, Sourangshu Bhattacharya, Sunil Gupta, and Svetha Venkatesh. Scalable backdoor detection in neural networks. In *ECML*, 2020.

[28] Zhen Xiang, David J. Miller, and George Kesidis. Detection of backdoors in trained classifiers without access to the training set. *IEEE TNL*, 2022.

[29] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. In *NIPS*, 2020.

[30] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. A simple framework for contrastive learning of visual representations. In *ICML*, 2020.

[31] Lucas Bourtoule, Varun Chandrasekaran, Christopher A Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, and Nicolas Papernot. Machine unlearning. In *IEEE S&P*, 2021.

[32] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

[33] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

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

[35] Pu Zhao, Pin-Yu Chen, Payel Das, Karthikeyan Natesan Ramamurthy, and Xue Lin. Bridging mode connectivity in loss landscapes and adversarial robustness. In *ICLR*, 2020.

[36] Yige Li, Xixiang Lyu, Nodens Koren, Lingjuan Lyu, Bo Li, and Xingjun Ma. Anti-backdoor learning: Training clean models on poisoned data. In *NIPS*, 2021.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Appendix G.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Section 6.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 4 and Appendix C.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section C.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See Section 4.
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A More algorithmic details and analysis on the proposed method

A.1 Sample-distinguishment module

We summarize the SD module in Algorithm 1. Note that it is a simplified version for an easy understanding and the detailed version is illustrated in Algorithm 2.

Algorithm 1 A Simplified Version of Sample-Distinguishment Module

Input: poisoned training set $\bar{D}_{train}$, a set of data transformations τ, randomly initialized model g_θ, number of training epochs e_t, proportion of separated clean (poisoned) samples α_c (α_p)

Output: separated clean (poisoned) samples $\hat{D}_c$ ($\hat{D}_p$), the remaining uncertain samples $\hat{D}_u$

```
# Inject backdoor into model
Train a backdoored model g_θ by minimizing the standard cross-entropy on D̄_train for e_t epochs
Let f(·) denote the feature representation of g_θ
# Assign threshold for choosing clean and poisoned samples
Initialize S ← {}
for each (x, y) ∈ D̄_train do
    S.append(Δ_trans(x; τ, f))
end for
S' ← sorted(S) in ascending order
γ_c ← [S']_{α_c|D̄_train|}, γ_p ← [S']_{(1−α_p)|D̄_train|}
# Separate samples
for each (x, y) ∈ D̄_train do
    if Δ_trans(x; τ, f) ≤ γ_c then
        D̂_c.append((x, y))
    else if Δ_trans(x; τ, f) ≥ γ_p then
        D̂_p.append((x, y))
    else
        D̂_u.append((x, y))
    end if
end for
return D̂_c, D̂_p, D̂_u
```

We omit some algorithmic details and state the SD module in Algorithm 1 for an easy understanding. Here, we continue to elaborate our mechanism in Algorithm 2. The main supplement is the step of fine-tuning the backdoored model, which is highlighted in blue.

Empirically, we discover that $e_t = 2$ is enough for injecting backdoor into model g_θ for the Train ASR is already higher than 90%. When we then use Algorithm 1 to distinguish samples, we observe that it works under dirty-label attacks (shown in Fig. 8(a,b,d,e)) since the distributions of clean and poisoned samples are quite distinct. However, it doesn't work under clean-label attacks (shown in Fig. 8(c,f)) since poisoned samples are mixed up with clean samples.

We first analyze the reason for the success under dirty-label attacks and then give an assumption of the reason for the failure under the clean-label attacks. In the feature space of a backdoored model [2, 23], poisoned samples are congregated into one cluster, called the *poisoned cluster*, as illustrated in black in Fig. 7. Let C_c^i, C_p^j denote the cluster made up of clean samples labelled as i and poisoned samples labelled as j, respectively. We formulate the *intra-cluster distance* between C_c^i and C_p^j as follows.

$$d_{intra}(C_c^i, C_p^j) = \|c_c^i - c_p^j\|_2^2 = \|\frac{\sum_{x_i \in C_c^i} f(x_i)}{|C_c^i|} - \frac{\sum_{x_j \in C_p^j} f(x_j)}{|C_p^j|}\|_2^2, \tag{6}$$

where c_c^i, c_p^j are the center of C_c^i and C_p^j, respectively. Under dirty-label attacks, when the trigger is perturbed by the the transformations τ, poisoned samples are no longer located at the poisoned cluster C_p^t (see black points). Instead, they move towards their *true-class clusters* $C_c^i (i \neq t)$ (see non-black points). The long intra-cluster distance between the two clusters, namely $d_{intra}(C_c^i, C_p^t)(i \neq t)$,

Algorithm 2 A Detailed Version of Sample-Distinguishment Module

Input: the poisoned training set $\bar{D}_{train}$, a set of data transformations τ, randomly initialized model g_θ, number of training epochs e_t, number of fine-tuning epochs e_{ft}, proportion of separated clean samples α_c, proportion of separated poisoned samples α_p, number of classes N

Output: separated clean samples $\hat{D}_c$, separated poisoned samples $\hat{D}_p$, the remaining uncertain samples $\hat{D}_u$

 # Inject backdoor into model

Train g_θ with the standard cross-entropy loss on $\bar{D}_{train}$ for e_t epochs

Let $f(\cdot)$ denote the feature representation of g_θ

Let $c^i = \frac{\sum_{x_i \in C^i} f(x_i)}{|C^i|}$ denote the center of cluster C^i, which consists of samples labelled as i

