Appendix

A. Optimization of CW

The CW L^0 attack first searches for perturbations on all pixels that can cause misclassification using the L^2 norm. It then uses a processing step external to the optimization to remove the perturbations that are the least important after each optimization epoch. The objective of optimization is the following.

$$\min_{\boldsymbol{w}} \mathcal{L}_{\text{CW}} = \|\boldsymbol{x}' - \boldsymbol{x}\|_2^2 + c \cdot f(\boldsymbol{x}'), \tag{7}$$

$$\text{where } f(\boldsymbol{x}') = \max\big(\max\{Z(\boldsymbol{x}')_i : i \neq t\} - Z(\boldsymbol{x}')_t, -\kappa\big), \tag{8}$$

$$\text{and } \boldsymbol{x}' = \boldsymbol{v} \circ \frac{1}{2}\tanh(\boldsymbol{w}) + (1 - \boldsymbol{v}) \circ \boldsymbol{x}. \tag{9}$$

The first objective in $\mathcal{L}_{\text{CW}}$ is to minimize the L^2 distance between the perturbed input $\boldsymbol{x}'$ and the original input $\boldsymbol{x}$. The second objective $f(\boldsymbol{x}')$ is for inducing misclassification by enlarging the target label t's logits value (denoted by $Z(\cdot)$) and reducing the largest logits of other labels as shown in Equation 8. The parameter κ controls the confidence of the misclassification. Equation 9 explains how an input $\boldsymbol{x}$ is perturbed. The perturbation is controlled by a mask vector $\boldsymbol{v}$ similar to the mask in NC. Its value is either 0 or 1, with the former indicating the corresponding pixel cannot be perturbed and the latter meaning that the pixel is replaced with the perturbation value. Variable $\boldsymbol{w}$ is a vector of arbitrary values denoting the perturbations. Function $\frac{1}{2}\tanh(\boldsymbol{w})$ projects these values to $[-0.5, 0.5]$. Note that in CW, each pixel value is normalized to $[-0.5, 0.5]$. To reduce the number of perturbed pixels, CW leverages a processing step external to the optimization after each epoch. Specifically, let $\boldsymbol{\delta} = \frac{1}{2}\tanh(\boldsymbol{w})$ be the perturbations and $\boldsymbol{g} = \nabla\mathcal{L}_{\text{CW}}$ be the gradient of objective function. It computes the importance of the perturbation of a pixel i by $\boldsymbol{g}_i \cdot \boldsymbol{\delta}_i$. CW sets $\boldsymbol{v}_i$ to 0 if the importance is smaller than a threshold. The change of $\boldsymbol{v}$ is monotonic. The algorithm can be easily adapted to generate backdoor: instead of optimizing $\boldsymbol{w}$ for one input, we optimize it for a set of inputs.

B. Optimization of Universal Adversarial Perturbation (UAP)

The existing UAP algorithm mainly produces perturbations for *untargeted* attacks. Since backdoor attacks are usually targeted, we extend the algorithm as follows.

$$\min_{\boldsymbol{\delta}} \mathcal{L}_{\text{UAP}} = \frac{1}{N}\sum_{i=1}^{N} \hat{\mathcal{L}}\big(\mathcal{M}(\boldsymbol{x}_i + \boldsymbol{\delta}), y_t\big) \text{ s.t. } \|\boldsymbol{\delta}\|_\infty \leq \epsilon, \tag{10}$$

$$\text{where } \hat{\mathcal{L}}\big(\mathcal{M}(\boldsymbol{x}_i + \boldsymbol{\delta}), y_t\big) = \min\{\mathcal{L}(\boldsymbol{x}_i + \boldsymbol{\delta}, y_t), \beta\}. \tag{11}$$

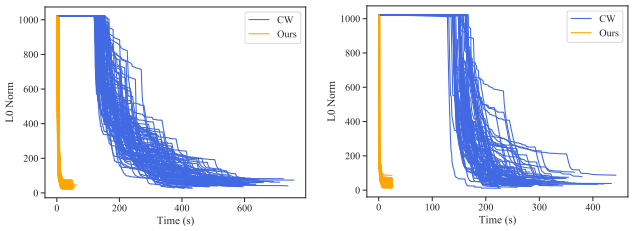
Variable $\boldsymbol{\delta}$ denotes the backdoor perturbation, bounded by an L^∞ size ϵ. Parameter β is a threshold for the loss of individual samples so as to avoid the loss of a single sample dominating the whole objective and to achieve a higher ASR [58]. Intuitively, the method aims to bound the maximum perturbation on a single pixel while minimizing the adversarial loss. Such bound is needed. Otherwise with an unbounded L^∞ distance, the optimization might completely change the input image in order to achieve its goal. We do not add a loss term to minimize the L^∞ distance because its ASR is low to begin with. Our results in Appendix E and F show that UAP cannot achieve high ASRs. Furthermore, almost all the pixels on an input image are perturbed. This makes its application in the physical world very difficult. The second and third images in the first row in Figure 2 (see Section 1) present the generated backdoors by UAP and its reduced version, respectively. Observe that almost all the pixels ($50k \approx 224 \times 224$) are perturbed while the ASR is only 2%. Reducing the number of perturbed pixels leads to 0% ASR. Its performance on CIFAR-10 is better. The green line in Figure 3 (see Section 3) shows that the ASR for a UAP trigger can be as high as 0.6 (the right end of the line). However, applying only the top 200 perturbations of UAP backdoor has nearly zero ASR, indicating most perturbations in the backdoor are important.

C. Detailed Experiment Setup

CIFAR-10 [27] is an object recognition dataset for a 10-class classification task, which contains 60,000 images. We split the whole dataset into three sets: 48,000 images for training, 2,000 for validation and 10,000 for testing. Two different models are utilized for this dataset: ResNet20 [20], Network in Network (NiN) [36].

SVHN (Street View House Numbers) [47] dataset contains house number digits extracted from Google Street View images, which consist of 73,257 training images and 26,032 test images. We further split the original training set into 67,257 samples for training and 6,000 samples for validation. We employ two models, NiN [36] and ResNet32 [20].

