Randomized Channel Shuffling: Minimal-Overhead Backdoor Attack Detection without Clean Datasets

Anonymous Author(s)
Affiliation
Address
email

Abstract

Deep neural networks (DNNs) typically requires massive data to train on, which is a hurdle for numerous practical domains. Facing the data shortfall, one viable option is to acquire domains-specific training data from external uncensored sources, such as the open web or the third-party data collectors. However, the quality of such acquired data is often not rigorously scrutinized, and one cannot rule out the risk of "poisoned" examples being included in such unreliable datasets. That casts a **new "self-supervised" backdoor detection setting** that was hardly explored by prior arts: in particular, for one specific collected dataset, users (1) have no prior knowledge whether it is poisoned, or on the target class/percentage of poisoned samples, and (2) have no access to a clean sample set from the same domain distribution, nor any trusted model trained on such clean data. This new backdoor threat hence incurs overlooked risks for many data-hungry, or high-stake applications. This paper reports **the first pilot study** on this new setting, by investigating the contrasting channel-level statistics between backdoor trigger and clean features, and consequently, how the former can be differentiated by progressive channel shuffling. The method leads to detecting the backdoor-targeted class with only a few feedforward passes, incurring minimal overheads, and demanding no clean sample nor prior knowledge. Extensive experiments are conducted with two datasets (CIFAR-10, GTSRB), two architectures (AlexNet, ResNet-20), and three strong attacks (BadNets[1], clean label attack [2], and WaNet [3]). Results consistently endorse the effectiveness of our technique in backdoor model detection, with margins of $0.291 \sim 0.640$ AUROC[1] over the current state-of-the-art methods. Codes will be released.

1 Introduction

Numerous deep learning based methods significantly outperform traditional methods and become useful in many critical domains. To achieve good results, the enormous deep neural networks (DNNs) require a large amount of training data. Due to the *absence of large enough self-gathered datasets* in many applications, many users are forced to use data samples from external sources, e.g., third-party data collection companies and online websites. However, these external sources are untrustworthy because the companies or the uploaders could be attackers who inject "poison" into the datasets by maliciously crafting the data samples, when they deliver or upload the datasets.

Recently, such poisoning attacks are found fatal specifically to deep learning applications [4, 5]. Even worse, it is hard to determine whether a DNN or the dataset that it is trained on is attacked or not, due to the lack of transparency and interpretability. Users are left in desperate needs for a reliable method that can achieve such a goal, which we call *attack detection*. While efforts from

[1] *AUROC* stands for the area under the receiver operating characteristic curve.

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

researchers have been put into this direction, we find that most existing work of attack detection try to obtain a classifier that predicts if a given data sample is *poisoned*. This classifier is trained on a human-craft dataset containing both poisoned and clean samples, with the labels being whether a sample is poisoned. This approach assumes the access to at least a small portion of *clean samples* as reference [6, 7, 8, 9, 10, 11]. Unfortunately, this assumption does not always hold in real-world scenarios. We lay out three-fold reasons:

- Users have zero control nor prior knowledge over their received or collected datasets in the wild. Of course, the users may not know whether such a dataset is attacked or not. Beyond that, they also do not know (even approximately) the portion of samples that are maliciously crafted, nor can they reliably draw a small set of clean samples out of the dataset. This directly challenges most of existing work for attack detection.
- Public datasets online are not necessarily helpful, because they can be totally different from the target datasets (potentially attacked), in terms of content and statistics. Samples in the public datasets may even be out-of-distribution samples, and therefore provide little meaningful information for the users to tell the if their specific datasets are attacked.
- Labeling a small clean set with human labor is also not a feasible fix. On one hand, the "backdoor" changes made to to data samples are often hard to perceive or notice by human, compared to other benign image artifacts. On the other hand, the attacked DNNs behave completely normal on the clean samples. It is thus difficult to distinguish clean samples from the attacked ones by simply investigating superficial behaviors such as model performance and testing accuracy, especially when the testing set is likely to be attacked too.

In view of those, this work strives to derive a new attack detection method for in a **brand-new setting**, i.e., a "self-supervised" post-training attack detection scenario, where we assume no prior knowledge over the datasets of interest, nor the access to a collection of clean samples, nor the need to retrain models. We propose novel hypotheses about the statistics of the exploited features and how they behave differently from normal data features. Based on these hypotheses, we design a shuffling-based method that stratifies attacked and clean samples according to their different behaviors towards the shuffling operation. The validity of the hypotheses and the efficacy of our proposed method are supported by extensive empirical observations.

1.1 Background and Our Contributions

We focus ourselves on the *backdoor attack*, which is notably stealthy among various poisoning attacks. Backdoor attack methods poison data samples by injecting small triggers into and applying certain transformations to them. The triggers and transformations are usually unnoticeable to human perception. The poisoned dataset is called a *backdoored* dataset. A backdoored dataset usually contains both poisoned samples and *benign* or clean samples that are not contaminated. Traditional backdoor attackers will also maliciously label the poisoned data into a target class [1, 4, 12, 3], which corresponds to scenarios where the attackers have full control over the datasets. More recent work [2, 13] does not require this control and only craft the raw data. For both types of backdoor attack methods, the *backdoored model*, i.e., the DNN that is trained on the backdoored dataset, will behave normally on benign inputs while classifying most malicious inputs into chosen target classes.

Many detection methods have been proposed to tackle the backdoor attack [14, 15, 16, 10, 17, 9, 6, 7, 18, 19, 8, 20, 21, 19, 11]. However, most of them assume the access to a hold-out dataset only containing benign samples. A common practice by previous work is to learn a predictor of a certain form on this assumed training set to distinguish between benign and backdoored samples. However, these methods totally fail in the scenarios we discussed above, which are much more realistic as users usually have little explicit control over the cleaniness of datasets that they can receive or collect.