Let $\mathcal{L}_{intra} = \frac{1}{N^2} \sum_{i,j \in \{0,1 \ldots N-1\}, i \neq j} \frac{< c^i, c^j >}{\|c^i\|_2 \cdot \|c^j\|_2}$

Fine-tune g_θ with $\mathcal{L}_{intra}$ on $\bar{D}_{train}$ for e_{ft} epochs

 # Assign threshold for choosing clean and poisoned samples

Initialize $S \leftarrow \{\}$

for each $(\boldsymbol{x}, y) \in \bar{D}_{train}$ **do**

  S.append($\Delta_{trans}(\boldsymbol{x}; \tau, f)$)

end for

$S' \leftarrow sorted(S)$ in ascending order

$\gamma_c \leftarrow [S']_{\alpha_c |\bar{D}_{train}|}$, $\gamma_p \leftarrow [S']_{(1-\alpha_p)|\bar{D}_{train}|}$

 # Separate samples

for each $(\boldsymbol{x}, y) \in \bar{D}_{train}$ **do**

  if $\Delta_{trans}(\boldsymbol{x}; \tau, f) \leq \gamma_c$ **then**

    $\hat{D}_c$.append($(\boldsymbol{x}, y)$)

  else if $\Delta_{trans}(\boldsymbol{x}; \tau, f) \geq \gamma_p$ **then**

    $\hat{D}_p$.append($(\boldsymbol{x}, y)$)

  else

    $\hat{D}_u$.append($(\boldsymbol{x}, y)$)

  end if

end for

return $\hat{D}_c$, $\hat{D}_p$, $\hat{D}_u$

makes the $\Delta_{trans}(x; \tau, f)$ of poisoned samples large. By contrast, under clean-label attacks where poisoned samples are labeled as their true classes (*i.e.*ground-truth classes), the poisoned cluster C_p^t (see black points) adjoins C_c^t (see red points) which is also the true-class cluster, leading to the short intra-cluster distance, namely $d_{intra}(C_c^t, C_p^t)$, and the resulting small $\Delta_{trans}(x; \tau, f)$ of poisoned samples.

In order to validate our assumption, we fine-tune the model g_θ backdoored by clean-label attacks with the *intra-cluster loss* $\mathcal{L}_{intra}^p$ for $e_f = 5$ epochs, which aims to enlarge $d_{intra}(C_c^t, C_p^t)$.

$$\mathcal{L}_{intra}^p = \frac{1}{N} \sum_{i \in \{0,1 \ldots N-1\}} \frac{< c_p^t, c_c^i >}{\|c_p^t\|_2 \cdot \|c_c^i\|_2}, \tag{7}$$

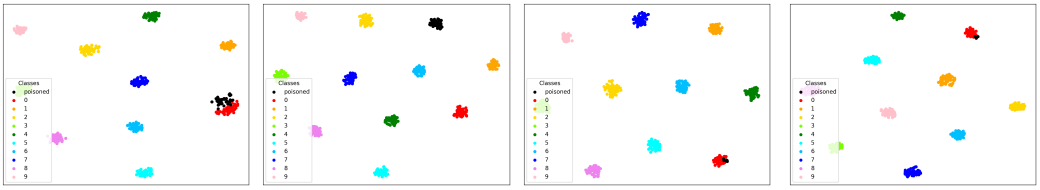
where N denote the number of classes. $\mathcal{L}_{intra}^p$ can successfully increase $d_{intra}(C_c^t, C_p^t)$ from 0.25 to 17.19. Then, we reuse the SD module and find that clean and poisoned samples can be well separated.

However, in the realistic scenario, we can not exactly know the poisoned cluster. So, the fine-tuning strategy above is infeasible. Instead, we fine-tune the model g with the *intra-class loss* $\mathcal{L}_{intra}$ for $e_f = 5$ epochs. Denote C^i as the cluster made up of all samples labelled as i. $\mathcal{L}_{intra}$ is formulated as:

$$\mathcal{L}_{intra} = \frac{1}{N^2} \sum_{i,j \in \{0,1 \ldots N-1\}, i \neq j} \frac{< c^i, c^j >}{\|c^i\|_2 \cdot \|c^j\|_2}, \tag{8}$$

where $c^i = \frac{\sum_{x_i \in C^i} f(x_i)}{|C^i|}$ is the center of C^i. Empirically, we discover that $\mathcal{L}_{intra}$ also helps to enlarge $d_{intra}(C_c^t, C_p^t)$, making it increase to 4.96. Subsequently, with the application of the SD

module, poisoned samples can be separated from clean samples (Fig. 8(i,l)). Note that $\mathcal{L}_{intra}$ can even boost the separation under dirty-label attacks (Fig. 8(g,h,j,k)). In conclusion, combined with the fine-tuning step, our proposed SD module can work under all kinds of attacks.



(a) BadNets (b) Blend (c) SIG (d) CL

Figure 7: T-SNE [4] visualization of feature space of models backdoored by different attacks on CIFAR-10. Target class is 0. The poisoned cluster is in black while other clean clusters are in non-black.

A.2 Distinguishment and secure training (D-ST) method

The D-ST method is shown in Algorithm 3.

