Final Year Project Final Report

Identification of Collusive Adversaries from a Single Adversarial Example via Machine Learning Watermarks

by

Wenbin Hu, Yize Cheng, Peihao Dou

Advised by

Prof. Minhao Cheng

Submitted in partial fulfillment

of the requirements for COMP 4981

in the

Department of Computer Science and Engineering

The Hong Kong University of Science and Technology

2023-2024

Abstract

Deep neural networks have been proven vulnerable to adversarial attacks. Despite there has been numerous defence mechanism proposed, most of them suffer from poor generalizability and scalability, making them impractical for real-life applications. Therefore, to help mitigate the threat of adversarial attacks, a new approach has emerged, which involves adopting a forensic perspective to trace the attacker responsible for generating the adversarial example. However, the current tracing mechanism can only tackle the case where only a single attacker is involved in the attack. But in reality, multiple attackers may collaborate with each other and produce a colluded adversarial example that would render the existing tracing framework ineffective. Therefore, in this project, we leverage K-Resilient Anti-Collusion Codes (ACC) and propose a novel watermark design. Particularly, we adopt an attractive class of ACCs, known as AND-ACCs, in our watermark design. We embed this watermark into machine learning models as a unique identifier for each model copy, and demonstrate both theoretically and empirically how this identifier and its "Anti-Collusion" characteristic can be leveraged to trace all attackers involved in a collusive attack.

Contents

1 Introduction

1.1 Overview

Deep Neural Networks (DNNs) have achieved remarkable success, yielding state-of-the-art results in various applications. However, it has also been discovered that DNNs are vulnerable to adversarial attacks [1, 2], which severely undermine the reliability of modern machine learning algorithms, hindering their real-world deployment. Numerous attacking algorithms [3–6], including both white-box and black-box approaches, have demonstrated the feasibility of carrying out such attacks in practical settings. These attacks share a common characteristic: the attacker generates an "adversarial example" by manipulating the test image used for model inference. The modified image appears visually indistinguishable to the original one but causes the model to make incorrect predictions.

To address the challenges posed by these attacks, several defense mechanisms [4, 7–9] have been proposed to enhance the resilience of models. However, these defenses often encounter issues related to scalability and generalizability, while also sacrificing accuracy when applied to clean images. As a result, these defense methods are not practical for real-world deployment. Considering the impracticality of defending against such attacks, an alternative approach to mitigate the threat is to establish traceability of the attackers. By implementing a tracing mechanism capable of identifying the culprits, potential attackers would be deterred due to the fear of being detected. Unfortunately, limited research has been conducted on the forensic aspect of tracing the model copy exploited by an attacker during an attack. Previous work by Cheng *et al.*[10] focuses solely on scenarios involving a single attacker. However, in reality, multiple attackers may collude and collaborate to launch adversarial attacks, rendering the existing tracing method by Cheng *et al.*[10] ineffective. Therefore, the objective of this project is to design a tracing mechanism capable of identifying attackers even in scenarios involving collusion.

1.2 Setting and Objective

We consider the problem in the following setting. Suppose there is a machine learning service provider that trained its K-way classification model, and distributes this model to n clients. Each client i is provided with API access to query the distributed model f_i to obtain the classification result of the client's input. However, the client has no internal information regarding the model structure, model parameters, or the model training process. In other words, client i only have black-box access to the distributed model f_i. This is a valid assumption as for the sake of protecting intellectual property, the machine learning provider has the incentive to conceal the model internal information from the clients. Furthermore, having API access to query the distributed model is sufficient for the client to use the provided service. Unfortunately, one of the clients maliciously conduct a black-box adversarial attack using the methods proposed in [5][6] by querying the distributed model. Moreover, some of the clients are in collusion with one another and may collaborate in attacking the model. Since all clients' model copies are trained in the same manner with the same training data, an adversarial example crafted for one or a few of the clients may also successfully achieve transfer attacks on other distributed model copies. The setting is illustrated in Figure 1.

Our objective is to trace all participated malicious clients in a collusive adversary setting using the generated colluded adversarial example $x_{adv}^{collude}$ and the clean image x_{clean}. A practical example of this is that one or several

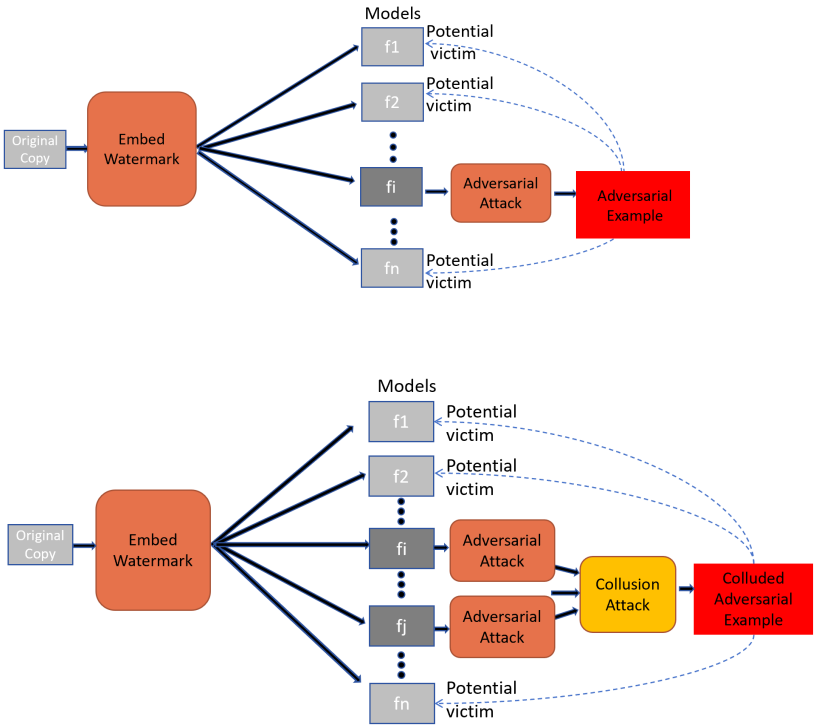

Figure 1: Illustration of the setting of our task. The upper figure shows the case where only one attacker is involved, which is what Cheng *et al*. [10] focused on. The lower figure shows the case where multiple clients collaborate in conducting the attack and result in a collusion attack. The light gray models represent the models distributed to benign clients and the dark gray models are the ones queried by malicious clients to generate the adversarial examples. In both cases, the generated adversarial example can potentially fool other clients using the same machine learning service. Our objective is to trace all the malicious clients from the adversarial example generated by them.