In this work, we propose that it is still possible to detect backdoor attacks without access to such a training set but let the backdoor samples reveal themselves. Laid on the foundation of this work is our core hypotheses over the unique channel-level distribution of the features of backdoored samples.

Trigger Feature Hypothesis We hypothesize that *the trigger features are sparsely encoded in only a few channels, while clean image features need to be encoded across many channels for the effective classification*. This is a key difference from normal data features that are presumably distributed more evenly across channels, which indicates that these two types of features might behave differently in certain situations, leading to our main technical contribution. More illustrations of trigger features are provided in App. A3.

Feature Differentiation Based on the the proposed hypotheses, we devise a novel randomized channel shuffling mechanism which introduces minimal computation cost to reveal trigger features. Specifically, given an unreliable dataset and a model trained by it, we control the channel shuffling strength and observe how representations vary according to it. By observing the abnormally different variation, we can detect the target class with only a few feedforward passes.

In addition, we further utilize the proposed mechanism to separate those clean image features and trigger features. Specifically, we retrain the model using the channel shuffling operation, and then recover triggers based on the shuffle model, obtaining more effective trigger than [6].

We summarize our novel contributions as follows:

- We raise that the unreliability of training data source has become a serious security threat. We find that clean image features and trigger features are different in terms of their activation distribution in channels, as well as their sensitivity to channel re-ordering.
- Utilizing trigger features' unique characteristic, we propose an effective, generalizable approach with minimal overhead based on channel shuffling to help data users examining datasets. We randomly shuffle weights of convolutional layers at channel level, revealing the secretly hidden trigger features. We also use the proposed shuffling method to retrain the models, helping generate more effective triggers.
- On two datasets (CIFAR-10, GTSRB), two architectures (AlexNet, ResNet-20), and three strong attacks (BadNets[1], clean label attack [2], and WaNet [3]), experiment results consistently endorse the effectiveness of our technique in backdoor model detection, e.g., $0.331 \sim 0.640$ higher AUROC than NC [6], and $0.291 \sim 0.543$ higher than STRIP [8].

2 Related Work

Backdoor Attack and Defense. Several backdoor attacks on deep learning models have been proposed recently. The attack features stealthiness since the attacked model will behave normally on clean images, but classify maliciously crafted images from any source class into the maliciously chosen target class. Among them, most backdoor attacks can be classified as trigger-driven attacks. Specifically, during the training phase, to launch an attack, the adversary injects an arbitrary trigger into a small fraction of training pictures and maliciously label them to the target class. Traditional triggers are a pre-defined pattern which can simply stamp on benign images [1, 4]. Many advanced triggers injection methods are proposed recently [12, 3] for stealthiness under both human inspection and former detection methods [6, 7, 8]. Another category of backdoor attack is clean-label attack [2, 22, 13, 23, 24, 25, 26], which was formulated without control over the labeling function. Among them, [23] targeted on large scale dataset and scale up the poisoning attack method via gradient matching. Additionally, both [22] and [24] focused on improving the transferability of attack methods, while [25] utilized meta-learning and proposed an effective and transferable poisoning methods.

Earlier works detect and mitigate backdoor models based on abnormal neuron responses [14, 16, 19], feature representation[15], entropy [8], and evolution of model accuracy [17]. [20] model the distribution of Trojan attacks by meta neural analysis, [18] utilized Deep k-NN with focus on clean-label attack, [21] tried to learn the Universal Litmus Patterns (ULPs) from poisoned dataset.

Another category of work [6, 7, 19, 9, 10, 11] focus on recovering the trigger and detect backdoor models based on the properties of the recovered triggers. Utilizing clean testing images, Neural Cleanse (NC) [6, 7] calculates minimal perturbation that causes misclassification towards every putative incorrect label. Backdoor model detection is completed by MAD outlier detection, due to the observation that the size of minimal perturbation to classify images from all classes to the target class is markedly smaller than that to other classes. However, the aforementioned methods suffer from occasional failures in detecting backdoor models. In addition, most methods except [19] that fall in this category generally assume the defender has a clean validation set, which may not be practical. Instead, we do not assume we can obtain a clean validation set for effective detection.

We also clarify the difference between our new setting, and the more classical backdoor detection setting as adopted in [27, 18, 28]. Those prior arts perform *pre-training* detection of backdoor training samples, without needing clean validation sets either. They all follow the same workflow: first detecting the poison examples in the training set, then removing them prior to training, and finally training the model using the filtered set. As the downside, to find out whether a pre-trained model has been poisoned or not, those methods also have to (re)train from scratch, which may often

be too costly or impractical. In contrast, [8] and our method belong to *post-training* detection that can detect from a given pre-trained model without costly (re-)training from scratch. That is more practical when the model is already actively deployed in the field. While [8] needs a clean "reference" set, our method requires no prior-training sample selection and (re-)training (hence differing from [27, 18, 28]), while eliminating the necessity of clean samples (hence differing from [8]).

Channel Shuffling. The idea of channel shuffling has mostly been used in efficient model design, intending to enable information flow between branches and consequently strengthen representation. Specifically, aiming at high efficiency, [29] and [30] utilized channel shuffling method to mitigate the problem induced by group convolution[31, 32]. Similarly, [33] included channel split and shuffle method to reduce computation cost while maintaining higher segmentation accuracy. [34] included channel shuffling to enable information communication between different sub-features. Different from aforementioned works, [35] adopted channel shuffling method with randomness as a regularization technique, in order to reduce overfitting and enhance generalization capabilities.