Algorithm 3 Distinguishment and secure training (D-ST) method

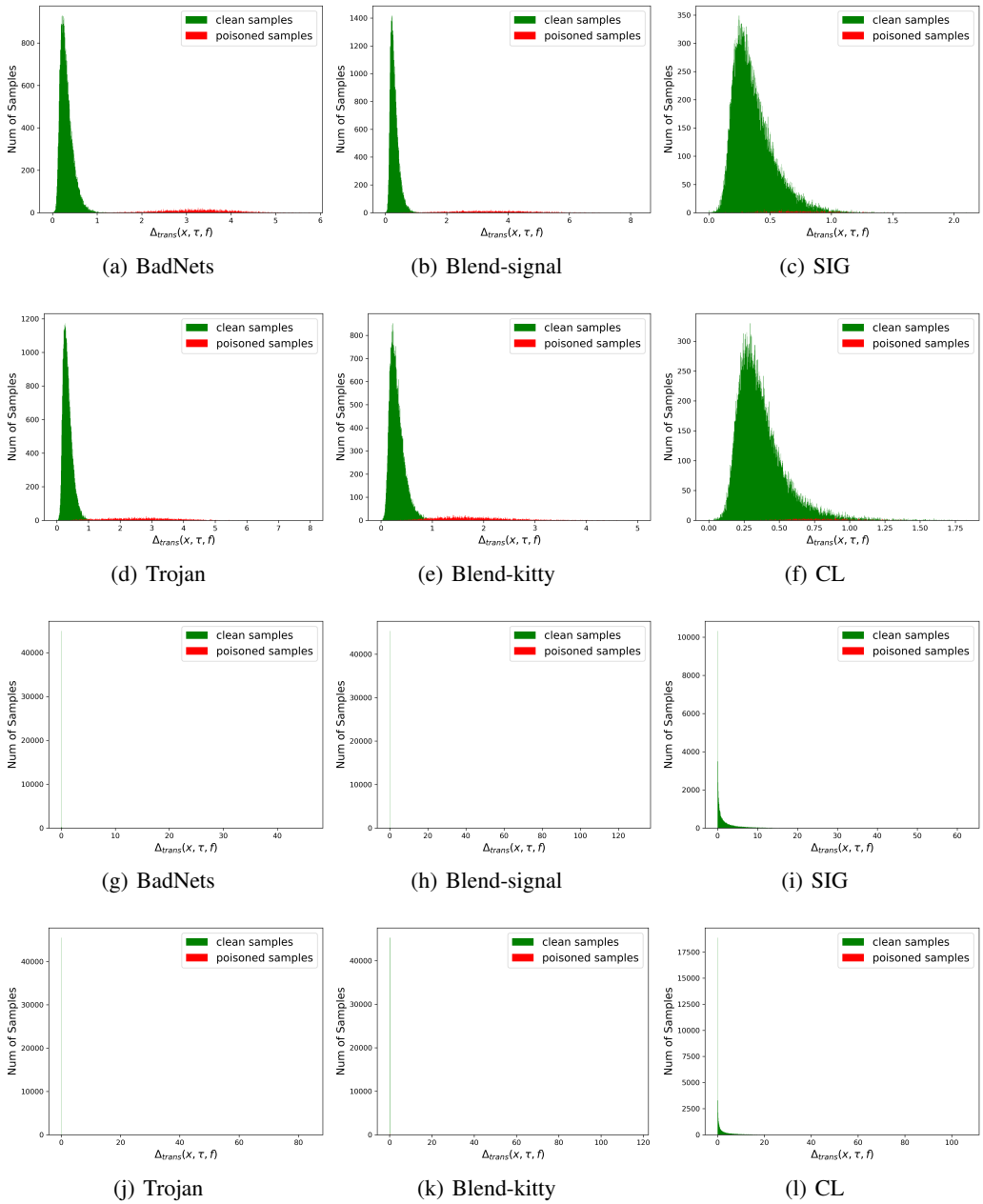
Input: The training set $\bar{D}_{train}$, number of training epochs for feature extractor e_e, number of training epochs for classifier e_c
Output: A secure model $g_{\boldsymbol{\theta}}$
 # Sample distinguishment module
Separate $\bar{D}_{train}$ to three subsets, *i.e.*, $\hat{D}_p$, $\hat{D}_c$, $\hat{D}_u$ (see Section 3.2)
 # Secure training module
Stage 1: learning the feature extractor $f_{\boldsymbol{\theta}_e}$ through SS-CTL, *i.e.*, minimizing $\mathcal{L}_{SS-CTL}$ on $\bar{D}_{train}$ (see Eq. (2)), for e_e epochs
Stage 2: learning the classifier $h_{\boldsymbol{\theta}_c}$ via minimizing the mixed cross-entropy loss, *i.e.*, minimizing $\mathcal{L}_{MCE}$ on $\bar{D}_c$ and $\bar{D}_p$ (see Eq. (3)), for e_c epochs
return $g_{\boldsymbol{\theta}} = h_{\boldsymbol{\theta}_c}(f_{\boldsymbol{\theta}_e}(\cdot))$

A.3 Distinguishment and backdoor removal (D-BR) method

The D-BR method is shown in Algorithm 4.

Algorithm 4 Distinguishment and backdoor removal (D-BR) method

Input: The training set $\bar{D}_{train}$, number of training epochs e_t, number of fine-tuning epochs e_{ft}
Output: A secure model $g_{\boldsymbol{\theta}}$
 # Sample distinguishment module
Separate $\bar{D}_{train}$ to three subsets, *i.e.*, $\hat{D}_p$, $\hat{D}_c$, $\hat{D}_u$ (see Section 3.2)
 # Backdoor removal module
Train a backdoored model $g_{\boldsymbol{\theta}}$ by minimizing the standard cross-entropy on $\bar{D}_{train}$ for e_t epochs
for each $e \in \{1, \dots, e_{ft}\}$ **do**
  Unlearning: Fine-tune $g_{\boldsymbol{\theta}}$ by minimizing $\mathcal{L}_{unlearn}$ on $\hat{D}_p$ for an epoch (see Eq. (4))
  Relearning: Fine-tune $g_{\boldsymbol{\theta}}$ by minimizing $\mathcal{L}_{relearn}$ on $\hat{D}_c$ for an epoch (see Eq. (5))
end for
return $g_{\boldsymbol{\theta}}$



(a) BadNets (b) Blend-signal (c) SIG

(d) Trojan (e) Blend-kitty (f) CL

(g) BadNets (h) Blend-signal (i) SIG

(j) Trojan (k) Blend-kitty (l) CL

Figure 8: Distribution of clean and poisoned samples with respect to the FCT metric on CIFAR-10. The top(bottom) two rows represent the distribution before(after) the step of fine-tuning. Note that after fine-tuning, clean and poisoned samples are too far apart to be seen clearly.

