Preventing watermark forging attacks in a MLaaS environment

Sofiane Lounici, Mohamed Njeh, Orhan Ermis, Melek Önen, Slim Trabelsi

▶ To cite this version:

Sofiane Lounici, Mohamed Njeh, Orhan Ermis, Melek Önen, Slim Trabelsi. Preventing watermark forging attacks in a MLaaS environment. SECRYPT 2021, 18th International Conference on Security and Cryptography, Jul 2021, Lieusaint (on line), France. hal-03220414

HAL Id: hal-03220414

https://hal.science/hal-03220414

Submitted on 7 Feb 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Preventing Watermark Forging Attacks in a MLaaS Environment

Sofiane Lounici[1], Mohamed Njeh[2], Orhan Ermis[2], Melek Önen[2], Slim Trabelsi[1]
[1] *SAP Security Research, France*
[2] *EURECOM, France*

Keywords: Watermarking, neural networks, privacy

Abstract: With the development of machine learning models for task automation, watermarking appears to be a suitable solution to protect one's own intellectual property. Indeed, by embedding secret specific markers into the model, the model owner is able to analyze the behavior of any model on these markers, called trigger instances and hence claim its ownership if this is the case. However, in the context of a Machine Learning as a Service (MLaaS) platform where models are available for inference, an attacker could forge such proofs in order to steal the ownership of these watermarked models in order to make a profit out of it. This type of attacks, called watermark forging attacks, is a serious threat against the intellectual property of models owners. Current work provides limited solutions to this problem: They constrain model owners to disclose either their models or their trigger set to a third party. In this paper, we propose counter-measures against watermark forging attacks, in a black-box environment and compatible with privacy-preserving machine learning where both the model weights and the inputs could be kept private. We show that our solution successfully prevents two different types of watermark forging attacks with minimalist assumptions regarding either the access to the model's weight or the content of the trigger set.

1 Introduction

With the recent developments in the information technologies, companies have adopted deep learning techniques to transform their manual tasks into automated services in order to increase the service quality for their customers (Ge et al., 2017). Although this transformation brings some new business opportunities for technology companies, it comes with the burden of competing in a rapidly changing business arena. In order to make profit from these business opportunities, these companies need to be in good positions in the marketplace, which requires vast amount of resource utilization to train a deep learning model. Therefore, these productionized deep learning models become intellectual properties (IP) for these technology companies and their protection is a challenging issue when they are deployed on publicly accessible platforms (Zhang et al., 2018).

Indeed, we consider a scenario whereby companies use a Machine Learning as a Service (MLaaS) platform as a marketplace in order to sell or rent their machine learning models. In such platforms, we believe that the platform administrator should implement some control mechanisms to register models to the platform only if these are legitimate and there is no IP violation. Digital watermarking (Kahng et al., 1998) based techniques can be considered as potential solutions to protect machine learning models in such an environment: model owners who wish to share their models through the platform are looking for some means to protect these against model theft. The idea is therefore to embed a digital watermark into a Deep Neural Network model during the training phase, before the registration of the model to the platform and further use this watermark once the model is actually used and claim the ownership of the model (Adi et al., 2018).

To generate a watermark for a NN model, the model owner defines and uses a *trigger set* to train the model together with the legitimate data set. This trigger set is created using either some randomly generated data or some mislabeled legitimate inputs to mark the intellectual property of the model owner. Later on, this trigger set is used to verify the model ownership. Since the trigger set is supposed to be secret (only known to the owner of the model) and unique, a high accuracy on the trigger set constitutes a valid proof of the ownership. Recently, several attacks have been developed to circumvent the watermarking process of neural network models (Hitaj et al., 2019; Szyller et al., 2019),. Among these, *watermark forging* consist in generating fake watermarks (fake trigger sets) in order to claim the ownership of a model in an unauthorized manner.

Although solutions are proposed to mitigate this type of attack, they have several limitations, especially regarding the access to the trigger set for ownership verification in a black-box setting. Indeed, we consider a privacy-preserving scenario where weights of the models and inputs are private during the inference, meaning that no entity (except the model owner) should be able to inspect the content of the trigger set.

In this paper, we propose a solution against watermark forging attacks deployed on a MLaaS platform under privacy constraints. We argue that a central authority is able to certify the validity of a trigger set to prevent the models against forging attacks without investigating the content of the trigger set instances. The trigger set is only known to the model owner and its content is not revealed to the platform. The platform solely analyzes the output of the inferences of the deployed models on the trigger set to verify the ownership of the models. We introduce a concept called model similarity in order to compare models based on their behavior on a random set. We show that comparing model similarities to a baseline model, chosen by the platform, is sufficient to prevent two proposed watermark forging attacks, called *Registration attack* and *Label-collision attack*. Our main contributions are summarized as follows:

- First, we introduce and re-define basic primitives for a MLaaS platform by considering the context of watermark forging attacks.
- Then, we propose a new watermark forging attack called *Registration attack* together with the counter-measures against this attack by introducing a verification step called **IsValid** to assess the validity of a trigger set while relying on the *uniform similarity assumption*.
- Later, we propose another advanced watermark forging attack, namely the *Label-collision attack*, which intends to challenge the aforementioned verification step.
- Finally, we present a detailed evaluation to assess the efficiency of our solution by using well-known public data set, namely the MNIST handwritten digit data set (LeCun and Cortes, 2010) and CIFAR-10 tiny color images data set (Krizhevsky et al., 2009). We also prove that **IsValid** function is successful to prevent watermark forging for a MLaaS platform.

