Tracing the Origin of Adversarial Attack for Forensic Investigation and Deterrence

Han Fang[1] Jiyi Zhang[1] Yupeng Qiu[1] Jiayang Liu[1]
Ke Xu[2] Chengfang Fang[2] Ee-Chien Chang[1,*]
[1]National University of Singapore [2]Huawei International
{fanghan, ljyljy}@nus.edu.sg {jiyizhang, qiu_yupeng}@u.nus.edu
{xuke64, fang.chengfang}@huawei.com changec@comp.nus.edu.sg

Abstract

Deep neural networks are vulnerable to adversarial attacks. In this paper, we take the role of investigators who want to trace the attack and identify the source, that is, the particular model which the adversarial examples are generated from. Techniques derived would aid forensic investigation of attack incidents and serve as deterrence to potential attacks. We consider the buyers-seller setting where a machine learning model is to be distributed to various buyers and each buyer receives a slightly different copy with the same functionality. A malicious buyer generates adversarial examples from a particular copy $\mathcal{M}_i$ and uses them to attack other copies. From these adversarial examples, the investigator wants to identify the source $\mathcal{M}_i$. To address this problem, we propose a two-stage separate-and-trace framework. The model separation stage generates multiple copies of a model for the same classification task. This process injects unique features into each copy so that adversarial examples generated have distinct and traceable features. We give a parallel structure which pairs a unique tracer with the original classification model in each copy and a variational autoencoder (VAE)-based training method to achieve this goal. The tracing stage takes in adversarial examples and a few candidate models, and identifies the likely source. Based on the unique features induced by the tracer, we could effectively trace the potential adversarial copy by considering the output logits from each tracer. Empirical results show that it is possible to trace the origin of the adversarial example and the mechanism can be applied to a wide range of architectures and datasets.

1. Introduction

Deep learning models are vulnerable to adversarial attacks. By introducing specific perturbations on input samples, the network model could be misled to give wrong predictions even when the perturbed sample looks visually close to the clean image [4,8,21,27]. There are many existing works on defending against such attacks [9,12,16,20]. Unfortunately, although current defenses could mitigate the attack to some extent, the threat is still far from being completely eliminated. In this paper, we look into the forensic aspect: from the adversarial examples, can we determine which model the adversarial examples were generated from? Techniques derived could aid forensic investigation of attack incidents and provide deterrence to future attacks.

We consider a buyers-seller setting [28], which is similar to the buyers-seller setting in digital copyright protection [19].

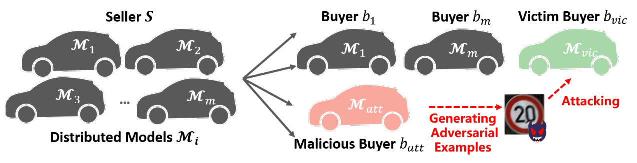


Figure 1: *Buyers-seller setting.* The seller has multiple models $\mathcal{M}_i, i \in [1, m]$ that are to be distributed to different buyers. A malicious buyer b_{att} attempts to attack the victim buyers b_{vic}s by generating the adversarial examples with his own model $\mathcal{M}_{att}$.

Buyers-seller Setting. Under this setting, the seller ***S*** distributes m classification models $\mathcal{M}_i, i \in [1, m]$ to different buyers b_i's as shown in Fig. 1. These models are trained for a same classification task using a same training dataset. The models are made accessible to the buyer as black boxes, for instance, the models could be embedded in hardware such as FPGA and ASIC, or are provided in a Machine Learning as a Service (MLaaS) platform. Hence, the buyer only has black-box access, which means that he can only query the model for the hard label. In addition, we assume that the buyers do not know the training datasets. The seller has

[*]Corresponding author.

full knowledge and thus has white-box access to all the distributed models.

Attack and Traceability. A malicious buyer wants to attack other victim buyers. The malicious buyer does not have direct access to other models and thus generates the examples from his own model and then deploys the found examples. For example, the malicious buyer might generate an adversarial example of a road sign using its self-driving vehicle, and then physically defaces the road sign to trick passing vehicles. Now, as forensic investigators who have obtained the defaced road sign, we want to understand where the adversarial example is generated from and trace the model used in generating the example.

Proposed Framework. There are two stages in our solution: *model separation* and *origin tracing*. During the model separation stage, given a classification task, we want to generate multiple models that have high accuracy on the classification task and yet are sufficiently different for tracing. In other words, we want to proactively enhance differences among the models in order to facilitate tracing. To achieve that, we propose a parallel network structure that pairs a unique tracer with the original classification model. The role of the tracer is to modify the output, so as to induce the attacker to generate adversarial examples with unique features. We give a variational autoencoder (VAE)-based training method for training the tracer.

During the tracing stage, given m different classification models $\mathcal{M}_i, i \in [1, m]$ and the found adversarial example, we want to determine which model is most likely used in generating the adversarial example. This is achieved by exploiting the different tracers that are earlier embedded into the parallel models. Our proposed method compares the output logits of those tracers to identify the source.

In a certain sense, traceability is similar to neural network watermarking and can be viewed as a stronger form of watermarking. Neural network watermarking schemes [2] attempt to generate multiple models so that an investigator can trace the source of a modified copy. In traceability, the investigator can trace the source based on the generated adversarial examples.

Contributions.

1. We point out a new aspect in defending against adversarial attacks, that is, tracing the origin of adversarial samples among multiple classifiers. Techniques derived would aid forensic investigation of attack incidents and provide deterrence to future attacks.

2. We propose a framework to achieve traceability in the buyers-seller setting. The framework consists of two stages: a model separation stage, and a tracing stage. The model separation stage generates multiple "well-separated" models and this is achieved by a parallel network structure that pairs a tracer with the classifier. The tracing mechanism exploits the characteristics of the paired tracers to decide the origin of the given adversarial examples.

3. We investigate the effectiveness of the separation and the subsequent tracing. Experimental studies show that the proposed mechanism can effectively trace to the source. For example, the tracing accuracy achieves more than 97% when applying to "ResNet18-CIFAR10" task for 5 distributed models. We also observe a clear separation of the source tracer's logits distribution, from the non-source's logits distribution (e.g. Fig. 4a).