LISA [45] is a U.S. traffic sign dataset that contains 47 different road signs. However, the number of samples of different classes is not well-balanced, with some classes having very few images. We use the same setting as in an existing work [11] by choosing 18 most common classes based



(a) CIFAR-10 (b) SVHN

Figure 8. Comparison of CW and ours on the generation efficiency for all class pairs on CIFAR-10 and SVHN datasets. The x-axis denotes the time in seconds and the y-axis shows the size of backdoors during the trigger generation.

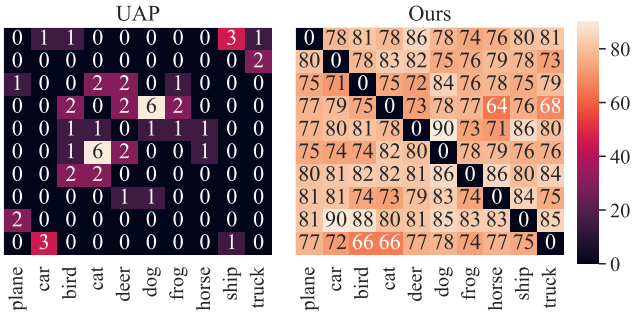


Figure 9. Comparison of UAP and ours on the ASR for all class pairs on the CIFAR-10 dataset

on the number of training examples, and split the dataset into 5,635 training samples, 704 validation samples and 704 test samples. We use two model structures for this dataset: A CNN model [11] that consists of three convolutional layers and one fully-connected layer, and a ResNet20 [20].

ImageNet [54] dataset is a large set for image classification with 1,000 labels, which contains 1,281,167 training images and 50,000 validation images. We use a ResNet50 [20] model downloaded from a widely-used model repository [25].

For backdoor scanning, we randomly select 100 models (half clean and half poisoned) in each round from rounds 2-4 of the TrojAI competition [50]. We exclude the round 1 models due to its simple poisoning settings. In total, we have 150 clean models and 150 poisoned models. The task of backdoor scanning is to determine whether a given model is poisoned or not. TrojAI models utilize 16 different structures such as DenseNet121, InceptionV3, MobileNetV2, etc. Each model is trained to classify synthetic traffic signs to between 5 and 45 classes. Input images are created by compositing a foreground object, e.g., a synthetic traffic sign, with a random background image from five different datasets in three categories from the KITTI dataset [14], the Cityscapes dataset [9] and the Swedish Roads dataset [29]. The organizer provides 2-50 clean images per class for each model in different rounds. Poisoned models are trojaned with various kinds of backdoors, including universal, label-specific and position-specific. We consider polygon triggers in this paper which are pixel patterns (e.g., polygons with solid color). Random transformations, such as shifting, titling, lighting, blurring, and weather effects, are applied during training to improve dataset diversity. Also, adversarial training with PGD (Projected Gradient Descent) [44] and FBF (Fast is Better than Free) [70] is leveraged to improve model quality in rounds 3-4.

D. Comparison of Time Cost with CW

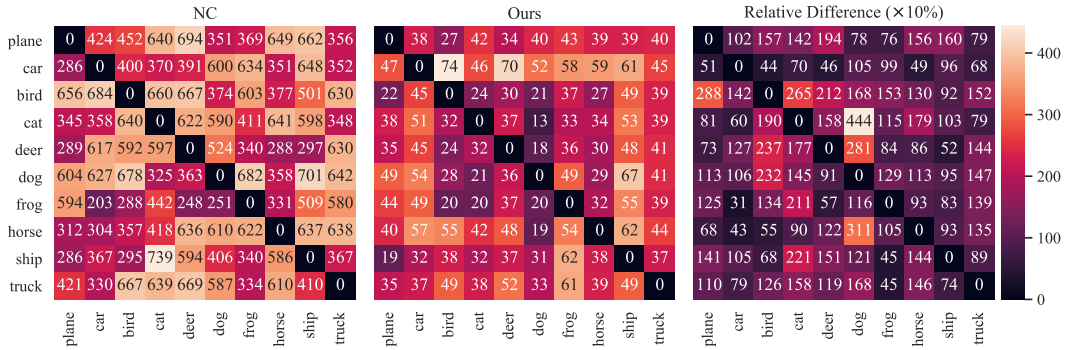
We compare the time cost of CW and ours in Figure 8. The x-axis denotes the time in seconds and the y-axis denotes the trigger size. We record the generation for all class pairs from CIFAR-10 and SVHN. Observe that CW spends a large amount of time finding a feasible solution at the beginning (around 120 seconds). After that, it aims to reduce the number of perturbed pixels using an external step as discussed in Section 3.2. This leads to the staircase phenomenon in Figure 8 as CW tries to find a feasible solution with the given set of pixels allowed for perturbing. Our method converges significantly faster than CW. On average, ours is 10.88 times faster on CIFAR-10 and 11.53 times faster on SVHN.

E. Comparison with UAP on CIFAR-10

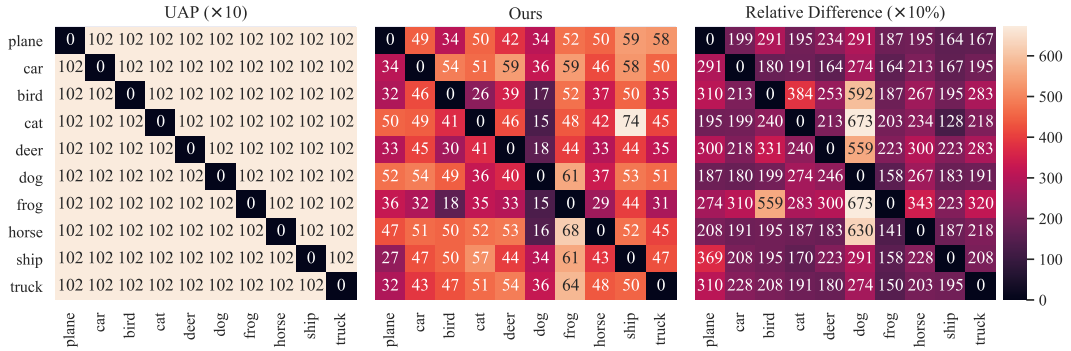
UAP is based on L^∞ and hence not directly comparable with our method. We follow the same procedure as before (see comparison with NC in the evaluation section): aligning by trigger sizes and then comparing ASRs, and aligning by ASRs and then comparing trigger sizes. The left heat map in Figure 9 shows the results when aligning trigger sizes. The average ASR of reduced UAP triggers is only 0.96%, whereas the original triggers have 68.21% ASR. It is clear that triggers generated by UAP are ineffective with a small number of perturbed pixels like ours. The left heat map in Figure 10b shows the results of aligning ASRs. Observe that the UAP triggers perturb all pixels, whereas ours are two orders of magnitude smaller. This is expected as UAP is an L^∞ method. Triggers generated by such a method can hardly be used in physical attack.