Based on our observation that clean image features and trigger features have different channel distributions which is shown in Section 3.1, we formulated the randomized channel shuffling mechanism to reveal trigger features, detecting backdoor models. To the best of our knowledge, we are the first to connect channel shuffling and feature differentiation goal.

Shortcut Features. Several recent works [36, 37] analyzed that trigger features can serve as "shortcut" features [5]. Such shortcut features are observed to be simpler and easier to learn than the discriminative clean image features [38]. In addition, [16] observed that a small number of neurons are activated by backdoor images but remain dormant for clean images. In contrast, [39] demonstrated the importance of the combination of channels when encoding complex and diversified image features on large datasets.

3 Methodology

In Section 3.1, we will first present our main hypothesis on trigger features' unique characteristics compared to clean image features. In Section 3.2, we propose our backdoor model detection method using randomized channel shuffling mechanism, directly motivated by the hypothesis. In Section 3.3 we further show that we can improve trigger recovery with the proposed channels shuffling scheme.

3.1 Differentiating Trigger and Image Features: A Hypothesis

Authors of [5] pointed out that DNNs tend to unintendedly exploit features *unrelated* to the task of interest to achieve good performance. These "shortcut" features are usually *simpler* and *easier* to learn [38] but highly correlated to the task-related features, such as sky background and the bird objective. It is then verified in [36, 37] that the backdoor triggers play a similar role as shortcuts exploited by the backdoored models.

Moreover, a small channels are observed to remain dormant for clean images but are activated by backdoor images [1, 16]. Considering all these observations, we conjecture that the trigger features have a key difference from clean images features in terms of channel-level activation, and raise the following **hypothesis**: *Trigger features are sparsely encoded and activated in only a few channels, while clean image features need to be encoded across many channels for the effective classification.*

To validate the hypothesis, we investigate the *activation level* of a specific channel, defined as the average of elements in its (output) feature map. In each layer, we consider channels with activation levels larger than a ratio p of the highest activation level as 'activated'. We then observe how channels are activated differently for clean and backdoor images. In Figure 1, we take the second last layer in a model backdoored by BadNets [1] as example. When the input is backdoored, only one single channel shows exceptionally high activation level. In contrast, a clean input has more balanced activations across all channels. In Figure 2, we have consistent observations on more advanced backdoor methods such as clean label attack [2] and transformation-based attack, WaNet [3]. Moreover, we can see that the sparse encoding of trigger

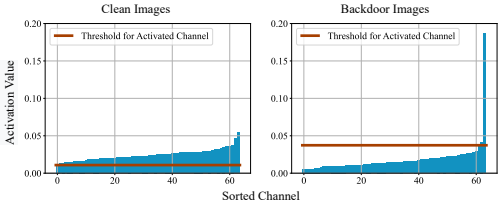

Figure 1: Blue columns represent the detailed distribution of activation values of *the second last layer*. Red lines indicate threshold for activated channels, 20% of max activation value in this figure.

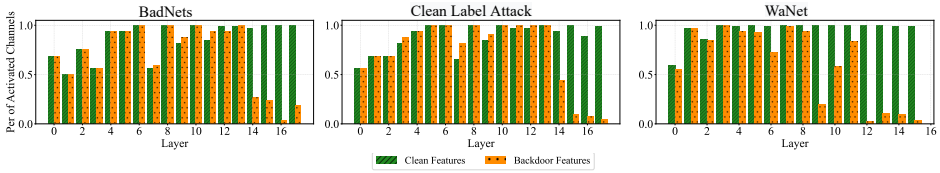

Figure 2: Green and orange columns represent the percentages of activated channels in each layer when inputting clean images and backdoor images, respectively. We empirically setting the $p = 20\%$ of max activation magtitude as the threshold, and considers channels with higher activation values than threshold as activated.

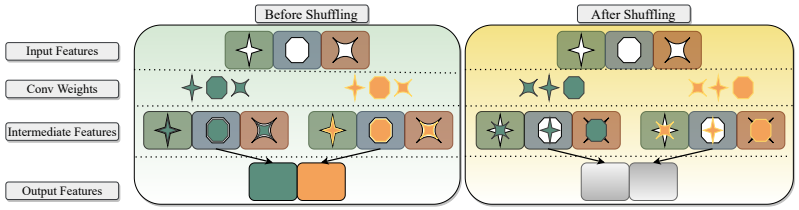

Figure 3: The overview of randomized channel shuffling operation. Before shuffling, the convolutional weights and feature maps are 'matched' in terms of channels (e.g., input feature with a hole in the shape of ploygon is supposed to encode by the polygon weight). After channel shuffling, we deliberately induce a mismatch (input feature with a hole in the shape of ploygon has to be encode by the star-shaped weight), interfering with the feature encoding process and generating meaningless output features.

features is more obvious in latter layers, which are known to encode more discriminate semantic features while the shallow layers usually encode histogram features.

3.2 Trigger Feature Revealing via Channel Shuffling

The next question is, what can be implied from the significant imbalance in activation distribution of trigger features? Consider the encoding of a poisoned image at a latter layer in backdoored model, where only one channel is vital with a high activation level. This channel needs to match well with convolutional weights in the next layer for the attack to succeed. However, if we deliberately create a mismatch between them, the lack of a second "vital channel" will lead to unsuccessful encoding with high probability. In contrast, channel features of clean images are activated more evenly, leaving a lot of backup channels to substitute in when mismatch is created. Motivated by this insight, we design a randomized channel shuffling mechanism to manually induce a mismatch between the activated channels and channel weights, and propose a way to observe how much the encoding process is being affected, thus revealing trigger features.

