Model Orthogonalization: Class Distance Hardening in Neural Networks for Better Security

Guanhong Tao, Yingqi Liu, Guangyu Shen, Qiuling Xu, Shengwei An, Zhuo Zhang, Xiangyu Zhang
Department of Computer Science, Purdue University
Email: {taog, liu1751, shen447, xu1230, an93, zhan3299, xyzhang}@cs.purdue.edu

2022 IEEE Symposium on Security and Privacy (SP) | 978-1-6654-1316-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/SP46214.2022.9833688

***Abstract*—The distance between two classes for a deep learning classifier can be measured by the level of difficulty in flipping all (or majority of) samples in a class to the other. The class distances of many pre-trained models in the wild are very small and do not align well with humans' intuition (e.g., classes turtle and bird have smaller distance than classes cat and dog), making the models vulnerable to backdoor attacks, which aim to cause misclassification by stamping a specific pattern to inputs. We propose a novel model hardening technique called model orthogonalization which is an add-on training step to pretrained models, including clean models, poisoned models, and adversarially trained models. It can substantially enlarge class distances with reasonable training cost and without much accuracy degradation. Our evaluation on 5 datasets with 22 model structures show that our technique can enlarge class distances by 177.63% on average with less than 1% accuracy loss, outperforming existing hardening techniques such as adversarial training, universal adversarial perturbation, and directly using generated backdoors. It reduces 80% false positives for a state-of-the-art backdoor scanner as the enlarged class distances allow the scanner to easily distinguish clean and poisoned models, and substantially outperforms three existing techniques in removing injected backdoors.**

I. Introduction

A backdoor in a deep learning model makes any inputs stamped with a specific pattern to be misclassified to a target class. While adversarial sample attack requires generating perturbations on the fly to cause an input sample, e.g., a video frame, to be misclassified, backdoor attack can have prompt effect by simply stamping a pattern. While backdoors can be injected through various methods, such as data poisoning [1]–[5], clean label poisoning [6]–[9], and neuron hijacking [10], they widely exist in naturally trained models (see Section II). We call them *natural triggers*. Natural triggers could be due to (1) the similarity between classes, e.g., a small fixed patch on any dog images can make the classifier predict cat, and (2) the model undesirably learning strong low-level features. We will show in Section II that there is a small backdoor between the turtle and bird classes even though they are unlike in humans' eyes. With the increasing applications of deep learning models in security-critical tasks such as autonomous driving [11], [12], surveillance [13], [14], access control [15], etc., backdoors are becoming a prominent security threat.

Existing defense techniques can be categorized to *backdoor scanning* that determines if a model has an injected backdoor [16]–[24], *backdoor attack detection* that determines on-the-fly if an input contains a backdoor pattern [24]–[35], and *backdoor elimination* that removes an injected backdoor [36]–[39]. Most these techniques focus on defending *injected* backdoors. For example, backdoor scanners Neural Cleanse (NC) [40] and ABS [41] rely on the assumption that injected backdoors tend to be small as they want to be stealthy. Backdoor attack detection techniques such as NIC [25] and SCAn [26] rely on the observation that an input sample stamped with a backdoor likely causes different model internal behaviors than a clean sample. Elimination techniques such as NAD [39] rely on benign samples to suppress injected backdoors. Detailed discussion of these techniques can be found in Section VII. However, they are much less effective for natural backdoors which are not injected but rather due to problems in training and even the nature of data. For example, natural backdoors cause a lot of false warnings for scanners as they cannot distinguish natural and injected backdoors [42]; a sample with a natural backdoor may likely evade detection as natural backdoors are usually benign features that may not induce abnormal internal behaviors; and using clean data cannot eliminate natural backdoors which are rooted exactly in the clean data.

Adversarial training is a widely used technique for model hardening which can force a model to unlearn unrobust (low-level) features. It aims to enforce any input undertaking adversarial perturbations in an L^p bound to be correctly classified by the hardened model. According to our studies (Section II and Section V), the improved robustness can help mitigate backdoors including natural backdoors. However, due to its L^p bounded training, which only considers local perturbations around individual samples, the protection against backdoors is limited. Also, it is known that adversarial training may cause non-trivial model performance degradation. In addition, our study shows that directly using backdoors generated by scanners to adversarially train a model does not work well due to either the extremely high computation cost or the longer convergence time. Intuitively, using a backdoor (which denotes much larger perturbations compared to those in adversarial samples) in adversarial training is like imposing a substantial displacement of the decision boundary. If not done properly, the decision boundary will oscillate. And since natural backdoors could exist in between any pair of classes, the training has quadratic complexity, which is very costly when the number of classes is large.

We propose a novel model hardening method to improve resilience to backdoors. It is an add-on training step for pre-

DOI 10.1109/SP46214.2022.00078

trained models, including adversarially pre-trained models. It substantially suppresses backdoors, including both injected and natural backdoors, making backdoor attacks more difficult (e.g., having to use much larger patterns and/or easier to be detected by scanners), with reasonable training cost and without substantial accuracy degradation. Specifically, we consider the minimal backdoor between two classes (that flips samples in a *victim class* to a *target class*) the *distance between the classes*. As such, we aim to achieve the possible maximum distances for all class pairs. Intuitively, if we project all high dimensional data points to a 2-dimensional space, *a decision boundary allows the maximum distance between two classes when it is perpendicular to the line between the centers of mass of the two classes* (visualization and more explanation in Section III). We hence call the process of finding such a decision boundary *model orthogonalization*. Our technique hardens all class pairs for a model. For each pair a and b, it repetitively generates the minimal backdoors from a to b and from b to a and adversarially trains the model with the two backdoors. In other words, it forces the model to gradually unlearn the low-level backdoor features (and focus on high-level features with more semantics). The symmetric training of the two directions of a pair substantially alleviates oscillation and improve effectiveness. To address the high computation cost induced by the quadratic complexity, our technique features a scheduler that selects a most promising pair to harden in each training round based on the potential of distance enlargement. It leverages the observation that different class pairs have different distance capacities. For example, a pair of turtle and bird has more potential than a pair of cat and dog as the latter two are so close that their distance is hard to enlarge no matter how much training effort is spent. A number of methods are further used to speed up the training process such as reusing backdoors from previous rounds and dynamically adjusting bounds. Details are in Section IV.

Our contributions are summarized as follows.

- We intuitively and formally define the problem of model orthogonalization, which is an add-on training step. As part of it, we define the distance between a pair of classes, which serves as the basis for the hardening process.
- We devise a new training process to achieve orthogonalization. It features symmetric training (training the two directions of a pair together), pair scheduling, and a few other designs.
- We develop a prototype MOTH (Model ORTHogonalization). Our evaluation on four standard datasets and six different model structures shows that MOTH can improve class distance for naturally trained models by 119.87% and adversarially trained models by 52.87% with less than 1% accuracy degradation on average. With similar hardening performance, our technique is 9x faster than a baseline that does not use scheduling. It achieves 29.72% more distance improvement than a baseline that does not perform symmetric training. It can achieve 95.80% more distance improvement and 2.58x faster than using *universal adversarial perturbations* [43]. We also conduct experiments on 30 pre-trained models downloaded from the TrojAI competition [42], a competition for backdoor scanning. MOTH improves the class distance by 232.39% over the original models and is 11x faster than the baseline. We apply MOTH in two applications including reducing false positives for backdoor scanning and eliminating injected backdoors in existing models. It can reduce false positives by 81.25% for the first application. Regarding the second application, the attack success rate (ASR) of injected backdoors is reduced (by orthogonalization) from almost 100% to 1% on average, outperforming three state-of-the-art backdoor elimination approaches with the best performance of reducing ASR to 26.75% on average. MOTH is publicly available at [44].

Threat Model. We consider backdoors between individual pairs. That is, a backdoor can flip samples from a victim class to a target class, called *label-specific backdoor* in the literature [40], [41], [45], which is more general than *universal backdoor* that flips any sample of any class to a target class. We consider backdoors that are either injected (in a poisoned/hijacked model) or naturally present (in a clean model). We consider both are equally harmful. Our goal is to enlarge class distances such that it is more difficult to find backdoors, without sacrificing much accuracy. That is, to launch attack, the attacker needs to use a large pixel pattern that may already possess a lot of semantic features of the target class.

In this paper, we only consider static backdoors, in which the backdoor patterns are input agnostic, like patch backdoors [1]. There are dynamic backdoors such as reflection backdoors [3], composite backdoors [4], and feature space backdoors [46]. We conduct a preliminary study on a few dynamic backdoors and MOTH can reduce the ASR to some extent (see Section VIII). We will leave more exploration to our future work. We argue that our contributions are still very valuable because model hardening for static backdoors is still an open problem. Finally, we assume only a subset of the original training dataset (5%) is available when MOTH is used to remove injected backdoors.

II. MOTIVATION

Figure 1 shows sample images from a normally trained ImageNet model downloaded from a widely-used model repository [47] and its natural backdoors (derived using NC [40]). The first column shows the backdoors, with the first row flipping dog images to cat, the second one turtle to bird, and the third one cat to bird. The second and third columns show the victim class samples; the fourth and fifth columns the victim class samples stamped with the backdoor patterns; the sixth column the target class samples, and the last column the size of backdoor in terms of the aggregated pixels in the R, G and B channels (i.e., L^1 norm). For instance, a trigger of size 615 (second row) has around $615/3 = 205 \approx 14 \times 14$ changed pixels. This is a very small backdoor compared to an input image of 224×224 pixels for ImageNet models. In other words, the model is very vulnerable, even though it is not poisoned by dirty data. In fact, over 90% of the samples

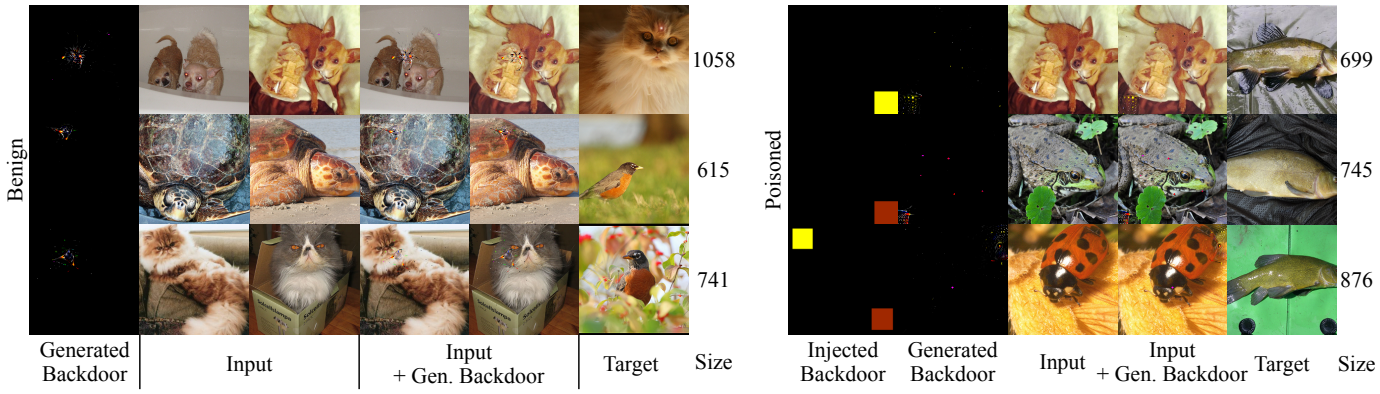


Fig. 1: Natural backdoor versus injected backdoor. The left figure shows natural backdoors (1st column) generated by a model scanner (i.e., NC) for a benign ImageNet model. The right figure presents three poisoned models with injected backdoors with different colors/shapes (1st column). Columns Size denote the size of generated backdoors.

in the victim classes are misclassified to the target class when they are stamped with the small backdoors. Also observe that these backdoors seem to be very low level features that do not constitute any meaningful features in humans' eyes. The model might have overfitted on these low level (strong) features of the target class, causing the natural backdoors. We observe that the backdoor sizes do not meet our intuitions of the distances of these classes. For example, intuitively cat and dog have more similarity than turtle and bird. However, this is not reflected by the sizes of natural backdoors. In some sense, we can say the model is not "*orthogonal*" (more discussion in Section III).

Similar information is presented for three poisoned ImageNet models from [41]. Each model is poisoned in a way that all samples of the victim classes (the third column) stamped with the backdoors in the first column are misclassified to the target class as shown in fifth column. While the models were poisoned with the backdoors shown in the first column, the models overfit on some low-level features of the backdoor patterns used in poisoning such that after poisoning, the injected backdoors are just the small pixel patterns shown in the second column. The last column shows the backdoor sizes. Observe that they are not distinguishable from the sizes of natural backdoors. In the TrojAI multi-round competitions for backdoor scanning organized by IARPA [42], regularly trained clean models have a large number of small natural backdoors whose sizes are not distinguishable from the injected ones in the round 2 competition, which is a round dedicated to finding backdoors in image classification models. As a result, many performers suffered from a large number of false positives (i.e., reporting a clean model as poisoned). On the other hand, if a pretrained model used the hardening technique proposed in the paper and had the hardened class distances published as part of the model specification, the substantially enlarged distances would make stealthy poisoning attempts of the hardened model easily detectable (Section V-C).

Existing Technique I: Adversarial Training. Adversarial training is the most widely used model hardening technique. It aims to train a subject model in such a way that samples in a L^p bound of each training input have the same classification result, with the sacrifice of some classification accuracy. It can move the decision boundary to make the model robust. It can enlarge class distances. In fact, since round 3 of the TrojAI competition, the red team (by NIST) responsible for producing the clean and poisoned models for performers to classify uses adversarial training to suppress natural backdoors. In Figure 2, we show some backdoors for an adversarially trained ImageNet model (downloaded from existing work [48]) in the second column. We use the same three class pairs as in Figure 1, with each pair taking two rows. The natural backdoors are in the second column with their sizes presented on the top of the backdoor images in red in the odd rows. The backdoors are also enlarged in the even rows. The classification confidence of a stamped sample (in the third column) is depicted on the sample.