F. Evaluation on SVHN

Comparison with CW Optimization. In this experiment, we use CW and our method to generate natural triggers for all the class pairs for a clean NiN model on SVHN. Figure 12 shows the comparison. Each cell in a heat map denotes the result for a natural backdoor flipping all the test samples from a victim class (row) to a target class (column). Figure 12a and Figure 12b show the trigger sizes and the ASRs for CW (the left heat map) and ours (the middle



(a) Comparison with NC



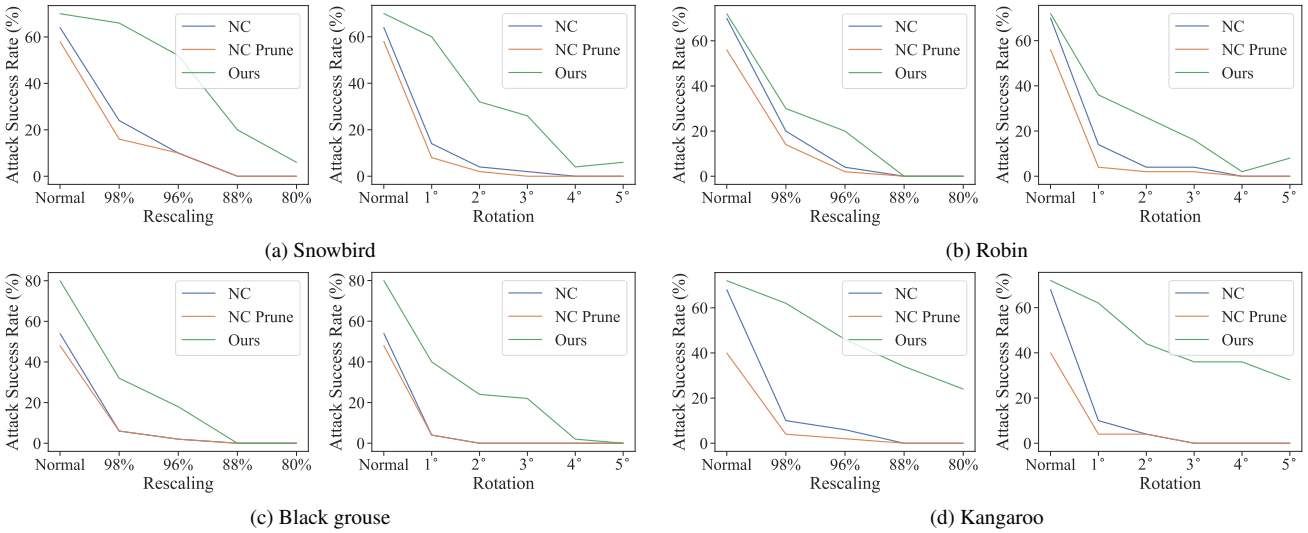
(b) Comparison with UAP

Figure 10. Comparison of the number of perturbed pixels with the same ASR for all class pairs on the CIFAR-10 dataset. The first two heat maps in each subfigure illustrate the results for NC/UAP and ours, respectively. The last heat map shows how much larger of generated backdoors by NC/UAP compared to ours.

heat map), respectively. The right heat map in Figure 12a shows how much larger the CW triggers are compared to ours. Observe that there are a few class pairs where CW and ours have the same trigger size, such as $3 \rightarrow 0$ and $5 \rightarrow 2$. However, for other pairs, CW has a significantly larger trigger size than ours. For instance, for pair $2 \rightarrow 0$, the trigger by CW is 110% larger than ours. Even with a much larger trigger, CW however still has lower ASR (59% vs 83% for $2 \rightarrow 0$). This is because CW uses an external procedure to reduce the number of perturbed pixels (removing unimportant pixels based on $\boldsymbol{g}_i \cdot \boldsymbol{\delta}_i$ as discussed in Section 3.2).

Comparison with NC. NC tends to generate triggers with a large number of small perturbations. The generated triggers hence cannot be easily applied in physical attacks. We conduct two experiments: (1) align the number of perturbed pixels of the NC triggers and our triggers and then compare the corresponding ASRs; (2) align the ASRs and compare the trigger sizes. For the first experiment, we use the sizes of our triggers as the reference, and align the triggers by NC by gradually removing their smallest perturbations until they have the same sizes as ours. We then compare the ASRs of our triggers and the reduced NC triggers. Figure 13 presents the results. Observe that for most class pairs, the reduced NC triggers have reasonable ASRs with an average of 75.87%, degraded from 76.55% without reduction. The results on SVHN are better than those on CIFAR-10 and ImageNet. Because SVHN is a dataset for digital number (from 0 to 9) recognition, which is a simpler task than objection recognition in CIFRA-10 and ImageNet. Optimization methods like NC can generate a digit shape (e.g., 1) with a few pixels change. This also explains the similar ASRs of triggers with and without reduction. Our triggers have higher ASRs than NC's for all class pairs, with the largest difference of 21% for $2 \rightarrow 8$ and $8 \rightarrow 2$. On average, ours have 83.18% ASR, even higher than the original NC triggers without size reduction. In the second experiment, we use NC's ASR as the reference and then gradually remove the smallest perturbations in our triggers until their ASRs drop to the same level as NC's and then compare the sizes. Figure 14a presents the results. Observe that NC has one order of magnitude larger trigger sizes than ours for all class pairs (except for $6 \rightarrow 5$ and $3 \rightarrow 8$). Since the reduced NC triggers have similar ASRs as the non-reduced ones, many pixel perturbations by NC are redundant and can be pruned for SVHN. After alignment, our generated triggers perturb fewer pixels.

