A1 More Methodology Details

A1.1 Classic Trigger Injection

[14] defined a generic form of trigger injection. Let $\boldsymbol{x}_i$ be an image sample and Δ denote a trigger pattern. The trigger is *stamped* on the image sample at a specific region characterized by a binary mask $\mathtt{M}$. We say the resulting image is a backdoor image, denoted by $\boldsymbol{x}_i'$. Formally, the injection process is formulated as follows, where $\odot$ is the point-wise multiplication operator:

$$\boldsymbol{x}' = \boldsymbol{x} \odot (\mathbf{1} - \mathtt{M}) + \mathtt{M} \odot \Delta \tag{2}$$

We generate 5×5 gray-scale triggers [1] or RGB triggers [7] and place them at the lower right corner of 10% of training samples to get backdoored datasets. The trigger patterns are provided in Figure A10

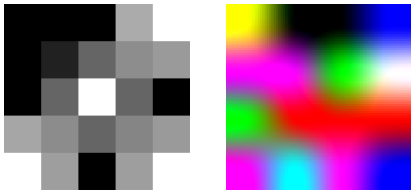

Figure A10: Trigger pattern used to form backdoor dataset. Note that the provided pattern will be resized according to the pre-defined patch size.

A1.2 Classic Trigger Recovery

As is mentioned in [14], trigger recovery is formulated as solving the following optimization problem:

$$\underset{\mathbf{M},\Delta}{\operatorname{argmin}} \sum_{\mathrm{i=1}}^{n} \mathcal{L}\left(\boldsymbol{x}_i, \mathtt{M}, \Delta\right) + \lambda \cdot |\mathbf{M}|, \tag{3}$$

where $|\mathbf{M}|$ is to regularize overly larger triggers.

Classic trigger recovery methods solely use the original model $\mathcal{G}$, and $\mathcal{L}\left(\boldsymbol{x}_i, \mathtt{M}, \Delta\right)$ is formulated as:

$$\mathcal{L}\left(\boldsymbol{x}_i, \mathtt{M}, \Delta\right) = \mathcal{L}_{\mathrm{XE}}(\mathcal{G}(\boldsymbol{x}'), y_t) \tag{4}$$

A2 More Experimental Details

Experiment details are provided in table A7. We perform experiments on 8 2080Ti GPUs.

Table A7: Detailed configurations of detection methods.

Method	# Epoch	Learning Rate	Optimizer	Batch Size	Num. of Trainset	Num. of Cleanset	Num. of Layer	Rounds
NC [14]	200	0.1	SGD	128	-	1024	-	-
STRIP [18]	-	-	-	100	1000	1000	-	10
Ours(CIFAR-10)	-	-	-	128	1024	-	4	-
Ours(GTSRB)	-	-	-	128	4096	-	4	-

A3 More Experimental Results

A3.1 Using Mean of Representation Shift for Detection

Instead of using the standard deviation of representation shift as depicted in Section 3.2, we will show that using the mean of representation shift for backdoor detection is also a plausible choice. Specifically, $y^k(n) = \mathtt{mean}_{\boldsymbol{x}_i \in \mathcal{D}_k}\{s_n(\boldsymbol{x}_i)\}$. Other steps remain the same.

To verify the effectiveness, we use the same setting as that in Section 4.2. Detection results are provided in Table A8, which shows using the mean of representation shift for detection can achieve comparable results as using the standard deviation of representation shift.

Table A8: The detection result when using mean of representation shift for detection. The reported numbers represent (*# correctly detected models*) / (*# models*). Note that "'CLA' stands for the clean label attack.

Attack	Dataset	Arch	Ours(Mean)		
			benign	backdoor	AUROC
BadNets	CIFAR-10	ResNet-20	25/25	23/25	0.981
		AlexNet	21/25	20/25	0.880
	GTSRB	ResNet-20	25/25	18/25	0.910
CLA	CIFAR-10	ResNet-20	25/25	22/25	0.976
WaNet	CIFAR-10	ResNet-18	25/25	24/25	0.998
	Total		121/125	107/125	-

A3.2 Comparison with prior-training methods

The more classical backdoor detection setting as adopted in [30, 16, 31] perform *pre-training* detection of backdoor training samples, which requires (re-)training from scratch. They do not need clean validation sets, which is similar to our setting. In contrast, our method belongs to *post-training* detection that can detect from a given pre-trained model without costly (re-)training. For dataset CIFAR-10 with ResNet-20, as is shown in Table A9, Deep-KNN [16] needs 1.47×10^4 times more FLOPs than ours, which makes our approach more practical when the model is already actively deployed in the field. We further compare the proposed method with Deep-KNN[16], which is a more recent work in this thread. As shown in Table A9, the AUROC of our method is significantly higher than Deep-KNN, showing the superior effectiveness of our approach.

Note that similar to [18], [16] are instance-based methods that cannot directly apply to attack detection settings. Following the method in [3], we classify the dataset with any input that is identified poisoned by the instance-based method as a backdoor dataset.

Table A9: Comparison on effectiveness and efficiency with Deep-KNN on BadNets, CIFAR-10, and ResNet-20.

Attack	Dataset	Arch	Method	FLOPs($\times 10^{10}$)	AUROC
BadNets	CIFAR-10	ResNet-20	KNN	9.85×10^4	0.495
			Ours	6.72	1.000

A4 Limitations and Discussions

In this work, following the mainstream setting[14, 18, 20, 11, 13, 15, 17], we only consider backdoor attacks when target classes are the minority. Several current works[19] consider another setting where all of the classes are infected, which is known as the "all-to-all" attack. In addition, we did not discuss about backdoor attack to modern architectures like vision-transformer [48] since only a few recent work, e.g., [49], consider this case. And we also did not show our effectiveness on real-world data. We will explore these in future work.

A5 Boarder Impact

We do not believe that this research has any negative social impact. Defending machine learning models against backdoor attacks is crucial toward the goal of making neural networks more trustworthy and reliable. Our proposed method can be a powerful tool to help users identify the existence of the backdoor in an unreliable dataset, ensuring the trustworthiness of generated models.