clients of an autonomous driving system use their autonomous driving cars to generate an adversarial sticker for a road sign. Since all clients share the same system, the adversarial sticker can very likely fool other autonomous driving cars that use the same autonomous driving system. Hence, the objective is to trace the involved clients in this attack using the adversarial sticker and the original road sign.

1.3 Related Works

1.3.1 Adversarial Attacks

Generally, adversarial attacks involve computing gradients to obtain a perturbation pattern that can deceive a trained model. Initially, several studies highlighted the potential threat of adversarial attacks [1, 11]. These early works primarily focus on the white box attack setting, where attackers have full access to the model's internal structure, weights, and output logits. However, the white box setting is not very realistic as model providers often aim to keep the model internal structure and weights confidential to the model users. As a result, the soft-label black-box attack setting, which only permits attackers to access the output classification logits of the trained models, has gained more attention by researchers. They have discovered that by leveraging the output logits of the models, attackers can estimate gradient flows inside the model and carry out adversarial attacks using optimization techniques [3, 12]. In an even more challenging scenario known as hard-label black-box attack, where attackers can only access the model's final decision result, successful adversarial attacks can still be performed [5, 6] via

zeroth-order optimization. Given the significant threat posed by adversarial attacks, there is an urgent and critical need to mitigate this risk.

1.3.2 Adversary Tracing

Given the difficulties encountered by defense methods in effectively countering adversarial attacks, particularly in terms of generalizability and scalability, researchers have recently redirected their efforts towards exploring methods for forensic investigation [10, 13]. These works involve embedding different watermarks in the model training pipeline for different model copies and utilizing the resulting differences to trace adversaries. For example, Cheng *et al.* [10] embed watermarks into training images, while Fang *et al.* [13] introduce an additional watermarking module in the model. Although existing methods have achieved decent tracing accuracy when dealing with individual adversarial attackers, they are ineffective in a collusive attack, where multiple adversaries collaborate to produce collusive adversarial examples. Surprisingly, there is little research investigating the tractability problem in collusive settings. Hence in this project, we have developed a framework that is capable of tracing all participated attackers in a collusive adversarial attack.

1.3.3 Anti-Collusion Codes

Anti-collusion codes (ACCs) are cryptographic techniques designed to prevent collusion among multiple parties in various scenarios, such as online auctions, voting systems, or distributed computing environments. The primary purpose of ACCs is to ensure fairness, integrity, and confidentiality in situations where multiple participants might attempt to collude to gain an unfair advantage. The concept of ACCs revolves around the use of cryptographic primitives and protocols to detect and deter collusion. These techniques employ various cryptographic tools, including encryption, digital signatures, zero-knowledge proofs, and secure multiparty computation.

AND-ACCs, introduced by Trappe *et al.* in 2003 [14], represents an intriguing category of ACCs. Trappe *et al.* define an AND-ACC as follows: "A code consisting of n binary vectors, each with a length of v, is referred to as a K-resilient AND-ACC if the element-wise AND operation between any subset of K or fewer code vectors yields a distinct result compared to the element-wise AND of any other subset of K or fewer code vectors." A code that satisfies this definition is called a (v, n, K) AND-ACC. In the subsequent sections, we will demonstrate how our watermark design exploits the unique "Anti-Collusion" characteristic of (v, n, K) AND-ACCs.

2 Methodology

The general tracing mechanism largely follows the idea proposed by Cheng *et al.* [10] for identifying the attacker in the single adversary setting. However, we improve this idea by using Anti-Collusion Codes (ACC) in the watermark design to enable the model owner to trace all participated clients in the case of a collusive attack.

The overall pipeline is shown in Figure 2, which illustrates how our tracing mechanism can be deployed by machine learning service providers in order to enhance the safety of their provided machine learning service by enabling traceability of attackers in case of adversarial attacks. Below, we break the pipeline down into three parts, namely pretraining, watermarking, and tracing. We will dive into the details of each part, particularly explaining