Comparing to the class distances in Figure 1, it is evident that adversarial training can enlarge class distance (e.g, from 1058 to 1598 for the dog→cat pair). In addition, the backdoors start to possess some human perceptible features. For instance, the backdoor for dog→cat resembles a cat face and the backdoor for turtle→bird has the beak of bird. However, due to the nature of adversarial training, the accuracy has nontrivial degradation (see Section V). Moreover, the order of class distances still does not align well with our intuitions of the two class pairs dog→cat and turtle→bird, indicating that it may not have achieved the maximum class distances. More discussion of the reasons of such insufficiency can be found in Section III.

Existing Technique II: Universal Adversarial Perturbation. While adversarial training generates adversarial perturbations separately for each sample, *universal adversarial perturbation* (UAP) aims to derive common adversarial perturbations for a (large) set of samples from different classes such that the derived UAPs can cause misclassification when they are applied to any samples. Similar to adversarial training, UAPs are usually derived using L^∞. As such, a straightforward idea is to use UAP to adversarially train a model to harden it like in [43]. Figure 3 shows the results. Observe that the derived UAPs in column 1 have noise-like pixel patterns. UAPs can enlarge class distances (e.g, from 1058 to 1118 for the dog→cat pair) to some extent. However, the distance

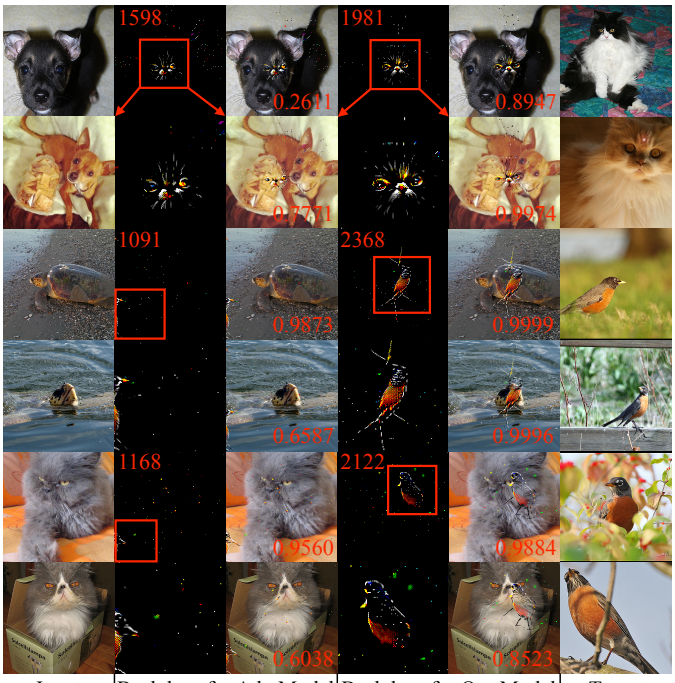


Fig. 2: Adversarial training vs. our training. The 2nd and 4th columns present the generated backdoors for an adversarially trained model and our model, respectively. The patterns highlighted in red boxes are enlarged in even rows. The sizes of backdoors are marked on the top in red. The numbers in 3rd and 5th columns are prediction confidences to the target labels.

enlargement is small. This is because UAPs are untargeted, meaning that a UAP for a victim class may flip samples of the class to different target classes. Also note that due to its use of L^∞ bound, targeted UAPs are very difficult to derive. They either cannot be found within the bound or the enlarged bound leads to substantial accuracy degradation after hardening. Natural backdoors, on the other hand, do not have any constraints on the magnitude of perturbations. They can easily achieve a high ASR for a specific target, which can reveal vulnerabilities of the decision making between two classes of a subject model.

Technique III: Directly Using Generated Backdoors in Training. Another method is to directly use optimization to derive the smallest backdoor between a pair of classes, e.g., by adapting optimizations in NC [40] and ABS [41]. The training is enhanced such that any samples stamped with the backdoor should retain their ground truth classifications. As we will show in Section V, such a method is expensive as its complexity is quadratic. In addition, its training loss fluctuates a lot (see Section IV-B), causing inferior results in distance improvement compared to ours (with 30% performance difference as shown in Section V-D). Note that the Pairwise baseline evaluated in Section V is already more sophisticated than simply using generated backdoors in hardening as described above. It leverages symmetric hardening and speedup methods discussed later in this section and in detail in Section IV.

Our solution. We propose a novel training method that can

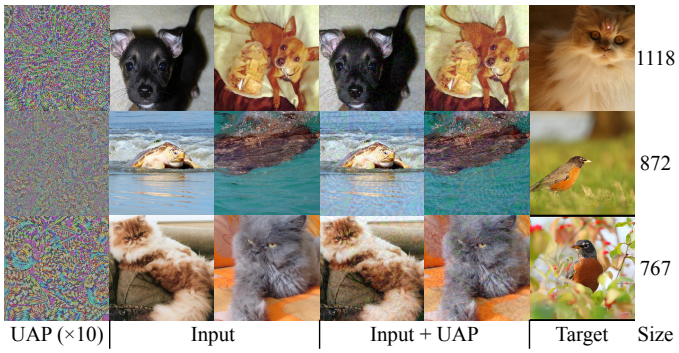
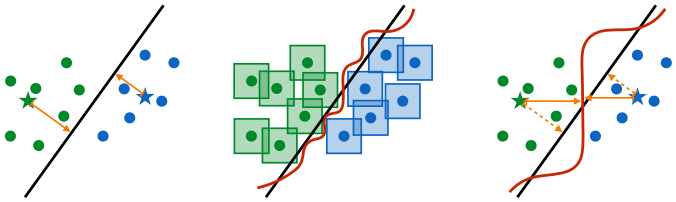


Fig. 3: Universal adversarial perturbation. The 1st column shows the generated UAPs for three different victim classes in the 2nd and 3rd columns. The last column presents the sizes of backdoors generated for a hardened model by UAP training.

effectively enlarge class distances without sacrificing much accuracy. We call it *model orthogonalization* (see Section III). Different from adversarial training that independently perturbs individual inputs, our training considers individual pairs of classes. Specifically, for a pair such as a and b, it derives a minimal backdoor from a to b and another backdoor from b to a. Intuitively, the former can be considered the distinctive (low level) feature of b with respective to a, and vice versa. It then stamps samples with these backdoors and ensures the classification results do not change, which essentially expels these low level features and forces the model to learn high level and more semantic features distinguishing the pair. The two directions of a pair are hardened together, which we call the *symmetric hardening*. As will be discussed in Section IV-B, asymmetric hardening, i.e., hardening only one side or hardening the two sides in separate batches, leads to inferior results. Considering each class pair uniformly (with a total of $\mathcal{O}(n^2)$ pairs for an n-class model) is not cost-effective as different class pairs have different distance capacities. Some pairs quickly reach their maximum distance (e.g., cat and dog) and others need more training to get there. We hence develop a scheduler that schedules the most promising pairs for each batch, substantially improving cost-effectiveness. The training also features a few special designs such as gradually growing optimization bounds and reusing backdoors to speedup the process. Details are in Section IV. The fourth and fifth columns of Figure 2 present example results using our training (on the adversarially trained model). The class distances are substantially enlarged without further accuracy degradation. The distances are much larger compared to the regularly trained model in Figure 1. With hardened distances, the data poisoning in Figure 1 can be easily detected, as the compromised classes would have much smaller distances than the hardened ones. Further, we observe that the order of the distances of the three pairs aligns well with our intuition after hardening. The zoomed-in trigger patterns in the even rows clearly exhibit high level semantic features of the target classes (e.g., the whole body of a bird). In other words, the model learns high level features of individual classes, which substantially mitigates its vulnerabilities to (natural) backdoors.



(a) Natural Training (b) Adversarial Training (c) Our Training

Fig. 4: A conceptual illustration of standard vs. adversarial vs. our decision boundaries. Green and blue dots are samples of two classes. The black straight line is the standard decision boundary separating these two sets of samples. The orange arrows in (a) and (c) denote the backdoor transformation that can move a set of samples to the other side of the boundary. The red curly line in (b) is the adversarial decision boundary. The red line in (c) is the decision boundary after our training.

III. Problem Definition

Intuitively, a backdoor from a victim class to a target class is a (uniform) transformation function that transforms an input of the victim class such that it is classified to the target class by the subject model. A backdoor ought to be stealthy otherwise it can be easily detected by automatic techniques or humans. For instance, replacing majority of a victim class input with a target class input likely causes the intended misclassification. However, such a "*large* and *obvious*" backdoor can hardly constitute a meaningful attack. The level of stealthiness is proportional to the magnitude of the transformation. One way of measuring such magnitude is to use L^p distance between the original sample and its transformed version.

As demonstrated in Section II, even clean models are vulnerable to *natural backdoors*. Such backdoors are so small that stealthy attacks can be easily conducted. In these attacks, the L^p distances introduced by the (uniform) transformations are small. Adversarial training can enlarge the L^p norm of the needed backdoor transformation but it cannot achieve the maximum distance. Intuitively, we define *the L^p norm of the minimum backdoor from the victim class to the target class their class distance*. Note that such distance is *directional* and usually *asymmetric*, that is, the distance from a to b is different from that from b to a. Our goal is hence to enlarge class distances such that models become less vulnerable to backdoor attacks without sacrificing classification accuracy.

Figure 4(a) illustrates the intuition. The blue and green points are samples of two classes and the black line denotes the decision boundary by the subject model that achieves the minimum cross-entropy loss. The orange arrows denote the backdoor transformations. Intuitively, most the green points undertaking the displacement denoted by the arrow starting from the green star (a.k.a. the center of mass of green points) fall into the other side of the decision boundary. Similarly, most the blue points can be moved to the other side by the arrow starting from the blue star. Observe the two arrows have different lengths. Figure 4(b) shows that adversarial training allows the L^p ball of each sample to have consistent classification results. Consequently, the decision boundary is altered as shown by the red line. It enlarges the arrow lengths. Figure 4(c) shows our technique. The red curve denotes the new decision boundary after our training, with which the solid arrows are much longer than before (the dashed arrows). We say *a classifier is orthogonal when all its class pairs have the maximum distance* and the training to achieve this effect *model orthogonalization*. Intuitively, as shown by Figure 4(c), in order to maximize both the distances from a to b and from b to a, the decision boundary tends to be perpendicular to the line linking the two centers, namely, orthogonal.

An important observation is that maximum class distances are bounded and different class pairs may have completely different distances. As demonstrated in Section II, the maximum distance from cat to dog is smaller than that from turtle to bird. This is determined by the nature of these classes. As such, while we can use special training methods like the one proposed in this paper to enlarge such distances, the enlargement is bounded by the natural differences of classes.

Next, we formally define these concepts. Considering a typical classification problem, where the samples $\boldsymbol{x} \in \mathbb{R}^d$ and the corresponding label $y \in \{0, 1, \ldots, n\}$ jointly obey a distribution $\mathcal{D}(\boldsymbol{x}, y)$. A classifier $\mathcal{M} : \mathbb{R}^d \to \{0, 1, \ldots, n\}$ with parameter θ is supposed to satisfy the following property $\arg\max_\theta P_{(\boldsymbol{x},y)\sim\mathcal{D}}[\mathcal{M}(\boldsymbol{x};\theta) = y]$.

Definition III.1. Given two classes a and b, the backdoor from a (the victim class) to b (the target class) is a transformation $\mathbb{T} : \mathbb{R}^d \to \mathbb{R}^d$ that transforms any sample of a, denoted as for all $(\boldsymbol{x}, y = a)$, such that $\mathcal{M}(\mathbb{T}(\boldsymbol{x});\theta) = b$. A backdoor is **stealth** if $\|\mathbb{T}(\boldsymbol{x}) - \boldsymbol{x}\|$ is small.

A backdoor could be natural (when the model is normally trained) or injected (e.g., by data poisoning [1], [3]–[5]). A backdoor may be *dynamic*, meaning that the perturbations are different for different inputs (e.g., feature space backdoors [46]), or *static*, meaning that the perturbations are input agnostic, like patch backdoors [1]. In this work, we focus on static backdoors.

Definition III.2. Given two classes a and b, the distance from a to b, denoted as $\|a \to b\|$, is $\min_{\mathbb{T}}(\mathbb{E}(\|\mathbb{T}(\boldsymbol{x}) - \boldsymbol{x}\|))$ with $(\boldsymbol{x}, y = a)$ a sample of a and $\mathbb{T}$ a backdoor from a to b.

Intuitively, class distance is determined by the smallest backdoor. In practice, the expectation $\mathbb{E}(\|\mathbb{T}(\boldsymbol{x}) - \boldsymbol{x}\|)$ is approximated by a set of samples. Note that for static backdoors, $\|\mathbb{T}(\boldsymbol{x}) - \boldsymbol{x}\|$ is constant for different samples (i.e., only a property of the two classes).

Our definition of class distance differs from existing class separation notions [49], [50] in two ways: (1) the distance measure in this work is carried out for a set of samples, which measures the distance from the center of sample mass to the decision boundary as shown in Figure 4(c), whereas the measures in existing works are performed for every sample to obtain the smallest distance from a sample to the decision boundary (the margin). The notion of class distance is intrinsically related to the underlying trigger inversion method, which is modular, meaning any suitable inversion method can be leveraged for measuring class distance; (2) our class

distance does not measure the robustness of models under input-specific adversarial attacks. As class distance does not consider the smallest distance for every sample, the hardened model does not defend against adversarial examples. As shown in Figure 4(c), there are blue and green dots that are close to the decision boundary (after our hardening) and hence the model may still be vulnerable to traditional adversarial attacks. We consider the two kinds of measures complementary.

Definition III.3. Model orthogonalization aims to derive a classifier $\overline{\mathcal{M}} : \mathbb{R}^d \rightarrow \{0, 1, \ldots, n\}$ with parameter $\overline{\theta}$ such that $\arg\max_{\overline{\theta}} P_{(\boldsymbol{x},y)\sim\mathcal{D}}[\overline{\mathcal{M}}(\boldsymbol{x};\overline{\theta}) = y]$ and $\arg\max_{\overline{\theta}} \sum_{(a,b)} \|a \rightarrow b\|$.

Intuitively, model orthogonalization aims to derive a classifier that has the highest accuracy and the largest aggregated class distance. While the first condition can be ensured by a cross-entropy loss, the second condition cannot be easily represented as a loss function as it entails deriving the distance of each pair of classes, having quadratic complexity. In the following sections, we explain the design of MOTH.