2. Related Work

In this paper, we adopt black-box settings where the adversary can only query the model and get the hard label (final decision) of the output. Many existing attacks assume white-box settings. Attacks such as FGSM [8], PGD [16], JSMA [23], DeepFool [21], CW [4] and EAD [6] usually directly rely on the gradient information provided by the victim model. As the detailed information of the model is hidden in black-box settings, black-box attacks are often considered more difficult and there are fewer works. Chen *et. al.* introduced a black-box attack called Zeroth Order Optimization (ZOO) [7]. ZOO can approximate the gradients of the objective function with finite-difference numerical estimates by only querying the network model. Thus the approximated gradient is utilized to generate the adversarial examples. Guo *et. al.* proposed a simple black-box adversarial attack called "SimBA" [10] to generate adversarial examples with a set of orthogonal vectors. By testing the output logits with the added chosen vector, the optimization direction can be effectively found.

Brendel *et. al.* developed a decision-based adversarial attack which is known as "Boundary attack" [3], it worked by iteratively perturbing another initial image that belongs to a different label toward the decision boundaries between the original label and the adjacent label. By querying the model with enough perturbed images, the boundary, as well as the perturbation, can be found thus generating adversarial examples. Chen *et. al.* proposed another decision-based attack named hop-skip-jump attack (HSJA) [5] recently. By only utilizing the binary information at the decision boundary and the Monte-Carlo estimation, the gradient direction of the network can be found so as to realize the adversarial examples generation. Based on [5], Li *et. al.* [17] proposed a query-efficient boundary-based black-box attack named QEBA which estimate the gradient of the boundary in several transformed space and ef-

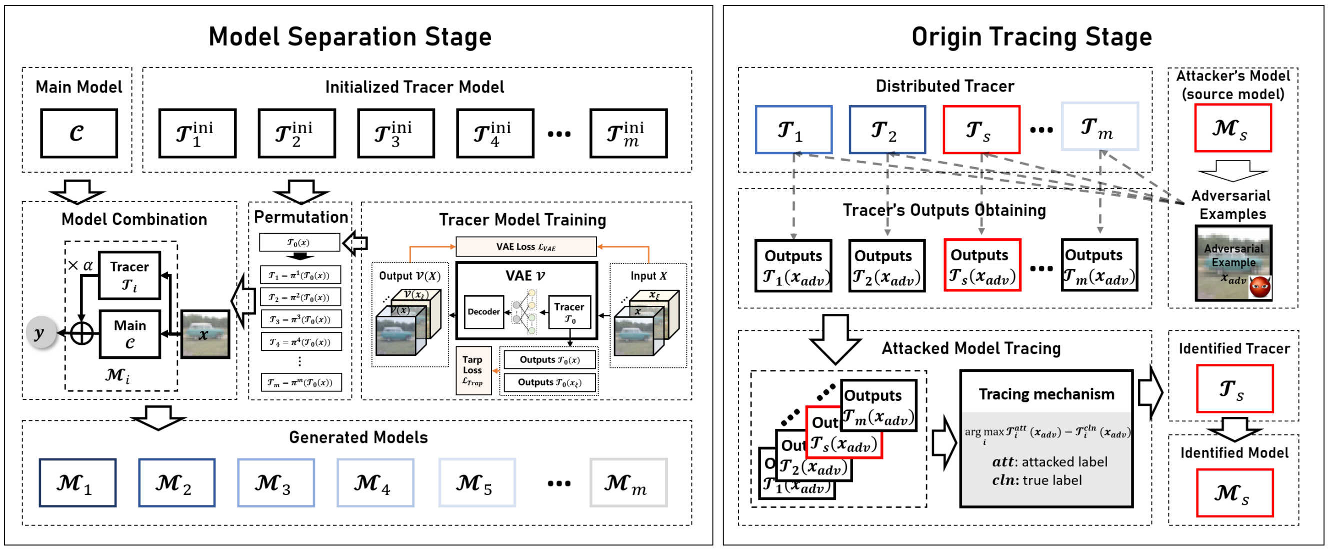


Figure 2: The framework of the proposed method. The left part of the framework indicates the separation process of the seller's distributed models $\mathcal{M}_i, i \in [1, m]$. The right part of the framework illustrates the origin tracing process.

fectively reduce the query numbers in generating the adversarial examples. Maho *et. al.* [18] proposed a surrogate-free black-box attack which does not estimate the gradient but searches the boundary based on polar coordinates, compared with [5] and [17], [18] achieves less distortion with fewer query numbers.

3. Proposed Framework

3.1. Main Idea

Considering that black-box adversarial attacks are performed by estimating and attacking across the boundary of the model, for the purpose of tracing, each distributed model should maintain different boundaries, besides, a unique feature of the source boundary should be carried on the generated adversarial example.

To achieve this goal, two essential questions should be addressed: **Q1.** How to generate multiple models with different boundaries while remaining highly accurate on the classification task? **Q2.** How to inject the unique features of the source boundary during the black-box adversarial attack process?

Tackling these two questions at the same time is not a trivial task, especially for **Q2**, where the black-box adversarial attack method is unknown to us (the defender). In this paper, we designed a parallel-network-based method to meet the requirements, where the basic component of the model is a paralleled structure that pairs a unique network named tracer with the original classifier. The tracer could effectively fluctuate the boundary and inject unique features during the attack. Such features could be further reflected by the output logits of the tracer.

The proposed framework contains two main stages: the model separation stage and the origin tracing stage, as illustrated in Fig.2. During the model separation stage, we want to generate multiple models which are sufficiently different under adversarial attack while remaining highly accurate on the classification task. As for origin tracing, we exploit unique features of different tracers in the parallel structure, which can be observed in the tracers' logits. Hence, our tracing process is conducted by feeding the adversarial examples into the tracers and analyzing their output.