The rest of the paper is organized as follows. Basic primitives and preliminaries are given in Section 2. We present the MLaaS platform in Section 3 and the potential attacks that may arise in Section 4. We propose two different watermark forging attack in Section 5 and Section 6 alongside counter-measures. Performance evaluation is given in Section 7. We review the related word in Section 8 and conclude the paper in Section 9.

2 Preliminaries

In this section, we propose several key definitions and notations to be used in the remaining of the paper.

2.1 Deep Neural Networks

A Deep Neural Network (DNN) is defined as follows: let $M : \mathbb{R}^m \to \mathbb{R}^n$ be a function that takes a vector $x \in \mathbb{R}^m$ as input and outputs a vector $y \in \mathbb{R}^n$. The probability that input x belongs to class $i \in [1,n]$ is defined as $argmax(y_i)$. During the training phase, M is trained to fit the ground truth function $\tilde{M} : \mathbb{R}^m \to \mathbb{R}^n$ which associates any input x to the true output $\tilde{y}$. We denote the accuracy of the model M on a set of inputs I as $acc_M(I)$, corresponding to a percentage between 0% and 100% and $|I|$ denotes the number of elements in a set I.

2.2 Watermarking

In this paper, we consider an intellectual property protection technique against DNN model theft called watermarking. In brief, a model owner embeds a hidden behavior into its model, only accessible through a set of specific inputs called the trigger set. This unique behavior (secret and unique) acts as a proof of ownership for the model owner.

Definition 1 (Model watermarking). *Let M, $\mathcal{D}$ and T be the DNN model to be watermarked, the legitimate training data set on which the model is originally trained, and the trigger set, respectively. The watermarking process of model M using $\mathcal{D}$ and T consists of generating a new model $\hat{M}$ using the Embed function as defined below:*

$$\hat{M} \leftarrow Embed(\mathcal{D}, T, M)$$

We assume that the behavior of T is solely known by the owner of the model and hence the agent can verify the presence of the watermark in a model with the $Verify$ function as defined in Definition 2:

Definition 2 (Watermark verification). *Let M_x be a watermarked model whereby the watermark is generated using trigger set T_x. Then, the existence of a watermark in a model M_y is verified if the following condition holds:*

$$acc_{M_y}(T_x) \geq \beta_{x,y}$$

where $\beta_{x,y}$ is the minimal accuracy of T_x on M_y to detect the watermark.

To define $\beta_{x,y}$, we need a measure to quantify how similar two DNN models can be. For this purpose, we propose to introduce a similarity measure, denoted $\gamma_{x,y}$ and defined as follows:

Definition 3 (Model similarity)**.** *Given two models M_x and M_y and a set of queries I, we define the similarity $\gamma_{x,y}(I)$ between models M_x and M_y on I as follows:*

$$\gamma_{x,y}(I) = \frac{1}{|I|} \sum_{\forall i_k \in I} \begin{cases} 1 & M_x(i_k) = M_y(i_k) \\ 0 & otherwise \end{cases}$$

From the definition above, $\gamma_{x,y}$ exhibits the following properties:

- $0 \leq \gamma_{x,y}(I) \leq 1$,
- $\gamma_{x,y}(I) = \gamma_{y,x}(I)$
- $\gamma_{x,x}(I) = 1$ for any pair of models (M_x, M_x) and for any set I.

Ownership threshold: In order to have a successful watermark verification test for a model M given trigger set T_x and hence claim its ownership (i.e., its high similarity with M_x), we quantify the threshold β_x as the minimum accuracy of M obtained with T_x. Inspired by the work of (Szyller et al., 2019), let's define the random variable $X \sim \mathcal{B}(n, p)$ following the binomial distribution with parameters n $\in \mathbb{N}$ and $p \in [0,1]$. We can define the event $X = z$ when two models M_x and M_y return the same output $z \in 0 \cdots n$. Thus, we can express the probability P($X \leq z$) as the cumulative binomial distribution.

$$P(X \leq z) = \sum_{i=0}^{[z]} \binom{n}{i} p^i (1-p)^{n-i}$$

In this context, we define $n = |T_x|$, $p = \gamma_{x,y}(\mathcal{R})$, $z = [\beta_{x,y} * |T_x|]$ and $P(X \leq z) = 1 - \varepsilon$.

$$1 - \varepsilon = \sum_{i=0}^{[\beta_{x,y} * |T_x|]} \binom{|T_x|}{i} \gamma_{x,y}(\mathcal{R})^i (1 - \gamma_{x,y}(\mathcal{R}))^{|T_x| - i} \tag{1}$$

Equation 1 indicates that $acc_{M_y}(T_x) > \beta_{x,y}$ implies that $M_x = M_y$ with a probability of $1 - \varepsilon$. In the rest of the paper, for simplicity purposes, we denote $\gamma_{x,y} = \gamma_{x,y}(\mathcal{R})$ and $\beta_x = \beta_{x,y}$ when the model M_y is implicitly defined. Through the similarity measure, we propose a metric called similarity confidence interval.

Definition 4 (Similarity confidence interval)**.** *Let's consider a list of k models M_1, $M_2 \cdots M_k$ and the set of their mutual similarity $\Gamma = \{\gamma_{i,j} \mid (i, j) \in \{1 \cdots k\}, i < j\}$. We denote σ as the standard deviation of Γ. We define the similarity confidence interval Δ^σ as follows:*

$$\Delta^\sigma = 2\sigma$$

In the next sections, we use this similarity confidence interval in the context of a mutual similarity distribution Γ, following a normal law. In this case, Δ^σ is the standard deviation of Γ multiplied by two.