IV. DESIGN

A. Overview

Figure 5 illustrates the whole training procedure. Given a pre-trained model and training data, MOTH first initiates a warm-up phase, where each class is treated as the target class for generating universal backdoors (the top rectangle). That is, for a set of samples not in the target class, we aim to generate a backdoor that can flip their predictions to the target class. During the optimization process for each target class, the changes of loss for different source classes are recorded for updating a prior-selection matrix (the bottom rectangle). Details are discussed in Section IV-C. Once all the classes are warmed up, we start the main training process. This phase consists of four steps as shown in Figure 5. Based on the prior-selection matrix, step ① uses a K-arm scheduler for selecting a promising class pair that will likely have the largest class distance increase. Assume the model has N classes. We create $N \times (N-1)/2$ arms (i.e., all the pairwise combinations without direction). Intuitively, an arm represents the optimization objective for a pair. The scheduler then selects the most promising arm. In step ②, MOTH applies a symmetric backdoor generation algorithm (for the selected pair) to yield two backdoors, which are stamped on samples of the classes. The modified training batch is used for updating model parameters according to the cross-entropy loss. The entries for the selected pair in the post-selection matrix are updated in step ③ to record the distance variations. For the next training round, we rely on a reward function (see Section IV-C) that combines both the prior-selection and post-selection matrices for class pair selection (step ④). MOTH iterates this process. It terminates when the model accuracy degradation reaches a preset bound. Finally, it outputs the hardened model that has larger class distances. We discuss individual components and algorithms in detail in the following sections.

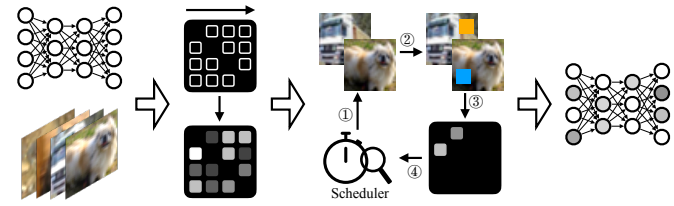
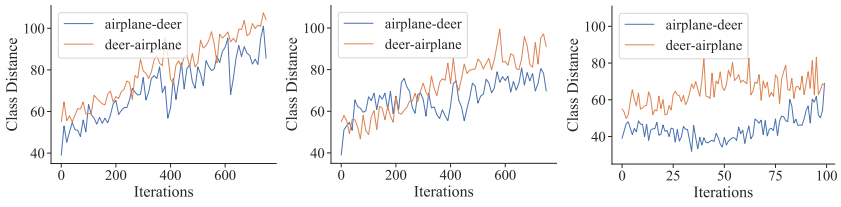


Fig. 5: MOTH overview

B. Symmetric Hardening

For a class pair a and b, there are two directions for generating backdoors. The goal of class distance hardening is to enlarge distance by training with the generated backdoors. A straightforward design is to leverage a backdoor generation method (e.g., NC) to produce a backdoor for one direction (e.g., from a to b), and stamp it on the samples of class a for training. In the next iteration, it randomly chooses another pair direction for training. However, such a design, we call it *asymmetric hardening*, can lead to undesirable oscillation of class distances during training. Figure 6b shows the class distances of airplane and deer using such a design. The x-axis denotes the training iteration, and the y-axis denotes the class distance. Observe that between iteration 250 and 400, the class distance from airplane to deer decreases while the other direction continues to grow. We show a closer look for this period in Figure 6c. It is evident that asymmetric hardening causes the two directions of a pair competing and leads to oscillation of class distances. As such, we propose a *symmetric hardening* method, where we harden the two directions of a pair simultaneously. In Figure 6a, observe that the symmetric hardening has continuous growth for both directions. We also evaluate class distances of all pairs using symmetric hardening and asymmetric hardening, and compare them with those of the original model. Figure 7 illustrates the differences of class distances for the two hardening methods. The heat map on the left is for symmetric hardening and the right for asymmetric hardening. Each cell in the heat map denotes the class distance improvement from a source class (row) to a target class (column). The brightness of the color denotes how much of the class distance is increased compared to the original model (the brighter the larger). We can clearly see bright colors for most cells in the symmetric heat map. The asymmetric heat map has many cells with dark colors. In addition, the distances from other classes to a class may have large improvement (e.g., the penultimate column of the ship class), but not the other direction (e.g., the penultimate row). As illustrated in Figure 4(c), the hardening procedure aims to push the decision boundary towards the opposite side. If only one direction is hardened at one time, the decision boundary will skew towards one side. It is hence pushed back and forth through multiple rounds, causing oscillation as demonstrated in Figure 6b.

Many existing backdoor generation methods follow an optimization procedure like NC [40]. It aims to flip a set of samples from a victim class to the target class, which can be



(a) Symmetric (b) Asymmetric (c) Zoom in for asymmetric

Fig. 6: Symmetric versus asymmetric hardening

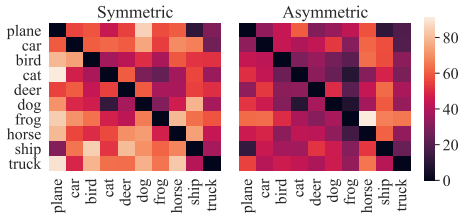


Fig. 7: Symmetric versus asymmetric hardening on all class pairs

formalized as follows.

$$\forall \boldsymbol{x} \in \boldsymbol{X},\ \min_{\boldsymbol{m},\boldsymbol{\delta}} \mathcal{L}\big(\mathcal{M}(\boldsymbol{x}'), y_t\big) + \lambda \cdot \|\boldsymbol{m}\|_1, \tag{1}$$

$$\mathbb{T}(\boldsymbol{x}, \boldsymbol{m}, \boldsymbol{\delta}) = \boldsymbol{x}' = (1 - \boldsymbol{m}) \cdot \boldsymbol{x} + \boldsymbol{m} \cdot \boldsymbol{\delta}, \tag{2}$$

where $\mathcal{L}(\cdot)$ is the loss function of model $\mathcal{M}(\cdot)$; y_t is a target label different from the ground truth of input $\boldsymbol{x}$; λ adjusts the weight of the second objective; $\mathbb{T}(\cdot)$ is the transformation that applies the generated backdoor to an input $\boldsymbol{x}$; $\boldsymbol{m}$ is a mask matrix whose values range from 0 to 1; $\boldsymbol{\delta}$ is the backdoor, which has the same shape and value range as input $\boldsymbol{x}$. A straightforward way for symmetric hardening is to run some existing backdoor generation method twice. However, this will, on one hand, introduce an extra optimization step that doubles the training cost, and on the other hand, fail to consider both directions such that the generated backdoors may be too aggressive in pushing the boundary, causing oscillation. We hence devise a *two-sided backdoor generation* method that generates two backdoors for a class pair at once. Algorithm 1 illustrates our method. Input parameter $\boldsymbol{X}$ denotes a set of inputs belonging to class a or b and $\boldsymbol{p}$ is an indicator vector of these inputs, with 1 denoting class a and 0 class b. We initialize the backdoor variables with the input values if available or random values otherwise (lines 2-6). In line 7, the target labels are set to the opposite of ground truth labels (of the class pair). During the optimization, backdoors are stamped on the samples of the corresponding class (lines 10-11). Line 13 computes the loss for both directions together. This is the key as the backdoor generation along either direction is constrained by loss from both directions, avoiding any bias to one side. Attack success rates are computed for individual classes (line 14). For each backdoor, we check whether it reaches the desired attack success rate and also has a smaller size compared to the previous result. If so, we record the best results and apply random perturbation ($\xi = 0.01$) to backdoor variables to avoid local minima (line 16-20).

Algorithm 1 Two-sided Backdoor Generation

1: **function** BIGENERATION(model M, data $\boldsymbol{X}$, label (a, b), class $\boldsymbol{p}$, initialization $(\boldsymbol{m}_{init}, \boldsymbol{\delta}_{init})$)
2: **if** $\boldsymbol{m}_{init}$ is not None **and** $\boldsymbol{\delta}_{init}$ is not None **then**
3: $\boldsymbol{m}, \boldsymbol{\delta} \leftarrow \boldsymbol{m}_{init}, \boldsymbol{\delta}_{init}$
4: **else**
5: $\boldsymbol{m}, \boldsymbol{\delta} \leftarrow$ random init with twice shape of $\boldsymbol{x} \in \boldsymbol{X}$
6: **end if**
7: $\boldsymbol{y}_t = \boldsymbol{p} \cdot b + (1 - \boldsymbol{p}) \cdot a$
8: **for** $step$ **in** $0 \ldots max\_steps$ **do**
9: $\boldsymbol{X}_n \leftarrow$ a batch of $\boldsymbol{x} \in \boldsymbol{X}$ ▷ use all $\boldsymbol{x}$ for simplicity
10: $\boldsymbol{X}'_n = \boldsymbol{p} \cdot \big((1 - \boldsymbol{m}[0]) \cdot \boldsymbol{X}_n + \boldsymbol{m}[0] \cdot \boldsymbol{\delta}[0]\big)$
11: $+(1 - \boldsymbol{p}) \cdot \big((1 - \boldsymbol{m}[1]) \cdot \boldsymbol{X}_n + \boldsymbol{m}[1] \cdot \boldsymbol{\delta}[1]\big)$
12: $\boldsymbol{y}'_n = M(\boldsymbol{X}'_n)$
13: minimize the loss defined in Equation 1
14: $acc = [\boldsymbol{p} \cdot \text{equal}(\boldsymbol{y}'_n, \boldsymbol{y}_t), (1 - \boldsymbol{p}) \cdot \text{equal}(\boldsymbol{y}'_n, \boldsymbol{y}_t)]$
15: **for** i **in** $\{0, 1\}$ **do**
16: **if** $acc[i] >= asr$ **and** $\|\boldsymbol{m}[i]\|_1 < size[i]$ **then**
17: $size[i], \boldsymbol{m}_{best}[i], \boldsymbol{\delta}_{best}[i] \leftarrow \|\boldsymbol{m}[i]\|_1, \boldsymbol{m}[i], \boldsymbol{\delta}[i]$
18: $\boldsymbol{m}[i] = \boldsymbol{m}[i] + \text{Uniform}(-\xi, \xi)$
19: $\boldsymbol{\delta}[i] = \boldsymbol{\delta}[i] + \text{Uniform}(-\xi, \xi) \cdot 255$
20: **end if**
21: **end for**
22: **end for**
23: **return** $\boldsymbol{m}_{best}, \boldsymbol{\delta}_{best}$
24: **end function**

C. Pair Scheduling

A naïve method is to train each pair to their maximum distance in turn. However, this often leads to substantial accuracy degradation. Another simple method is to do it iteratively and randomly by selecting a pair in each iteration. However, this is suboptimal as well because different pairs have various capacities. For instance, deer and horse are close by their nature, spending a lot of optimization cycles on this pair may not be as productive as spending on deer and bird. As such, we prioritize the promising pairs using a scheduler. We say a pair is promising if the return (i.e., the growth of class distance) of spending a fixed number of optimization cycles is larger than others. To validate our hypothesis, we select three class pairs and study their distance changes during training. We use the CIFAR-10 dataset and a naturally trained ResNet20 model as our subject. Figure 8 shows the results. Observe that for a close pair like deer and horse (the orange line in the figure), the class distance does not grow much over time. It is hence not cost-effective to spend a lot of cycles on this pair. In contrast, the class distance of bird→horse (the blue line), increases from 37 to 61 within 38 iterations. Such a pair is more cost-effective as its class distance can be improved quickly. We also observe that after iteration 38, there is no further improvement (or degradation) for bird→horse, meaning it reaches the maximal. Continuous training this pair does not bring in more benefits. Our scheduler predicts the potential of a pair on-the-fly based on its past performance. We discuss details in the following.

Warm-up. Initially, there is not enough information to predict promising pairs. We hence generate backdoors for all the pairs first before further selective training. A straightforward idea is to warp up all class pairs by running a few rounds of optimization for each pair. However, such an idea is expensive due to the quadratic complexity. We hence resort to an approximate approach, which only has a linear complexity.

Specifically, instead of optimizing individual class pairs, we leverage a universal backdoor generation step to probe the

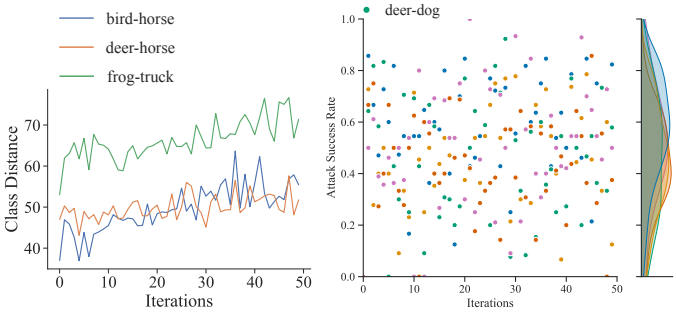


Fig. 8: Class distance change over training iterations

Fig. 9: Attack success rate of prior backdoors

potential for each source class to a target class. The universal trigger generation aims to generate a trigger that can flip samples of all the classes (other than the target class) to the target class y_t. Formally,

$$\mathcal{M}(\boldsymbol{x}_i') = \mathcal{M}(\mathbb{T}(\boldsymbol{x}_i, \boldsymbol{m}, \boldsymbol{\delta})) = y_t \neq y_i, \forall \boldsymbol{x}_i \in \boldsymbol{X}, \quad (3)$$

where $\boldsymbol{x}_i'$ is acquired by optimizing Equation 1 and y_i is the ground truth label of input $\boldsymbol{x}_i$. Since the universal trigger applies to all the source classes, the optimization needs to find a backdoor that can flip the hardest class, which is far from the target class. As such, this pair has good potential and should be prioritized. To measure the level of difficulty of forcing a class to the target class, we leverage the change of loss during the optimization. Classes distinct from the target will have large initial loss values as they are quite different from the target class. Once the predictions of all the samples are successfully flipped, the loss values are small for all classes. We use a set of samples for each source class to approximate their distance to the target class, which is denoted in the following.

$$u_{s \to t} = \frac{1}{m} \sum_{i, y_{i,s} \neq y_t}^{m} l_i^{init} - l_i^{final}, \quad (4)$$

where s and t are the source and target classes; m is the number of samples; l_i^{init} and l_i^{final} are the initial and final loss values for sample i, whose label $y_{i,s}$ is different from the target class y_t. Here, a cross entropy loss function is adopted for calculating the loss value. We set the number of samples $m = 10$ unless stated otherwise. The warm-up phase repeats the above process for each class (i.e., considering it the target class) and computes a prior-selection matrix. We use $\boldsymbol{U}$ to denote the matrix. Each entry represents the loss change computed by Equation 4. For instance, the entry at row 0 and column 1 of matrix $\boldsymbol{U}$ denotes the loss value change from class 0 to class 1, i.e., $\boldsymbol{U}[0, 1] = u_{0 \to 1}$. All the entries on the diagonal are initialized with $-\infty$. This prior-selection matrix will be used for pair selection in the later steps.