3.2. Model Separation

One goal of model separation is to generate m distributed models $\mathcal{M}_i, i \in [1, m]$ with similar classification functions but different boundaries. To achieve this goal, we design a parallel network structure, which contains a tracer model $\mathcal{T}_i, i \in [1, m]$ and a main model $\mathcal{C}$, as shown in Fig. 2. $\mathcal{C}$ is the network trained for the original task. Each $\mathcal{T}_i$ is used for fluctuating the boundaries of $\mathcal{C}$. The final results are determined by both $\mathcal{C}$ and each $\mathcal{T}_i$ with a weight parameter α. The specific workflow of each specific $\mathcal{M}_i$ can be described as: For input image x, $\mathcal{T}_i$ and $\mathcal{C}$ both receive the same x and output two different vectors $\mathcal{T}_i(x)$ and $\mathcal{C}(x)$ respectively. $\mathcal{T}_i(x)$ and $\mathcal{C}(x)$ have the same size and will be further added in a weighted way to generate the final outputs $\mathcal{M}_i(x)$, as shown in Eq. 1.

$$\mathcal{M}_i(x) = \mathcal{C}(x) + \alpha \times \mathcal{T}_i(x) \tag{1}$$

where α is the weight parameter that controls the weight of $\mathcal{T}_i$ in the final results. The output value of each $\mathcal{T}_i$ ranges from [-1,1] and the output value of $\mathcal{C}(x)$ is normalized into

[0,1]. In each $\mathcal{M}_i$, $\mathcal{C}$ is fixed and only $\mathcal{T}_i$ is different. For the main classification task, $\mathcal{C}$ only has to be trained once. $\mathcal{C}$ and $\mathcal{T}_i$'s are trained separately. Training of $\mathcal{T}_i$'s has access to some data with similar domain of $\mathcal{C}$.

In addition to the goal of fluctuating the boundaries, $\mathcal{T}_i$ also takes the responsibility of injecting unique features during attacks, where the source tracer $\mathcal{T}_s$ should give unique responses on the adversarial examples.

3.2.1 $\mathcal{T}_0$ generation

We can treat each $\mathcal{T}_i$ as a classifier of K classes where K is the number of classes of $\mathcal{C}$.

Requirements: The goals of $\mathcal{T}_i$ are to induce different boundaries among the $\mathcal{M}_i$'s. Intuitively, the tracers should meet the following requirements:

1. Each $\mathcal{T}_i$ is easier to be attacked than $\mathcal{C}$.
2. The T_i's have a similar feature space as $\mathcal{C}$.
3. The classes in each $\mathcal{T}_i$ do not overlap with classes in $\mathcal{C}$.

Recap that the adversarial attack perturbs the clean input so as to cross boundaries of $\mathcal{M}_i$. Both $\mathcal{T}_i$ and $\mathcal{C}$ would contribute to the perturbation. If $\mathcal{T}_i$ is easier to be attacked, $\mathcal{T}_i$ would contribute more to the perturbation and thus induce tracer's features to the adversarial example. Hence, we have the first requirement, which lurks the adversarial attack toward the features of the tracer. The second requirement ensures the same feature space of each $\mathcal{T}_i$ and $\mathcal{C}$, such that the feature-based attack of $\mathcal{C}$ can also work on $\mathcal{T}_i$. The last requirement requires the boundaries of $\mathcal{T}_i$ and $\mathcal{C}$ are not overlapped, which makes sure the perturbations of $\mathcal{T}_i$ and $\mathcal{C}$ will not significantly interfere each other.

Method: There are many ways of designing $\mathcal{T}_i$'s, in this paper, we propose a simple but effective method, which first obtains a $\mathcal{T}_0$ by a variational autoencoder (VAE)-based training method, then uses a perturbation-based method to separate each $\mathcal{T}_i$ with the well-trained $\mathcal{T}_0$. Specifically, as shown in Fig. 3, $\mathcal{T}_0$ is the encoder part of a VAE $\mathcal{V}$, which is linearly cascaded with one "SingleConv" block (Conv-BN-ReLU), two "Res-block" [11], one "Conv" block, one full connection block and one "Tanh" activation layer. For the decoder part of the VAE, it consists of three "Double-Conv" block (2-"SingleConv"), two "up-conv" (UpSample-Conv-BN-ReLU) one "Conv" block and one full connection block. The training process of $\mathcal{T}_0$ can be described as:

1) Given $\mathcal{V}$ and the image $x \in \mathbb{R}^{B \times C \times H \times W}$ from the training dataset of $\mathcal{C}$, we first initialize $\mathcal{V}$ with random parameters.

2) For each training epoch, we add random noise ξ [1] on the input images x to generate the noised image $x_\xi \in \mathbb{R}^{B \times C \times H \times W}$, where B indicates the batch size and $C \times H \times W$ represents the size of the image.

[1]ξ follows a uniform distribution over [0, 0.01)

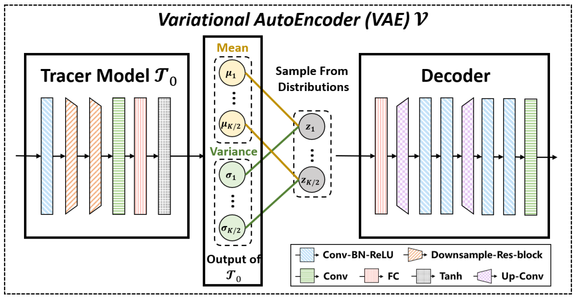


Figure 3: The architecture of VAE $\mathcal{V}$ and tracer $\mathcal{T}_0$.

3) Then we concatenate x and x_ξ in batch-dimension (denoted as $X \in \mathbb{R}^{2\dot{B} \times C \times H \times W}$) and feed them into $\mathcal{V}$ to get the outputs $\mathcal{V}(X) \in \mathbb{R}^{2\dot{B} \times C \times H \times W}$ and $\mathcal{T}_0(X) \in \mathbb{R}^{2\dot{B} \times K}$, where K is the number of classes. $\mathcal{T}_0(X)$ can be split in the batch-dimension to $\mathcal{T}_0(x) \in \mathbb{R}^{B \times K}$ and $\mathcal{T}_0(x_\xi) \in \mathbb{R}^{B \times K}$. Besides, same as the traditional VAE training, $\mathcal{T}_0(X)$ also should be divided into two parts in another dimension $\mu \in \mathbb{R}^{2\dot{B} \times K/2}$ and $\sigma \in \mathbb{R}^{2\dot{B} \times K/2}$, aiming for providing the mean and the variance for sampling the latent variables $Z \in \mathbb{R}^{2\dot{B} \times K/2}$, which is the input of the decoder.