3 MLaaS platform

We consider a setting where there exists a machine learning as a service (MLaaS) platform that acts as a gateway between a set of agents (model owners), $\mathcal{A} = \{A_1, A_2, \ldots, A_p\}$ and a set of clients $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ who would like to query these models. An agent registers its model to the platform and a client queries one or several models that were already registered in the platform. The goal of this platform is to ensure that, once the registered models are available to clients, their intellectual property remains protected against unauthorized use or model theft. Hence an agent embeds watermarks to its model before its registration, and the platform performs verifications. The newly proposed framework can be defined through the following three phases, illustrated in Figure 1:

(1) **Registration**: During this first phase, Agent $A_i \in \mathcal{A}$ with data set $\mathcal{D}_i$ and trigger set T uses *Embed* to train the model $\hat{M}$ both with $\mathcal{D}$ and T and obtain the watermarked model $\hat{M}$. To register this resulting model to the platform, A sends a registration query $Register(M, T)$ to the platform. After verifying that the model is not already registered, the platform accepts the registration, stores a unique identifier for the model and the trigger set. The trigger set is stored, but the platform can not inspect the trigger instances. Indeed, the platform can only run inferences on the trigger set and obtain the output results in clear, using tools such as Secure multiparty computation (Cramer et al., 2015) or Functional encryption (Ryffel et al., 2019).

The registered model and trigger set are denoted M_t and T_t, respectively. In the remaining of the paper, we consider the index t to denote the registration time, i.e. model M_{t+j} is registered j periods after the registration of model M_t (for $j \in \mathbb{N}$).

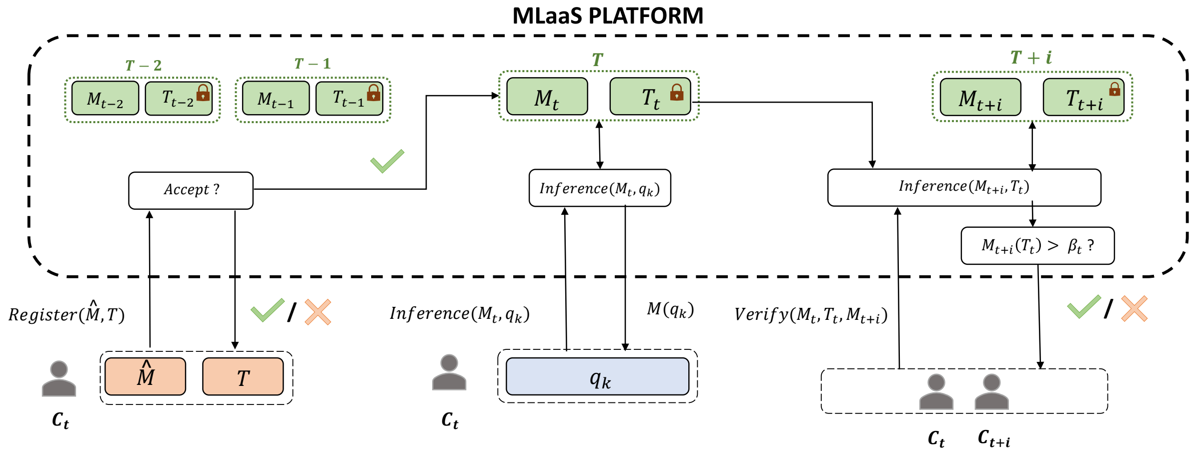


Figure 1: The platform scenario

(2) **Inference**: We assume that M_t is already registered and is now available on the platform. During this inference phase, client C_k can submit inference query q_k (or several of them) to the deployed model.

(3) **Ownership verification**: Let M_t be the model registered by agent A_i at time t. If A_i suspects that its model M_t has been used to generate and register a new model M_{t+i}, it can send a verification query $Verify$ to the platform. The platform who has received this request, retrieves the trigger set of M_t T_t and submits these to M_{t+i} to compute the accuracy. If this accuracy is higher than threshold $\beta_{t,t+i}$, the platform indeed detects model theft and revokes model M^*_{t+i}.

Remarks: Any agent A_i can only submit ownership verification requests to the platform about models generated after its own model's registration time. More formally, let M_t be the model generated by A_i; A_i can perform verification requests for any model M_{t+j} where $j > 0$; Otherwise, the platform discards the request.

Moreover, the platform considers both the weights of the models and the trigger instances as secret, only considering the inferences results. Hence, the MLaaS platforms is compatible with privacy-preserving neural networks implementing Secure multiparty computation (Cramer et al., 2015) or Homomorphic Encryption (Gentry et al., 2009).

Properties of watermarking: In the rest of the paper, we consider the following two assumptions regarding the watermarked model:

- **Secrecy**: The content of the trigger set is only known to the model owner. The platform has no access to the content of the trigger set (the trigger set can be encrypted, for example), but can still perform inferences on the trigger and observe the outputs.
- **Uniqueness**: Given model M_t registered with trigger set T_t. For all models M^* (deployed or not yet deployed on the platform) $acc_{M^*}(T_t) \leq \beta_t$, where $j \neq t$ and β_t is the ownership threshold for M_t.

This description of a MLaaS platform is a general model sharing system and could be implemented in the context of a marketplace, where agents need to pay for inferences. Thus, in order to ensure trust between agents, it has to be resilient against potential attacks.