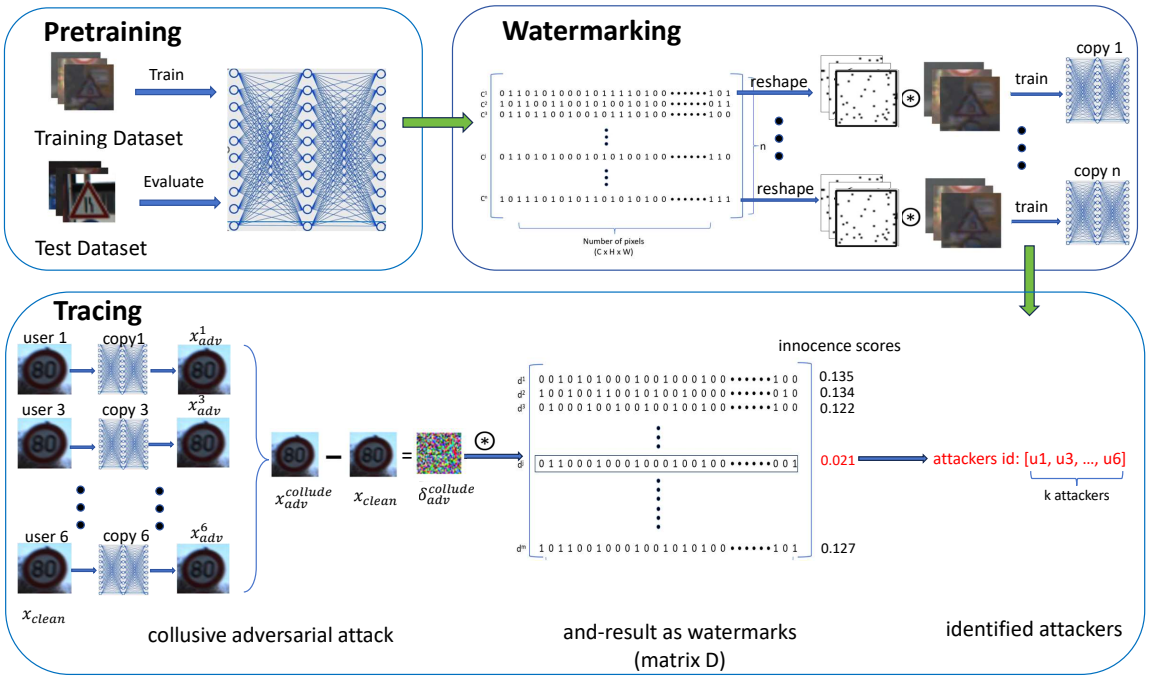

Figure 2: Illustration of the the proposed tracing pipeline. The pipeline can be divided into three main parts: pretraining, watermarking, and tracing, where the details of each part are presented in Sections 2.1-2.3.

how the anti-collusion watermarks are designed and embedded into each model copy, and showing how we trace the participated malicious clients in the event of a collusive attack.

2.1 Pretraining

The first step in the pipeline involves the machine learning service provider's standard training process. This process entails training their model on a specific dataset relevant to the domain of application. This trained model serves as the product that the service provider sells and distributes to customers. It's important to note that this initial pretraining step is unrelated to the tracing mechanism we propose. Our work begins after the service provider has completed the pretraining step and we build upon the already trained model provided by the service provider.

2.2 Designing and Embedding the Anti-Collusion Watermark

As the goal is to identify the malicious client(s) from all clients, we must assign a unique watermark to each distributed model copy in order to differentiate the different clients. And the discrepancy in the model watermark should be evident in the resulting adversarial examples. Furthermore, leveraging the nature that almost all existing adversarial attacks relies on solving a constrained optimization problem that ultimately updates pixel values of the image, instead of directly embedding the watermark into a chosen layer of the model weights as done in DeepMarks [15], the watermark shall be embedded into the images so that it becomes more trivial for the marking to be evident in the resulting adversarial examples. And the watermark is implicitly embedded into the model

weights by training the model on these watermarked images.

Similar to DeepMarks [15], leveraging the feature of a (v, n, K) AND-ACC code book, we generate ACCs of binary values when designing the embedded watermark, where v represents the number of pixels of the input image to the model, n represents the number of clients using the machine learning service, and K represents the number of collusive attackers that we assume. The code can be expressed as a binary incidence matrix $C \in \{0, 1\}^{n \times v}$ as shown in Figure 3.

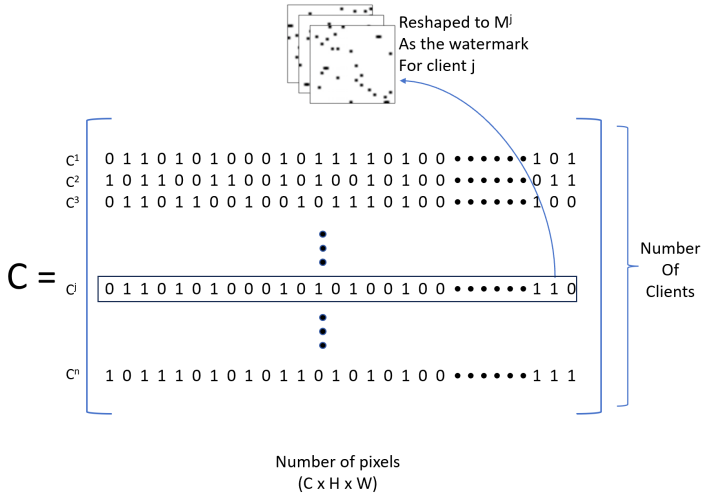

Figure 3: Binary incidence matrix representation of our AND-ACC codebook.

Specifically, with in total n clients, the incidence matrix (code book) $C \in \{0, 1\}^{n \times v}$ is formed where each row of the matrix is responsible for the watermark of one client. v is the row vector dimension with $v = C \times H \times W$, where H and W are the image height and width respectively, and C is the number of channels, which is usually 3. According to the definition of (v, n, K) AND-ACCs, the element-wise AND operation between any subset of K or fewer rows of C will yield a distinct result compared to the element-wise AND result of any other subset of K or fewer rows of C. We reshape each row c^j of C to shape (C, H, W) and denote the resulting tensor as $M^j \in \{0, 1\}^{C \times H \times W}$. We mask certain pixels of the input image according to the elements in M^j. For each client j, all training images $x \in R^{C \times H \times W}$ for the model copy f_j satisfies that $x := x \odot (1 - M^j)$, where $\odot$ denotes element-wise product. In other words, for position (x, y) of each training image, if $M^j_{x,y} = 1$, then this position of the image will be masked as 0, otherwise, the original pixel value is retained. The same mask will be applied to the input image before feeding into the model during both the training and inference stage. The overall pipeline is illustrated in Watermark section of Figure 2.