The whole loss function of $\mathcal{V}$ can be written as:

$$\mathcal{L} = \lambda \mathcal{L}_{VAE} + \mathcal{L}_{Trap} \tag{2}$$

where

$$\begin{aligned} \mathcal{L}_{VAE} &= \mathcal{L}_{con} + \mathcal{L}_{KL} \\ &= \|\mathcal{V}(X) - X\|_2 + KL(\mathbb{N}(\mu, \sigma^2) \| \mathbb{N}(0,1)) \end{aligned} \tag{3}$$

and

$$\mathcal{L}_{Trap} = \frac{\mathcal{T}_0(x) \otimes \mathcal{T}_0(x_\xi)}{\|\mathcal{T}_0(x)\|_2 \|\mathcal{T}_0(x_\xi)\|_2} - \cos(\theta) \tag{4}$$

$\|\|$ indicates the mean square error loss and KL indicates the Kullback-Leibler divergence. $\otimes$ represents the Hadamard product. λ is the parameter to control the weight of $\mathcal{L}_{VAE}$, which is set as 0.001. θ is the parameter that controls the cosine similarity of $\mathcal{T}_0(x)$ and $\mathcal{T}_0(x_\xi)$. The selection and influence of θ will be discussed in Section 5.1.

3.2.2 $\mathcal{T}_i$ Separation

From $\mathcal{T}_0$, we want to generation $\mathcal{T}_i, i \in [1, m]$. In order to realize tracing, each distributed $\mathcal{T}_i$ for different buyers should maintain different boundaries so that the adversarial perturbation of the i^{th} copy will not produce the same output logits on the j^{th} copy. So we proposed a permutation-based method:

$$\mathcal{T}_i(x) = \pi_i(\mathcal{T}_0(x)) \tag{5}$$

where π_i indicates the i^{th} permutation, x indicates the input images. π should satisfy: No two permutations "overlap" more than u elements, where u is a pre-defined constant. That is, for any two permutations, say π_i and π_j,

$$\left|\{k|\pi_i(k) = \pi_j(k)\}_{k=1}^{K}\right| \leq u$$

where K indicates the number of classes. When u is small, the accuracy of tracing would improve, but trade-off with smaller possible copy numbers. In our experiment, we choose u=1. To illustrate the permutation, we give an example: assume $\mathcal{T}_0$ is a four-class classifier, a possible permutation could be $\pi_i = (1,2,3,4) \rightarrow (2,3,4,1)$, and $\pi_j = (1,2,3,4) \rightarrow (3,4,1,2)$.

Discussion: To say how our training method meets the requirement, we give the following analysis. For the first requirement, it is satisfied by the trap loss $\mathcal{L}_{Trap}$. Through Eq. 4 we can see when $\theta = 90°$, $\mathcal{T}_0(x)$ and $\mathcal{T}_0(x_\xi)$ will be orthogonal, which means $\mathcal{T}_0(x)$ will be significantly different from $\mathcal{T}_0(x_\xi)$. Hence by setting appropriate θ, we could guarantee that the output of $\mathcal{T}_0$ is easy to be changed by small noise, such that $\mathcal{T}_0$ is easy to be attacked to cross the boundary. Training $\mathcal{T}_0(X)$ with the same distribution of training dataset of $\mathcal{C}$ satisfies the second requirement. As for the third requirement, it is satisfied by the loss function of Eq. 3. Through training, $\mathcal{T}_0$ could effectively squeeze the feature of a single image into a K-dimension vector, where each dimension can be regarded as a class, and such classification criterion is not based on the original classification task but on the VAE reconstruction task. So the classes in $\mathcal{T}_0$ do not overlap with classes in $\mathcal{C}$. Besides, the permutation-based method will not influence the properties of $\mathcal{T}_0$, so each $\mathcal{T}_i$ will satisfy these three requirements.

3.3. Tracing the Origin

Given an adversarial example x_{adv}, which is obtained by attacking one of m copies, we want to trace/determine which copy it is derived from. w.l.o.g., let us assume that the m copies are $\mathcal{M}_1, \mathcal{M}_2, ..., \mathcal{M}_m$. By earlier argument in section 3.2.1, the adversarial perturbations would be contributed more by the tracer compared to the classifier. Hence, we propose the following logits-based tracing mechanism:

1) Given an appeared adversarial example denoted as x_{adv}, we feed x_{adv} into all $\mathcal{M}_i, i \in [1,m]$ and obtain the output logits of $\mathcal{M}_i$ and $\mathcal{T}_i$, noted as $\mathcal{M}_i(x_{adv}), \mathcal{T}_i(x_{adv}), i \in [1,m]$.

2) Then we sort $\mathcal{M}_i(x_{adv})$ and take out the index corresponding to the first sort noted as att and second sort noted as cln, att indicates the potential attack label and cln indicates the potential clean label.

3) We obtain the outputs of $\mathcal{T}_i(x_{adv})$ corresponded to the label att and cln, denoted as $\mathcal{T}_i(x_{adv})[att]$ and $\mathcal{T}_i(x_{adv})[cln]$.

4) The source model can be determined by:

$$s = \underset{i, i \in [1,m]}{\arg\max}(\mathcal{T}_i(x_{adv})[att] - \mathcal{T}_i(x_{adv})[cln]) \quad (6)$$

To simplify the description, we denote the difference of output logits $(\mathcal{T}_i(x_{adv})[att] - \mathcal{T}_i(x_{adv})[cln])$ as DOL. The tracer corresponding to the largest DOL is identified as the source model.