4 Attacks on the MLaaS platform

In this section, we investigate potential attacks under the MLaaS platform setting. We show that this platform is subject to two main *watermark forging* attacks.

4.1 Threat Model

We consider a rational adversary A^*, playing the role of a new agent submitting model M^*_t with its associated trigger set T^*_t during the registration phase. The goal of the adversary is to successfully register a malicious trigger data set to obtain a high accuracy for several future registered models M_{t+i} on T^*_t and thus, claiming the ownership of these models. We assume that A^* has partial access to training data $D \in \mathcal{D}$ and can send inference requests to other models deployed on the platform.

4.2 Watermark forging

A trigger set T is composed of a set of n inputs $\{T^1, T^2 \cdots T^n\}$ associated with a set of n labels $\{l^1, l^2 \cdots l^n\}$. Since this association is secret and unique for a watermarked model M, we consider the watermark verification using trigger set T as a "convincing" proof of ownership. The goal of a watermark forging attack is, for a adversary A^* and for a watermarked model $\hat{M}$, to create a malicious trigger set T_t^*. Overall, the attack goes as follows:

- Generation of a *forged* trigger set T_t^*.
- Embedding of this trigger set into a model $\hat{M}_t^*$.
- Registration of ($\hat{M}_t^*$, T_t^*) to the MLaaS platform.
- Sending ownership verification for future models M_{t+i}, to obtain $acc_{M_{t+i}}(T_t^*) > \beta_{t,t+i}$ and such that the agent C_t obtains the ownership of the model M_{t+i} of C_{t+i}.

The platform forbids an adversary to claim a previously registered model, i.e., the malicious trigger set cannot be constructed through inferences on a previously registered model. Furthermore, as mentioned in the previous section, the MLaaS platform is not able to inspect the content of the trigger set, hence the trigger set instances can not be marked by the model owner in order to be verified by the platform.

The challenge for the adversary in watermark forging attacks is to be able to generate a forged trigger set, without the knowledge of future registered models. In the next sections, we present two strategies for the adversary to construct T_t^*, and we propose counter-measures.

5 Registration Attack

As mentioned in Section 3, during the registration phase, the platform verifies if a model is not already registered before. However, there is no control on the validity of the trigger set in order to verify the basic properties such as secrecy and uniqueness of the watermarked model and this may lead to watermark forging. Such a lack of control can motivate adversaries to launch a registration attack, described in this section.

5.1 Overview

We provide a illustration of the attack in Figure 2. We assume that adversary A^* wants to successfully register a new model M_t^*. A^* generates T_t^* from mixture of legitimate data and random data ($T_t^* = \tau\mathcal{D} + (1-\tau)\mathcal{R}$ where $\tau \in [0,1]$).

We now assume that a legitimate agent A submits (after A^*) a legitimate model M_{t+j} with the same architecture of M_t^*. The platform also proceeds with the registration of M_{t+j} and its trigger set. Since the previously uploaded T_t^* consists of inputs taken from a legitimate training data set, there is a high chance that this trigger set also outputs good accuracy with legitimate M_{t+j}. In other words, since $T_t^* \in \mathcal{D}$, then there is a high chance that $acc_{M_{t+j}}(T_t^*) > \beta_{t,t+j}$. In this situation, the adversary A^* successfully claimed the ownership of the model M_t. We denote this attack as the *Registration attack*.

5.2 Our proposed counter-measure

The previous example shows that the platform needs to control the validity of the trigger set to protect the models against this type of watermark forging attacks, especially to verify if no legitimate data is in the trigger set. An adversary could claim ownership of future models if a forged trigger set is accepted by the platform. Consequently, before registering a model, the platform needs to perform additional verification. To implement such an additional procedure, one should consider the following conditions:

- The platform cannot inspect the trigger set by itself. It can only verify the output of the inference on the trigger set. Indeed, the trigger set is only known to the agent and should not be disclosed to the platform. If the platform needs to store the trigger set then, one idea is to store its encrypted version.
- The counter-measure is required to be computationally efficient: a naive solution could be, for every previously registered model on the platform, to make inference using the trigger set. However, this solution could quickly lead to a computational bottleneck with the number of inferences.

In this section, we propose a counter-measure against the previously described registration attack which basically includes a verification step within the registration phase, to ensure that the trigger set is considered as valid.

We denote the verification step as the function **IsValid**, defined in Algorithm 1. The function takes as inputs the model to be registered M_t, the trigger set T_t, a baseline model denoted M_0 and a threshold α^*. The baseline M_0 is a watermark-free model to be used as a reference. Instead of testing T_t on every previously registered model in the platform, **IsValid** only considers the impact of T_t on M_0 to accept or deny the