Comparison with UAP. UAP is based on L^∞ and hence not directly comparable either. We follow the same proce-



(a) Snowbird (b) Robin

(c) Black grouse (d) Kangaroo

Figure 11. Evaluation of NC and our triggers under transformations on ImageNet. The victim class is turtle and the target classes are in the caption of each subfigure.

dure as before: aligning by trigger sizes and then comparing ASRs, and aligning by ASRs and then comparing trigger sizes. The middle heat map in Figure 13 shows the results when aligning trigger sizes. The average ASR of reduced UAP triggers is only 0.51%, whereas the original triggers have 18.76% ASR, which is also very low. It is clear that triggers generated by UAP are completely ineffective with a small number of perturbed pixels like ours. The left heat map in Figure 14b shows the results of aligning ASRs. Observe that the UAP triggers perturb all pixels, whereas ours are two orders of magnitude smaller.

G. Comparison with NC Variants

NC variants are based on NC and hence limited by the performance of NC. We test on a NC variant, ABS [38], for generating a trigger for pair plane→car on a ResNet20 model on CIFAR-10. The generated trigger by ABS has 84 perturbed pixels, which is twice larger than ours (38). The ASR on the test set is 38% by ABS and 78% by ours. If we reduce the number of perturbed pixels by ABS to match the size of our trigger, it has only 16% ASR. This empirically demonstrates that NC variants have the same limitation as NC. Besides, as those variants are built upon NC, they can be modified to use our method as the core optimization, which can boost their performance.

H. Evaluation on Desktop

We conduct an experiment on a desktop equipped with one Intel i7-8700 Processor, 16 GB of RAM, and a single NVIDIA GeForce GTX 1070 Ti GPU, which is a common affordable desktop machine. We use the case shown in Figure 2 in Section 1 and the results are shown in Table 4. Observe that the runtime for different methods on the desktop is similar to that on the server with minor differences. Our method still has the lowest time cost, and is around 2 times faster than NC and 4 times faster than UAP. Our method can be easily deployed on machines with limited resources.

Table 4. Experimental comparison on different machines

Method	Machine	Time	#Pixels	ASR
UAP	Server	22.69	50173	2.00%
	Desktop	21.12	50170	2.00%
NC	Server	9.02	26583	28.00%
	Desktop	10.62	17516	20.00%
Ours	Server	4.27	822	70.00%
	Desktop	5.40	818	70.00%

I. Robustness of Generated Triggers

We study the robustness of generated triggers under various image transformations. As the triggers by CW and UAP have very low ASRs, we hence only consider NC in this study. For some target classes, NC still has a low ASR. We then increase the strength of the adversary by utilizing 100 images for both NC and ours during trigger generation (50 images for the study in Section 5.3). This yields better test ASRs in general. We test on two types of transformations with different parameters: rescaling and rotation. Figure 11 presents the results for a source class turtle, with the target classes in individual subfigures. For each target class, we show the results under rescaling transformation on the left and under rotation on the right. The x-axis de-

Table 5. Comparison of different methods augmented with transformations during generation on a victim class turtle from the ImageNet dataset. The first column shows the target classes. The second column shows backdoor generation methods. The third column is the ASR of original triggers. The 4th-7th columns show the ASR under different rescaling transformations. The 8th-12th columns show the ASR under different rotation transformations. The last column shows the average ASR.

Target	Method	Normal	Rescaling				Rotation					Average
			98%	96%	88%	80%	1°	2°	3°	4°	5°	
Snowbird	NC	80.00%	74.00%	46.00%	0.00%	0.00%	82.00%	46.00%	8.00%	8.00%	2.00%	34.60%
	NC Prune	64.00%	44.00%	16.00%	0.00%	0.00%	50.00%	18.00%	0.00%	0.00%	0.00%	14.22%
	Ours	78.00%	66.00%	64.00%	16.00%	6.00%	74.00%	26.00%	18.00%	4.00%	4.00%	35.60%
Robin	NC	80.00%	70.00%	56.00%	16.00%	12.00%	68.00%	50.00%	20.00%	18.00%	10.00%	40.00%
	NC Prune	42.00%	38.00%	22.00%	6.00%	4.00%	32.00%	18.00%	4.00%	6.00%	4.00%	17.60%
	Ours	84.00%	78.00%	70.00%	46.00%	32.00%	80.00%	46.00%	40.00%	20.00%	12.00%	50.80%
Grouse	NC	82.00%	70.00%	48.00%	0.00%	0.00%	70.00%	6.00%	2.00%	0.00%	0.00%	27.80%
	NC Prune	76.00%	68.00%	42.00%	0.00%	0.00%	54.00%	4.00%	2.00%	0.00%	0.00%	18.89%
	Ours	82.00%	70.00%	68.00%	30.00%	28.00%	78.00%	42.00%	28.00%	24.00%	24.00%	47.40%
Kangaroo	NC	80.00%	72.00%	52.00%	0.00%	0.00%	74.00%	38.00%	34.00%	28.00%	20.00%	39.80%
	NC Prune	66.00%	46.00%	28.00%	0.00%	0.00%	52.00%	24.00%	16.00%	18.00%	8.00%	21.33%
	Ours	78.00%	70.00%	60.00%	50.00%	34.00%	76.00%	54.00%	52.00%	48.00%	40.00%	56.20%

notes the transformation strengths with the first one without any transformations, and the y-axis denotes the ASR. As we discussed in Section 5.3, NC has a much larger number of perturbed pixels than ours. Hence besides the original NC triggers, we also reduce the NC triggers to match our numbers of perturbed pixels and study their robustness as well. They are denoted as NC Prune in the figure. Observe that most of NC triggers become ineffective after 96% rescaling or 2° rotation (near 0% ASR). NC Prune has a lower ASR, even without transformations. For the target kangaroo, NC Prune has 28% lower ASR compared to NC, indicting that NC does require a large number of perturbed pixels to be effective, which is not so desirable for physical attacks. Our method has a consistently higher ASR than NC. For target kangaroo, our trigger has around 40% ASR with most of the transformations. We further study the robustness of triggers by applying transformations during the trigger generation. The observations are similar (see the following section).