Randomized Channel Shuffling with Controlled Strength. Consider the convolutional weight θ^l at the lth layer of shape $C^l_{\text{out}} \times C^l_{\text{in}} \times F^l \times F^l$, where C^l_{out} and C^l_{in} are the number of output channels and input channels, respectively; F^l is the kernel size. We shuffle the order of *input channels* (the second dimension) of θ^l. The overview of randomized channel shuffling is illustrated in Figure 3.

Let $\Theta = \{\theta^1, \ldots, \theta^L\}$ denote the set of all weights in the network. We create a shuffled version of itself, $\Theta(n)$ by randomly shuffling the last n layers convolutional layers, $n = 0, 1, ..., L-1$, since trigger features show more obvious sparsity characteristic in the latter layers according to Section 3.1. We control the strength of channel shuffling by setting n for shuffling; the larger n the stronger shuffling is applied. Then, for a given input $\boldsymbol{x}$, we calculate its representation (the output of the last convolutional layer) using Θ and Θ_n, denoted as $f_\Theta(\boldsymbol{x})$ and $f_{\Theta(n)}(\boldsymbol{x})$, respectively, where f is the representation extraction part of a model. Finally, we compute the representation shift s_n using the ℓ_2-norm distance: $s_n(\boldsymbol{x}) = \parallel f_\Theta(\boldsymbol{x}) - f_{\Theta(n)}(\boldsymbol{x}) \parallel_2$.

Detect the Class-wise Abnormal Variation. First, for a dataset $\mathcal{D}$ with K class, we split it into K subsets according to the labels. For $\mathcal{D}_k$, the subset with class k, we calculate the standard deviation of representation shift caused by the shuffling operation on the last n layers, over all samples in that subset, denoted by $y^k(n) = std_{\ \boldsymbol{x}_i \in \mathcal{D}_k}\{s_n(\boldsymbol{x}_i)\}$. Then, by changing n, we can obtain a curve $y^k[0:n]$ for class k showing the variation of standard deviation of representation shift versus different strengths of channel shuffling. Finally, We can detect the target class by observing the abnormal curve, and its rationale is rooted in our sparsity hypothesis in 3.1. Note that using mean of representation shift instead of using standard deviation still works for detection, and results are provided in **App.** A3. We fix the range of n (maximum layer number to shuffle, starting from the top/last layer) as four, and we will analyze the choice in Section 4.4.

To describe how a specific curve $y^k[0:n]$ deviates from a cluster of curves $\{y^j[0:n], j = 0, ..., K-1\}$ and how to detect anomaly, we formulate the following two metrics:

(i) *Distance*: We use the ℓ_1 distance between the curve and the median to measure how far the curve is away from the cluster, formulated as $\Phi_{\text{d}} = \| \, y^k[0:n] - median_j\{y^j[0:n], \; j = 0, ..., K-1\} \, \|_1$.

(ii) *Tendency*: First, we preprocess each curve to achieve 'zero-mean' and get $\hat{y}^j[0:n]$. Then, we calculate the standard deviation of the difference between the curve and the median. To be specific, $\Phi_{\text{t}} = std_n\{ \, \hat{y}^k[0:n] - median_j\{\hat{y}^j[0:n], j = 0, ..., K-1\} \, \}$.

Then, we formulate $\Phi_{\text{dev}} = \Phi_{\text{d}} + \lambda * \Phi_{\text{t}}$ to take both deviation in distance and tendency of a curve into account, and λ is to balance two factors. A larger value of Φ_{dev} imply that its corresponding curve is less similar to others, and more likely to be the target class. We set λ as 0.01 for main experiments, but we will analyze the role of λ in Section 4.4, showing the effectiveness of the proposed method is not sensitive to the choice. After calculating Φ_{dev} for each curve, we use *Median Absolute Deviation* to detect outliers, which is a common choice for numerous backdoor detection methods[6, 7]. A anomaly index can be obtained for each data point, measuring the probability of the data point being an outlier and the curve relating to the target class. By properly setting threshold, the target class and backdoor model can be detected. In Section 4.2, we will measure the detection ability of our proposed method by AUROC. In addition, our method only need a few feedforward passes, showing exceptional efficiency compared to existing methods. More details can be found in **App.** A1.

3.3 Improved Trigger Recovery with Shuffled Models

In this section, we improve the challenging trigger recovery by leveraging the distinctive properties of channel-level feature distributions from clean image and triggers. Specifically, we first design a training procedure with randomly channel shuffling to destroy clean image features while retain trigger related features. Such models perform terribly for predicting clean images (i.e., nearly random guess clean testing accuracy), but can be easily attacked by samples with a trigger (i.e., almost perfect $\sim 100\%$ attack successful rate). Then, this "impaired" model is adopted for the subsequent recovery where substantial performance improvements can be observed.

The rational of these two steps lie in: (1) Based on our verified hypothesis in Section 3.1 - *trigger features are sparsely encoded while clean image features are encoded across multiple channels*, randomized channel shuffling alters the channel order every iteration, damages the relationship among channels, and therefore stops the model to encode meaningful clean image features. On the contrary, trigger features are more amenable to the perturbed channel orders thanks to its sparse encoding manner. (2) Models with rich trigger and suppressed clean feature information are more tractable for trigger recovery, since they get rid of the interruptive effects from clean features. This intuition is echoed with recent findings in [40].