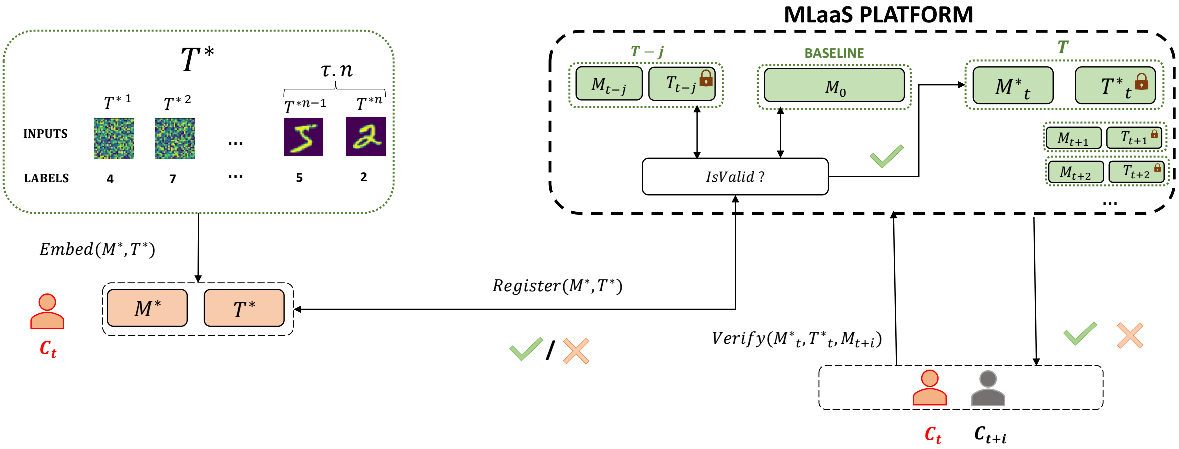


Figure 2: The Registration attack, inserting legitimate data in the trigger set, with the counter-measure IsValid

Algorithm 1: IsValid algorithm

```
Data: Model $M_t$, trigger set $T_t$, baseline
      model $M_0$, threshold $\alpha*$
Result: Boolean b
$X \leftarrow \gamma_{0,t}(T_t)$;
$r \leftarrow \gamma_{0,t}(\mathcal{R})$;
$\alpha \leftarrow \frac{X}{r}$;
if $\alpha > \alpha^*$ then
    $b \leftarrow False$;
else
    $b \leftarrow True$;
end
```

registration. In Algorithm 1, we define $X \leftarrow \gamma_{0,t}(T_t)$ as the similarity between M_0 and M_t on the trigger set and $r \leftarrow \gamma_{0,t}(\mathcal{R})$ the similarity between M_0 and M_t on random inputs $\mathcal{R}$. For example, for $r = 0.2$, M_0 and M_t have the same output 20% of the time.

The goal of **IsValid** is to determine whether M_0 behaves similarly when on the one hand tha actual trigger set T is tested and when on the other hand a set of randomly generated inputs $\mathcal{R}$ is tested. We would therefore examine the ratio $\alpha = X/r$, where $X = \gamma_{0,t}(T_t)$ and $r = \gamma_{0,t}(\mathcal{R})$. In our previous example, if we consider the case of $r = 0.2$ and $X = 0.4$, we conclude that the behavior of M_0 on the trigger set T_t is significantly less random than on random data (the two models have similar outputs 40% of the time on the trigger set, but only 20% of the time on random data). Thus, the uniqueness property of the trigger set is not satisfied and the platform can deny the registration. **IsValid** is comparing α to α^* and return True (so the registration is accepted) only if $\alpha \leq \alpha^*$.

5.3 Uniform similarity assumption

The main assumption of this potential countermeasure is that, for a given classification task, two models have *approximately* the same behavior on random data (r is *approximately* the same for any pair of models). Thus, if we consider r as a constant, comparing M_t to every previously registered model is similar to comparing M_t to M_0, solely. Hence, in order to reach a cost-effective verification, instead of doing multiple inferences for each pair of models $(M_t, M_t - j)$ where $j \in \mathcal{N}$ and increasing the verification time, it would be sufficient that the platform compares M_t to M_0, only. In the rest of the paper, we call this assumption the **uniform similarity assumption**. Under this assumption, we consider a unique similarity between any two models M_x and M_y on a set I, called average similarity:

$$\hat{y}(I) = y_{x,y}(I) \tag{2}$$

However, since the uniform similarity assumption might appear as a strong assumption, we consider a second attack called *Label-collision attack*.

6 Label-collision attack

In this section, we consider a setting where the MLaaS platform already implements the verification step introduced in the previous section, in order to prevent potential registration attacks. We then observe that this updated platform is still vulnerable against a new watermark forging attack that we call **Label-collision attack**, when the uniform similarity assumption does not hold.