J. Augmentation during Backdoor Generation

In this section, we study the robustness of triggers by applying transformations during trigger generation. Specifically, for input samples stamped with triggers, we randomly rescale 1% and rotate 1° for those samples. Using larger transformations would increase the trigger size, which is not desired for physical attacks. Table 5 shows the ASR results for augmented triggers on a victim class turtle from the ImageNet dataset. The first two columns show the target classes and backdoor generation methods. The third column is the ASR of original triggers. The 4th-7th columns show the ASR under different rescaling transformations. The 8th-12th columns show the ASR under different rotation transformations. The last column shows the average ASR. As NC has a much larger number of perturbed pixels than ours. Hence besides the original NC triggers, we also reduce the NC triggers to match our trigger sizes and study their robustness as well, which is denoted as NC Prune in the table. Observe that the ASRs of both NC and ours are increased on small scale transformations (98%-96% rescaling and 1°-2° rotation) compared to the results in Figure 11 in the previous section. However, for larger transformations, NC still has near 0% ASR on most cases. NC Prune has a low ASR even with the augmentation during the trigger generation. On average, it has only 18.01% ASR. Backdoors generated by our method can maintain a reasonable ASR. For instance, under the largest rescaling transformation (80%), our triggers still have around 30% ASR on the bottom three target classes. On average, our method has 47.50% ASR, 11.95% higher than NC (35.55%) and 29.49% higher than NC Prune (18.01%), indicating our generated triggers are more suitable for physical attacks.

K. Ablation Study

Our method introduces two components for approximating the number of perturbed pixels: the $\tanh$ loss and the two variables $\boldsymbol{b}_p$ and $\boldsymbol{b}_n$ for the positive and negative perturbations. We study individual components to understand their effects. In particular, we consider four settings, namely, (1) excluding the $\tanh$ loss; (2) replacing the two variables with a single variable for the positive perturbation; (3) replacing the two variables with a single variable for both positive and negative perturbations; (4) excluding both the $\tanh$ loss and the two variables. The results for pair plane→dog from CIFAR-10 on a ResNet20 model with 10 random runs are shown in Table 6. The number of perturbed pixels (#Pixels) is presented in the second column

Table 6. Ablation study on effects of different components. The results are collected from 10 random runs for the class pair plane→dog for a ResNet20 model on the CIFAR-10 dataset.

Method	#Pixels	ASR
Ours	38.70± 5.10	80.07±3.96%
- tanh loss	46.40±10.07	79.73±2.11%
- two variables (positive)	84.50±12.24	72.64±3.82%
- two variables	1024.00± 0.00	90.99±4.40%
- tanh loss & two variables	1023.10± 0.88	88.39±4.85%

and the ASR in the third column. Observe that without using the tanh loss, the trigger size increases by 20% from 38.70 to 46.40 on average. The standard deviation is twice of ours (10.07 vs. 5.10). Replacing the two variables with a single positive variable doubles the size of the generated trigger (from 38.70 to 84.50), and the ASR also drops (from 80.07% to 72.64%). Using a single variable for both positive and negative perturbations cannot reduce the number of perturbed pixels (the last two rows). As discussed in Section 4, there is a steep slope where the perturbation is 0 when using such a variable. It is unlikely for the optimization to stabilize at the 0 point. Hence, almost all the pixels will be perturbed during the trigger generation. This indicates the necessity of separating the positive and negative perturbations during the optimization. Using the tanh loss can further reduce the trigger size without sacrificing the ASR.

L. Applications

In this section, we evaluate our method in two applications including model hardening and backdoor scanning.

L.1. Model Hardening

In the introduction section, we have shown that a small backdoor generated by our method can flip the majority of samples of turtle to kangaroo in the ImageNet dataset. This is a critical security threat. We hence utilize generated triggers to harden the model by training on normal inputs stamped with triggers. After hardening, an adversary is supposed to produce a large and visible backdoor, which can be easily detectable by automated tools or human inspectors. CW is extremely expensive in trigger generation (as shown in Section 5 and Appendix D), which is not computationally feasible for model hardening that requires a large number of triggers generated on-the-fly during training. We hence only consider UAP, NC and ours for model hardening. In the original paper [65], NC generates universal backdoors for all classes beforehand, and then hardens the model using this set of backdoors by stamping on training inputs. We further improve the hardening process by generating universal backdoors on-the-fly, similar to adversarial training. We call it iterative NC. We use the same procedure for UAP and ours to harden models. The L^∞ bound for UAP training is determined according to the normal accuracy drop. We use L^∞ bound of $4/255$ for CIFAR-10, 0.05 for SVHN, and 0.03 for LISA.

We use the mask size by NC and the number of perturbed pixels by our method to measure the class distance from a victim class to a target class. The goal of model hardening is hence to enlarge the class distance for all pairs. We use the relative improvement of pairwise class distance as the metric. That is, we compute the improvement percentage for every class pair and obtain the average, defined as follows.

$$\frac{1}{n \times (n-1)} \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} \frac{\hat{d}_{i \to j} - d_{i \to j}}{d_{i \to j}}, \tag{12}$$

where n is the number of classes; $d_{i \to j}$ and $\hat{d}_{i \to j}$ are the class distances from i to j for the original model and the hardened model, respectively. We randomly select 100 samples from the validation set of class i and apply NC/ours for 1,000 epochs to generate a backdoor that can flip 90% of those samples to the target class j. As a backdoor is randomly initialized during generation, to avoid the bias from randomness, we run the generation on the same pair for 3 times and use the smallest backdoor size as the class distance. We show the average relative improvement along with the average class distance in the following results.