Scheduling. We introduce a K-arm scheduler [51] for pair selection. As discussed earlier, we use the variation of distance to identify promising pairs. Assume there are N classes. We then create $K = N \times (N-1)/2$ arms (i.e., all the pairwise combinations without direction). At each training iteration, the scheduler selects a promising arm to optimize. Our goal is to prioritize pairs that have fast class distance improvements. As such, we use local class distance changes in a batch to guide selection (since we only have a batch of samples at each iteration). The local distance may not fully align with the global distance (over all examples), inducing uncertainty during arm selection. To handle such an issue, we leverage the ϵ-greedy algorithm [52] to introduce randomness in our selection. Specifically, a random sample is drawn from a uniform distribution over $[0, 1)$. If the sample is greater than a threshold ϵ, the selection is based on an objective function; otherwise, a random arm (class pair) is selected. Formally, we have the following scheduling policy.

$$P = \begin{cases} \arg\max_{(a,b)} \mathcal{W} & , s > \epsilon \\ \text{rand}(K) & , s < \epsilon \end{cases}, \text{ with } s \sim \mathcal{U}(0, 1), \quad (5)$$

where P is the selected class pair. $\mathcal{W}$ is the objective function for selecting the promising class pair. ϵ determines the level of randomness. Equipped with this ϵ-greedy method, even if a globally-promising but not locally-promising pair is not selected in the early stage, it can still be picked up in the following iterations with a probability of ϵ. We set $\epsilon = 0.3$. The objective function $\mathcal{W}$ is the combination of two components: prior-selection matrix $\boldsymbol{U}$ and post-selection matrix $\boldsymbol{V}$. The prior-selection matrix $\boldsymbol{U}$ stores the loss changes in the warp-up phase as discussed earlier. The post-selection matrix $\boldsymbol{V}$ monitors and records the class distance change for every pair during training. As shown in Figure 8, some pairs have a large distance increase within a few iterations and we aim to prioritize those pairs. As such, we select a class pair that has the largest increase of class distance between two iterations. From the figure, observe that the class distance oscillates during training even for a pair with great potential. It indicates that only looking at the difference between two iterations, the scheduler will miss class pairs with potential. We hence consider an accumulated class distance change with exponential decay for early changes as follows.

$$v_{s \to t} = \sum_{i=1}^{q} (\frac{1}{2})^{q-i} \cdot \frac{d_{s \to t}^{i} - d_{s \to t}^{i-1}}{d_{s \to t}^{i-1}}, \quad (6)$$

where q is the number of times a pair being selected up to the current iteration; $d_{s \to t}^{i}$ is the L^1 norm of the backdoor mask matrix $\boldsymbol{m}$ from a source class s to a target class t at iteration i, denoting their distance. We use the sizes of universal backdoors during warm-up phase as the initial $d_{s \to t}^{0}$. Entries in the post-selection matrix $\boldsymbol{V}$ are updated every iteration with Equation 6. The meaning of each entry in $\boldsymbol{V}$ is similar to $\boldsymbol{U}$, where rows denote source classes and columns denote target classes. For instance, $\boldsymbol{V}[0, 1] = v_{0 \to 1}$ denotes the accumulated class distance change from class 0 to class 1. At the early stage of training, since we have not explored many class pairs, we rely on prior-selection matrix $\boldsymbol{U}$ as guidance for selecting pairs. With the training iteration grows, more class pairs are explored, whose distance changes in $\boldsymbol{V}$ represent

their priority of being selected. Hence, the objective function $\mathcal{W}$ combines these two together as follows.

$$\mathcal{W} = (1-\alpha)\cdot \boldsymbol{U} + \alpha \cdot \boldsymbol{V}, \quad \text{where } \alpha = \min(\frac{i-n}{1000}, 1). \quad (7)$$

Parameter α controls how much the scheduler relies on the information from the two phases; i is the number of training iterations and n is the number of classes. Since the warm-up phase goes through n target classes (in n iterations), we exclude those iterations in the objective function.

D. Speeding-up Training

The hardening procedure discussed earlier requires generating minimal backdoors for stamping on training samples. Usually, it takes several hundred steps of optimization to obtain a minimal backdoor for a set of samples. Our scheduler reduces the number of iterations for training as it proactively prioritizes promising pairs. However, it still suffers from high training cost due to the need of generating a minimal backdoor for each iteration. Recall that the backdoor generation aims to find a minimal mask $\boldsymbol{m}$ and a pattern $\boldsymbol{\delta}$ that can induce misclassification to the target class for a set of samples (see Equation 1 and 2 in Section IV-B). There are two objectives in Equation 1, which are the cross entropy loss and the L^1 norm of $\boldsymbol{m}$. The first objective is to induce misclassification and the second to obtain a minimal backdoor. If the first objective can be satisfied, the optimization will then focus on reducing the size of a backdoor. The common practice of generating backdoors starts from a randomly initialized mask $\boldsymbol{m}$ and pattern $\boldsymbol{\delta}$, which usually has low or zero attack success rate (ASR). It means the backdoor generation needs to spend time on the first objective to achieve a high ASR, which slows down the process for minimizing the backdoor size. If there exists a backdoor that has a reasonable ASR, the optimization can quickly concentrate on reducing the backdoor size.

We study the ASR of backdoors generated during training. Particularly, we are interested in the performance of prior backdoors (backdoors generated in the previous training iterations). Figure 9 shows the ASR of backdoors for five class pairs. The x-axis denotes the training iteration, and the y-axis denotes the ASR. Each dot in the figure denotes the ASR of applying a backdoor *from the last iteration* on samples in the current batch. The figure on the right shows distributions of ASR for different pairs. Observe that many dots have higher than 0.4 ASR. It is clearer from the distribution figure on the right, where the peaks for different pairs are between 0.4 and 0.6 with pair airplane-automobile having the largest value 0.6. This indicates that leveraging backdoors from previous iterations can give us a reasonable ASR, which can reduce the cost of backdoor generation. Based on this observation, we hence make use of a previous backdoor as the initialization for generating the current backdoor. We also enlarge the weight λ in Equation 1 for minimizing the backdoor size as the first objective is easier to satisfy with backdoor reuse. We set $\lambda = 0.001$ for generating the initial backdoor and $\lambda = 0.2$ for the follow-up ones. In case no successful backdoor is found, we rollback to the smaller λ for the next iteration.

V. Evaluation

The evaluation is conducted on various standard datasets and model structures. We also leverage pre-trained models from the TrojAI competition [42] with a variety of classification tasks and model types in the experiment. For different design choices discussed earlier, we carry out an ablation study to understand effects of different components of MOTH. Most experiments were conducted on a server equipped with two Intel Xeon Silver 4214 2.20GHz 12-core processors, 256 GB of RAM, and eight NVIDIA Quadro RTX 6000 GPU cards.

A. Experiment Setup

Datasets and Models. We employ four standard datasets in the evaluation: CIFAR-10, SVHN, LISA, and GTSRB. We also conduct experiments on 30 pre-trained models from round 4 of the TrojAI competition [42]. Details of the setup can be found in Appendix X-A.

Baselines. Three techniques discussed in Section II, namely, adversarial training, universal adversarial perturbation (UAP), and directly applying generated backdoors in training (Pairwise), are employed as baselines to compare with our hardening approach MOTH. We also include training with universal backdoors (flipping all classes to the target class) as our baseline (Universal). Since our technique can be applied to any trained models in computer vision, we further harden adversarially trained models for evaluation. Please see detailed settings in Appendix X-A. Note that the Pairwise baseline is already more sophisticated than simply using generated backdoors in hardening as it hardens the two directions of a pair together (which could lead to 30% performance difference as shown in Section V-D) and uses the speedup methods discussed in Section IV-D. In addition to the above baselines, we also study two existing backdoor-erasing approaches, namely, NC [40] and NAD [39] in hardening class distance. NC first determines if a model has a backdoor by checking if an exceptionally small backdoor can be found. If so, it stamps the generated backdoor on 20% of the available training set for retraining the model. It is suitable for mitigating injected backdoors (see Section VI-B). In contrast, MOTH enlarges distances for all class pairs, aiming at general hardening. Symmetric hardening, pair scheduling, etc. are hence needed to achieve our purpose. NAD uses a model finetuned on the poisoned model as the teacher network, and the poisoned model as the student network. It then minimizes the internal feature differences between the teacher and the student networks to update the student network. NAD produces the student model as the final output. All the trained models are considered valid based on their normal accuracy drop. For TrojAI models, we do not consider UAP as it is expensive to train from scratch for such large models. Our evaluation on the four standard datasets already shows the ineffectiveness of UAP (see Section V-B).

Metrics. We consider the following criteria in the evaluation. The prediction accuracy on the test set is used for measuring normal functionalities. For adversarially trained models, we measure model robustness within the given L^∞ bound. As

stated in Section III, the goal of orthogonalization is to enlarge the aggregated class distances for all pairs. We use the relative improvement of pair-wise class distance as the metric. That is, we compute the improvement percentage for every class pair and obtain the average, which is defined as follows.

$$\frac{1}{n \times (n-1)} \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} \frac{\hat{d}_{i \to j} - d_{i \to j}}{d_{i \to j}}, \tag{8}$$

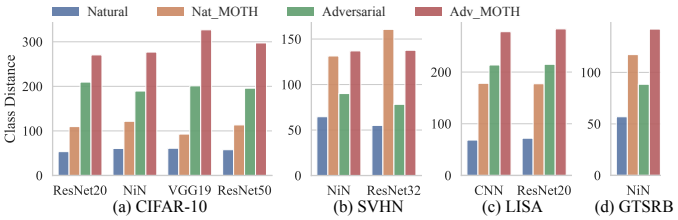
where n is the number of classes; $d_{i \to j}$ and $\hat{d}_{i \to j}$ are the class distances from i to j for the original model and the hardened model, respectively. We leverage an existing backdoor generation method NC [53] to measure the distance. Specifically, we randomly select 100 samples from the validation set of class i and apply NC 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 NC on the same pair for 3 times and use the smallest backdoor size as the class distance. As the distance is approximated by the above process, we further study the stability of the measurement using different sets/numbers of samples. Our experiment on CIFAR-10 for a naturally trained ResNet20 model shows that the distance measured on 100 samples with 100 random runs is 57.48 with a standard deviation of 4.56, which is quite stable. We also study using another backdoor generation method ABS [41] for measuring distance. Our results show that ABS can be a reasonable alternative. Details are in Appendix X-B. We show the average relative improvement along with the average class distance in the following results. In addition, the training time (in minutes) of each method is also measured.

B. Evaluation on Standard Datasets

We conduct experiments on both naturally trained and adversarially trained models for the four standard datasets, and the results are presented in Table I and Table II, respectively. Results for LISA and GTSRB are shown in Table V and Table VI in Appendix X-C. As hardening with universal adversarial perturbation needs to train a model from scratch [43], we only evaluate it in comparison with other techniques on naturally trained models. From Table I, we can observe that with a very small accuracy drop, MOTH can improve the class distance from 55.21% to 190.40% (the largest increase on ResNet32 model with SVHN dataset) compared to the original model. We also evaluate the robustness (using PGD) for a naturally trained ResNet20 model on CIFAR-10 before and after applying MOTH. The robustness does not change. Baseline UAP can only harden the class distance on a few datasets and models. For some models such as Network in Network (NiN) on CIFAR-10, UAP is not able to increase the class distance. The average class distance is even smaller than the original model (57.56 vs. 60.67), rendering UAP ineffective for class distance hardening. Training with universal backdoors has reasonable improvement over the original models, from 18.88% to 113.92%. However, it is inferior to MOTH, with 46.68% improvement difference on average. Pairwise considers all class pairs equally for hardening and has similar results as MOTH. However, due to its poor cost-effectiveness, Pairwise can take up to 1683.52 minutes to train a model, which is 22.75 times slower than MOTH (see results on a NiN model for GTSRB in Table V in Appendix X-C). The two backdoor-erasing techniques have limited improvements on class distance, with 7.60% for NC and -4.59% for NAD on average. This is reasonable as these backdoor-erasing approaches were originally designed for removing potential backdoors injected in the model. They are supposed to have little impact on benign models and hence do not enlarge class distances for normal pairs. Overall, MOTH outperforms NC, NAD, UAP and Universal in terms of class distance hardening, and Pairwise in terms of efficiency with similar distance improvement.

TABLE I: Comparison of different methods on hardening class distance for naturally trained models. 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 column shows the average class distance across all class pairs. The seventh column denotes the improvement of pairwise class distance by different techniques compared to that of original models (Natural). The last column shows the degradation of test accuracy.