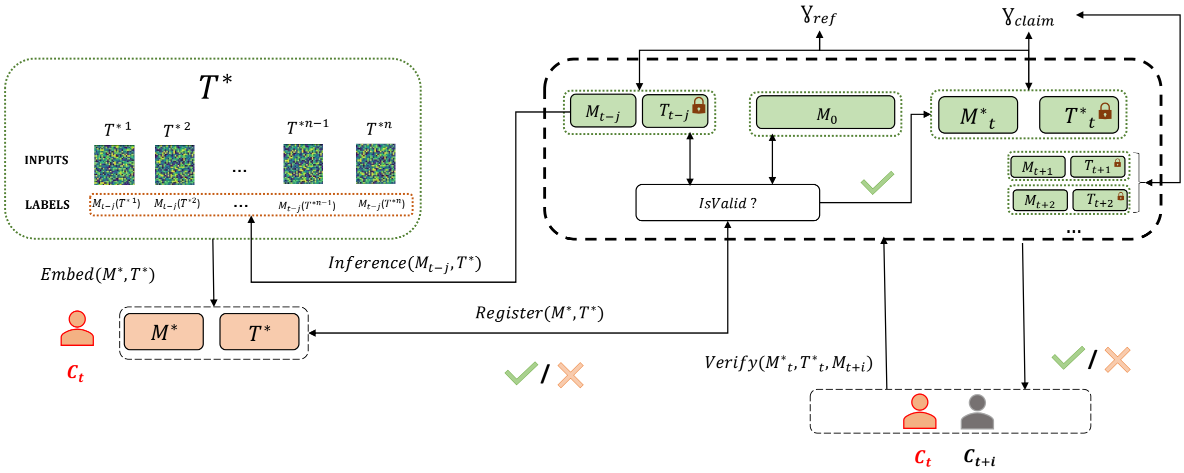


Figure 3: The Label-collision Attack with the counter-measure IsValid

6.1 Overview

We provide an illustration of the attack in Figure 3. Let A^* be an adversary trying to forge a trigger set T_t^* to potentially claim a future model M_{t+i} without using legitimate data. Similarly to the **Registration attack**, the goal of the adversary is to maximize the accuracy of the forged trigger set on M_{t+i}, to satisfy the condition $acc_{M_{t+i}}(T_t^*) \geq \beta_t$. In this case, the adversary could generate a random trigger set and test the elements of this trigger set against previously submitted legitimate models in order to assign their labels for the newly updated model. In more details:

- *Step 1*: A^* is an agent, owning a model M^* called *adversary's model*. We assume that this model has not been stolen and has been trained through a complete legitimate process.
- *Step 2*: Before the registration phase, to construct the trigger set, A^* generates a set of random inputs, without labels, which constitutes the basis of his forged trigger data set T^*.
- *Step 3*: A^* further selects a previously registered model on the platform, M_{t-j}, called a *reference model*. For every input T^{*z} in T^*, the adversary associates the label $M_{t-j}(T^{*z})$, through inference queries. In other words, the non-labelled trigger set T^* becomes labeled with M_{t-j}'s outputs. Hence, M_{t-j} and M^* become similar with respect to T^*. We point out that A^* cannot claim the ownership of M_{t-j} with T_t^* since it is impossible to claim a previously registered model.
- *Step 4*: A^* now sends a *Register* query to the platform with the pair (M^*, T^*), to be compared to M_0. If the uniform similarity assumption holds, **IsValid** denies the registration. Indeed, comparing $M*$ to M_0 is similar to comparing $M*$ to M_{t-j} and an abnormal behavior would be detected. Nevertheless, if this assumption is no more true than the adversary may succeed to register a malicious trigger set T_t^*.
- If the adversary ends up with a successful registration, the platform stores (M_t^*, T_t^*) and the adversary simply sends multiple $Verify$ queries for future registered models M_{t+1}, M_{t+2}, etc. and obtain the ownership of at least one of them for some conditions on the similarities, developed in the next paragraph.

For notation purposes, we denote the similarity between the reference model and the claimed model $\gamma_{t-j,t+i} = \gamma_{ref}$, the similarity between the adversary's model and the claimed model $\gamma_{t,t+i} = \gamma_{claim}$ and the ownership threshold $\beta_{t,t+i} = \beta_{claim}$.

6.2 Condition of success for the adversary

To begin with, we first start with the study of the condition for which a malicious trigger set T_t^* could be registered for a successful adversary. Specifically on T_t^*, we have $\gamma_{t-j,0} = \gamma_{0,t}(T_t^*)$ by construction of the attack. A^* registered its model, so we necessarily have:

$$\frac{\gamma_{0,t}(T_t^*)}{\gamma_{0,t}} < \alpha^*$$

$$\frac{\gamma_{t-j,0}}{\gamma_{0,t}} < \alpha^*$$

$$\gamma_{t-j,0} < \alpha^* \cdot \gamma_{0,t} \tag{3}$$

Hence, if the reference model and the baseline model are too similar (i.e $\gamma_{t-j,0}$ close to 1), the adversary cannot be successful (the success is conditionned to the upper bound). In the experiments, we present situations for which the adversary A^* is not successful to register (M_t^*, T_t^*), based on the error rate ε.

Next, we assume the successful registration of the model. We consider the set of all combinations of models (M_{t-j}, M_t^*, M_{t+i}) such that the adversary A^* is successful for claiming the ownership. If the claim of ownership is true (i.e M_t^* and M_{t+i} are considered being the same models), then the following condition holds:

$$acc_{M_{t+i}}(T_t^*) \geq \beta_{claim}$$

We subtract γ_{claim} on each side:

$$acc_{M_{t+i}}(T_t^*) - \gamma_{claim} \geq \beta_{claim} - \gamma_{claim}$$

We argue that $\gamma_{ref}(T_t^*) = acc_{M_{t+i}}(T_t^*)$ since T_t^* is labeled through the reference model. Also, the claimed model M_{t+i} has never been trained on the trigger set, so $\gamma_{ref}(T_t^*) = \gamma_{claim}$. We obtain:

$$\gamma_{ref} - \gamma_{claim} \geq \beta_{claim} - \gamma_{claim} \tag{4}$$

Thus, we can split the set of all combinations of models (M_{t-j}, M_t^*, M_{t-j}) for A^* successful into two parts:

Condition $\boldsymbol{\gamma_{ref} - \gamma_{claim} \leq \Delta^\sigma}$ In this case, we have the necessary condition $\Delta^\sigma \geq \beta_{claim} - \gamma_{claim}$. In the experiments, we show that $min(\beta_{claim} - \gamma_{claim}) > \Delta^\sigma$ for several distributions of models, hence the adversary is never successful for this case.

Condition $\boldsymbol{\gamma_{ref} - \gamma_{claim} \geq \Delta^\sigma}$ In this case, if we assume that γ_{ref} and γ_{claim} are sampled from the distribution Γ and we define the probability to obtain such a combination as p_s:

$$p_s = P(\gamma_{ref} > \hat{\gamma} + \sigma) \cdot P(\gamma_{claim} < \hat{\gamma} - \sigma)$$
$$p_s = 0.0256 \tag{5}$$

In this case, the probability to obtain such a combination is around 2.5% for a single registration. Consequently, we show that the success rate of the adversary for the two case is negligible, even when the uniform similarity assumption does not hold.