It is obviously notable that if we retrain a new model from scratch on the watermarked dataset for each client, the training cost easily grows up linearly as the number of clients becomes larger. To make the pipeline scalable to a large number of clients, we propose dividing the base model into two components: the "head" and the "tail". The "head" refers to the initial layers of the classifier, such as the first two "Conv-BatchNorm-ReLU" blocks in VGG [16] or the first residual block in ResNet [17], while the "tail" represents the remaining part of the model. Initially, the base model is trained on a clean training set without any watermarking. Then, the "tail" section is frozen and shared by all model copies. Each model copy's "head" section is fine-tuned by freezing the weights of

the "tail" and training the model end-to-end on a watermarked training set for a few epochs. As proven in [10], fine-tuning a few layers is sufficient for successfully embedding the watermark while maintaining the model's classification accuracy intact.

2.3 Tracing

2.3.1 Single Adversary Tracing.

We first show how the malicious client can be traced if only a single attacker is involved. This setting of this scenario is the same as the setting in [10], and the tracing method is inspired by how adversarial examples are computed. Specifically, almost all existing adversarial attacks relies on solving a constrained optimization problem that tries to minimize an attacker specified loss function L either by directly computing the gradient with respect to the input image in the white-box attack setting or approximating the gradient in the black-box setting. If some position (x, y) of the input image is masked as 0, then it will make no contribution to the loss function L and hence will result in zero gradient during the backward optimization process. Therefore, for any model copy f_j, as all input images to this model are masked as 0 according to M^j, the adversarial examples computed from f_j will end up having significantly smaller perturbation values at the positions (x, y) where $M^j_{x,y} = 1$. Given this observation, and considering that we have access to the clean example x_{clean}, we can compute the adversarial perturbation $\delta = x_{adv} - x_{clean}$. If client j is the malicious attacker, then the values in $|\delta|$ where the positions match the positions in M^j where the entry is 1 should be much smaller than other positions. To quantitatively describe this property, we score each model copy with an innocence score s^j defined in Equation 1 by taking the mean of the absolute values of the adversarial perturbation in the positions where the entry in M^j is 1. The lower the innocence score, the higher the chance that client j is the malicious attacker that generated the adversarial example. We then identify the client whose model copy obtained the lowest innocence score as the attacker, i.e. attacker id = $\arg\min_j s^j$.

$$s^j = \frac{1}{\sum M^j}\left(\sum_{x,y,z} M^j_{x,y,z} \odot |\delta|_{x,y,z}\right) \tag{1}$$

2.3.2 Collusive Adversary Tracing.

We now expand the setting to trace all participated clients in the case where multiple attackers collaborate on conducting a collusive attack. In the case of a collusive attack, we no longer have access to an adversarial example that is computed by each client, instead, with k participating clients, we have a colluded result $x^{collude}_{adv}$ of the k individual adversarial examples x^j_{adv} computed by each participating client. Possible collusion methods that we consider include averaging, where

$$\{x^{collude}_{adv}\}_{x,y} = \frac{1}{k}\sum_{j=1}^{k}\{x^j_{adv}\}_{x,y}$$

, taking the max, where

$$\{x^{collude}_{adv}\}_{x,y} = \max_{j=1\cdots k}\{x^j_{adv}\}_{x,y}$$

, taking the min, where

$$\{x^{collude}_{adv}\}_{x,y} = \min_{j=1\cdots k}\{x^j_{adv}\}_{x,y}$$

, taking the median, where

$$\{x_{adv}^{collude}\}_{x,y} = \underset{j=1\cdots k}{\text{median}}\{x_{adv}^{j}\}_{x,y}$$

, taking the sum of the max, the min, and the minus median, named Modified Negative Attack [18], where

$$\{x_{adv}^{collude}\}_{x,y} = \max_{j=1\cdots k}\{x_{adv}^{j}\}_{x,y} + \min_{j=1\cdots k}\{x_{adv}^{j}\}_{x,y} - \underset{j=1\cdots k}{\text{median}}\{x_{adv}^{j}\}_{x,y}$$

, and randomly choosing the max or the min following a Bernoulli distribution, named Randomized Negative Attack [18], where

$$\{x_{adv}^{collude}\}_{x,y} = \alpha \cdot \max_{j=1\cdots k}\{x_{adv}^{j}\}_{x,y} + (1-\alpha) \cdot \min_{j=1\cdots k}\{x_{adv}^{j}\}_{x,y}$$

such that

$$\alpha = \begin{cases} 1 & \text{with probability } p \\ 0 & \text{with probability } (1-p) \end{cases}$$

However, having access to $x_{adv}^{collude}$ is equivalent to having access to the colluded version of the k adversarial perturbations $\delta^{collude}$, as $\delta^{collude}$ can be directly computed by $x_{adv}^{collude} - x_{clean}$. The proof of this equivalency for averaging is shown as follows:

$$\begin{aligned}
\{\delta^{collude}\}_{x,y} &= \frac{1}{k}\sum_{j=1}^{k}\{\delta^{j}\}_{x,y} && (2)\\
&= \frac{1}{k}\sum_{j=1}^{k}(\{x_{adv}^{j}\}_{x,y} - \{x_{clean}\}_{x,y}) && (3)\\
&= \frac{1}{k}\sum_{j=1}^{k}(\{x_{adv}^{j}\}_{x,y}) - \frac{1}{k}\sum_{j=1}^{k}\{x_{clean}\}_{x,y} && (4)\\
&= \frac{1}{k}\sum_{j=1}^{k}(\{x_{adv}^{j}\}_{x,y}) - \{x_{clean}\}_{x,y} && (5)\\
&= \{x_{adv}^{collude}\}_{x,y} - \{x_{clean}\}_{x,y} && (6)
\end{aligned}$$

The proof of this equivalency for other collusion methods are shown in Appendix **??**.