Training with Randomly Channel Shuffling. Given a backdoor model $\mathcal{G}$, the proposed randomized channel shuffling is applied at each training iteration, and produces its shuffled variant $\tilde{\mathcal{G}}$ in the end. Actually, we find that a few training epochs (e.g., 5 epochs in our case) on a subset (e.g., 10%) of the original training set, are sufficient to generate a high quality $\tilde{\mathcal{G}}$ with low clean testing accuracy and high attack successful rate.

Enhanced Backdoor Trigger Recovery. On the top of classic approaches (refer to Section A1), we improve its trigger recovery performance by utilizing $\tilde{\mathcal{G}}$. To be specific, we introduce an enhanced objective function as depicted below:

$$\mathcal{L}(\boldsymbol{x}, \mathtt{M}, \Delta, y_t) \quad = \overbrace{\mathcal{L}_{\text{XE}}(\mathcal{G}(\boldsymbol{x}'), y_t)}^{\text{Classical Term [6]}} + \overbrace{\mathcal{L}_{\text{XE}}(\tilde{\mathcal{G}}(\boldsymbol{x}'), y_t)}^{\text{Our Term with Shuffled Models}} \quad , \tag{1}$$

where $\boldsymbol{x}' = \boldsymbol{x} \odot (\mathbf{1} - \mathtt{M}) + \mathtt{M} \odot \Delta$, $\boldsymbol{x}$ is the input sample and y_t denotes the target label detected in our previous stage (Section 3.2). $\mathcal{L}_{\text{XE}}(\cdot)$ refers to a cross entropy loss, Δ presents a recovered trigger pattern, and the mask $\mathtt{M}$ characterize the region that the trigger stamped on. Here $\mathtt{M}$ is a binary mask, belonging to $\{0, 1\}^{d \times d}$, where $d \times d$ denotes the resolution of input images.

4 Experiments

4.1 Implementation Details

Trojan Details. Based on whether the attacker manipulates labels, backdoor attacks can be roughly grouped into (i) traditional backdoor attacks like BadNets [1] and WaNet [3]; (ii) clean label attack

Table 1: Comparisons between ours and classic detection approaches. The reported numbers represent (*# correctly detected models*) / (*# models*). Note that "'CLA' stands for the clean label attack.

Attack	Dataset	Arch	Neural Cleanse			STRIP			Ours		
			benign	backdoor	AUROC	benign	backdoor	AUROC	benign	backdoor	AUROC
BadNets	CIFAR-10	ResNet-20	14/25	20/25	0.669	18/25	8/25	0.459	25/25	25/25	1.000
		AlexNet	19/25	13/25	0.622	16/25	11/25	0.454	24/25	25/25	0.997
	GTSRB	ResNet-20	24/25	2/25	0.341	12/25	21/25	0.587	23/25	21/25	0.880
CLA	CIFAR-10	ResNet-20	14/25	20/25	0.635	20/25	15/25	0.706	24/25	25/25	0.997
WaNet	CIFAR-10	ResNet-18	25/25	1/25	0.314	25/25	1/25	0.442	21/25	23/25	0.954
Total			96/125	56/125	-	91/25	56/125	-	117/125	119/125	-

(CLA) [2][2]. BadNets and CLA are classic attack methods, while WaNet is an advanced approach designed to become more stealthy to human inspection [3].

For BadNets [1], we adopt 5×5 gray-scale triggers and place them at the bottom right corner. 10% of the full training samples are injected with the trigger. As for CLA [2], we craft all training images from the target class and stamp gray-scale triggers at their bottom right corner. Experiments with WaNet attackers follow the default hyperparameter configurations in [3], and warp 10% of total training images. It is worthy to mention that the "noise mode" of WaNet is chosen in our case, which is reported as a more stealthy and challenging setup. More details are recorded in Table **??**.

Detection and Evaluation Details. $\triangleright$ *Detection.* To examine the reliability of training dataset with N classes, we first divide it into N subsets based on their label. For each subset, we then feed them into the target model as well as its randomly shuffled variant, and compute the associated representation shifts over different number of shuffled layers. Based on our observations in Section 3.1, the last few layers mainly encode discriminate features and therefore they are used in our detection. In our implementation, we only shuffle the channel order within the last four layers and generate feature sensitivity curves as $\{y^k[n], k = 0, ..., \mathrm{N}-1, n = 0, 1, 2, 3\}$. $y^i[j]$ denotes the standard deviation of representation shift from the subset of class i after randomly shuffling the last j convolutional layer. Note that the range of number of layers used for our detection play a minor role in the achievable performance, as evidenced in Section 4.4. Lastly, we utilize the metric Φ_{dev} shown in Section 3.2 to measure whether there exist a outline curve significantly different from other curves. An abnormal curve usually indicates: $i)$ the dataset and model contains backdoor triggers; $ii)$ its corresponding class is likely to be the backdoor target class. $\triangleright$ *Evaluation.* Similar to [19], we use *area under the receiver operating characteristic curve* (AUROC) to measure the backdoor model detection's performance. More details are provided in Section A2.

4.2 Backdoor Detection by Leveraging Abnormal Variation

Superior Performance across Diverse Datasets and Architectures. In this section, we verify the effectiveness of proposed detection algorithm with randomized channel shuffling on five different combinations of datasets, network architectures and backdoor attacks. Detection's ROC results are presented in Figure 4, where orange, green, and blue curves are generated by Ours, Neural Cleanse [6], and STRIP [8], respectively. Consistent performance improvements are observed: ① Compared to NC and STRIP, the AUROCs of BadNets on CIFAR-10 are increased from $0.669, 0.459$, and $0.662, 0.454$ to 1.000 and 0.997 for ResNet-20 and AlexNet, respectively; ② We outperform NC and STRIP by a substantial margin of $0.539, 0.293$ AUROCs values on GTSRB, respectively; ③ Even for more advanced clean label attack (CLA) and WaNet, ours still gain $0.362, 0.291$, and $0.64, 0.512$ AUROCs on CIFAR-10 compared to NC and STRIP.