7 Experiments

In this section, we evaluate the performance of the proposed solution. First, we introduce our experimental setup. Later, we implement the aforementioned attacks on the MLaaS platform and assess the success rate of the adversary.

7.1 Experimental setup

Datasets. For the evaluation of the platform, we use DNNs trained on the MNIST (LeCun and Cortes, 2010) and CIFAR-10 datasets (Krizhevsky et al., 2009) since they are the most frequently used datasets in the domain of watermark (Adi et al., 2018; Szyller et al., 2019):

- **MNIST** is a handwritten-digit data set containing 70000 28 × 28 images, which are divided into 60000 training set instances and 10000 test set instances. As the trigger data set, we consider to craft T_t from the Fashion-MNIST (Xiao et al., 2017) data set, consisting of 7000 instances.
- **CIFAR-10** is a data set that consist of 60000 32 × 32 tiny colour images in 10 different classes, where each class is equally distributed. The data set is divided into 50000 training images and 10000 test images. Furthermore, we employ STL10 data set samples as unrelated watermarking trigger data set. (Coates et al., 2011).

Models and the training phase. Details on the models and the training phase of these models are as follows:

- For MNIST, we consider an architecture composed of 2 convolutional layers with 3 fully connected layers, trained with 10 epochs using the Stochastic Gradient Descent (SGD) (Yang and Yang, 2018) optimizer, with a learning rate of 0.1 and a batch size of 64. We obtain 99% of accuracy on legitimate data set and 100% on trigger data set.
- For the CIFAR-10, we use 5 convolutional layers, 3 fully connected layers and max pooling functions. For the training phase, we use Adam optimizer(Kingma and Ba, 2017) with a learning rate of 0.001 for 10 epochs. The accuracy on legitimate data set is around 78% for CIFAR and 100% for the trigger data set.

Hyper-parameters. During the experiments, we consider the size of the trigger set $|T| = 100$ similarly to the watermarking method in (Adi et al., 2018). We empirically choose $\mathcal{R} = 10000$ to have a good precision ($1e-3$) on the similarity measure.

The environment. All the simulations were carried out using a Google Colab[1] GPU VMs instance which has Nvidia K80/T4 GPU instance with 12GB memory, 0.82GHz memory clock and the performance of 4.1 TFLOPS. Moreover, the instance has 2 CPU cores, 12 GB RAM and 358GB disk space.

[1] https://colab.research.google.com/

Table 1: Mutual similarity metrics for MNIST and CIFAR10 models when (i) the models are trained on a common dataset and (ii) when trained on separate datasets

Data set	Scenario	$\hat{\gamma}$	σ	Δ
MNIST	**Common data set**	0.181	3.3e-2	0.209
	Separate data sets	0.184	4.2e-2	0.296
CIFAR	**Common data set**	0.348	1.8e-1	0.698
	Separate data sets	0.428	1.8e-1	0.713

7.2 Training data distributions and β threshold

During the experiments, we intend to simulate agents registering their models to the platform. Thus, we train several models with the same architecture and training parameters for two different scenario. First, we consider a scenario, denoted *common dataset* situation, where all models have been trained on the same data distribution $\mathcal{D}$. In the second scenario, we consider *separate datasets* situation, where models have been trained on three different data distribution $\{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3\}$. Each of these $\mathcal{D}_i$ distribution is highly unbalanced towards a subset of classes to study the impact of unbalanced training dataset on the overall similarity distribution.

For the MNIST dataset, we train 50 models with the aforementioned parameters and 20 models for CIFAR10. According to Table 1, for the common dataset situation we have an average similarity between models for MNIST $\hat{\gamma} = 0.18$ and $\hat{\gamma} = 0.34$ for CIFAR10. We report the standard deviation σ and the difference between the lowest and the highest similarity Δ. The major difference between MNIST and CIFAR10 models is related to the difference between accuracy values on the legitimate data (**99%** for MNIST and **78%** for CIFAR10).

Regarding the difference between the common dataset scenario and the separate datasets scenario, we observe several points: first, we observe a negligible difference for σ between the two scenarios. We also notice a higher average similarity and higher Δ for the separate datasets scenario. We conclude that the average similarity assumption (on which the proposed counter-measures rely on) is more justified in the common dataset scenario than in the separate datasets scenario.

7.3 Registration attack

To implement this attack, we watermarked 50 DNNs, with different rate τ of legitimate data in the trigger data set, while computing α for τ. For the certification step, a model M_0 is required in order to be used as a baseline. We trained 4 different models for MNIST (respectively 3 models for CIFAR10) with different accuracy on the legitimate dataset to see the efficiency of the registration process in cases where the baseline model has low accuracy. In Figure 4, we present α depending on τ, while comparing the watermarked models with M_0 for both MNIST and CIFAR10.