4. Experimental Results

4.1. Implementation Details

In order to show the effectiveness of the proposed framework, we perform the experiments on two network architectures (ResNet18 [11] and VGG16 [25]) with two small image datasets (CIFAR10 [15] of 10 classes and GTSRB [13] of 43 classes) and two deeper network architecture (ResNet50 and VGG19) with one big image dataset (mini-ImageNet [24] of 100 classes). The main classifier $\mathcal{C}$ in experiments is trained for 200 epochs. All the model training is implemented by PyTorch and executed on NVIDIA RTX 2080ti. For gradient descent, Adam [14] with learning rate of 1e-4 is applied as the optimization method.

4.2. The Classification Accuracy of The Proposed Architecture

The most influenced parameter for the classification accuracy is the weight parameter α. α determines the participation ratio of $\mathcal{T}_i$ in final outputs. To investigate the influence of α, we change the value of α from 0 (baseline) to 0.2 and record the corresponding classification accuracy of each task, the results are shown in Table 1.

α	CIFAR10		GTSRB		Mini-ImageNet	
	ResNet18	VGG16	ResNet18	VGG16	ResNet50	VGG19
0	94.30%	93.68%	96.19%	97.59%	81.81%	75.81%
0.05	94.24%	93.64%	96.14%	97.52%	81.73%	75.80%
0.1	94.24%	93.63%	96.07%	97.36%	81.56%	75.75%
0.15	94.07%	93.63%	95.72%	96.84%	81.52%	75.73%
0.2	93.95%	93.57%	95.09%	95.52%	81.38%	75.70%

Table 1: The classification accuracy with different α.

It can be seen from Table 1 that for all the classification tasks, the growth of α will seldom decrease the accuracy of the classification task. Compared with the baseline ($\alpha = 0$), when α is in the range of 0.05 to 0.15, the reduction in classification accuracy will not exceed 1%. But when $\alpha = 0.2$, the accuracy decrease of the dataset "GTSRB" is more than 1%. We intended the influence of α on classification accuracy to be as small as possible, so the subsequent experiments are completed with $\alpha = \{0.05, 0.1, 0.15\}$.

4.3. Traceability of different black-box attack

Setup and Code. To verify the traceability of the proposed mechanism, we conduct experiments on 5 distributed models $\mathcal{M}_i, i \in [1,5]$. All the tracers of $\mathcal{M}_i$ are generated with the permutation-based method with a well-trained $\mathcal{T}_0$, and $\mathcal{T}_0$ is trained with $\theta = 75°$ for 400 epochs. We set one model as the source model $\mathcal{M}_s$ to perform the adversarial attack and set the other models as the victim models $\mathcal{M}_{v_i}, i \in [1,5] \backslash s$, where the tracer of $\mathcal{M}_s$ and $\mathcal{M}_{v_i}$ are denoted as $\mathcal{T}_s$ and $\mathcal{T}_{v_i}$. The goal is to test whether the proposed scheme can effectively trace the source model from the generated adversarial examples. The black-box attack we choose is Boundary [3], HSJA [5], QEBA [17] and SurFree [18]. For Boundary [3] and HSJA [5], we use Adversarial Robustness Toolbox (ART) [22] platform to conduct the experiments. For QEBA [17] and SurFree [18], we pull implementations from their respective GitHub repositories [2] [3] with default parameters. For each α, each network architecture, each dataset and each attack, we generate 1000 ***successfully attacked adversarial examples*** of $\mathcal{M}_s$ and conduct the tracing experiment.

Evaluation Metrics. Traceability is evaluated by the successful tracing accuracy, which is calculated by:

$$\text{Acc} = \frac{N_{\text{correct}}}{N_{\text{All}}} \tag{7}$$

where N_{correct} indicates the number of correct-tracing samples and N_{All} indicates the total number of samples, which is set as 1000 in the experiments. The tracing performance of different attacks with different settings is shown in Table 2.

The influence of α. We can see from Table 2 that the tracing accuracy increases with the increase of α. We conclude the reason as: α determines the participation rate of tracer $\mathcal{T}_i$ in final output logits, the larger α will make the final decision boundary rely more on $\mathcal{T}_i$. Therefore, when α gets larger, the DOL of $\mathcal{T}_i$ is more likely to be pushed larger, thus leading to better tracing performance.

The influence of network architecture. The tracing results vary with different networks and different datasets. With the same dataset, the tracing accuracy of ResNet18 will be higher than that of VGG16. We attribute the reason to the complexity of the model architecture. According to [26], compared with ResNet, the structure of VGG is less robust, so VGG-based $\mathcal{C}$ might be easier to be adversarially attacked. Therefore, once $\mathcal{C}$ is attacked, there is a certain probability that $\mathcal{T}_i$ is not attacked as we expected, so the DOL of $\mathcal{T}_i$ will not produce the expected features for tracing. Fortunately, the network architecture can be designed by us, so in practice, choosing a robust architecture of $\mathcal{C}$ would be better for tracing.

[2]QEBA:https://github.com/AI-secure/QEBA

[3]SurFree:https://github.com/t-maho/SurFree

The influence of classification task. In our experiments, we test the classification task with the different numbers of classes. It can be seen that with the increase in classification task complexity, traceability performance decreases slightly. But when $\alpha = 0.15$, the traceability ability can still reach more than 94% in most cases.

The influence of black-box attack. The mechanism of the black-box attack greatly influences the tracing performance. For Boundary attack [3], HSJA [5] and QEBA [17], the tracing accuracy shows similar results, but for SurFree [18], the tracing accuracy will be worse than that of the other attacks. The reason is that Boundary attack, HSJA, QEBA are gradient-estimation-based attacks, which try to use random noise to estimate the gradient of the network and further attack along the gradient. Since the gradient is highly related to $\mathcal{T}_i$, such attacks are more likely to be trapped by $\mathcal{T}_i$. But SurFree [18] is attacking based on geometric characteristics of the boundary, which may ignore the trap of $\mathcal{T}_i$ especially when α is small. So compared with Boundary attack, HSJA and QEBA, the proposed mechanism may get slightly worse performance when facing SurFree attack.