B More details on semi-supervised contrastive learning (SS-CTL)

We explain ℓ_{CTL} and ℓ_{S-CTL} in details here. For clarity, we denote $f_{\boldsymbol{\theta}_e}(\tilde{\boldsymbol{x}}_{\boldsymbol{i}}^{(\mathbf{1})})$ as f_i^1 and $f_{\boldsymbol{\theta}_e}(\tilde{\boldsymbol{x}}_{\boldsymbol{i}}^{(\mathbf{2})})$ as f_i^2.

$$\begin{aligned}\ell_{CTL}\big(f_{\boldsymbol{\theta}_e}(\tilde{\boldsymbol{x}}_{\boldsymbol{i}}^{(\mathbf{1})}), f_{\boldsymbol{\theta}_e}(\tilde{\boldsymbol{x}}_{\boldsymbol{i}}^{(\mathbf{2})})\big) = &-\log \frac{\exp\left(f_i^1 \cdot f_i^2/T\right)}{\sum_{k=1,2}\sum_{j=1}^{|\bar{D}_{train}|}\mathbb{I}(f_i^1 \neq f_j^k)\exp\left(f_i^1 \cdot f_j^k/T\right)} \\ &-\log \frac{\exp\left(f_i^2 \cdot f_i^1/T\right)}{\sum_{k=1,2}\sum_{j=1}^{|\bar{D}_{train}|}\mathbb{I}(f_i^2 \neq f_j^k)\exp\left(f_i^2 \cdot f_j^k/T\right)}.\end{aligned} \tag{9}$$

If $y_i = y_j$, then

$$\begin{aligned}&\ell_{S-CTL}\big(f_{\boldsymbol{\theta}_e}(\tilde{\boldsymbol{x}}_{\boldsymbol{i}}^{(\mathbf{1})}), f_{\boldsymbol{\theta}_e}(\tilde{\boldsymbol{x}}_{\boldsymbol{i}}^{(\mathbf{2})}), f_{\boldsymbol{\theta}_e}(\tilde{\boldsymbol{x}}_{\boldsymbol{j}}^{(\mathbf{1})}), f_{\boldsymbol{\theta}_e}(\tilde{\boldsymbol{x}}_{\boldsymbol{j}}^{(\mathbf{2})}); y_i, y_j\big) \\ &= -\frac{1}{3}\{\log \frac{\exp\left(f_i^1 \cdot f_i^2/T\right)}{\sum_{k=1,2}\sum_{j=1}^{|\bar{D}_{train}|}\mathbb{I}(f_i^1 \neq f_j^k)\exp\left(f_i^1 \cdot f_j^k/T\right)} \\ &\quad + \log \frac{\exp\left(f_i^1 \cdot f_j^1/T\right)}{\sum_{k=1,2}\sum_{j=1}^{|\bar{D}_{train}|}\mathbb{I}(f_i^1 \neq f_j^k)\exp\left(f_i^1 \cdot f_j^k/T\right)} \\ &\quad + \log \frac{\exp\left(f_i^1 \cdot f_j^2/T\right)}{\sum_{k=1,2}\sum_{j=1}^{|\bar{D}_{train}|}\mathbb{I}(f_i^1 \neq f_j^k)\exp\left(f_i^1 \cdot f_j^k/T\right)}\} \\ &\quad -\frac{1}{3}\{\log \frac{\exp\left(f_i^2 \cdot f_i^1/T\right)}{\sum_{k=1,2}\sum_{j=1}^{|\bar{D}_{train}|}\mathbb{I}(f_i^2 \neq f_j^k)\exp\left(f_i^2 \cdot f_j^k/T\right)} \\ &\quad + \log \frac{\exp\left(f_i^2 \cdot f_j^1/T\right)}{\sum_{k=1,2}\sum_{j=1}^{|\bar{D}_{train}|}\mathbb{I}(f_i^2 \neq f_j^k)\exp\left(f_i^2 \cdot f_j^k/T\right)} \\ &\quad + \log \frac{\exp\left(f_i^2 \cdot f_j^2/T\right)}{\sum_{k=1,2}\sum_{j=1}^{|\bar{D}_{train}|}\mathbb{I}(f_i^2 \neq f_j^k)\exp\left(f_i^2 \cdot f_j^k/T\right)}\} \\ &\quad -\frac{1}{3}\{\log \frac{\exp\left(f_j^1 \cdot f_i^1/T\right)}{\sum_{k=1,2}\sum_{j=1}^{|\bar{D}_{train}|}\mathbb{I}(f_j^1 \neq f_j^k)\exp\left(f_j^1 \cdot f_j^k/T\right)} \\ &\quad + \log \frac{\exp\left(f_j^1 \cdot f_i^2/T\right)}{\sum_{k=1,2}\sum_{j=1}^{|\bar{D}_{train}|}\mathbb{I}(f_j^1 \neq f_j^k)\exp\left(f_j^1 \cdot f_j^k/T\right)} \\ &\quad + \log \frac{\exp\left(f_j^1 \cdot f_j^2/T\right)}{\sum_{k=1,2}\sum_{j=1}^{|\bar{D}_{train}|}\mathbb{I}(f_j^1 \neq f_j^k)\exp\left(f_j^1 \cdot f_j^k/T\right)}\} \\ &\quad -\frac{1}{3}\{\log \frac{\exp\left(f_j^2 \cdot f_i^1/T\right)}{\sum_{k=1,2}\sum_{j=1}^{|\bar{D}_{train}|}\mathbb{I}(f_j^2 \neq f_j^k)\exp\left(f_j^2 \cdot f_j^k/T\right)} \\ &\quad + \log \frac{\exp\left(f_j^2 \cdot f_i^2/T\right)}{\sum_{k=1,2}\sum_{j=1}^{|\bar{D}_{train}|}\mathbb{I}(f_j^2 \neq f_j^k)\exp\left(f_j^2 \cdot f_j^k/T\right)} \\ &\quad + \log \frac{\exp\left(f_j^2 \cdot f_j^1/T\right)}{\sum_{k=1,2}\sum_{j=1}^{|\bar{D}_{train}|}\mathbb{I}(f_j^2 \neq f_j^k)\exp\left(f_j^2 \cdot f_j^k/T\right)}\}.\end{aligned} \tag{10}$$