Table 7 shows the results for model hardening on CIFAR-10, SVHN and LISA datasets. The first three columns denote the dataset, model structure and training methods, respectively. The 4th column shows the model accuracy on the test set. The 5th column presents the training time in minutes. The 6th and 8th columns show the average class distance measured by NC and ours, respectively. The 7th and 9th columns show the relative improvement of class distance measured using Equation 12. UAP has the lowest improvement on distances by both NC and ours on CIFAR-10 and SVHN, except for the ResNet20 model on CIFAR-10 measured by NC. For cases such as ResNet32 on SVHN, UAP instead reduces the class distance measured by both NC and ours. NC can achieve a reasonable improvement from 4.58% to 39.10% measured by NC and from 3.96% to 49.56% measured by ours. The iterative version of NC further improves the class distance. It has an average of 68.61% improvement on the distance measured by NC and 89.70% by ours. Using our method for hardening produces the largest class distance improvement on both metrics (73.02% by NC and 106.37% by ours) on average. Specifically, models hardened by our method have much more improvements on the distance measured by ours, and are also better when measured by NC. This demonstrates that our backdoor generation method is better than NC in exposing model vulnerabilities. Models hard-

Table 7. Comparison of different methods on model hardening. First three columns denote different datasets (D), models (M) and training methods for the evaluation. The fourth column denotes model accuracy on the test set. The fifth column shows the training time in minutes. The sixth and the eighth columns show the average class distance across all class pairs measured by NC and ours, respectively. The seventh and the ninth columns denote the improvement of pairwise class distance (measured by NC and ours) by different techniques compared to that of original models (Natural).

D	M	Method	Accuracy	Time (m)	Adv_{NC}	$\text{Increase}_{\text{NC}}$	Adv_{Ours}	$\text{Increase}_{\text{Ours}}$
CIFAR-10	ResNet20	Natural	91.52%	56.77	53.49	-	43.91	-
		UAP	90.04%	243.11	96.00	81.57%	61.62	42.91%
		NC	90.83%	84.88	72.56	37.84%	62.77	49.56%
		Iterative NC	90.57%	65.00	93.54	78.16%	89.24	112.12%
		Ours	90.32%	64.11	95.17	79.21%	100.72	139.77%
	NiN	Natural	88.09%	68.30	60.67	-	38.17	-
		UAP	86.61%	196.67	57.56	-5.22%	36.92	-2.05%
		NC	86.64%	40.35	75.49	26.49%	52.20	42.89%
		Iterative NC	86.76%	33.90	90.69	54.09%	66.69	87.75%
		Ours	86.32%	28.46	93.77	59.08%	74.31	108.59%
SVHN	NiN	Natural	95.61%	10.50	64.63	-	37.17	-
		UAP	94.63%	45.47	69.31	6.99%	41.56	12.37%
		NC	94.89%	24.72	82.90	32.19%	53.16	48.96%
		Iterative NC	95.03%	53.40	107.15	65.97%	67.53	88.99%
		Ours	94.86%	42.71	108.43	68.47%	71.18	99.86%
	ResNet32	Natural	95.15%	26.70	55.11	-	32.83	-
		UAP	93.16%	228.95	49.40	-8.86%	23.69	-27.08%
		NC	93.45%	31.51	75.77	39.10%	45.57	39.69%
		Iterative NC	94.60%	109.30	120.20	113.92%	76.20	128.26%
		Ours	94.18%	97.55	122.79	121.07%	83.24	152.02%
LISA	CNN	Natural	97.30%	0.15	68.47	-	32.92	-
		UAP	95.60%	1.79	65.90	-1.23%	32.31	-0.43%
		NC	96.88%	8.27	71.69	4.58%	35.09	6.70%
		Iterative NC	96.45%	11.34	101.48	46.13%	53.38	60.36%
		Ours	96.02%	10.41	107.34	54.34%	56.83	70.11%
	ResNet20	Natural	98.86%	1.70	72.05	-	43.62	-
		UAP	96.16%	6.33	97.11	36.32%	56.77	30.77%
		NC	99.29%	34.34	75.51	4.67%	45.21	3.96%
		Iterative NC	98.30%	25.37	113.35	53.38%	72.18	60.74%
		Ours	98.30%	27.03	115.63	55.96%	75.58	67.86%

ened by our method are more resilient to existing natural backdoor attacks.

L.2. Backdoor Scanning

Backdoor scanning aims to scan a given model to decide if it contains a backdoor, without assuming any inputs stamped with the backdoor pattern [19, 21, 22, 26, 52, 63, 68, 72, 76]. This is one of the popular defense solutions against backdoor attack. Many existing backdoor scanners are built on top of NC's trigger generation method. For instance, a state-of-the-art approach K-arm [59] uses NC as the base optimization method to generation backdoor. It iteratively and stochastically selects the most promising labels (potentially poisoned) for optimization with the guidance of an objective function. It achieves the top performance on the TrojAI competition organized by IARPA [50], outperforming NC [65], ABS [38], TABOR [18], DLTND [68], and other existing methods. To evaluate the performance of our backdoor generation method in downstream applications, we replace the NC method with ours in the K-arm scanner. We conduct the experiment on 300 pre-trained models from the TrojAI competition. Detailed setup can be found in Appendix C. For a fair comparison, we use the same setting in K-arm (including the pre-selection and the scheduler) and only replace the optimization component. The comparison results are shown in Table 8. We also include the results of a few other baselines from the K-arm paper [59]. The first column denotes detection methods. The following six columns present detection accuracy and time (per model in seconds) on different rounds. Observe that using our method can further boost the performance of K-arm for 1% on round 3 and 2% on round 4, surpassing the state-of-the-art results. Note that since the original K-arm already has high accuracy, the room to improve is small. The time cost is comparable using our method. We also compare our method with another detection approach GangSweep [79] on the TrojAI round 3. We randomly select 20 benign models and 20 poisoned models to conduct the experiment. The

Table 8. Scanning backdoored models on the TrojAI dataset

Method	Round 2		Round 3		Round 4	
	Acc.	Time(s)	Acc.	Time(s)	Acc.	Time(s)
ABS	62%	1527	71%	1435	79%	525
TABOR	55%	$>$ 32000	60%	$>$ 30000	60%	$>$ 35000
DLTND	60%	$>$ 26000	65%	$>$ 29000	65%	$>$ 31000
K-arm	85%	210	91%	183	87%	292
Ours	85%	231	92%	198	89%	320

detection accuracy of GangSweep is only 57.50%, much lower than ours (92%).