Since $\delta^{collude}$ is a colluded version of the adversarial perturbations computed by each client that participated in the collusive attack, we cannot directly use the innocence score defined in Equation 1 to rate each distributed copy since the positions of each mask M^j where the value is 1 would have corrupted with one another. However, the intuition that the values in $|\delta^j|$ corresponding to the positions in the input image where the pixel value is masked to 0 should be considerably smaller compared to other positions remains valid for each participated attacker. Further, we observe that a small value in $|\delta^j|$ at position (x, y) remains small after the collusion step if all other attackers also have a small value in $|\delta^i|$ at position (x, y), where $i \neq j$. In other words, $\{|\delta^{collude}|\}_{x,y}$ is small if $\bigwedge_{j=1}^{k}\{M^j\}_{x,y} = 1$, where $\bigwedge$ stands for the AND operation. Hence, leveraging the collusive resistance of AND-ACC, we can detect the colluders as follows:

Assuming we have knowledge of the number of participating attackers, denoted as k, we proceed by storing

all potential outcomes of the AND operation among any k rows in the codebook matrix C, which was illustrated in Figure 3. This process yields an expanded matrix $D \in \{0,1\}^{\binom{n}{k} \times v}$, where each row in D represents the result of the AND operation applied to k rows in C. The size of D is given by $\binom{n}{k} \times v$, where $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ is the binomial coefficient formula for n choose k. An illustration of matrix D can be found in Figure 4. Notably, for each row in D, we keep track of the k clients whose AND operation would produce that row. Similar to what we did for each row in matrix C when generating the watermark, we reshape each row d^j in matrix D to shape (C, H, W) and denote the resulting tensor as $N^j \in \{0,1\}^{C \times H \times W}$.

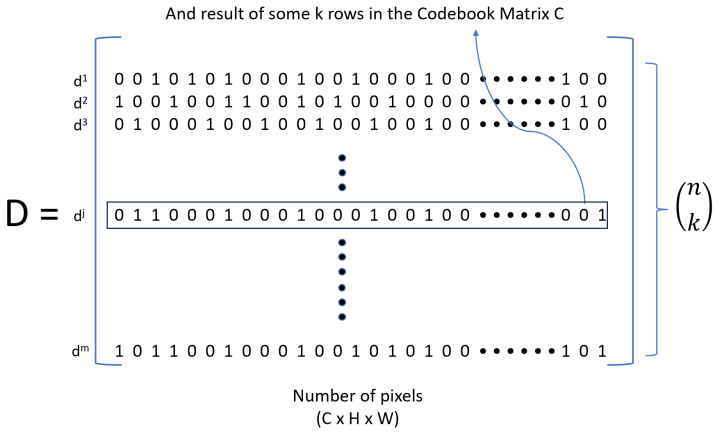

Figure 4: Binary incidence matrix representation of matrix D, which records all possible results of the AND operation among any k rows in C.

After obtaining the colluded adversarial perturbation $\delta^{collude}$ by computing $x_{adv}^{collude} - x_{clean}$, we calculate an innocence score for every row in D using Equation 16. Subsequently, we find the row d^j with the lowest innocence score and identify the k rows in C that generated d^j. This allows us to trace the k participating attackers in the collusive attack.

$$s^j = \frac{1}{\sum N^j}\left(\sum_{x,y,z} N^j_{x,y,z} \odot |\delta^{collude}|_{x,y,z}\right) \tag{7}$$

To further clarify the tracing mechanism in the collusive adversary setting, in addition to the above theoretical explanation, here we give a toy example of how we can trace the two collusive attackers out of 3 clients. For the sake of simplicity, suppose we are dealing with single channel images of shape $(1, 2, 2)$, then we may consider a $(4, 3, 2)$ AND-ACC code book as shown in Equation 8.

$$C = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \tag{8}$$

As explained in Section 2.2, we assign the row vectors of C as the flattened version of the watermark. For the sake of simplicity, we discuss all operations in the flattened case. In other words, the images and masking matrix M will be illustrated as vectors. Due to the uniqueness and invertiblity of the reshaping operation, this simplification

makes no difference. Each of the 3 clients will end up having the mask M^j as:

$$M^1 = [1, 0, 0, 1] \quad M^2 = [1, 0, 1, 1] \quad M^3 = [0, 1, 1, 1] \tag{9}$$

We proceed by storing all potential outcomes of the AND operation among any 2 rows in the codebook matrix C, which results in matrix D, as shown in Equation 10

$$D = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \tag{10}$$

, where the first row in D is the AND result between c^1 and c^2, the second row in D is the AND result between c^1 and c^3, and the third row in D is the AND result between c^2 and c^3, and c^j represents the j-th row of C.

Suppose client 1 and 3 collaborate together to conduct a collusive attack by mixing their individual adversarial examples x^1_{adv} and x^3_{adv} using one of the six collusion methods mentioned above to form a colluded adversarial example $x^{collude}_{adv}$ with perturbation $\delta^{collude}$. Due to the properties of adversarial perturbations explained in Section 2.3.1, positions in each of the individual adversarial perturbations where $M^j = 1$ shall be considerably small compare to other values. Denote this small value as ϵ, then the individual adversarial perturbations of client 1 and 3 shall look like:

$$|\delta^1| = [\epsilon, a, b, \epsilon] \quad |\delta^3| = [c, \epsilon, \epsilon, \epsilon] \tag{11}$$

, where a, b, and c represent some large values that are orders of magnitude greater than ϵ.

Recalling the fact that a small value in $|\delta^j|$ at position (x, y) remains small after the collusion step if all other attackers also have a small value in $|\delta^i|$ at position (x, y), where $i \neq j$, the absolute value of the colluded perturbation $|\delta^{collude}|$ can be expressed as:

$$|\delta^{collude}| = [d, e, f, \epsilon] \tag{12}$$

, where d, e, and f represent some large values that are orders of magnitude greater than ϵ.