4.4. The influence of distributed copy numbers

It can be expected that as the number of distributed copies increases, the differences between different boundaries will become less and less, making it more difficult to complete the tracing. In this section, we mainly discuss how traceability will evolve with the number of copies.

Since we ensure that each $\mathcal{T}_i$ maintains different boundaries, when feeding the adversarial examples, we have the following assumption: the DOL of $\mathcal{T}_s$ and that of $\mathcal{T}_{v_i}$ should follow different distributions, and the DOL of each $\mathcal{T}_{v_i}$ should follow a similar distribution.

Distribution: In order to verify the correctness of the assumption, we perform the following experiments. We use "ResNet" as the backbone and QEBA [17] as the attack method. The datasets we test are CIFAR10, GTSRB and MiniImageNet. α is fixed as 0.15. For each task, We first generate 10 different $\mathcal{T}_i$ according to the permutation-based method, then randomly choose one as the source model $\mathcal{M}_s$ to generate the adversarial examples. Then we feed the adversarial examples on each $\mathcal{T}_i$ and record the distribution of the resulting DOL of the source tracer and victim tracers (denoted as $\mathbb{D}_s$ and $\mathbb{D}_{v_i}, i \in [1,9]$). We perform the Kolmogorov-Smirnov test between every two possible $\mathbb{D}_{v_i}, \mathbb{D}_{v_j} i,j \in [1,9], i \neq j$ and between $\mathbb{D}_s$ and $\mathbb{D}_{v_i}$. Then we record Kolmogorov's D statistic (larger values indicate larger differences) to measure the similarity of these distributions. Results are shown in Table 3.

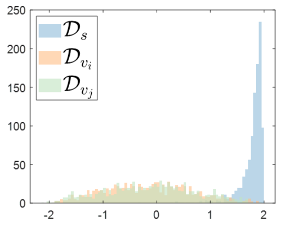
$\{\mathbb{D}_s, \mathbb{D}_{v_i}\}$ indicates the Kolmogorov's D statistic (KD) value between $\mathbb{D}_s$ and $\mathbb{D}_{v_i}$ where $\{\mathbb{D}_{v_i}, \mathbb{D}_{v_j}\}$ indicates the mean KD value between $\mathbb{D}_{v_i}$ and $\mathbb{D}_{v_j}$. It can be seen that

Attack		Boundary			HSJA			QEBA			SurFree		
alpha		0.05	0.1	0.15	0.05	0.1	0.15	0.05	0.1	0.15	0.05	0.1	0.15
CIFAR10	ResNet18	97.3%	97.4%	**98.0%**	95.2%	97.3%	**97.5%**	94.4%	97.1%	**98.0%**	83.3%.	92.3%	**95.9 %**
	VGG16	85.7%	96.1%	97.8%	87.3%	88.9%	90.2%	73.1%	90.5%	92.8%	70.0%	87.8%	94.2%
GTSRB	ResNet18	87.3%	94.2%	**95.7%**	87.3%	92.5%	**96.4%**	84.7%	95.3%	**96.7%**	77.8%	92.1%	**94.7%**
	VGG16	89.3%	93.9%	95.6%	88.9%	92.7%	95.2%	81.0%	91.8%	95.6%	81.4%	92.8%	94.0%
mini-ImageNet	ResNet50	89.9%	93.4%	**94.5%**	89.4%	93.7%	**94.2%**	89.0%	94.4%	**96.4%**	85.1%	92.5%	**93.3%**
	VGG19	80.6 %	91.0 %	93.4%	81.7 %	89.3 %	90.7%	80.6%	88.6%	92.7 %	85.7%	90.7%	93.3%

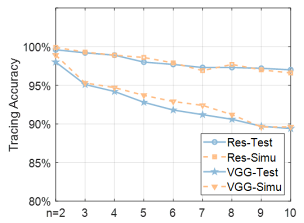
Table 2: The trace accuracy of different attacks.

KD	CIFAR10		GTSRB		Mini-ImageNet	
	ResNet18	VGG16	ResNet18	VGG16	ResNet50	VGG19
$\{\mathbb{D}_s,\mathbb{D}_{v_i}\}$	0.925	0.818	0.850	0.807	0.873	0.821
$\{\mathbb{D}_{v_i},\mathbb{D}_{v_j}\}$	0.040	0.047	0.034	0.051	0.063	0.092

Table 3: The Kolmogorov's D statistic of $\mathbb{D}_s$ and $\mathbb{D}_{v_i}$.



(a) The distribution of DOL with task "ResNet-CIFAR10".



(b) The tracing results of multiple models with dataset "CIFAR10".

Figure 4: The distribution of DOL and tracing performance of multiple branches.

for all the tasks, the KD value between $\mathbb{D}_s$ and $\mathbb{D}_{v_i}$ is large, which indicates that $\mathbb{D}_s$ and $\mathbb{D}_{v_i}$ follows the different distribution. Meanwhile, the KD value between different $\mathbb{D}_{v_i}$ is small, which means that each $\mathbb{D}_{v_i}$ follows the same distribution. Besides, we also visually show $\mathbb{D}_s$ and two random selected $\mathbb{D}_{v_i}$ and $\mathbb{D}_{v_j}$ with "ResNet-CIFAR10" task in Fig. 4a for better illustration.

Tracing Rate Estimation: Since $\mathbb{D}_s$ and $\mathbb{D}_{v_i}$ follows the different distribution and $\mathbb{D}_{v_i}$ and $\mathbb{D}_{v_j}$ follows the same distribution, we could effectively estimate the tracing performance of m distributed copies by Monte-Carlo sampling based on $\mathbb{D}_s$ and one random selected $\mathbb{D}_{v_i}$. We have conducted experiments to evaluate the performance of such estimation on $n, n \in [2, 10]$ copies. Specifically, for n tracers, we take one sample S_s from $\mathbb{D}_s$ and $n-1$ samples $\mathbb{S}_v^{n-1}$ from $\mathbb{D}_{v_i}$. Such sampling is repeated 1000 times. For each sampling, if $S_s > max(\mathbb{S}_v^{n-1})$, we consider it as a successful tracing sample. We record the total number of successful tracing samples N_C^n in 1000 samplings. The final tracing accuracy of n models can be calculated with N_C^n/1000.