D	M	Method	Accuracy	Time (m)	Distance	Increase	Degrad.
CIFAR-10	ResNet20	Natural	91.52%	56.77	53.49	-	-
		NC	89.95%	84.96	60.57	14.26%	1.57%
		NAD	91.09%	3.06	55.80	4.84%	0.43%
		UAP	90.04%	243.11	96.00	81.57%	1.48%
		Universal	90.57%	65.00	93.54	78.16%	0.95%
		Pairwise	88.78%	120.47	111.09	112.84%	2.74%
		MOTH	90.34%	29.68	109.70	108.62%	1.18%
	NiN	Natural	88.09%	68.30	60.67	-	-
		NC	87.18%	40.38	67.40	14.39%	0.91%
		NAD	83.68%	1.14	62.04	4.35%	4.41%
		UAP	86.61%	196.67	57.56	-5.22%	1.48%
		Universal	86.76%	33.90	90.69	54.09%	1.33%
		Pairwise	86.35%	64.51	120.38	104.13%	1.74%
		MOTH	86.81%	36.63	121.64	107.60%	1.28%
	VGG19	Natural	92.30%	68.42	61.14	-	-
		NC	91.23%	134.08	52.87	-11.07%	1.07%
		NAD	91.51%	3.83	45.08	-25.17%	0.79%
		UAP	90.78%	226.55	77.20	29.53%	1.52%
		Universal	91.71%	95.80	71.17	18.88%	0.59%
		Pairwise	90.01%	243.96	99.32	61.83%	2.29%
		MOTH	91.48%	44.80	92.93	55.21%	0.82%
SVHN	NiN	Natural	95.61%	10.50	64.63	-	-
		NC	94.39%	24.75	65.76	8.12%	1.22%
		NAD	92.48%	1.42	59.36	-4.93%	3.13%
		UAP	94.63%	45.47	69.31	6.99%	0.98%
		Universal	95.03%	53.40	107.15	65.97%	0.58%
		Pairwise	95.16%	152.63	113.64	79.20%	0.45%
		MOTH	94.99%	47.77	131.39	102.56%	0.62%
	ResNet32	Natural	95.15%	26.70	55.11	-	-
		NC	94.09%	31.59	60.04	12.31%	1.06%
		NAD	93.91%	0.89	53.68	-2.03%	1.24%
		UAP	93.16%	228.95	49.40	-8.86%	1.99%
		Universal	94.60%	109.30	120.20	113.92%	0.55%
		Pairwise	94.74%	530.43	126.60	129.21%	0.40%
		MOTH	94.49%	172.63	160.65	190.40%	0.66%
Average		Natural	92.53%	46.14	59.01	-	-
		NC	91.37%	63.15	61.33	7.60%	1.16%
		NAD	90.53%	2.07	55.19	-4.59%	2.00%
		UAP	91.04%	188.15	69.89	20.80%	1.49%
		Universal	91.73%	71.48	96.55	66.20%	0.80%
		Pairwise	91.01%	222.40	114.21	97.44%	1.52%
		MOTH	91.62%	66.30	123.26	112.88%	0.91%



(a) CIFAR-10 (b) SVHN (c) LISA (d) GTSRB

Fig. 10: Comparison of the average class distance

We have similar observations on adversarially trained models as shown in Table II. Adversarially trained models usually have large accuracy degradation compared to their naturally trained counterparts. Their class distances are much larger. However, those class distances are still not optimal. MOTH can further improve the class distance by 53.18% with only 0.58% accuracy degradation and no robustness degradation on average. We observe a slight robustness drop (0.30%) for a NiN model on CIFAR-10. This is because there may still exist data points close to the decision boundary and hence the model is vulnerable to adversarial attacks as discussed in Section III. Universal has a similar performance on adversarially trained models as on natural ones. It can increase class distance from 19.02% to 57.85% with an average 16.73% lower than MOTH. Pairwise has low efficiency as discussed earlier. For adversarially trained models, it has a even larger time cost, with a maximum training time of 2122.95 minutes, which is around one and a half days (see Table VI in Appendix X-C). Its class distance improvement is similar to MOTH. NC and NAD have slightly better performances on adversarially trained models than naturally trained models, with an average of 10.91% and 4.84%, respectively, which are inferior to other baselines and MOTH. Since Pairwise has a similar performance on both naturally trained and adversarially trained models to MOTH, we further study their efficiency for producing a hardened model. Please see details in Appendix X-D.

We also compare the class distances of the naturally trained, adversarially trained models, and their hardened versions by MOTH. Figure 10 presents the average class distance for the different models. For CIFAR-10, we observe that MOTH can improve the class distances of both the naturally and adversarially trained models. But the hardened models based on the natural models have smaller distances compared to the adversarial models. This is because these adversarial models have a low accuracy from 73.94% to 78.45% as shown in Table II, whereas the hardened models have 86.81% to 91.48% as shown in Table I. For the SVHN and GTSRB datasets, observe that the hardened models based on the natural models have larger class distances than the adversarially trained models. The accuracy of the hardened models is also higher than that of the adversarial models, e.g., 94.99% vs. 91.63% for NiN on SVHN (see Table I and Table II). This is consistent with our discussion in Section III that adversarial training can alter the decision boundary to have larger class distances. However, it may not achieve maximum as shown in Figure 4(b). On the other hand, orthogonalization can achieve much larger distances starting from robust models, with the sacrifice of accuracy (due to adversarial training).

TABLE II: Comparison of different methods on hardening class distance for adversarially trained models. The fifth column (Rob.) denotes model robustness on the validation set. The last two columns (ADeg. and RDeg.) show the degradation of test accuracy and model robustness, respectively.

D	M	Method	Acc.	Rob.	Time	Dist.	Increase	ADeg.	RDeg.
CIFAR-10	ResNet20	Adversarial	74.87%	42.80%	462.50	209.59	-	-	-
		NC	74.96%	42.50%	78.73	210.73	2.99%	0.00%	0.30%
		NAD	74.29%	42.50%	2.61	212.16	2.39%	0.58%	0.30%
		Universal	73.92%	43.30%	73.00	259.93	25.62%	0.95%	0.00%
		Pairwise	73.34%	43.70%	128.46	288.12	39.90%	1.53%	0.00%
		MOTH	74.04%	43.00%	38.71	270.57	32.16%	0.83%	0.00%
	NiN	Adversarial	73.94%	41.40%	430.39	189.49	-	-	-
		NC	73.75%	38.00%	43.41	188.32	3.90%	0.19%	3.40%
		NAD	73.61%	39.50%	1.26	192.24	1.23%	0.33%	1.90%
		Universal	73.54%	41.10%	35.60	221.53	19.02%	0.40%	0.30%
		Pairwise	73.46%	40.50%	75.66	271.83	47.97%	0.48%	0.90%
		MOTH	73.47%	41.10%	25.57	276.93	51.18%	0.47%	0.30%
	VGG19	Adversarial	76.00%	41.30%	1166.18	201.07	-	-	-
		NC	75.37%	41.60%	141.38	226.70	17.13%	0.63%	0.00%
		NAD	73.81%	39.90%	4.12	210.16	5.53%	2.19%	1.40%
		Universal	75.06%	42.90%	113.42	298.06	41.56%	0.94%	0.00%
		Pairwise	75.31%	42.30%	322.68	359.57	62.19%	0.69%	0.00%
		MOTH	75.08%	42.50%	81.54	326.85	53.33%	0.92%	0.00%
SVHN	NiN	Adversarial	91.63%	51.20%	132.00	90.10	-	-	-
		NC	92.02%	41.50%	27.38	91.62	4.35%	0.00%	9.70%
		NAD	88.73%	31.10%	1.42	102.76	16.39%	2.90%	20.10%
		Universal	92.10%	51.80%	56.00	124.04	38.18%	0.00%	0.00%
		Pairwise	91.55%	51.10%	156.10	126.88	42.01%	0.09%	0.10%
		MOTH	91.40%	52.30%	51.42	136.96	54.10%	0.24%	0.00%
	ResNet32	Adversarial	92.94%	58.30%	300.08	78.27	-	-	-
		NC	92.25%	49.90%	43.20	95.08	26.17%	0.69%	8.40%
		NAD	90.02%	48.70%	5.81	75.93	-1.32%	2.92%	9.60%
		Universal	92.74%	58.90%	111.42	123.73	57.85%	0.20%	0.00%
		Pairwise	92.34%	57.10%	365.30	124.26	58.27%	0.59%	1.20%
		MOTH	92.52%	58.90%	111.74	137.68	75.12%	0.42%	0.00%
Average		Natural	81.88%	47.00%	498.23	153.70	-	-	-
		NC	81.67%	42.70%	66.82	162.49	10.91%	0.21%	4.30%
		NAD	80.09%	40.34%	3.04	158.65	4.84%	1.79%	6.66%
		Universal	81.47%	47.60%	77.89	205.46	36.45%	0.41%	0.00%
		Pairwise	81.20%	46.94%	209.64	234.13	50.07%	0.68%	0.06%
		MOTH	81.30%	47.56%	61.80	229.80	53.18%	0.58%	0.00%

C. Evaluation on TrojAI Models

The models from the TrojAI competition were trained by NIST [42] with various model structures and classification tasks. These models have already been hardened by adversarial training as described in the experiment setup. We download a random set of 30 models from the official website [42] (see Appendix X-E for how these models are selected) and study whether we can further improve class distance. Detailed results can be found in our supplementary material [44]. We have the same observation as in Table II. Universal can improve the class distance to some extent with an average of 154.70% improvement over the original models, inferior to MOTH with 232.39% improvement. TrojAI models are larger and more complex than the ones used in the previous section. Universal has lower efficiency, taking 167.92 minutes on average to train a model, which is 42% slower than MOTH (118.10). Pairwise has a similar performance (239.98%) on hardening class distance as MOTH. But it suffers from low efficiency (1121.97 minutes on average), 10.81 times slower than MOTH. The results delineate the effectiveness and the efficiency of MOTH on hardening class distance for pre-trained models.

TABLE III: Ablation study on effects of design choices.

Method	Accuracy	Time (min)	Distance	Increase	Degrad.
Natural	91.52%	56.77	53.49	-	-
MOTH	90.34%	29.68	109.70	108.62%	1.18%
w/o symmetric	90.69%	35.40 (+19%)	94.16	78.90%	0.83%
w/o dynamic adjustment	90.65%	44.72 (+51%)	103.45	95.59%	0.87%
w/o approx. warm-up	90.44%	52.00 (+75%)	105.28	99.53%	1.08%
w/o scheduler	88.78%	120.47 (+306%)	111.09	112.84%	2.74%

D. Ablation Study

MOTH features a few design choices to effectively and efficiently improve class distance. We study each component individually to understand the effects of those designs. In detail, we consider four major parts in MOTH, namely, (1) symmetric hardening; (2) backdoor reuse and weight adjustment; (3) warm-up through approximation by loss changes; (4) K-arm scheduler. We conduct an ablation study using a ResNet20 model on CIFAR-10. Table III shows the effects of these design choices. Row Natural denotes the original model. MOTH is the final result of our technique. The following four rows correspond to the aforementioned four components that are excluded individually during training. Observe that without symmetric training, the overall improvement degrades by 30%, rendering its importance. This is consistent with our discussion in Section IV-B. The training time without symmetric hardening also increases as the training needs to consider both directions of a pair separately. Excluding dynamic adjustment, the training cost increases by 51% from 29.68 min to 44.72 min. The class distance improvement also degrades by 10%, indicating that backdoor reuse indeed boosts performance. The result in the sixth row of Table III shows that without the approximation of warm-up, the training cost grows substantially by 75%. The situation can deteriorate for tasks with more classes. We also investigate the final pairwise class distances for different warm-up strategies and the results are almost identical (see details in Appendix X-F). The result in the last row evidently demonstrates the effectiveness the scheduler in reducing training cost. Equipped with the scheduler, MOTH has a 4x speedup over the one without it, and the distance improvement is similar.

VI. APPLICATIONS

In this section, we evaluate MOTH in two applications including reducing false positives for backdoor scanning and eliminating injected backdoors in existing models.

A. Reducing False Positives

In the TrojAI multi-round competitions for backdoor scanning, performers are asked to identify poisoned models from a large set consisting of both clean and poisoned models. As discussed in Section II, due to the existence of small natural backdoors in clean models, performers struggled in reducing false positives (i.e., reporting clean models as poisoned), especially for round 2. The TrojAI organizer then leveraged adversarial training to mitigate this issue in the following rounds, e.g., round 4. However, the top performer still reports 32 clean models as poisoned in round 4. We apply MOTH to harden the 32 clean models. The accuracy degradation is unnoticeable (0.13% on average). Details can be found in our supplementary material [44]. The distance improvement is similar to that shown in Section V-C and elided. After orthogonalization, we apply the top-performers' scanner downloaded from their site [54] to the round 4 datasets again (with the 32 models hardened). The number of false positives is reduced from 32 to 6, without losing any true positives. For the remaining 6 false positive models, we observe that some classes are very similar. We show an example case of model ID 905 in Figure 11. The first and the second images are from classes 1 and 20, respectively. The two signs in the foreground look very similar to each other, with only one arrow difference in the center. The last image shows the backdoor generated by the top performer, which has only small black dots in the center. The class distance of the pair is 1842 for the original model, and 3964 for the hardened model. Although MOTH has already improved the distance by 115%, it is still too small as it corresponds to around $3964/3 \approx 36 \times 36$ pixel changes (224×224 for the input image), which is not distinguishable from the injected backdoor whose size is 4215. The class distance is bounded by the natural similarity between the two classes. This further stresses the importance of publishing orthogonalized class distances as part of the model specification such that the users are aware of the inevitable security risks of natural backdoors between these classes.

(a) Class 1 (b) Class 20 (c) Backdoor

Fig. 11: Example images of class 1 and class 20 from model ID 905. The backdoor is generated by the top performer.