Given the a fixed threshold, the percentage of successfully detected models can be computed. As shown in Table 1, we choose the threshold that maximize the total detection accuracy, i.e., *correctly detected benign models and backdoor models / total models*, in each combination of {dataset, architecture, attack and detection method}. Results evidence the superior performance of our detection methods in all settings.

Feature Sensitivity Curves and The Distribution of Indicator Φ_{dev}. We present the feature sensitivity curves and the distribution of indicator Φ_{dev} in Figure 5 and Figure 6 respectively. Without loss of the generality, for each attack, we randomly select one model for further analyses. Each point in Figure 5 denotes the standard deviation of feature shifts between the original and randomly

[2] Note that we follow the trigger version of [41]

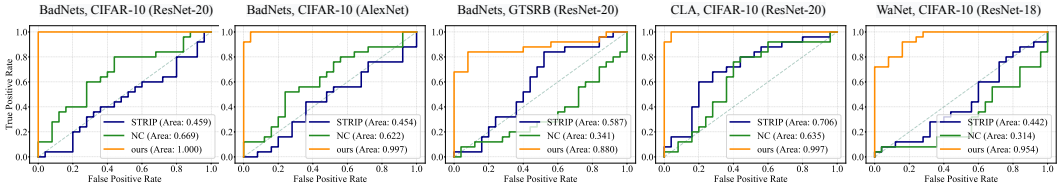

Figure 4: Performance of existing and our detection approaches, measured by ROC and related AUROC values.

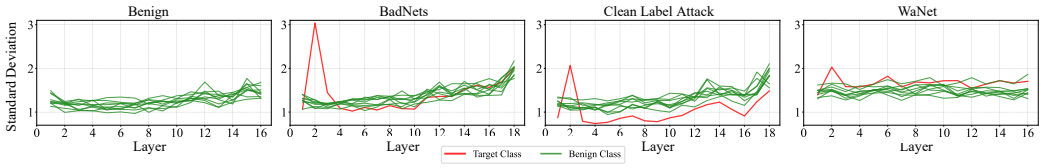

Figure 5: Standard deviation of feature shifts after applying randomized channel shuffling operation over the number of shuffled convolutional layers. Red curves are generated from subsets of target class, while green curves are calculated from subsets of benign classes.

shuffled models, given a subset of input images from certain class. After we vary the number of shuffled layers, a feature sensitive curve can be drawn. From the results in Figure 5, we see the standard deviation of feature shifts on target class's samples appears a clearly distinctive pattern (red curves), especially for the first a few points. It lies the foundation of our detection algorithm.

On the top of it, we compute the indicator Φ_{dev} to quantitatively measure the deviation from the majority of curves. Then, in order to locate the abnormal curve, a MAD outlier detection [42] is performed to Φ_{dev}. Figure 6 shows that the Φ_{dev} of the target labels tends to have significantly large values compared to the ones of benign labels, which allows it can be easily identified from the others.

4.3 Improved Triggers Recovery by Models Trained with Randomized Channel Shuffling

Training with Randomized Channel Shuffling Leads to Low TA and High ASR. In this section, we first investigate how proposed training procedures with randomized channel shuffling shape the behaviors of backdoor models. Results of TA and ASR are collected in Table 2, where "Orig." and "Shuf." indicate the original and shuffly trained models. We observe that our proposal effectively produces models with nearly unimpaired ASR (e.g., $90 \sim 100\%$) but poor TA which closes the level of random guessing. Such consistent findings can be drawn from all {dataset, architecture} combinations. It again supports our claim that randomly channel shuffling tends to destroy clean images features while preserving trigger features.

Table 2: TA (%) and ASR (%) after randomly channel shuffling on various dataset-architecture combinations.

Attack	Dataset	Arch	Type	TA	ASR
BadNets	**CIFAR-10**	**ResNet-20**	Orig.	$91.4 \pm 0.5\%$	$100.0 \pm 0.0\%$
			Shuf.	$16.2 \pm 7.3\%$	$100.0 \pm 0.0\%$
		AlexNet	Orig.	$85.7 \pm 0.5\%$	$99.9 \pm 0.1\%$
			Shuf.	$45.4 \pm 6.7\%$	$95.4 \pm 4.6\%$
	GTSRB	**ResNet-20**	Orig.	$98.5 \pm 0.5\%$	$100.0 \pm 0.0\%$
			Shuf.	$25.9 \pm 17.4\%$	$98.4 \pm 1.6\%$
CLA	**CIFAR-10**	**ResNet-20**	Orig.	$83.3 \pm 0.4\%$	$90.1 \pm 8.0\%$
			Shuf.	$14.36 \pm 6.1\%$	$99.2 \pm 0.8\%$

Superior Trigger Recovery. To efficiently generate high quality triggers, it requires two key factors: (1) detected backdoor models and identified target labels, obtained from our detection algorithms; (2) shuffly trained models that have rich trigger feature but less clean image features. As evidenced in Table 3, ours recovered backdoor triggers reach consistently enhanced ASR. More results are referred to **App.** A3.

Table 3: ASR (%) of recovered triggers from "original model" ($\mathcal{G}$) and "retrained model" ($\tilde{\mathcal{G}}$).