The results are shown in Fig. 4b. It can be seen that with the increasing number of distributed copies, the tracing accuracy gradually decreases. But with 10 branches, it can still maintain more than 96% accuracy for "ResNet-CIFAR10". Besides, the estimated tracing performance is almost the same as the actual experiment results, which indicates the correctness of our analysis. Therefore, the tracing performance of a large number of copies could be effectively estimated by $\mathbb{D}_s$ and $\mathbb{D}_{v_i}$. The results of GTSRB and mini-ImageNet are shown in the supplementary material.

Besides, since the valid copy numbers with the permutation-based method are $K \times (K-1)$ when $u = 1$, which is limited when K is small, we also provide a possible way to enlarge the number by scrambling the input of $\mathcal{T}_i$, such a method is illustrated in the supplementary material.

5. Discussion

5.1. The influence of θ

In this section, we will introduce the influence of parameter θ on tracing accuracy. As shown in Eq. 4, θ controls the cosine similarity between $\mathcal{T}_0(x)$ and $\mathcal{T}_0(x_\xi)$. A larger θ will lead to larger output changes of $\mathcal{T}_0$ when facing a certain ξ, which may make $\mathcal{T}_0$ to be more vulnerable to attack. But θ should not be as large as possible too, because training with larger θ may on the other hand make $\mathcal{T}_0$ change too heavily with ξ, which is also not conducive to inducing adversarial attacks to find the best perturbations of $\mathcal{T}_0$. To illustrate the influence of θ, we conduct the following experiments.

We use the task of "ResNet-CIFAR10", the attack of QEBA [17], and train $\mathcal{T}_0$ with different θ to show the specific tracing results on 5 copies. α is set as 0.15. θ is chosen from 15° to 90° with an interval 15°. The results are shown in Table 4. It can be seen that when θ ranges from 15° to 75°, the tracing accuracy increases as the θ increases, but for $\theta = 90°$, the tracing accuracy is lower than $\theta = 75°$, which justify our analysis. Finding the best θ for training could be an interesting problem, but in this paper, we focus

θ	15°	30°	45°	60°	75°	90°
Acc	31.8%	80.0%	94.5%	95.1%	98.0%	95.0%

Table 4: The trace accuracy of different θ.

more on evaluating the possibility of tracing, so we utilize $\theta = 75^\circ$ as the experimental parameters.

5.2. Non-transferability and traceability

The concept of traceability is related to but not equivalent to non-transferability. A non-transferable adversarial example works only on the model it is generated from. Therefore, tracing such an example may be a straightforward task. On the other hand, a transferable sample may be generic enough to work on many copies/models. The task of tracing becomes more meaningful in this scenario. Our traceability towards the transferable example demonstrates that the process of adversarial attack introduces distinct and unique traceable features to the source model. In this sense, traceability can serve as a fail-safe property in defending against adversarial attacks. There are many defense methods that can satisfy non-transferability, but once the defense fails, the model will not be effectively protected. But our experimental results show that for the proposed method, even if the defense fails, we still have a certain probability to trace the attacked model, as shown in Table 5. We use "ResNet-CIFAR10" task with HSJA [5] and QEBA [17] as examples to show the specific tracing results on 5 copies.

Attack	α	**NTr**	**NTr**(+)	**Tr**	**Tr**(+)	**Tr** Rate	**Total** Rate
HSJA	0.05	314	314	686	638	93.00%	95.20%
	0.1	746	746	254	229	90.16%	97.50%
	0.15	892	892	108	81	75.00%	97.30%
QEBA	0.05	692	692	308	252	81.82%	94.40%
	0.1	682	682	318	289	90.88%	97.10%
	0.15	658	658	342	322	94.15%	98.00%

Table 5: Results of transferable/non-transferable samples.

In Table 5, **NTr** and **Tr** indicate the number of non-transferrable samples and transferrable samples respectively. **NTr**(+) and **Tr**(+) indicate the number of successful tracing samples of **NTr** and **Tr**. We can see that for QEBA with $\alpha = 0.1$ and 0.15, the traceability to transferrable samples is all keep at a high level which is greater than 90%. As for HSJA, when $\alpha = 0.05$, 686 samples can be transferred, and the traceability of transferrable examples achieves 93.00%. Although the traceability of transferrable examples decreases to 75.00% when $\alpha = 0.15$, only 108 samples are transferrable, which is much less compared to QEBA. So the total tracing rate is still at a high level of 97.30%. In general, the proposed method shows superior overall traceability and especially great tracing performance on transferrable samples.

5.3. Adaptive attacks & defense

In the buyers-seller setting, we assume only one buyer is a potential attacker, so the adversarial attack can only be conducted with 1 distributed model. However, if multiple buyers are the attackers, they could use multiple models to conduct the adaptive collusion attack.

Collusion attack: Assume the attacker can access multiple models, *e.g.* two models $\mathcal{M}_1$ and $\mathcal{M}_2$, he can generate the adversarial examples by iteratively attacking $\mathcal{M}_1$ and $\mathcal{M}_2$, and ensure the adversarial example works both on these two models. Such a combined model may offset the trap of $\mathcal{T}_1$ and $\mathcal{T}_2$ and make the attack focus more on the boundary of $\mathcal{C}$. Therefore, the generated adversarial examples will not carry much expected features of $\mathcal{T}_1$ and $\mathcal{T}_2$ thus causing troubles in tracing.

Adaptive Defense: One way to mitigate such attacks is making $\mathcal{C}$ in each $\mathcal{M}_i$ slightly different. For example, we could make each $\mathcal{C}_i$ maintain a different gradient direction by setting gradient-orthogonal loss [1]. Therefore, each $\mathcal{C}_i$ maintains different boundaries, so the boundary of the combined model (e.g. $\mathcal{C}_1$ and $\mathcal{C}_2$) will not same as the boundary of other models $\mathcal{C}_i, i \neq 1, 2$. Thus the attack may work on $\mathcal{M}_1$ and $\mathcal{M}_2$ but fail on $\mathcal{M}_i, i \neq 1, 2$, which ensures the non-transferability and traceability.