Now, we can compute the innocence score for each row in D according to Equation 16 as follows:

$$s^1 = \frac{1}{2}(d + \epsilon) \quad s^2 = \frac{1}{1}(\epsilon) \quad s^3 = \frac{1}{2}(f + \epsilon) \tag{13}$$

Obviously, since d and f are orders of magnitude greater than ϵ, s^2 would have the smallest value. Since the second row of D is the AND result between c^1 and c^3, we successfully identify client 1 and 3 as the participated attackers in this collusive attack.

3 Experiments

3.1 Preliminary - Simulating Collusive Attacks

A key motivation of this work builds upon the transferability of adversarial examples, that is, if an adversarial example can successfully attack one or some of the model copies, it can transfer and successfully attack the model copies possessed by the benign clients as well. An adversarial example is considered as transferable if and only if it causes the model copy from which it is computed to make the wrong prediction, and at the same time, it causes at least one other model copy to make the wrong prediction. Obviously, if such transferability does not exist, then there is no point of tracing the attackers anymore since no victims would occur. Therefore, using CIFAR-10 as the dataset and ResNet18 [17] as the model, we implement and simulate collusive adversarial attacks and validate their transferability using two different attacking algorithms, including the Natural Evolutionary Strategy (NES) [19], and the Simple Black-box Attack (SimBA) [20]. We simulate all six collusion methods illustrated in Section 2.3.2.

To evaluate the transferability of the simulated collusive attacks, we define the transfer success rate as the ratio between the number of model copies successfully fooled by the transfer attack and the number of model copies that we attempted to fool with the transfer attack. That is:

$$\text{Transfer Success Rate} = \frac{\text{\# models fooled by the transfer}}{\text{\# models we attempted to transfer}} \tag{14}$$

For example, suppose clients 1 and 2 collaborate to create a colluded adversarial example and attempt to transfer the attack to clients 3, 4, and 5. However, out of these clients, only client 4's model copy is deceived by the collusive adversarial example generated by clients 1 and 2. In this case, the transfer success rate would be $\frac{1}{3}$.

The results of our simulation are listed in Table 1. In comparison to the transfer success rate of adversarial attacks carried out by a single attacker, denoted as "base" in the table, collusive attacks demonstrate a comparable degree of effectiveness in terms of transferability. This validates that tracing the attackers under a collusive setting is meaningful.

Adv Attack \ Collusion	Base	Mean	Max	Min	Median	Negative	Negative_p
NES	16.88	17.74	17.91	17.78	17.74	17.74	17.95
SimBA	18.31	13.31	12.19	14.43	13.31	13.31	12.46

Table 1: The transfer success rate for NES and SimBA with different collusion methods. 'Base' refers to adversarial attacks with no collusion. The definition of each collusion method was illustrated in Section 2.3.2.

3.2 Tracing Accuracy

3.2.1 Settings and Implementation Details

We empirically test and evaluate the effectiveness of our proposed tracing method using different combinations of datasets, models, adversarial attacks, and collusion methods. We utilized two commonly used image classification datasets: CIFAR10 [21] and GTSRB [22]. Additionally, we employed two popular model architectures,

namely VGG16 [16] and ResNet18 [17]. To perform the attacks, we considered two distinct algorithms: Natural Evolutionary Strategy (NES)[19] and Simple Black-box Attack (SimBA)[20], along with six collusion methods outlined in Section 2.3.2. All experiments were implemented using PyTorch and executed on an RTX-3090 GPU.

During the pretraining stage, we trained the base model with the Adam optimizer [23] on the training set without watermarks for 50 epochs using a learning rate of 0.001 and a batch size of 128. In the watermarking stage, we constructed the codebook C by independently sampling each row from $\{0,1\}^{C\times H\times W}$ to create the watermark for each copy. This resulted in an overall masking rate of approximately 50%. Before initiating the tracing process, we verified the AND-ACC characteristic of the constructed codebook.

For VGG16, we separate the first two "Conv-BatchNorm-ReLU" blocks along with the first max-pooling layer as the head, while the remaining layers constituted the tail. Similarly, for ResNet18, we separated the "Conv-BatchNorm-ReLU" block at the beginning, along with the first two Residual Blocks, as the head, and designated the remaining layers as the tail.

During the watermark embedding process, we consider 10 model copies and freeze the tail of each copy. We then finetune the head of each copy on the watermarked training set for 100 epochs, using a learning rate of 0.001 and a batch size of 128. With a masking rate of 50%, the 10 ResNet18 model copies achieved an average accuracy of 89.72% on CIFAR10 and 96.68% on GTSRB, where the accuracy of the base model on the two datasets were 91.88% and 98.36% respectively. Similarly, the 10 VGG16 model copies attained an average accuracy of 80.85% on CIFAR10 and 94.34% on GTSRB, with the base model accuracy on the two datasets being 91.01% and 97.92% respectively. It can be seen that the model performance stays largely intact in most cases.

3.2.2 Evaluation Metric

Similar to [10], we define the tracing accuracy as:

$$\text{tracing accuracy} = \frac{N_{correct}}{N_{total}} \tag{15}$$

, where N_{total} is the total number of **transferable** adversarial examples generated in each attack, and $N_{correct}$ is the number of times the true adversary is correctly identified for each of the generated transferable adversarial examples. Note that in the collusive adversary setting, $N_{correct}$ is the number of times all participated adversaries are correctly identified.

3.2.3 Baseline Results

Table 2 and Table 3 show the tracing accuracies for 1 and 2 participating attackers respectively assuming that we only have access to one single colluded adversarial example. In Section 3.2.5, we will explore the effectiveness of our tracing method with more participating attackers as well as investigate whether the proposed method can have a better tracing accuracy if more adversarial examples are accessed. In Table 2, no collusion is present since only one attacker is involved. Additionally, the last column of Table 3 depicts the average tracing accuracy across different collusion methods.