B. Eliminating Injected Backdoors

To achieve stealthiness, injected backdoors are usually small. The essence of such data poisoning is to bring the victim and target classes close to each other, namely, only separated by the small backdoor. MOTH hence can be utilized to eliminate such backdoors since it enlarges class distance for all pairs. We utilize 59 randomly selected poisoned models (see selection details in Appendix X-E) from the TrojAI competition (round 4), where the models were poisoned by stamping a polygon (with size ranging from 903 to 6021) to foreground objects. We follow the same setup as in existing work [39] for eliminating backdoors, where *only 5% of the original training set* is used. We use three existing backdoor-erasing approaches, namely, Standard Finetuning, NC [40], and NAD [39], as the baselines. Standard Finetuning is a standard approach that was originally designed for transfer learning. It uses a small learning rate to update model parameters based on a small set of training samples. It is the same finetuning baseline from NAD [39]. Please see detailed descriptions of NC and NAD in Section V-A. We strictly

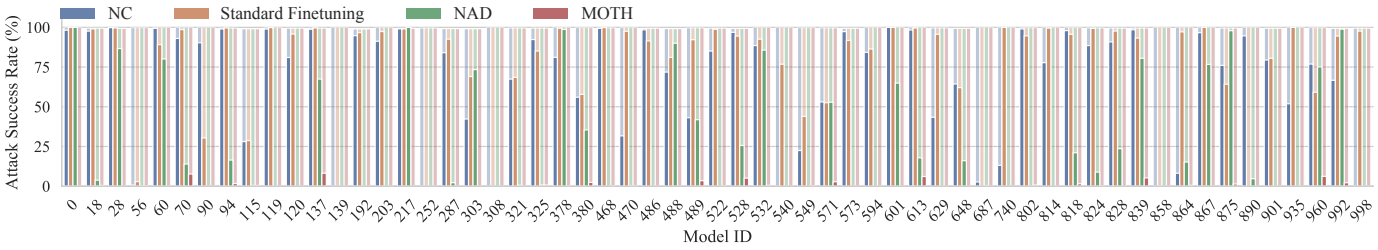


Fig. 12: Attack success rate of poisoned models before/after repair.

follow the setup of the baselines in the original papers [39], [40] and choose the best results by testing on different settings. The attack success rate of poisoned models before and after applying each technique is presented in Figure 12. The x-axis denotes the model IDs and the y-axis denotes the attack success rate (ASR). Bars in the light colors denote the ASR of the injected backdoors before erasing/hardening and the dark color after. Observe that NC (blue bars) can only successfully eliminate 7 (out of 59) backdoors and reduce ASR to below 25% for another 4 backdoors. The poisoned models in round 4 have injected backdoors only effective when stamping on a specific class. NC generates the smallest backdoor once and uses it in hardening. It is not iterative. Moreover, the smallest backdoor may not be the injected backdoor (but rather a natural backdoor). Standard Finetuning (the orange bars) is effective for seven cases as the injected backdoors were trained with clean samples. Finetuning only on clean samples may not affect the backdoor patterns learned by the poisoned models. NAD is built on top of Standard Finetuning, whose performance will be constrained by Standard Finetuning. From the green bars in Figure 12, we can observe that some models still have a high ASR after finetuning, and NAD reduces the ASRs to some extent but cannot eliminate the backdoors (e.g., ID 28 and ID 60). MOTH, on the other hand, can eliminate all the backdoors with an average ASR down to 1%. The accuracy degradation on clean samples is minimal for all the approaches on average ($< 0.2\%$). See details in our supplementary material [44].

VII. Related Work

Backdoor Attack and Defense. Existing attacks either poison the training set using backdoor injected samples with the target label like patch attacks [1], [2], or with the original label like clean label attacks [6]–[9]. There is another type of backdoor attacks that craft different backdoors for different inputs [45], [55]. Backdoor attacks can be launched on models with various applications [56]–[66]. To identify whether a model is poisoned [16]–[18], Existing works inverse backdoors [40], [41], use the difference between poisoned models and clean models when reacting to input perturbations [19]–[21]. Beside identifying poisoned models, existing techniques also detect and reject inputs stamped with backdoors [24]–[35]. These works are orthogonal to our technique as they do not consider natural backdoors. There are also works focusing on eliminating backdoors by pruning out compromised neurons [36] or retraining leveraging data augmentation technique [38]. A state-of-the-art technique NAD [39] makes use of the teacher-student training procedure to remove backdoors. Our evaluation in Section VI-B shows that MOTH outperforms NAD. In addition, it does not handle natural backdoors.

Adversarial Training. There are a large body of works on adversarial training aiming for better robustness and efficiency [53], [67]–[70]. They can enlarge class distances and on the other hand MOTH can further add to their improvement. Universal adversarial perturbation (UAP) differs from conventional adversarial attacks targeting individual samples. It aims to fool models on a set of samples with a universal perturbation [71], [72]. Researchers proposed to use UAP to improve model robustness [43]. Our results show that UAP's effectiveness on improving class distances is limited.

VIII. Discussion

The main focus of this paper is computer vision tasks. We discuss possible extensions of MOTH to other domains. Specifically, we discuss a possible proposal of leveraging a sigmoid function to approximate the discrete value (e.g., characters in natural language processing (NLP) and the executability of code in Android apps) for backdoor generation. Details are in Appendix X-G. Although the threat model of our paper focuses on static backdoors, we also test MOTH on other backdoor types, including reflection backdoors [3], composite backdoors [4], and filter backdoors [41]. MOTH is effective in defending against these attacks using the default metric. In addition, we investigate a different metric for the filter attack, which can be used for hardening and improves the effectiveness of MOTH. Please see details in Appendix X-G.

IX. Conclusion

We develop a novel model hardening technique that can enlarge class distances, making models resilient to backdoor attacks. Our evaluation on 5 datasets with 15 model structures show that it can improve class distance by 149.9% on average with only 1% accuracy loss, outperforming existing hardening techniques. It reduces 80% false positives for a state-of-the-art backdoor scanner and substantially outperforms three recent techniques in removing injected backdoors.

Acknowledgment

We thank the anonymous reviewers for their constructive comments. This research was supported, in part by IARPA TrojAI W911NF-19-S-0012, NSF 1901242 and 1910300, ONR N000141712045, N000141410468 and N000141712947. Any opinions, findings, and conclusions in this paper are those of the authors only and do not necessarily reflect the views of our sponsors.

REFERENCES

[1] T. Gu, K. Liu, B. Dolan-Gavitt, and S. Garg, "Badnets: Evaluating backdooring attacks on deep neural networks," *IEEE Access*, 2019.

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

[3] Y. Liu, X. Ma, J. Bailey, and F. Lu, "Reflection backdoor: A natural backdoor attack on deep neural networks," in *ECCV*, 2020.

[4] J. Lin, L. Xu, Y. Liu, and X. Zhang, "Composite backdoor attack for deep neural network by mixing existing benign features," in *CCS*, 2020.

[5] E. Bagdasaryan and V. Shmatikov, "Blind backdoors in deep learning models," *arXiv preprint arXiv:2005.03823*, 2020.

[6] A. Shafahi, W. R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein, "Poison frogs! targeted clean-label poisoning attacks on neural networks," in *NeurIPS*, 2018.

[7] C. Zhu, W. R. Huang, H. Li, G. Taylor, C. Studer, and T. Goldstein, "Transferable clean-label poisoning attacks on deep neural nets," in *ICML*, 2019, pp. 7614–7623.

[8] S. Zhao, X. Ma, X. Zheng, J. Bailey, J. Chen, and Y.-G. Jiang, "Clean-label backdoor attacks on video recognition models," in *CVPR*, 2020.

[9] A. Saha, A. Subramanya, and H. Pirsiavash, "Hidden trigger backdoor attacks," in *AAAI*, 2020.

[10] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang, "Trojaning attack on neural networks," in *NDSS*, 2018.

[11] Y. Cao, N. Wang, C. Xiao, D. Yang, J. Fang, R. Yang, Q. A. Chen, M. Liu, and B. Li, "Invisible for both camera and lidar: Security of multi-sensor fusion based perception in autonomous driving under physical-world attacks," in *S&P*, 2021.

[12] Y. Cao, C. Xiao, B. Cyr, Y. Zhou, W. Park, S. Rampazzi, Q. A. Chen, K. Fu, and Z. M. Mao, "Adversarial sensor attack on lidar-based perception in autonomous driving," in *CCS*, 2019.

[13] X. Liu, W. Liu, T. Mei, and H. Ma, "A deep learning-based approach to progressive vehicle re-identification for urban surveillance," in *ECCV*.

[14] S. Thys, W. Van Ranst, and T. Goedemé, "Fooling automated surveillance cameras: adversarial patches to attack person detection," in *CVPR Workshops*, 2019.

[15] B. Biggio, G. Fumera, F. Roli, and L. Didaci, "Poisoning adaptive biometric systems," in *SPR and SSPR*, 2012, pp. 417–425.

[16] S. Kolouri, A. Saha, H. Pirsiavash, and H. Hoffmann, "Universal litmus patterns: Revealing backdoor attacks in cnns," in *CVPR*, 2020.

[17] X. Xu, Q. Wang, H. Li, N. Borisov, C. A. Gunter, and B. Li, "Detecting ai trojans using meta neural analysis," *arXiv preprint arXiv:1910.03137*.

[18] X. Qiao, Y. Yang, and H. Li, "Defending neural backdoors via generative distribution modeling," in *NeurIPS*, 2019.

[19] S. Huang, W. Peng, Z. Jia, and Z. Tu, "One-pixel signature: Characterizing cnn models for backdoor detection," in *ECCV*, 2020.

[20] X. Zhang, A. Mian, R. Gupta, N. Rahnavard, and M. Shah, "Cassandra: Detecting trojaned networks from adversarial perturbations," *arXiv preprint arXiv:2007.14433*, 2020.

[21] R. Wang, G. Zhang, S. Liu, P.-Y. Chen, J. Xiong, and M. Wang, "Practical detection of trojan neural networks: Data-limited and data-free cases," in *ECCV*, 2020.

[22] W. Guo, L. Wang, Y. Xu, X. Xing, M. Du, and D. Song, "Towards inspecting and eliminating trojan backdoors in deep neural networks," in *ICDM*, 2020.

[23] X. Huang, M. Alzantot, and M. Srivastava, "Neuroninspect: Detecting backdoors in neural networks via output explanations," *arXiv preprint arXiv:1911.07399*, 2019.

[24] A. K. Veldanda, K. Liu, B. Tan, P. Krishnamurthy, F. Khorrami, R. Karri, B. Dolan-Gavitt, and S. Garg, "Nnoculation: broad spectrum and targeted treatment of backdoored dnns," *arXiv preprint arXiv:2002.08313*.

[25] S. Ma, Y. Liu, G. Tao, W.-C. Lee, and X. Zhang, "Nic: Detecting adversarial samples with neural network invariant checking," in *NDSS*.

[26] D. Tang, X. Wang, H. Tang, and K. Zhang, "Demon in the variant: Statistical analysis of dnns for robust backdoor contamination detection," in *USENIX Security*, 2021.

[27] Y. Gao, C. Xu, D. Wang, S. Chen, D. C. Ranasinghe, and S. Nepal, "Strip: A defence against trojan attacks on deep neural networks," in *ACSAC*, 2019, pp. 113–125.

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

[29] Y. Li, T. Zhai, B. Wu, Y. Jiang, Z. Li, and S. Xia, "Rethinking the trigger of backdoor attack," *arXiv preprint arXiv:2004.04692*, 2020.

[30] Y. Liu, Y. Xie, and A. Srivastava, "Neural trojans," in *ICCD*, 2017.

[31] E. Chou, F. Tramer, and G. Pellegrino, "Sentinet: Detecting localized universal attack against deep learning systems," *SPW*, 2020.

[32] B. Tran, J. Li, and A. Madry, "Spectral signatures in backdoor attacks," in *NeurIPS*, 2018, pp. 8000–8010.

[33] H. Fu, A. K. Veldanda, P. Krishnamurthy, S. Garg, and F. Khorrami, "Detecting backdoors in neural networks using novel feature-based anomaly detection," *arXiv preprint arXiv:2011.02526*, 2020.

[34] A. Chan and Y.-S. Ong, "Poison as a cure: Detecting & neutralizing variable-sized backdoor attacks in deep neural networks," *arXiv preprint arXiv:1911.08040*, 2019.

[35] M. Du, R. Jia, and D. Song, "Robust anomaly detection and backdoor attack detection via differential privacy," in *ICLR*, 2019.

[36] K. Liu, B. Dolan-Gavitt, and S. Garg, "Fine-pruning: Defending against backdooring attacks on deep neural networks," in *RAID*, 2018.

[37] E. Borgnia, V. Cherepanova, L. Fowl, A. Ghiasi, J. Geiping, M. Goldblum, T. Goldstein, and A. Gupta, "Strong data augmentation sanitizes poisoning and backdoor attacks without an accuracy tradeoff," *arXiv preprint arXiv:2011.09527*, 2020.

[38] Y. Zeng, H. Qiu, S. Guo, T. Zhang, M. Qiu, and B. Thuraisingham, "Deepsweep: An evaluation framework for mitigating dnn backdoor attacks using data augmentation," *arXiv preprint arXiv:2012.07006*.

[39] Y. Li, N. Koren, L. Lyu, X. Lyu, B. Li, and X. Ma, "Neural attention distillation: Erasing backdoor triggers from deep neural networks," in *ICLR*, 2021.

[40] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao, "Neural cleanse: Identifying and mitigating backdoor attacks in neural networks," in *S&P*, 2019, pp. 707–723.

[41] Y. Liu, W.-C. Lee, G. Tao, S. Ma, Y. Aafer, and X. Zhang, "Abs: Scanning neural networks for back-doors by artificial brain stimulation," in *CCS*, 2019, pp. 1265–1282.

[42] "TrojAI Leaderboard," https://pages.nist.gov/trojai/.

[43] A. Shafahi, M. Najibi, Z. Xu, J. Dickerson, L. S. Davis, and T. Goldstein, "Universal adversarial training," in *AAAI*, 2020.

[44] ModelOrth, "Moth," https://github.com/ModelOrth/MOTH.

[45] A. Salem, R. Wen, M. Backes, S. Ma, and Y. Zhang, "Dynamic backdoor attacks against machine learning models," *arXiv preprint arXiv:2003.03675*, 2020.

[46] S. Cheng, Y. Liu, S. Ma, and X. Zhang, "Deep feature space trojan attack of neural networks by controlled detoxification," in *AAAI*, 2021.

[47] "Keras Applications," https://keras.io/api/applications/.

[48] H. Salman, A. Ilyas, L. Engstrom, A. Kapoor, and A. Madry, "Do adversarially robust imagenet models transfer better?" in *NeurIPS*, 2020.

[49] G. Elsayed, D. Krishnan, H. Mobahi, K. Regan, and S. Bengio, "Large margin deep networks for classification," *NeurIPS*, 2018.

[50] Y. Yang, R. Khanna, Y. Yu, A. Gholami, K. Keutzer, J. E. Gonzalez, K. Ramchandran, and M. W. Mahoney, "Boundary thickness and robustness in learning models," *NeurIPS*, 2020.