We have conducted the corresponding experiments to show the possible influence of the collusion attack and the performance of the adaptive defense. The task we use is "ResNet18-CIFAR10" and the attack method is Boundary attack [3]. We randomly select two models $\mathcal{M}_1$ and $\mathcal{M}_2$ as the attack model and test the tracing performance on other 4 models $\mathcal{M}_i, i \in [3, 6]$ with the collusion attack. Meanwhile, we also use the adaptive defense method to train 6 different $\mathcal{C}_i$ and conduct the same attack and tracing procedure. The specific results are shown in Table 6.

Model	$\mathcal{M}_1$	$\mathcal{M}_1 + \mathcal{M}_2$	$\mathcal{M}'_1 + \mathcal{M}'_2$
Acc	98.0%	50.0%	97.5%

Table 6: The trace accuracy of adaptive attack and defense.

$\mathcal{M}_1$ indicates the traceability of the attack on the single model, $\mathcal{M}_1 + \mathcal{M}_2$ indicates the traceability of the attack on the combined model. $\mathcal{M}'_1 + \mathcal{M}'_2$ indicates the traceability of the attack on the adaptive defense model. It can be seen that when suffering a collusion attack, the tracing accuracy will decrease from 98.0% to 50.0%. But with the adaptive defense model, the tracing accuracy can return back to 97.5%, which indicates the effectiveness of the adaptive defense.

6. Conclusion

This paper researches a new aspect of defending against adversarial attacks that is the traceability of adversarial attacks. The techniques derived could aid forensic investigation of known attacks, and provide deterrence to future attacks in the buyers-seller setting. As for the mechanism, we design a framework which contains two related components (model separation and origin tracing) to realize traceability.

For model separation, we propose a parallel network structure which pairs a unique tracer with the original classifier and a VAE-based training method. The tracer model can effectively injects the unique features and ensures the differences between distributed models. As for origin tracing, we design a logits-based tracing mechanism based on the tracer model which could sufficiently tracing the origin. The experiment of multi-dataset, multi-network model and multi-black-box attacks shows the effectiveness of the method in achieving traceability through the adversarial examples.

Acknowledgement. This research/project is supported by the National Research Foundation, Singapore under its Strategic Capability Research Centres Funding Initiative and Huawei International. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore.

References

[1] Huanyu Bian, Dongdong Chen, Kui Zhang, Hang Zhou, Xiaoyi Dong, Wenbo Zhou, Weiming Zhang, and Nenghai Yu. Adversarial defense via self-orthogonal randomization super-network. *Neurocomputing*, 452:147–158, 2021.

[2] Franziska Boenisch. A survey on model watermarking neural networks. *arXiv preprint arXiv:2009.12153*, 2020.

[3] Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. In *International Conference on Learning Representations, ICLR 2018*, 2018.

[4] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (S&P)*, pages 39–57. IEEE, 2017.

[5] Jianbo Chen, Michael I Jordan, and Martin J Wainwright. Hopskipjumpattack: A query-efficient decision-based attack. In *2020 IEEE Symposium on Security and Privacy (S&P)*, pages 1277–1294. IEEE, 2020.

[6] Pin-Yu Chen, Yash Sharma, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh. Ead: elastic-net attacks to deep neural networks via adversarial examples. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

[7] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proceedings of the 10th ACM workshop on Artificial Intelligence and Security*, pages 15–26, 2017.

[8] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[9] Shixiang Gu and Luca Rigazio. Towards deep neural network architectures robust to adversarial examples. *arXiv preprint arXiv:1412.5068*, 2014.

[10] Chuan Guo, Jacob Gardner, Yurong You, Andrew Gordon Wilson, and Kilian Weinberger. Simple black-box adversarial attacks. In *International Conference on Machine Learning*, pages 2484–2493. PMLR, 2019.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[12] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[13] Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, and Christian Igel. Detection of traffic signs in real-world images: The german traffic sign detection benchmark. In *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. Ieee, 2013.

[14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015*, 2015.

[15] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[16] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.

[17] Huichen Li, Xiaojun Xu, Xiaolu Zhang, Shuang Yang, and Bo Li. Qeba: Query-efficient boundary-based blackbox attack. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1221–1230, 2020.

[18] Thibault Maho, Teddy Furon, and Erwan Le Merrer. Surfree: a fast surrogate-free black-box attack. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10430–10439, 2021.

[19] Nasir Memon and Ping Wah Wong. A buyer-seller watermarking protocol. *IEEE Transactions on image processing*, 10(4):643–649, 2001.

[20] Dongyu Meng and Hao Chen. Magnet: a two-pronged defense against adversarial examples. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 135–147, 2017.

[21] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2574–2582, 2016.

[22] Maria-Irina Nicolae, Mathieu Sinn, Minh Ngoc Tran, Beat Buesser, Ambrish Rawat, Martin Wistuba, Valentina Zantedeschi, Nathalie Baracaldo, Bryant Chen, Heiko Ludwig, et al. Adversarial robustness toolbox v1. 0.0. *arXiv preprint arXiv:1807.01069*, 2018.

[23] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 372–387. IEEE, 2016.

[24] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. 2016.

[25] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[26] Dong Su, Huan Zhang, Hongge Chen, Jinfeng Yi, Pin-Yu Chen, and Yupeng Gao. Is robustness the cost of accuracy?– a comprehensive study on the robustness of 18 deep image classification models. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 631–648, 2018.

[27] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *2nd International Conference on Learning Representations, ICLR 2014*, 2014.

[28] Jiyi Zhang, Wesley Joon-Wie Tann, and Ee-Chien Chang. Mitigating adversarial attacks by distributing different copies to different users. *arXiv preprint arXiv:2111.15160*, 2021.