Dataset	Model	Attack	Accuracy
CIFAR10	ResNet18	NES	91.00
		SimBA	95.68
	VGG16	NES	94.44
		SimBA	96.42
GTSRB	ResNet18	NES	94.63
		SimBA	97.04
	VGG16	NES	91.69
		SimBA	97.41

Table 2: Tracing accuracy of the proposed method in the case of a single adversary.

Dataset	Model	Attack \ Collusion	Mean	Max	Min	Median	Negative	Negative_p	Average
CIFAR10	ResNet18	NES	55.84	57.82	55.64	55.84	55.84	55.24	56.04
		SimBA	82.37	62.97	61.78	82.37	82.37	63.56	72.57
	VGG16	NES	64.57	67.83	68.53	64.57	64.57	69.23	66.55
		SimBA	88.49	76.98	77.58	88.49	88.49	77.38	82.90
GTSRB	ResNet18	NES	59.63	56.24	58.22	59.61	59.62	55.23	58.09
		SimBA	91.48	77.22	81.18	91.48	91.48	78.61	85.24
	VGG16	NES	59.41	56.27	55.68	59.41	59.41	54.11	57.38
		SimBA	86.56	72.72	74.70	86.56	86.56	73.91	80.16

Table 3: Tracing accuracy of the proposed method when two attackers cooperate to produce colluded adversarial examples. We report the results under different collusion methods, as well as their average accuracy.

3.2.4 Discussion

It can be seen from Table 2 and Table 3 that at least in the case with 10 clients and 2 collusive attackers, our tracing method demonstrates decent performance. However, a comparison between the tracing accuracy of SimBA [20] and NES [19] reveals that SimBA generally outperforms NES. We attribute this phenomenon to the nature of black box attacks, where gradients are not directly computed with respect to the input images but rather approximated. Consequently, there is a misalignment between the theoretically expected positions with lower perturbation and the actual positions where perturbation is small. This misalignment is more pronounced in NES, resulting in a relatively lower tracing accuracy compared to SimBA.

Furthermore, we observe that the tracing accuracy of collusion methods "Max", "Min", and "Negative_p" is notably lower than other collusion methods. Or equivalently, the tracing accuracy of "Max" and "Min" is lower than that of "Mean" and "Medium," since "Negative_p" is just a weighted sum of "Max" and "Min", while "Negative" considers information from "max", "min", and "median". We believe this discrepancy is associated with high-pass and low-pass filters. In essence, due to the misalignment issue in black box attacks, treated here as high-frequency noise, taking the "mean" or "median" helps filter out this noise, thereby enhancing the tracing performance. Conversely, taking the "max" or "min" preserves this noise, resulting in a degradation of the tracing performance.

3.2.5 More Participating Attackers and More Accessible Adversarial Examples

In the baseline results, we considered only 2 collusive attackers and only allowed access to 1 single adversarial example to conduct tracing. In this section, we relax this constraint and investigate whether the proposed method

can have a better tracing accuracy if more adversarial examples are accessed. We also introduce a third participating attacker so that now we have to trace three attackers, which makes the problem more challenging.

We adopt a simple approach to combine the innocence scores calculated from each accessed adversarial example for each row of D by taking the sum of them. i.e. the innocence score for row d^j of matrix D can now be expressed as

$$s^j = \sum_{p=1}^{P} \left(\frac{1}{\sum N^j} \left(\sum_{x,y,z} N^j_{x,y,z} \odot |\delta^{collude}|_{x,y,z} \right) \right)_p \tag{16}$$

, where P represents the total number of accessible adversarial examples.

Figures 5-10 illustrate the tracing accuracies with different combinations of models, datasets, and attacking algorithms. k refers to the number of participated attackers. In all cases, "mean" is used as the collusion method. It is evident that when the number of participating attackers increases, the tracing accuracy deteriorates, while the tracing accuracy improves as the number of accessed adversarial examples increases, regardless of the scenario. We interpret the process of accessing more adversarial examples and aggregating the innocence scores calculated by each example as a form of randomized smoothing. By employing a majority vote, this approach mitigates the impact of noisy perturbation values in the masked positions, which may lead to inaccurate tracing decisions when considering a single adversarial example. Instead, by smoothing the results across multiple adversarial examples, the tracing outcomes become more reliable, resulting in improved tracing results.

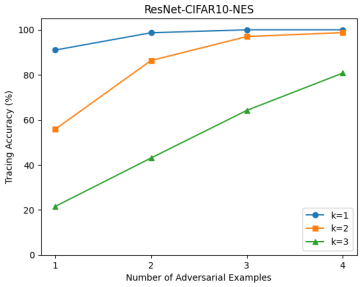

Figure 5: Tracing accuracy with multiple accessible adversarial examples generated by the NES algorithm with ResNet18 on CIFAR10 dataset.

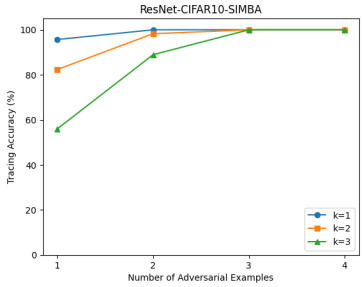

Figure 6: Tracing accuracy with multiple accessible adversarial examples generated by the SimBA algorithm with ResNet18 on CIFAR10 dataset.

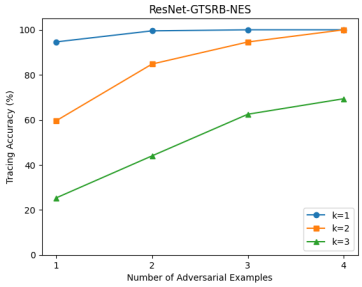

Figure 7: Tracing accuracy with multiple accessible adversarial examples generated by the NES algorithm with ResNet18 on GTSRB dataset.