Else,

$$\ell_{S-CTL}\big(f_{\boldsymbol{\theta}_e}(\tilde{\boldsymbol{x}}_{\boldsymbol{i}}^{(\mathbf{1})}), f_{\boldsymbol{\theta}_e}(\tilde{\boldsymbol{x}}_{\boldsymbol{i}}^{(\mathbf{2})}), f_{\boldsymbol{\theta}_e}(\tilde{\boldsymbol{x}}_{\boldsymbol{j}}^{(\mathbf{1})}), f_{\boldsymbol{\theta}_e}(\tilde{\boldsymbol{x}}_{\boldsymbol{j}}^{(\mathbf{2})}); y_i, y_j\big) = 0. \tag{11}$$

C More implementation details

All experiments are run on 3 RTX 5000 GPUs and are repeated over 5 runs with different random seeds.

C.1 Dataset details

We conduct experiments on three benchmark datasets, CIFAR-10 [3], CIFAR-100 [3] and an ImageNet subset [32]. We use the ImageNet subset provided by [9], which contains 200 classes with 100000 training samples (500 samples per class) and 10000 testing samples (50 samples per class).

C.2 Model details

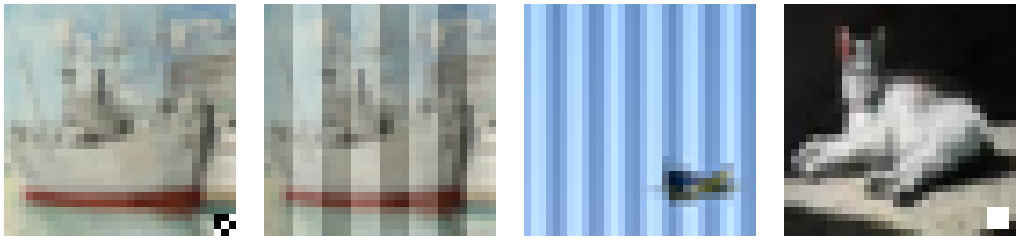
We conduct experiments with ResNet-18 [33] as base model. For the ST module, we leverage the model provided by [29] on CIFAR-10 and CIFAR-100. For the BR module, we use the model implemented by codes[2] on CIFAR-10, codes[3] on CIFAR-100 and [9] on the ImageNet subset.

C.3 Attack details

Referring to Section 2, we can categorize existing poisoning-based backdoor attacks according to various criteria. As shown in Table 4, we choose eight attacks to guarantee at least one method in each category. We also give some examples in Fig. 9 to display the patterns of different triggers.

Table 4: Categories of eight backdoor attacks based on different criteria.

Criterion	Size of trigger		Visibility of trigger		Variability of trigger		Label-consistency		Num of target classes	
	Patch	Blend	Visible	Invisible	Agnostic	Specific	Dirty	Clean	All2one	All2all
BadNets-all2one	✔		✔		✔		✔		✔	
BadNets-all2all	✔		✔		✔		✔			✔
Trojan	✔		✔		✔		✔		✔	
Blend-signal		✔	✔		✔		✔		✔	
Blend-kitty		✔	✔		✔		✔		✔	
SIG		✔	✔		✔			✔	✔	
CL	✔		✔		✔			✔	✔	
SSBA		✔		✔		✔	✔		✔	



(a) BadNets-all2one (b) Blend-signal (c) SIG (d) BadNets-all2all

(e) Trojan (f) Blend-kitty (g) CL (h) SSBA

Figure 9: Examples of poisoned samples attached with triggers crafted by different backdoor attacks.

Besides, we provide an explanation for the *poisoning rate* configured in attacks. Take CIFAR-10 (50000 training samples) as an example, if the poisoning rate is 10%, then 10% of samples with

[2] `https://github.com/huyvnphan/PyTorch_CIFAR10`

[3] `https://github.com/weiaicunzai/pytorch-cifar100`

non-target classes are poisoned in dirty-label (also all2one) attacks (4500 samples) while 10% of samples with target class are poisoned in clean-label (also all2one) attacks (500 samples).

C.4 Defense details

Here, we complement some details not mentioned in the main paper.