Attack	Dataset	Arch	Original	Ours
BadNets	CIFAR-10	ResNet-20	100.0%	100.0%
		AlexNet	88.5%	100%
	GTSRB	ResNet-20	68.0%	98.1%
CLA	CIFAR-10	ResNet-20	100.0%	100.0%

4.4 Ablation Study

Different Trigger Types in BadNets. Instead of fixing trigger at the lower right corner of the image, we change the trigger location to the upper left side, and randomly choose the location of trigger to generate two types of poisoned dataset with different stamping location strategy. We further change the gray-scale trigger into the RGB trigger. We use ResNet-20 and CIFAR-10 in the above settings. The ROC curves and AUROC values are shown in Table 4, showing the effectiveness of the proposed randomized channel shuffling across multiple trigger positions and trigger types.

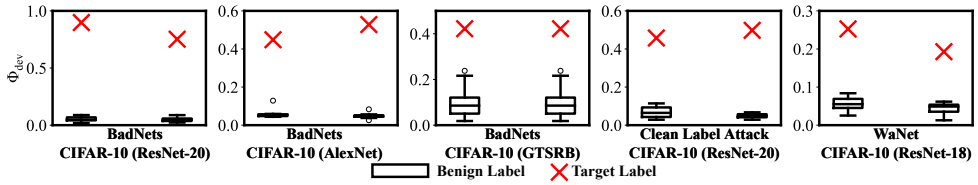

Figure 6: The Φ_{dev} distribution for backdoor models. Boxplots show the means and variances of Φ_{dev} of curves related to benign classes. The red crosses stands for Φ_{dev}s of curves for target classes. A larger value of Φ_{dev} imply that its corresponding curve is less similar to others, and thus more likely to be the target class.

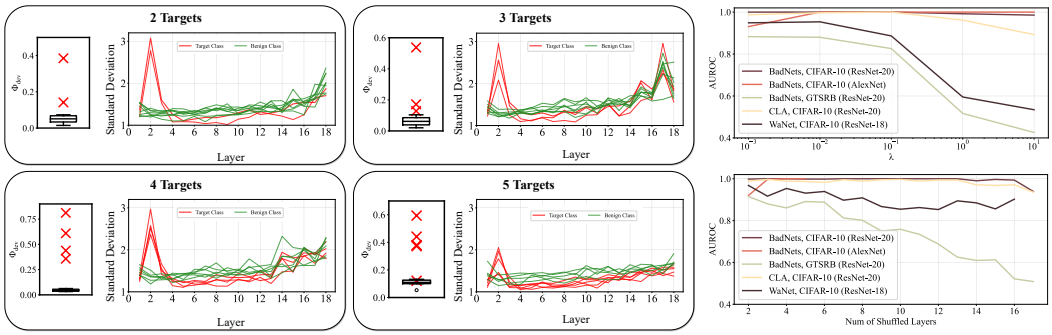

Figure 7: The *left* two columns provide visualization results when multiple backdoor target classes exist. For each attack scenario, the distribution of Φ_{dev} is collected in the *left* sub-figure, and its standard deviation of representation shifts is reported in the *right* sub-figure. The *upper right* figure provides AUROC values over different λ, the *bottom right* figure shows AUROC values over the number of shuffled layers.

Table 4: The detection performance (AUROC) of ablation studies.

Ablation Type	Different Trigger Types			Multiple Target Classes				Partial Backdoor
	Bottom Left	Random Location	RGB Trigger	2 Targets	3 Targets	4 Targets	5 Targets	
AUROC	1.000	1.000	1.000	0.978	0.981	0.902	0.770	0.945

Multiple Target Classes. Instead of just one, we consider the scenario where multiple backdoors are inserted into the dataset, with more than one class as the target class. Using CIFAR-10 as the dataset which has ten classes, ResNet-20 as architecture, and BadNets as the attack, we claim that our method is still effective when multiple target classes exist but target classes still being the minority compared to the benign classes. AUROCs are provided in Table 4. Detection performances remain acceptable when less than 5 classes are infected. But AUROC drops significantly when 5 out of 10 classes are target classes. More visualization results are provided in figure 7.

Source-Label-Specific (Partial) Backdoor A 'partial' backdoor is activated only when inputs are from specific source labels, which is more stealthy than a non-partial backdoor. We then verify the effectiveness of our method under this scenario with BadNets[1] as the attack, ResNet-20 as the model architecture, and CIFAR-10 as the dataset. The results are provided in Table 4, showing the effectiveness of the proposed method in the partial backdoor scenario.

Choice of Hyper-Parameters. As is mentioned in Section 3.2, $\Phi_{dev} = \Phi_d + \lambda * \Phi_t$, where λ is to balance two terms. Figure 7 shows the variation of AUROC when changing λ. In addition, we also verify how the number of layers included for detection affects the detection performance for different attacks. According to Figure 7, curves of AUROC remain high when only last several layers are included. But they gradually decrease when including more layers, since the standard deviation of representation shift from target class become mixed with benign ones, affecting the performance.

5 Conclusion

In this paper, we introduce and investigate a new backdoor detection setup without access to clean datasets. An effective detection algorithm is proposed by contrasting representations shifts before and after randomized channel shuffling of clean and trigger features. Moreover, by leveraging our shuffly trained backdoor models, the performance of trigger recovery is substantially boosted. Extensive experiments validate the effectiveness of our proposal.

References

[1] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain, 2019.

[2] Ali Shafahi, W Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. Poison frogs! targeted clean-label poisoning attacks on neural networks. *arXiv preprint arXiv:1804.00792*, 2018.