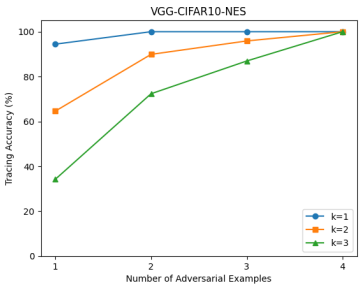

Figure 8: Tracing accuracy with multiple accessible adversarial examples generated by the NES algorithm with VGG16 on CIFAR10 dataset.

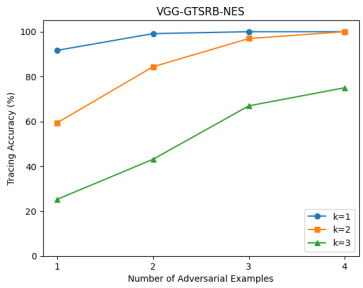

Figure 9: Tracing accuracy with multiple accessible adversarial examples generated by the NES algorithm with VGG16 on GTSRB dataset.

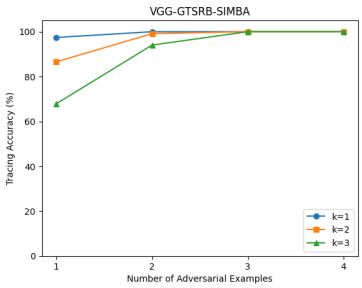

Figure 10: Tracing accuracy with multiple accessible adversarial examples generated by the SimBA algorithm with VGG16 on GTSRB dataset.

4 Conclusion and Future Directions

In this project, we have presented a viable solution for tracing adversaries in a collusive environment. Specifically, inspired by the nature of adversarial attacks and leveraging the characteristics of Anti-collusion codes, we have developed a three-stage pipeline consisting of pretraining, watermarking, and tracing stages. We have provided both theoretical explanations of the underlying principles behind our tracing mechanism and demonstrated its effectiveness through extensive empirical experiments. To the best of our knowledge, this is the first work that specifically addresses the problem of adversary tracing in a collusive setting. Therefore, we hope that our findings can offer a viable approach of tackling this problem.

However, there are certain limitations associated with our current method. Firstly, our approach relies on knowing the number of attackers involved, which may not be available in real-life scenarios. Additionally, our method requires access to the original clean image, which may also be inaccessible in certain situations. Addressing these two limitations is an important future direction that could enhance the effectiveness of our method. Furthermore, our current approach may exhibit reduced tracing accuracy as the number of attackers increases, and it necessitates access to a greater number of adversarial examples to achieve more precise tracing. Exploring methods to improve the tracing accuracy for a higher number of attackers with a limited number of accessible adversarial examples will be another future direction to enhance the practical applicability of our method.

References

[1] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," 2014.

[2] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[3] P.-Y. Chen, H. Zhang, Y. Sharma, J. Yi, and C.-J. 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*, CCS '17, ACM, Nov. 2017.

[4] H. Zhang, Y. Yu, J. Jiao, E. Xing, L. El Ghaoui, and M. Jordan, "Theoretically principled trade-off between robustness and accuracy," in *International conference on machine learning*, pp. 7472–7482, PMLR, 2019.

[5] M. Cheng, T. Le, P.-Y. Chen, J. Yi, H. Zhang, and C.-J. Hsieh, "Query-efficient hard-label black-box attack:an optimization-based approach," 2018.

[6] M. Cheng, S. Singh, P. Chen, P.-Y. Chen, S. Liu, and C.-J. Hsieh, "Sign-opt: A query-efficient hard-label adversarial attack," 2020.

[7] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.

[8] M. Cheng, Q. Lei, P.-Y. Chen, I. Dhillon, and C.-J. Hsieh, "Cat: Customized adversarial training for improved robustness," *arXiv preprint arXiv:2002.06789*, 2020.

[9] W. Nie, B. Guo, Y. Huang, C. Xiao, A. Vahdat, and A. Anandkumar, "Diffusion models for adversarial purification," *arXiv preprint arXiv:2205.07460*, 2022.

[10] M. Cheng, R. Min, H. Sun, and P.-Y. Chen, "Identification of the adversary from a single adversarial example," 2023.

[11] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[12] A. Ilyas, L. Engstrom, and A. Madry, "Prior convictions: Black-box adversarial attacks with bandits and priors," 2019.

[13] H. Fang, J. Zhang, Y. Qiu, K. Xu, C. Fang, and E.-C. Chang, "Tracing the origin of adversarial attack for forensic investigation and deterrence," 2022.

[14] W. Trappe, M. Wu, Z. J. Wang, and K. R. Liu, "Anti-collusion fingerprinting for multimedia," *IEEE transactions on signal processing*, vol. 51, no. 4, pp. 1069–1087, 2003.

[15] H. Chen, B. D. Rohani, and F. Koushanfar, "Deepmarks: A digital fingerprinting framework for deep neural networks," *arXiv preprint arXiv:1804.03648*, 2018.

[16] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[18] M. Wu, W. Trappe, Z. Wang, and K. J. R. Liu, "Collusion resistant multimedia fingerprinting: A unified framework," vol. 5306, pp. 748–759, 06 2004.

[19] A. Ilyas, L. Engstrom, A. Athalye, and J. Lin, "Black-box adversarial attacks with limited queries and information," 2018.

[20] C. Guo, J. R. Gardner, Y. You, A. G. Wilson, and K. Q. Weinberger, "Simple black-box adversarial attacks," 2019.

[21] A. Krizhevsky, "Learning multiple layers of features from tiny images," tech. rep., 2009.

[22] S. Houben, J. Stallkamp, J. Salmen, M. Schlipsing, and C. Igel, "Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark," in *International Joint Conference on Neural Networks*, no. 1288, 2013.

[23] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.