- FT: We fine-tune the backdoored model on 1% of the clean training samples with 100 epochs.
- MCR: We adopt t = 0.3.
- Baseline1: We first use self-supervised contrastive learning, SimCLR [30] particularly, to train the feature extractor. During the process, data is regarded as unlabeled. Thus, the *positive* of a sample is its augmented version. We then use standard supervised learning to train the classifier on the poisoned dataset.
- Baseline2: We first use supervised contrastive learning, SupContrast [29] particularly, to train the feature extractor. During the process, the *positives* of a sample are its augmented version and samples with the same label (and also their augmented versions). Note that since there are poisoned samples in the dataset, 'samples with the same label' may not be the genuine intra-class samples. We then use standard supervised learning to train the classifier on the poisoned dataset.

C.5 Details of proposed method

Choices of parameters. Here, we complement some details not mentioned in the main paper.

- Algorithm 2: τ =rotate+affine, $e_t = 2$, $e_f = 10$, $\alpha_c = 0.20$, $\alpha_p = 0.05$. No other data augmentations are used since they would hinder the effect of backdoor implantation.
- Algorithm 3: $e_e = 200$, $e_c = 10$, $\lambda_p = 0.001$. Applied data augmentations are the same as [30].
- Algorithm 4: $e_t = 200$, $e_{ft} = 20$. Data augmentations applied on CIFAR-10 are random crop, random horizontal flip and normalization. Data augmentations applied on CIFAR-100 are random crop, random horizontal flip, random rotation and normalization. Data augmentations applied on ImageNet are random rotation, random horizontal flip and normalization.

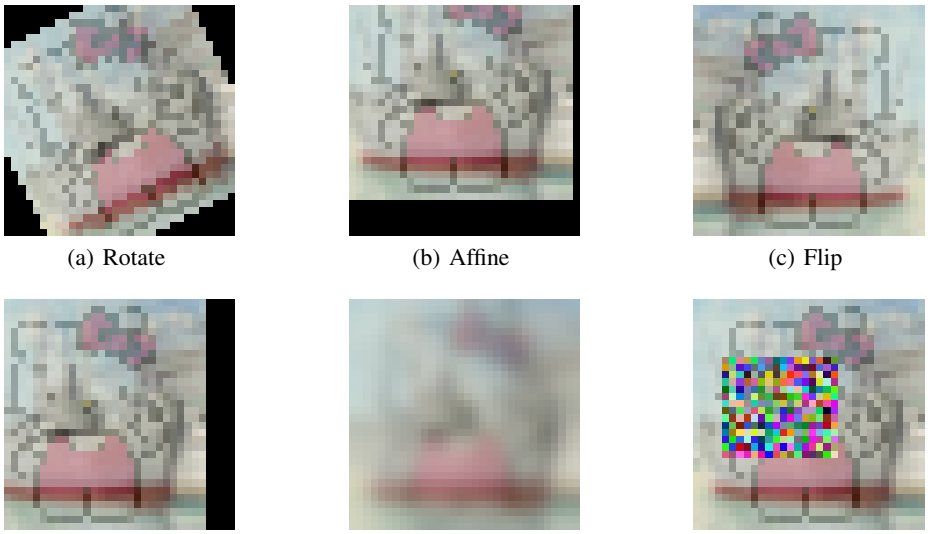
Choices of different data transformations τ. In Section 4.3, we select six types of τ to change the pattern or location of the trigger, which are listed as follows and shown in Fig. 10.

- Rotate: Rotate an image by a random degree less than $180°$.
- Affine: Translate an image horizontally and vertically.
- Flip: Flip an image horizontally.
- Crop: Crop an image at a random location.
- Blur: Perform Gaussian blurring on an image by given kernel.
- Erase: Select a random rectangle region in an image and erase its pixels.

D Results on ImageNet

Here, we exhibit the performance of our proposed methods on the ImageNet subset [32, 9] in Table 5. Since we aim to study successful backdoor attacks in this paper where ASR of the backdoored model is higher than 85%, results on unsuccessful attacks are not reported. For instance, the increase of number of classes adds to the difficulty of clean-label attacks (ASR < 5%). So, we do not report results on clean-label attacks.

Performance of the D-ST method. Note that SupCon [29] does not launch open-souce codes for ImageNet, so the related results are not reported here.



(a) Rotate (b) Affine (c) Flip

(d) Crop (e) Blur (f) Erase

Figure 10: Examples of six data transformations.

Performance of the D-BR method. We do not report results of ANP and MCR on ImageNet since they require particular modification to model and do not have an open-source version for ImageNet. As illustrated in Table 5, among the selected methods, ABL performs the best with ACC as 80.16% and ASR as 23.94% on average. However, it still fails on Blend-Strip attack (ASR = 95.75%). In contrast, D-BR performs steadily well on all attacks, even reducing ASR to 0% in most cases, which demonstrates the effectiveness of the SD module and the BR module.

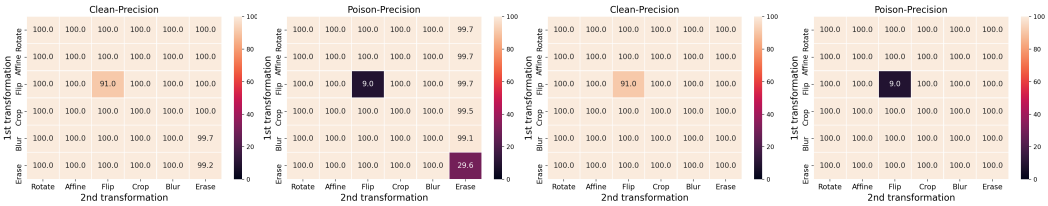
Table 5: Comparisons of D-BR with 3 backdoor-removal defense methods on ImageNet subset. The best result (with the largest ACC-ASR) is denoted in boldface.