We further evaluate our method on detecting three advanced backdoor attacks, namely, WaNet [48], invisible backdoor [32], and blind Backdoor [2].

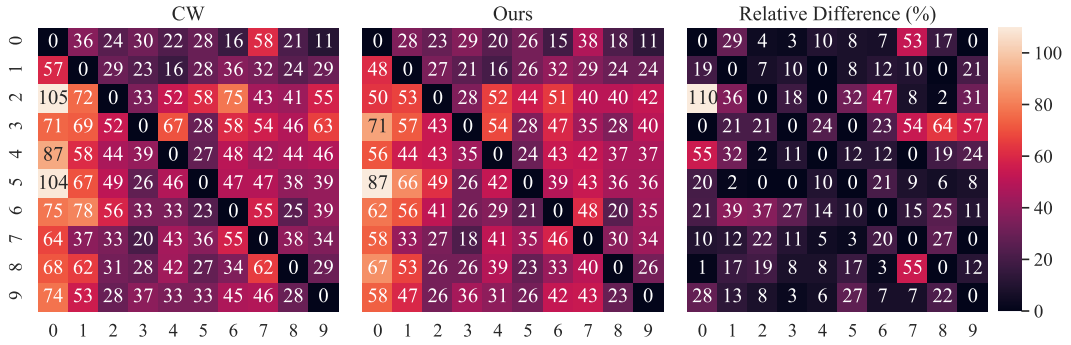
WaNet [48] uses distortion transformation (e.g., distorting straight lines) as the backdoor. At the pixel level, the backdoor varies for different inputs. We conduct an experiment on backdoored models downloaded from the official repository [48], which are trained on MNIST, CIFAR-10, GTSRB and CelebA, respectively. We use NC and our method to reverse engineer triggers for these models. We use the anomaly index to analyze generated triggers for backdoor detection as in the original NC paper [65]. A large index means that the generated trigger for a label is much smaller than those for other labels. A model with an anomaly index larger than 2 is considered backdoored (i.e., the default setting). Table 3 in Section 5.5 shows the anomaly indices for different models using NC and ours. We can see that NC cannot detect any of the evaluated models (consistent with the results reported in [48]), whereas our method can detect all the backdoored models (as we can generate a much smaller trigger for the target). Our inspection shows that although the injected triggers are pervasive, the models pick up low level features such as curly lines during poisoning. NC generates large triggers for the target class that are not distinguishable from those of benign classes, whereas our triggers are much smaller.

The invisible backdoor [32] uses a uniform perturbation with the smallest L^2 as the backdoor, which is hence pervasive. Since the authors did not provide the implementation or backdoored models in the paper, we have contacted the authors but haven't heard from them yet. We also tried to re-implement their attack based on the paper. With limited details of hyper parameters, the backdoor we got is larger than what was reported in the paper. We hence tested on a backdoored model on CIFAR-10 with the smallest L^2 backdoor that we can get. The normal test accuracy is 90.96% and the ASR is 99.63%. We then use NC and our method to evaluate this model. The anomaly index is 0.77 by NC and 2.28 by ours. The result shows that our method can detect the model as backdoored, whereas NC cannot.

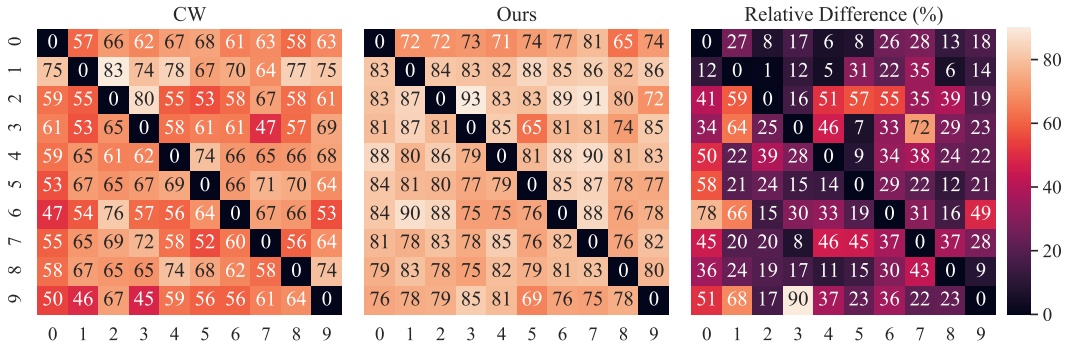
The blind backdoor attack [2] can evade the detection of NC. We study our method against such a strong attack. We use the official repository from the original paper [2] to conduct an experiment. We use the MNIST dataset and replace the optimization of NC with ours in the robust training. After training, we run our method on the trained model to generate triggers for all classes, and use anomaly detection to see whether the model is backdoored. For the above robustly trained model, our method has an anomaly index of 3.37, which can detect the model as backdoored. It indicates that the smoother loss function in our technique allows finding the true trigger despite the robust training. We further demonstrate that when applying our method on the robust model trained with NC (the same in [2]), we can also detect the model as backdoored with an anomaly index of 3.04.

Table 9. Comparison of different methods on more class pairs from ImageNet. The first column shows the victim→target class pairs. The second column shows the methods. The third column is the time cost (in minutes) and the fourth column the number of perturbed pixels (#Pixels). The last column shows ASR on the samples from the validation set.