[51] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine learning*, pp. 235–256, 2002.

[52] C. J. C. H. Watkins, "Learning from delayed rewards," 1989.

[53] E. Wong, L. Rice, and J. Z. Kolter, "Fast is better than free: Revisiting adversarial training," in *ICLR*, 2020.

[54] "Abs," https://github.com/naiyeleo/ABS.

[55] T. A. Nguyen and A. Tran, "Input-aware dynamic backdoor attack," *NeurIPS*, vol. 33, 2020.

[56] X. Zhang, Z. Zhang, and T. Wang, "Trojaning language models for fun and profit," in *European S&P*, 2021.

[57] X. Chen, A. Salem, M. Backes, S. Ma, and Y. Zhang, "Badnl: Backdoor attacks against nlp models," *arXiv preprint arXiv:2006.01043*, 2020.

[58] K. Kurita, P. Michel, and G. Neubig, "Weight poisoning attacks on pre-trained models," in *ACL*, 2020.

[59] S. Rezaei and X. Liu, "A target-agnostic attack on deep models: Exploiting security vulnerabilities of transfer learning," in *ICLR*, 2020.

[60] B. Wang, Y. Yao, B. Viswanath, H. Zheng, and B. Y. Zhao, "With great training comes great vulnerability: Practical attacks against transfer learning," in *USENIX Security*, 2018, pp. 1281–1297.

[61] Y. Yao, H. Li, H. Zheng, and B. Y. Zhao, "Latent backdoor attacks on deep neural networks," in *CCS*, 2019, pp. 2041–2055.

[62] C. Xie, K. Huang, P.-Y. Chen, and B. Li, "Dba: Distributed backdoor attacks against federated learning," in *ICLR*, 2019.

[63] H. Wang, K. Sreenivasan, S. Rajput, H. Vishwakarma, S. Agarwal, J.-y. Sohn, K. Lee, and D. Papailiopoulos, "Attack of the tails: Yes, you really can backdoor federated learning," *NeurIPS*, 2020.
[64] V. Tolpegin, S. Truex, M. E. Gursoy, and L. Liu, "Data poisoning attacks against federated learning systems," in *ESORICS*, 2020, pp. 480–501.
[65] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," in *AISTATS*, 2020, pp. 2938–2948.
[66] M. Fang, X. Cao, J. Jia, and N. Gong, "Local model poisoning attacks to byzantine-robust federated learning," in *USENIX Security*, 2020.
[67] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *ICLR*, 2018.
[68] A. Shafahi, M. Najibi, A. Ghiasi, Z. Xu, J. Dickerson, C. Studer, L. S. Davis, G. Taylor, and T. Goldstein, "Adversarial training for free!" *NeurIPS*, 2019.
[69] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial machine learning at scale," in *ICLR*, 2017.
[70] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, "Ensemble adversarial training: Attacks and defenses," in *ICLR*, 2018.
[71] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," in *CVPR*, 2017, pp. 1765–1773.
[72] J. Hendrik Metzen, M. Chaithanya Kumar, T. Brox, and V. Fischer, "Universal adversarial perturbations against semantic image segmentation," in *ICCV*, 2017, pp. 2755–2764.
[73] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.
[74] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.
[75] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
[76] A. Mogelmose, M. M. Trivedi, and T. B. Moeslund, "Vision-based traffic sign detection and analysis for intelligent driver assistance systems: Perspectives and survey," *T-ITS*, vol. 13, no. 4, pp. 1484–1497, 2012.
[77] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song, "Robust physical-world attacks on deep learning visual classification," in *CVPR*, 2018, pp. 1625–1634.
[78] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *CVPR*, 2017, pp. 4700–4708.
[79] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *CVPR*, 2016.
[80] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.
[81] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *IJRR*, 2013.
[82] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *CVPR*, 2016, pp. 3213–3223.
[83] F. Larsson, M. Felsberg, and P.-E. Forssen, "Correlating fourier descriptors of local patches for road sign recognition," *IET Computer Vision*, vol. 5, no. 4, pp. 244–254, 2011.
[84] F. Pierazzi, F. Pendlebury, J. Cortellazzi, and L. Cavallaro, "Intriguing properties of adversarial ml attacks in the problem space," in *S&P*, 2020.
[85] Y. Liu, X. Ma, J. Bailey, and F. Lu, "Reflection backdoor: A natural backdoor attack on deep neural networks," https://github.com/DreamtaleCore/Refool/tree/582e54a6b92a8a52.
[86] J. Lin, L. Xu, Y. Liu, and X. Zhang, "Composite backdoor attack for deep neural network by mixing existing benign features," https://github.com/TemporaryAcc0unt/composite-attack.

X. Appendix

A. Detailed Experiment Setup

CIFAR-10 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. Four different models are utilized for this dataset: ResNet20 [73], Network in Network (NiN) [74], VGG19 [75], and ResNet50 [73].

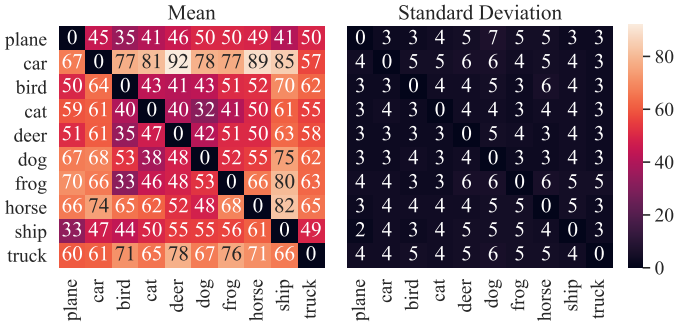


Fig. 13: Mean and standard deviation of class distances for 100 different sets of 100 random samples for a naturally trained ResNet20 model on CIFAR-10.

SVHN (Street View House Numbers) dataset contains house (digital) numbers extracted from Google Street View images, which consists 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 [74] and ResNet32 [73].

LISA is a U.S. traffic sign dataset that contains 47 different road signs [76]. 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 existing work [77] by choosing 18 most common classes based 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 [77] that consists of three convolutional layers and one fully-connected layer, and a ResNet20 [73].

GTSRB (German Traffic Sign Recognition Dataset) contains 43 different traffic signs, which is designed for training models in self-driving scenarios. We divide the dataset into three parts: 35,289 signs for training, 3,920 for validation, and 12,630 for testing. We use an NiN [74] model for this dataset.

We also conduct experiments on 89 pre-trained models (30 benign ones and 59 poisoned ones) from round 4 of TrojAI competition [42]. In this round, TrojAI models utilize 16 different structures such as DenseNet121 [78], InceptionV3 [79], MobileNetV2 [80], etc. Each model is trained to classify synthetic street signs into between 15 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 including three categories from KITTI dataset [81], Cityscapes dataset [82] and Swedish Roads dataset [83]. Random transformations, such as shifting, titling, lighting, blurring, and weather effects, are applied during training to improve dataset diversity. Adversarial training with PGD (Projected Gradient Descent) [67] and FBF (Fast is Better than Free) [53] is leveraged to improve model quality.

For adversarial training, we leverage PGD to harden models, with L^∞ bound of $8/255$ for CIFAR-10 dataset, 0.03 for SVHN and GTSRB, and 0.1 for LISA. For training with universal adversarial perturbation (UAP), we utilize an existing work [43] for hardening models. The L^∞ bound for UAP training is determined according to the normal accuracy

TABLE IV: Comparison of distance measures by different backdoor generation methods. The fifth and seventh columns show the average class distance measured by NC and ABS [41], respectively.

D	M	Method	Accuracy	$\text{Distance}_{\text{NC}}$	$\text{Increase}_{\text{NC}}$	$\text{Distance}_{\text{ABS}}$	$\text{Increase}_{\text{ABS}}$
CIFAR-10	NiN	Natural	88.09%	60.67	-	166.12	-
		NC	87.18%	67.40	14.39%	270.83	86.27%
		NAD	83.68%	62.04	4.35%	206.09	42.97%
		MOTH	86.81%	121.64	107.60%	490.25	289.68%

drop. We use L^∞ bound of $4/255$ for CIFAR-10, 0.05 for SVHN, and 0.03 for LISA and GTSRB. For directly applying generated backdoors in training (Pairwise), we harden every class pair for 100 iterations.

B. Stability of Class Distance

We study the stability of class distance by using different sets/numbers of samples on CIFAR-10 and SVHN. Specifically, we select 100 different sets of 100 random samples from the validation set, and apply NC [40] to measure the distance as described in Section V-A. We also study using 200 samples. We employ a naturally trained ResNet20 model for CIFAR-10 and a naturally trained NiN model for SVHN. Figure 13 shows the results for CIFAR-10 using 100 random samples. Results of using 200 random samples for CIFAR-10 and results for SVHN can be found in our supplementary material [44]. The heat map on the left denotes the means of class distances for all pairs and the heat map on the right denotes the standard deviations. Observe that the standard deviations of class distances are small for using 100 random samples (4.56) for the measurement (4.50 for 200 random samples), rendering the class distance measure quite stable. The average distances are slightly larger for using 200 samples (61.31) than using 100 samples (57.48). The observation is the same for SVNH using 100 samples (69.25$\pm$5.42) and using 200 samples (73.43$\pm$5.15).

We also study using another backdoor generation method ABS [41] for measuring the distance. The experiment is conducted on a naturally trained NiN model on CIFAR-10. Models hardened by three methods, namely, NC [40], NAD [39], and MOTH, are also considered for studying the distance by ABS [41]. Table IV shows the distances and relative improvements over the original model by different methods. ABS does not merge original pixels with trigger pixels. Instead, it either completely replaces them or keeps them untouched. As such, the class distance measured by ABS is larger than NC. When comparing the distances of different hardened models (by NC, NAD, and MOTH), the relative order is the same for the distance measure by NC and ABS. NAD has the smallest improvement (42.97%), and NC has a relatively large improvement (86.27%). MOTH achieves the best performance (289.68%) regardless of the measure. This delineates ABS an alternative tool for distance measure.

C. More Results on Standard Datasets

We evaluate on more models and datasets for both naturally trained and adversarially trained models, which are presented in Table V and Table VI, respectively. From Table V, we can observe that with a very small accuracy drop (0.69%), MOTH can improve the class distance by 128.60% on average compared to the original model. Baseline UAP can only harden the class distance on a few datasets and models. For some models such as CNN on LISA, UAP is not able to increase the class distance. Training with universal backdoors has reasonable improvement over the original models, from 46.13% to 69.74%. However, it is inferior to MOTH, with 74.36% improvement difference on average. Due to its poor cost-effectiveness, Pairwise can take up to 1683.52 minutes to train a model (on GTSRB), which is 22.75 times slower than MOTH. The two backdoor-erasing techniques have limited improvements on class distance, with 8.33% for NC and 5.13% for NAD on average. Overall, MOTH outperforms NC, NAD, UAP, and Universal in terms of class distance hardening, and Pairwise in terms of efficiency with similar distance improvement. We have similar observations on adversarially trained models as shown in Table VI. MOTH can improve the class distance by 52.49% with only 0.37% accuracy degradation and no robustness degradation on average. Universal has a similar performance on adversarially trained models as on natural ones. It can increase class distance from 16.26% to 37.13% with an average 23% lower than MOTH. Pairwise has low efficiency as discussed earlier. For adversarially trained models, it has a even larger time cost, with a maximum training

TABLE V: Comparison of different methods on hardening class distance for naturally trained models.

D	M	Method	Accuracy	Time (m)	Distance	Increase	Degrad.
CIFAR-10	ResNet50	Natural	92.71%	74.11	57.80	-	-
		NC	91.54%	183.00	67.65	18.65%	1.17%
		NAD	92.65%	6.01	57.75	0.60%	0.06%
		UAP	91.83%	345.61	98.94	73.29%	0.88%
		Universal	91.69%	145.02	97.30	69.74%	1.02%
		Pairwise	90.43%	209.00	112.46	93.57%	2.28%
		MOTH	91.25%	51.69	113.46	98.79%	1.46%
LISA	CNN	Natural	97.30%	0.15	68.47	-	-
		NC	82.67%	8.27	74.94	14.21%	14.63%
		NAD	95.45%	0.15	67.70	0.20%	1.85%
		UAP	95.60%	1.79	65.90	-1.23%	1.70%
		Universal	96.45%	11.34	101.48	46.13%	0.85%
		Pairwise	97.16%	204.98	193.41	166.57%	0.14%
		MOTH	96.31%	13.09	178.31	158.35%	0.99%
	ResNet20	Natural	98.86%	1.70	72.05	-	-
		NC	98.44%	34.42	73.61	3.08%	0.42%
		NAD	96.59%	0.72	81.42	11.79%	2.27%
		UAP	96.16%	6.33	97.11	36.32%	2.70%
		Universal	98.30%	25.37	113.35	53.38%	0.57%
		Pairwise	98.72%	738.92	192.70	168.74%	0.14%
		MOTH	98.30%	38.75	177.41	144.92%	0.57%
GTSRB	NiN	Natural	95.28%	4.60	56.93	-	-
		NC	95.65%	110.51	51.99	-2.61%	0.00%
		NAD	93.63%	0.51	57.56	7.93%	1.65%
		UAP	95.25%	20.55	53.97	4.27%	0.03%
		Universal	95.22%	25.20	78.66	47.69%	0.06%
		Pairwise	95.68%	1683.52	167.42	201.81%	0.00%
		MOTH	95.55%	74.01	117.30	112.34%	0.00%
Average		Natural	96.04%	20.14	63.81	-	-
		NC	92.08%	84.05	67.05	8.33	3.96%
		NAD	94.58%	1.85	66.11	5.13	1.46%
		UAP	94.71%	93.57	78.98	28.16%	1.33%
		Universal	95.42%	51.73	97.70	54.24%	0.62%
		Pairwise	95.50%	709.11	166.50	157.67%	0.54%
		MOTH	95.35%	44.39	146.62	128.60%	0.69%

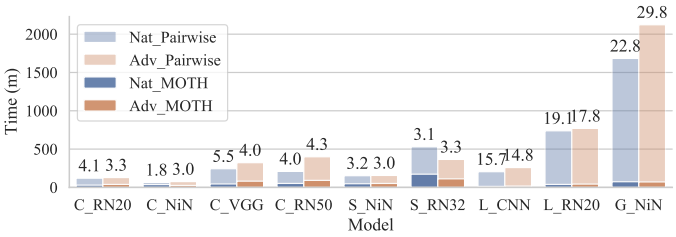


Fig. 14: Comparison of Pairwise and MOTH in terms of training cost. The x-axis denotes different models, where the first letter denotes the dataset (C for CIFAR-10, S for SVHN, L for LISA, and G for GTSRB) and the remaining letters denote the model structure. The y-axis denotes the training time in minutes. The numbers on top of bars show the speedup of MOTH over Pairwise.

time of 2122.95 minutes, which is around one and a half days. Its class distance improvement is similar to MOTH. NC and NAD have slightly better performances on adversarially trained models than naturally trained models, with an average of 13.09% and 7.21%, respectively, which are inferior to other baselines and MOTH.