ImageNet	Backdoored		FT*		NAD*		ABL		D-BR	
Attack	ACC	ASR	ACC	ASR	ACC	ASR	ACC	ASR	ACC	ASR
BadNets-all2one	84.72	95.80	82.20	56.66	63.07	0.41	82.72	0.00	**83.66**	**0.00**
Blend-Strip	84.36	97.64	82.35	78.82	59.66	16.19	79.71	95.74	**80.40**	**0.00**
Blend-Kitty	85.46	99.68	82.63	99.60	63.78	98.21	77.10	0.00	**84.29**	**0.00**
SSBA	85.24	99.64	82.35	97.63	63.82	34.62	81.10	0.00	**83.77**	**0.09**
Avg	84.95	98.19	82.38	83.18	62.58	37.36	80.16	23.94	**83.03**	**0.02**

E Performance with different data transformations

Here, we study how τ affects the performance of the SD module, specifically the precision of $\hat{D}_c$ (clean-precision) and $\hat{D}_p$ (poison-precision). We select six types of τ which change the pattern or location of trigger, detailed in Appendix C.5. In order to strengthen the perturbation to trigger, we adopt a combined τ, combining two of the six choices. The total 36 combinations are applied on the poisoned training set and the precision is illustrated in Fig. 11 and Appendix E.

Apparently, the lighter the color, the higher the precision. One worse case is applying 'flip' twice, equal to not using any transformation. Hence, genuinely poisoned samples can not be identified since trigger is not perturbed, and the poison-precision is extremely low. Another slightly worse case is applying 'erase' as the second transformation. We infer that adding a noise patch on the random position of an image, 'erase' may fail to obscure the trigger. In general, the SD module is highly effective in identifying genuinely clean and poisoned samples under most combined τ with 100% precision, also validating the sensitivity of poisoned samples to transformations. Note that the 100% clean-precision does not mean all distinguished clean samples being genuinely clean. In fact,



(a) BadNets (b) BadNets (c) Blend (d) Blend

Figure 11: Clean(poison)-precision of $\hat{D}_c$ and $\hat{D}_p$ under different τ. Samples are from CIFAR-10.

we discover that there are several genuinely poisoned samples, like five, in the distinguished clean samples. These tiny amount of poisoned samples are not reflected on the clean-precision due to the round-off.

Additionally, we show more results in Fig. 12. As a clean-label attack, SIG has high clean-precision but low poison-precision, which is different from BadNets and Blend attacks. We infer that the difference comes from the definition of the attack. Although we set a uniform poisoning rate (10%) for all attacks, the total number of poisoned samples is various. Take CIFAR-10 as an example where there are 50000 training samples (5000 samples per class). In a dirty-label (also all2one) attack, excluding the 5000 samples from target class, there are 4500 poisoned samples totally. By contrast, in a clean-label (also all2one) attack, since the attacker only targets at samples from target class, there are only 500 poisoned samples. Recalling $\alpha_p = 5\%$, we separate 2500 samples as poisoned samples, which are much more than the actual 500 poisoned samples. Therefore, the poison-precision under a clean-label attack is quite low.

Nevertheless, the seemingly adverse fact actually helps to validate one of the benefits of our proposed defense modules (the ST module and the BR module), which is the *robustness to the wrong distinguishment in the SD module*. Specifically, even with a low poison-precision, the proposed two defense methods, *i.e.*D-ST (SD module + ST module) and D-BR (SD module + BR module), are still effective in defending against clean-label attacks, which demonstrates that the two defense modules are robust to the inaccurate distinguishment in the SD module. The robustness probably attributes to the high clean-precision (> 99%). It could help our proposed methods 'survive' under extreme circumstances where there are only tiny amount of poisoned samples, further proved in Appendix G empirically. Hence, despite some newly introduced attacks focusing on reducing the poisoning rate to add to the difficulty in defense, our method is still capable of coping with these potential attacks.

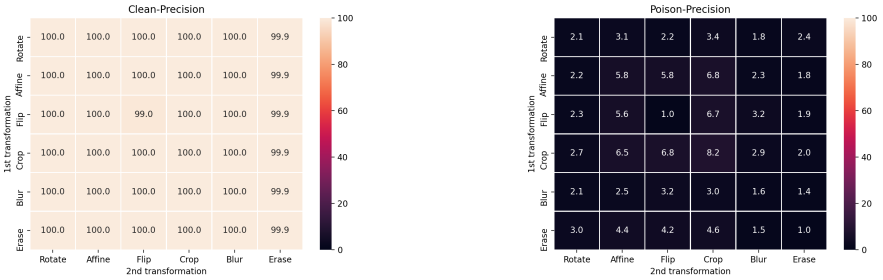


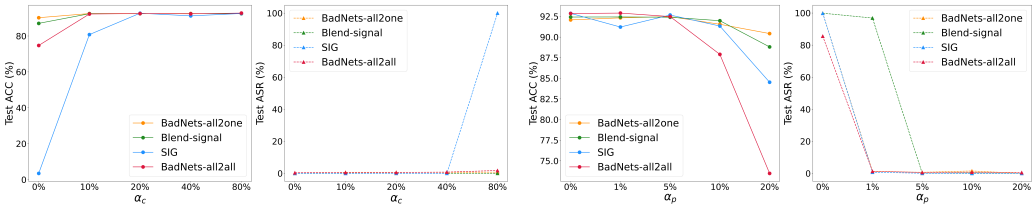
Figure 12: Clean(poison)-precision of the separated clean(poisoned) samples identified by our proposed FCT-based SD module under different τ. Samples are from CIFAR-10 poisoned by SIG attack.