Pair	Method	Time (min)	#Pixels	ASR
Bullfrog→Robin	UAP	10.28	50175	0.00%
	NC	9.20	26016	28.00%
	Ours	3.69	489	50.00%
Gorilla→Tench	UAP	12.37	50173	0.00%
	NC	9.19	26803	10.00%
	Ours	3.78	587	50.00%
Gorilla→Goldfish	UAP	12.63	50174	0.00%
	NC	9.19	26532	50.00%
	Ours	4.20	626	62.00%
Gorilla→Hammerhead	UAP	12.79	50173	0.00%
	NC	9.11	26854	30.00%
	Ours	4.54	643	54.00%
Treefrog→Goldfish	UAP	13.21	50175	44.00%
	NC	9.20	26714	48.00%
	Ours	4.04	551	58.00%
Treefrog→Tiger Shark	UAP	12.70	50175	14.00%
	NC	9.21	27383	28.00%
	Ours	4.12	967	58.00%
Peacock→Ostrich	UAP	13.04	50175	0.00%
	NC	9.47	29800	12.00%
	Ours	4.29	1447	54.00%
Peacock→Bulbul	UAP	12.70	50171	0.00%
	NC	9.37	28882	50.00%
	Ours	5.02	1512	62.00%
Chihuahua→Partridge	UAP	10.45	50175	0.00%
	NC	9.19	26228	36.00%
	Ours	3.77	666	54.00%
Chihuahua→Isopod	UAP	10.35	50173	8.00%
	NC	9.27	26397	26.00%
	Ours	3.69	626	56.00%



(a) #Pixels of generated triggers. The last heat map shows how much larger CW triggers are.



(b) ASRs of generated backdoors. The last heat map shows how much higher our ASRs are.

Figure 12. Comparison of CW and ours for all class pairs on SVHN. Each cell denotes the result of a generated trigger from a victim class (row) to a target class (column). The first two heat maps in each subfigure illustrate the results for CW and ours. The last shows the relative difference.

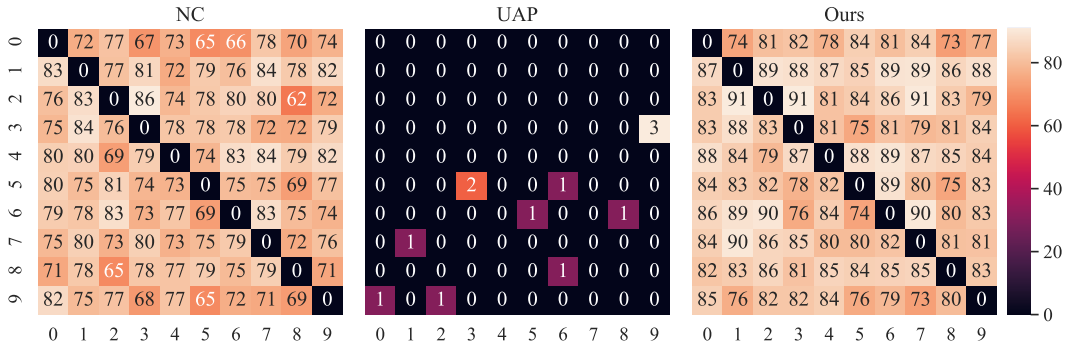
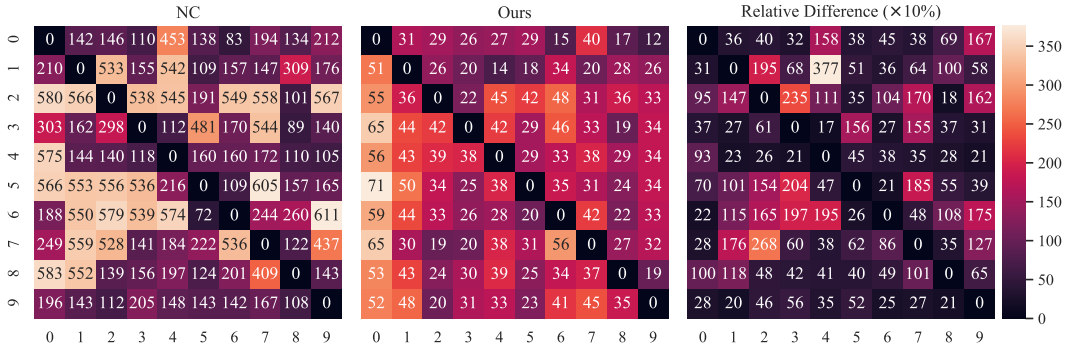
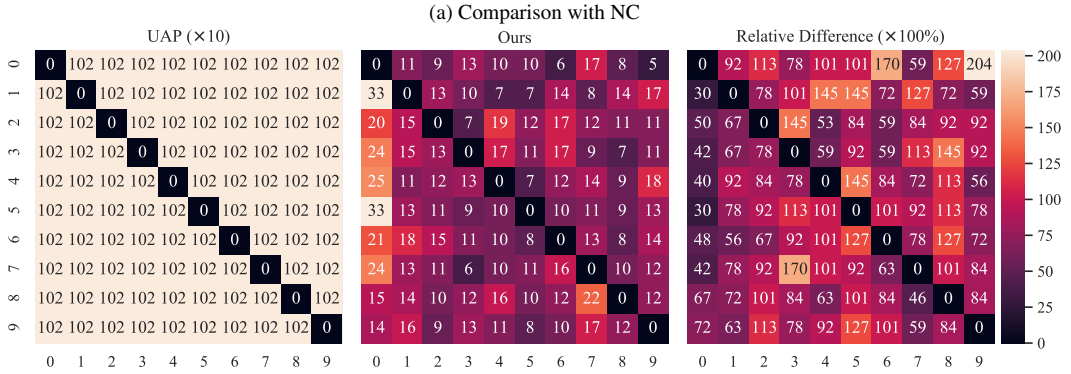


Figure 13. Comparison of NC, UAP and ours on the ASR for all class pairs on the SVHN dataset



(a) Comparison with NC



(b) Comparison with UAP

Figure 14. Comparison of the number of perturbed pixels with the same ASR for all class pairs on the SVHN dataset. The first two heat maps in each subfigure illustrate the results for NC/UAP and ours, respectively. The last heat map shows how much larger of generated backdoors by NC/UAP compared to ours.