D. Efficiency of MOTH

Since Pairwise has a similar performance on both naturally trained and adversarially trained models to MOTH, we further study their efficiency for producing a hardened model. Figure 14 shows the training time of Pairwise and MOTH. The x-axis denotes the models and the y-axis denotes the training time in minutes. The numbers on top of each bar show the speedup of MOTH over Pairwise. Observe that MOTH has a speedup of 1.8 to 22.8 for natural models and 3.0 to 29.8 for adversarial models. Moreover, Pairwise is extremely slow on datasets with many classes. For instance, the LISA datasets have 18 classes and the GTSRB dataset has 43 classes. Pairwise is 17x-19x slower on LISA and 22x-29x slower on GTSRB than MOTH. This is due to the quadratic complexity of orthogonalization, which becomes very expensive for models with a large number of classes if scheduling is not in place. MOTH, on the other hand, is efficient even for models with many classes and has a competitive performance with Pairwise on class distance improvement.

E. Selection of TrojAI Models

We evaluate on 30 randomly selected TrojAI benign models from the official website [42] and study the performance of different methods on hardening class distance. We use random seed 1030792629 to choose from the list of benign models from TrojAI round 4. We also use random seed 186270393 to select 59 poisoned models for the study in Section VI-B. We use Python 3.6.9 and NumPy 1.19.5.

As discussed in Section V-A, the class distance measurement is conducted for all class pairs, which is computationally expensive, especially for models with many classes. We also run 3 times on each model to have a more accurate measurement of class distances. For TrojAI models, it usually takes days to evaluate one single model (including MOTH and all the baselines). We hence resort to measuring on 100 randomly selected class pairs for studying the class distance. We set the random seed to be the sum of 165838010 and the model id, in order to avoid selecting the same set of class pairs for models with the same number of classes.

TABLE VI: Comparison of different methods on hardening class distance for adversarially trained models.

D	M	Method	Acc.	Rob.	Time	Dist.	Increase	ADeg.	RDeg.
CIFAR-10	ResNet50	Adversarial	78.45%	42.30%	973.47	195.90	-	-	-
		NC	77.21%	42.70%	155.98	224.85	27.84%	1.24%	0.00%
		NAD	77.60%	44.50%	6.11	201.52	3.73%	0.85%	0.00%
		Universal	77.60%	42.10%	150.53	262.28	37.13%	0.85%	0.20%
		Pairwise	77.63%	42.50%	399.74	276.04	43.32%	0.82%	0.00%
		MOTH	77.63%	41.70%	91.90	297.40	55.19%	0.82%	0.60%
LISA	CNN	Adversarial	75.43%	25.00%	5.00	213.70	-	-	-
		NC	73.44%	24.43%	8.54	228.81	16.05%	1.99%	0.57%
		NAD	70.17%	23.57%	0.14	198.26	3.51%	5.26%	1.43%
		Universal	74.15%	33.14%	4.00	270.17	29.08%	1.28%	0.00%
		Pairwise	76.85%	33.71%	258.99	312.77	59.74%	0.00%	0.00%
		MOTH	74.72%	31.14%	17.47	278.11	59.01%	0.71%	0.00%
	ResNet20	Adversarial	80.54%	36.57%	17.40	215.07	-	-	-
		NC	78.13%	41.86%	79.76	233.03	11.06%	2.41%	0.00%
		NAD	80.97%	40.00%	0.93	245.20	15.27%	0.00%	0.00%
		Universal	79.69%	42.29%	26.28	251.59	16.26%	0.85%	0.00%
		Pairwise	81.25%	41.71%	770.11	265.71	27.58%	0.00%	0.00%
		MOTH	81.25%	41.00%	43.34	283.53	35.00%	0.00%	0.00%
GTSRB	NiN	Adversarial	90.96%	68.70%	54.98	88.51	-	-	-
		NC	91.55%	63.60%	111.72	85.79	-2.60%	0.00%	5.10%
		NAD	88.12%	54.30%	0.50	95.25	6.34%	2.84%	14.40%
		Universal	90.25%	69.70%	26.00	118.41	35.47%	0.70%	0.00%
		Pairwise	91.62%	72.60%	2122.95	162.12	76.89%	0.00%	0.00%
		MOTH	90.32%	70.50%	71.29	142.10	60.76%	0.64%	0.00%
Average		Natural	81.35%	43.14%	262.71	178.30	-	-	-
		NC	80.08%	43.15%	89.00	193.12	13.09%	1.27%	0.00%
		NAD	79.22%	40.59%	1.92	185.06	7.21%	2.13%	2.55%
		Universal	80.42%	46.81%	51.70	225.61	29.49%	0.93%	0.00%
		Pairwise	81.84%	47.63%	887.95	254.16	51.88%	0.00%	0.00%
		MOTH	80.98%	46.09%	56.00	250.29	52.49%	0.37%	0.00%

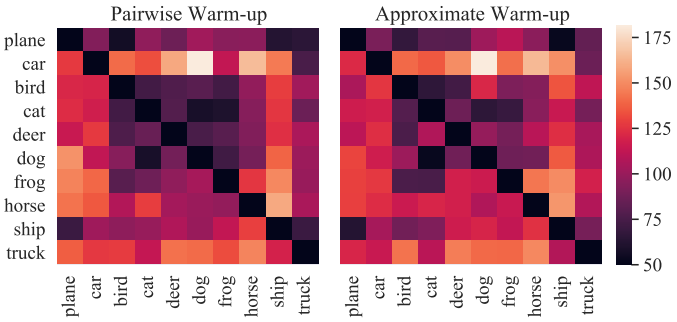


Fig. 15: Comparison of different warm-ups for naturally trained ResNet20 on CIFAR-10.

F. Comparison of Warm-up Strategies

We present the final pairwise class distances for different warm-up strategies in Figure 15. Each cell in the heat map denotes the class distance improvement from a source class (row) to a target class (column). The brightness of the color denotes the class distance (the brighter the larger). Observe that the two heat maps are almost identical, meaning our approximation is effective in solving the cold-start problem.

G. Extensions to Other Settings

Extension to Other Domains. The main focus of this paper is computer vision tasks. Backdoors in other domains have

TABLE VII: Evaluation on other backdoor types.

Backdoor Attack	Subject	Original		NAD		MOTH	
		Accuracy	ASR	Accuracy	ASR	Accuracy	ASR
Reflection	GTSRB	71.22%	83.33%	65.28%	20.83%	87.50%	0.00%
	CIFAR ID 1	86.13%	96.60%	78.63%	75.17%	82.30%	8.92%
	CIFAR ID 2	84.45%	95.78%	76.59%	92.56%	83.11%	15.33%
	CIFAR ID 3	84.93%	99.10%	77.83%	97.16%	81.30%	11.67%
Composite	MNIST	99.51%	99.51%	99.31%	97.10%	98.39%	17.60%
	FMNIST	93.09%	97.20%	90.90%	92.10%	90.04%	38.00%
	CIFAR	82.70%	87.60%	79.09%	46.80%	80.42%	44.50%
	SVHN	94.90%	89.24%	93.49%	66.15%	91.67%	37.00%

different definitions of being stealthy and semantic-aware. For example, in natural language processing (NLP) domain, backdoors are usually characters or words that do not change the overall meaning of original sentences. We can define the class distance as the number of characters or words of generated backdoors. As characters or words are discrete (i.e., either in the sentence or not), existing backdoor generation techniques may not be directly applicable. A possible proposal is to use a sigmoid function to approximate the discrete value such that existing optimization methods can be leveraged to generate minimal backdoors. The training process can then follow MOTH by inserting minimal backdoors in normal sentences. The measure of backdoors in domains such as Android apps [84] requires domain-specific constraints such as the injected code not being executed [84]. These constraints may be considered in the loss function during backdoor generation, similar to minimizing the L^1 norm of the mask. For instance, we can use a sigmoid function to approximate the executability of a piece of code (which is discrete) and add this to the loss function. A large weight can be applied on this loss part to encourage the loss to be zero. We will leave the experimental exploration to our future work.

Application to Other Backdoors. Although the threat model of our paper focuses on static backdoors, we also test MOTH on other backdoor types, including reflection backdoors [3], composite backdoors [4], and filter backdoors [41]. For the reflection attack, we download a pre-trained poisoned model from the original GitHub repository [85] and also train another three poisoned models. For the composite attack, we use the open-source repository from the original paper [86] to generate four poisoned models. For the filter attack, we leverage 28 poisoned models from the TrojAI round 4 dataset [42]. We use the same random seed as in Section VI-B for the selection. The experimental results on reflection and composite attacks are shown in Table VII. The first column denotes different backdoor attacks. The second column presents subject models for evaluation. The third and fourth columns show the accuracies and ASRs of poisoned models. The following columns show the results after applying NAD [39] and MOTH. We can observe that MOTH is effective against the reflection attack, reducing the ASR down to 8.98% on average, whereas NAD can only reduce the ASR to 71.43% on average. For the composite attack, MOTH is able to eliminate around half of the backdoor effect (reducing the ASR from 87.60-99.51% to 17.60-44.50%). NAD, on the other hand, is only able to reduce one model on CIFAR to 46.80%. The other three

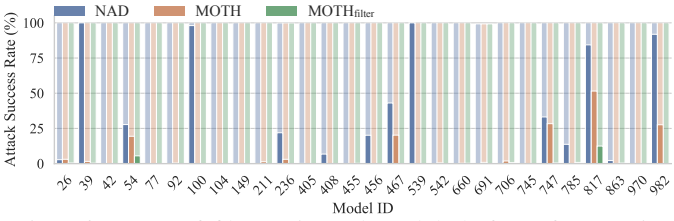


Fig. 16: ASRs of filter-poisoned models before/after repair.

models after applying NAD still have more than 65% ASR, rendering NAD ineffective against complex backdoor attacks such as composite backdoors. The experimental results on the filter attack are shown in Figure 16. The x-axis denotes the model IDs and the y-axis denotes the attack success rate (ASR). Bars in the light colors denote the ASR of the injected backdoors before erasing/hardening and the dark color after. Filter backdoors are more pervasive than static backdoors by transforming all the pixels on an input image. Model hardening using minimal static backdoors may not eliminate filter backdoors. In our results, for 23 out of the 28 filter-poisoned models, MOTH can still reduce the ASR down to $<$3.1%. For five other cases (ID 54, ID 467, ID 747, ID 817, and ID 982), MOTH is able to reduce the ASRs by 48.44-80%. The accuracy degradation on clean samples is minimal for all the approaches on average ($< 0.3\%$) and omitted.

As filter backdoors are dynamic, meaning that the backdoor transformation is input-specific. The class distance measured by static backdoors may not be optimal for filter backdoors. We hence resort to a different measurement for the filter case. Specifically, we use the magnitude of mean and standard deviation for transforming a set of samples as the distance in the following.

$$\forall \boldsymbol{x} \in \boldsymbol{X},\ \min_{\boldsymbol{\mu},\boldsymbol{\sigma}} \mathcal{L}\big(\mathcal{M}(\boldsymbol{x}'), y_t\big) + \lambda \cdot (|\boldsymbol{\mu} - \bar{\boldsymbol{\mu}}_{\boldsymbol{X}}| + |\boldsymbol{\sigma} - \bar{\boldsymbol{\sigma}}_{\boldsymbol{X}}|),$$
$$\mathbb{T}(\boldsymbol{x}, \boldsymbol{\mu}, \boldsymbol{\sigma}) = \boldsymbol{x}' = (\boldsymbol{x} - \boldsymbol{\mu}_{\boldsymbol{x}})/\boldsymbol{\sigma}_{\boldsymbol{x}} \cdot \boldsymbol{\sigma} + \boldsymbol{\mu},$$

where $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are the mean and standard deviation for the backdoor transformation, respectively. $\boldsymbol{\mu}_{\boldsymbol{x}}$ and $\boldsymbol{\sigma}_{\boldsymbol{x}}$ are the mean and standard deviation of individual inputs, and $\bar{\boldsymbol{\mu}}_{\boldsymbol{X}}$ and $\bar{\boldsymbol{\sigma}}_{\boldsymbol{X}}$ are the averages across all the samples. Instead of minimizing the L^1 of the mask in Equation 1, here we minimize the change of the mean and standard deviation with respect to those of input samples, i.e., $|\boldsymbol{\mu} - \bar{\boldsymbol{\mu}}_{\boldsymbol{X}}| + |\boldsymbol{\sigma} - \bar{\boldsymbol{\sigma}}_{\boldsymbol{X}}|$. This is the distance measurement for the filter scenario. Note that such a measurement does not change the definition of class distance $\min_{\mathbb{T}}(\mathbb{E}(\|\mathbb{T}(\boldsymbol{x}) - \boldsymbol{x}\|))$ in Definition III.2, where the distance is the expectation of some measurement between transformed data $\mathbb{T}(\boldsymbol{x})$ and original data $\boldsymbol{x}$ for a set of samples. Based on the above measurement, we modify the original MOTH by replacing the optimization in Equation 1 and Equation 2 with the above two equations. The results in green bars in Figure 16 show that the modified version MOTH$_{\text{filter}}$ can reduce the ASRs from 100% to nearly 0% for 27 models and to around 10% for one model (ID 817). These results demonstrate the capability of MOTH when extended to eliminate other types of backdoors. We plan to study more diverse backdoor types in the future.