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

[4] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning, 2017.

[5] Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A Wichmann. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2(11):665–673, 2020.

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

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

[8] Yansong Gao, Chang Xu, Derui Wang, Shiping Chen, Damith C. Ranasinghe, and Surya Nepal. Strip: A defence against trojan attacks on deep neural networks, 2020.

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

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

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

[12] Yunfei Liu, Xingjun Ma, James Bailey, and Feng Lu. Reflection backdoor: A natural backdoor attack on deep neural networks. In *European Conference on Computer Vision*, pages 182–199. Springer, 2020.

[13] Aniruddha Saha, Akshayvarun Subramanya, and Hamed Pirsiavash. Hidden trigger backdoor attacks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 11957–11965, 2020.

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

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

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

[17] Yanyao Shen and Sujay Sanghavi. Learning with bad training data via iterative trimmed loss minimization. In *International Conference on Machine Learning*, pages 5739–5748. PMLR, 2019.

[18] Neehar Peri, Neal Gupta, W. Ronny Huang, Liam Fowl, Chen Zhu, Soheil Feizi, Tom Goldstein, and John P. Dickerson. Deep k-nn defense against clean-label data poisoning attacks, 2019.

[19] Ren Wang, Gaoyuan Zhang, Sijia Liu, Pin-Yu Chen, Jinjun Xiong, and Meng Wang. Practical detection of trojan neural networks: Data-limited and data-free cases. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIII 16*, pages 222–238. Springer, 2020.

[20] Xiaojun Xu, Qi Wang, Huichen Li, Nikita Borisov, Carl A Gunter, and Bo Li. Detecting ai trojans using meta neural analysis. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 103–120. IEEE, 2021.

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

[22] Chen Zhu, W Ronny Huang, Hengduo Li, Gavin Taylor, Christoph Studer, and Tom Goldstein. Transferable clean-label poisoning attacks on deep neural nets. In *International Conference on Machine Learning*, pages 7614–7623. PMLR, 2019.

[23] Jonas Geiping, Liam Fowl, W Ronny Huang, Wojciech Czaja, Gavin Taylor, Michael Moeller, and Tom Goldstein. Witches' brew: Industrial scale data poisoning via gradient matching. *arXiv preprint arXiv:2009.02276*, 2020.

[24] Hojjat Aghakhani, Dongyu Meng, Yu-Xiang Wang, Christopher Kruegel, and Giovanni Vigna. Bullseye polytope: A scalable clean-label poisoning attack with improved transferability. In *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 159–178. IEEE, 2021.

[25] W Ronny Huang, Jonas Geiping, Liam Fowl, Gavin Taylor, and Tom Goldstein. Metapoison: Practical general-purpose clean-label data poisoning. *Advances in Neural Information Processing Systems*, 33:12080–12091, 2020.

[26] Junfeng Guo and Cong Liu. Practical poisoning attacks on neural networks. In *European Conference on Computer Vision*, pages 142–158. Springer, 2020.

[27] Jacob Steinhardt, Pang Wei W Koh, and Percy S Liang. Certified defenses for data poisoning attacks. *Advances in neural information processing systems*, 30, 2017.

[28] Andrea Paudice, Luis Muñoz-González, Andras Gyorgy, and Emil C Lupu. Detection of adversarial training examples in poisoning attacks through anomaly detection. *arXiv preprint arXiv:1802.03041*, 2018.

[29] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6848–6856, 2018.

[30] Zhang Ting, Qi Guo-Jun, Xiao Bin, and Wang Jingdong. Interleaved group convolutions for deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, 2017.

[31] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[32] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.

[33] Yu Wang, Quan Zhou, Jia Liu, Jian Xiong, Guangwei Gao, Xiaofu Wu, and Longin Jan Latecki. Lednet: A lightweight encoder-decoder network for real-time semantic segmentation. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 1860–1864. IEEE, 2019.

[34] Qing-Long Zhang Yu-Bin Yang. Sa-net: Shuffle attention for deep convolutional neural networks, 2021.

[35] Sudhakar Kumawat, Gagan Kanojia, and Shanmuganathan Raman. Shuffleblock: Shuffle to regularize deep convolutional neural networks. *arXiv preprint arXiv:2106.09358*, 2021.

[36] Da Yu, Huishuai Zhang, Wei Chen, Jian Yin, and Tie-Yan Liu. Indiscriminate poisoning attacks are shortcuts. *arXiv preprint arXiv:2111.00898*, 2021.

[37] Pedro Sandoval-Segura, Vasu Singla, Liam Fowl, Jonas Geiping, Micah Goldblum, David Jacobs, and Tom Goldstein. Poisons that are learned faster are more effective. *arXiv preprint arXiv:2204.08615*, 2022.

[38] Shiori Sagawa, Aditi Raghunathan, Pang Wei Koh, and Percy Liang. An investigation of why overparameterization exacerbates spurious correlations. In *International Conference on Machine Learning*, pages 8346–8356. PMLR, 2020.

[39] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[40] Tianlong Chen, Zhenyu Zhang, Yihua Zhang, Shiyu Chang, Sijia Liu, and Zhangyang Wang. Quarantine: Sparsity can uncover the trojan attack trigger for free. In *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

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

[42] Frank R Hampel. The influence curve and its role in robust estimation. *Journal of the american statistical association*, 69(346):383–393, 1974.

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] (Section A4)
   (c) Did you discuss any potential negative societal impacts of your work? [Yes]
   (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]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
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]
   (b) Did you mention the license of the assets? [Yes]
   (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]