F Performance with different proportion values

In this part, we explore how α_c and α_p, specified in the SD module, influence the final defense performance. To this end, we first conduct the SD module with different α_c and α_p, and then test the performance under the backdoor-removal paradigm for illustration. Recall the settings in the previous experiments where $\alpha_c = 20\%$ and $\alpha_p = 5\%$.

First, we fix α_p as 5% and vary α_c from 0% to 80%. Results are illustrated in Fig. 13 (a). Comparing $\alpha_c = 0$ and others, we see that $\hat{D}_c$ plays an important role in boosting ACC. As shown in the left figure, the model may suffer from extremely low ACC without these samples. As α_c grows, ACC increases steadily and finally converges. However, when α_c is too large (eg. 80%), D_c may contain genuinely poisoned samples, resulting in the rise of ASR, as shown in the right figure. Hence, 20% and 40% are appropriate choices for α_c.

Then, we fix α_c as 20% and alter α_p from 0% to 20%. Results are depicted in Fig. 13 (b). The comparisons between $\alpha_p = 0$ and others validate the effect of $\hat{D}_p$ on decreasing ASR. Specifically, as shown in the right figure, ASR could be extremely high if none of these samples are available. While with the increase of α_p, ASR declines steadily and finally converges. Nevertheless, excessive $\hat{D}_p$ could hurt the model and lead to a reduction in ACC, as shown in the left figure. We infer that the excessive samples may include a certain amount of genuinely clean samples and unlearning which is deleterious to the model. Therefore, a moderate α_p (eg. 5%) is preferred.



(a) Fixed α_p = 5% (b) Fixed α_c = 20%

Figure 13: Test ACC and Test ASR of the D-BR method with different α_p and α_c settings in the SD module, on CIFAR-10 poisoned by different attacks. α_p and α_c are fixed separately.

G Performance with different poisoning rates

In this section, we aim to show performance of our proposed methods, namely D-ST and D-BR, under different poisoning rates. To this end, we set the poisoning rate as 5% and 20% respectively, and conduct the methods on CIFAR-10 poisoned by different attacks. Here, we choose four representative attacks for illustration. Note that we keep $\alpha_c = 20\%$ and $\alpha_p = 5\%$ as before. The results are shown in Table 6.

Table 6: Performance of our proposed methods on CIFAR-10 poisoned under different poisoning rates.

Poisoning rate	5%						20%					
CIFAR-10	Backdoored		D-BR		D-ST		Backdoored		D-BR		D-ST	
Attack	ACC	ASR	ACC	ASR	ACC	ASR	ACC	ASR	ACC	ASR	ACC	ASR
BadNets-all2one	91.31	99.98	87.52	4.22	89.38	9.44	89.06	99.99	90.32	0.96	91.48	0.13
BadNets-all2all	91.67	83.23	91.71	0.78	89.84	1.42	91.02	87.01	91.41	0.60	89.98	1.29
Blend-Strip	91.70	99.99	89.07	2.49	89.56	0.00	90.78	100.00	90.11	7.42	90.72	0.00
SIG	92.00	93.26	87.9	0.75	89.56	0.63	92.00	99.87	90.76	0.44	89.13	0.19
Avg	91.67	94.11	89.05	2.06	89.59	2.87	90.71	96.72	90.65	2.36	90.33	0.40

Under the poisoning rate of 20%, D-BR can reduce the average ASR from 96.72% to 2.36%, while keep ACC barely unchanged. Meanwhile, D-ST can train a clean model from scratch with ACC as

90.33% and ASR as 0.40% on average. When the poisoning rate is as small as 5%, D-BR can still manage to decrease ASR by 92.05% on average with a mere drop in ACC (2.62% on average). The model trained with D-ST is relatively clean with ACC as 89.59% and ASR as 2.87% on average. The performance is slightly worse than that of 20%. We infer that in this case, poisoning rate and α_p both equal to 5%. Since we can not find all poisoned samples accurately, the poisoned samples separated by the proposed SD module definitely contain some extent of genuinely clean samples, which is harmful to the performance of the final model. Hence, better performance can be achieved with a careful adjustment in α_c and α_p.

H Complexity of the proposed two methods

Denote the complexity of one forward and backward pass in feature extractor as a_e while that in classifier as a_c.

Complexity of D-ST. Given the training epochs detailed in Appendix C.5, the complexity is $\mathcal{O}\big(e_e \cdot a_e \cdot [2 \cdot (|\hat{D}_p| + |\hat{D}_u|) + 12 \cdot |\hat{D}_c|^2]\big)$ and $\mathcal{O}\big(e_c \cdot a_c \cdot [|\hat{D}_c| + |\hat{D}_p|]\big)$ for stage 1 and stage 2, respectively. We adopt $e_e = 200$, $e_c = 10$ in the experiments.

Complexity of D-BR. The complexity of training a backdoored model is $\mathcal{O}\big(e_t \cdot (a_e + a_c) \cdot |\bar{D}_{train}|\big)$. The complexity of the BR module is $\mathcal{O}\big(e_{ft} \cdot (a_e + a_c) \cdot [|\hat{D}_c| + |\hat{D}_p|]\big)$. We adopt $e_t = 200$, $e_{ft} = 20$ in the experiments.

Take CIFAR-10 as an example. $|\bar{D}_{train}| = 50000$. $|\hat{D}_c|$ and $|\hat{D}_p|$ are approximately 10000 and 2500, respectively. Compared with a_e, a_c is relatively small and can be omitted. Thus, the complexity of D-ST is higher than that of D-BR by about $\mathcal{O}\big(12 \cdot a_e \cdot |\hat{D}_c|^2\big)$.