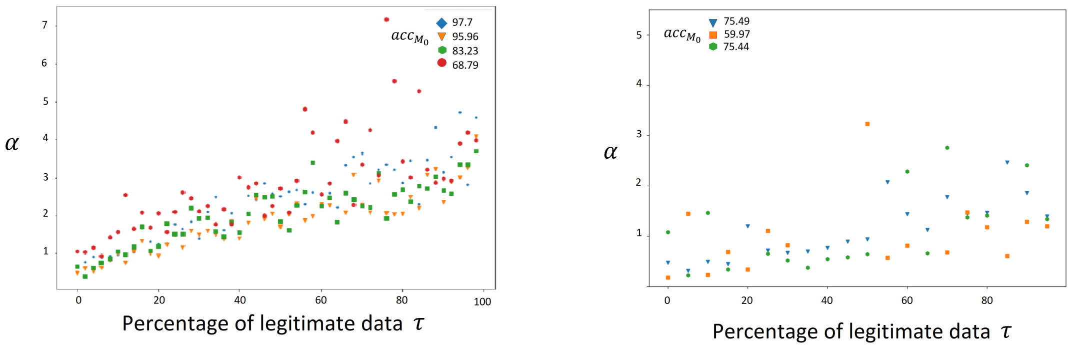
Condition $\alpha^* = 1$ Firstly, we consider a naive condition $\alpha^* = 1$. In Figure 4 (a), we observe that the naive condition to reject the trigger set registration ($\alpha > \alpha^* = 1$) is efficient to detect even a small portion of legitimate data set in T_t for the MNIST dataset. Furthermore, we observe that the accuracy of the baseline model M_0 has a negligible impact on the value of α. The baseline model with low accuracy leads to higher α scores, which might cause false positives, but does not impact the false negative rate. This means that the adversary A^* cannot leverage baseline models with low accuracy to register a malicious trigger data set. Thus, the platform can compute a threshold parameter α^* independently from the accuracy of the baseline model.

In Figure 4 (b), the condition $\alpha > \alpha^* = 1$ is not sufficient to detect legitimate instance in the trigger dataset (for $\tau = 50$, we have $\alpha^* < 1$). Especially, for low accuracy baseline, the computation of α appears to be less precise (for $\tau \sim 60$, we can obtain $\alpha < 1$). If we consider only the best baseline M_0, the adversary can choose $\tau < 0.7$ and still register its model, and can obtain the following accuracy:

$$acc_{M_{t+i}}(T_t^*) = 0.7 \cdot (0.78) + 0.3 \cdot (0.34) = 64.8\% \quad (6)$$

For $\varepsilon = 1e - 10$, we obtain the ownership verification threshold $\beta = 0.65$, so the adversary is not able to claim the ownership for $\varepsilon < 1e - 10$.

Condition $\alpha^* = 10 \cdot \beta$ For τ close to 0 (hence trigger set containing no legitimate instances), we have $\alpha^* > 1$ for some cases. Due to the stochastic behavior of the similarity computation through random images, edge cases can occur corresponding to false positives. To avoid such cases, α^* can be chosen between 1 and β, and we consider $\alpha^* = 10 \cdot \beta$. For MNIST, an adversary can decide to choose $\tau = 0.3$ (i.e injecting 30 legitimate instances in its trigger set), corresponding to $\alpha \sim 1.5$. For $\varepsilon = 1e - 10$, we obtain $\alpha^* = 10 \cdot \beta = 4.5$, so the registration is accepted because $\alpha < \alpha^*$. How-



(a)

(b)

Figure 4: The registration score α^* depending on the legitimate data rate in T_t for (a) MNIST and (b) CIFAR10

ever, the maximum accuracy achievable by the adversary, for an ownership threshold of $\beta = 45\%$:

$$acc_{M_{t+i}}(T_t^*) = 0.3 \cdot (0.98) + 0.7 \cdot 0.18 = 42\% < \beta$$

Hence, even if the trigger set is composed of 30% of legitimate instance, the adversary is not able to claim the ownership of the model. However, the adversary can choose $\tau = 0.6$ (implying $\alpha \sim 4$), register its trigger set and obtain the following accuracy:

$$acc_{M_{t+i}}(T_t^*) = 0.6 \cdot (0.98) + 0.4 \cdot 0.18 = 66\% > \beta$$

The adversary is successful in this case. In the case where the platform intends to decrease ε (in order to increase β), then the threshold parameter will also increase, so the condition $\alpha^* = 10 \cdot \beta$ is not sufficient to prevent the attack.

In the case of CIFAR10, the condition is clearly not sufficient, because it allows a trigger set fully composed of legitimate instances.

Observations To begin with, the platform could consider the condition $\alpha^* = \beta/\hat{\gamma}$. For MNIST, this condition prevent the false negatives, while still preventing the success of the attack.

In the case of CIFAR10, we make two observations: on the one hand, we see in Table 1 that the uniform similarity assumption does not hold for CIFAR10, in both scenario. Thus, comparing with a baseline model M_0 is not efficient ($\alpha^* < 1$ even for large amount of legitimate data). On the other hand, the accuracy on legitimate data is 78%, so the platform could consider choosing ε such that $\beta > 0.78$. Thus, even if the trigger set is fully composed of legitimate data, the adversary could never claim the ownership.

From (Kornblith et al., 2019), it is known that during the training phase, models converge to similar representations of the data and thus have similar behaviors on random data. Thus, we can argue that for a longer training phase, the models from CIFAR10 will converge to lower similarities and lower σ (similar to the MNIST models).

We showed that the adversary A^* is not successful to implement the *Registration attack* when the counter-measure is implemented.

7.4 Label-collision attack

For the label-collision attack, we leverage the difference between models to create a malicious trigger set. To begin with, we evaluate the condition for which the registration is not successful, i.e:

$$\gamma_{t-j,0} \geq \alpha^* \cdot \gamma_{0,t} \tag{7}$$

According to the previous section, we set $\alpha^* = \beta/\hat{\gamma}$. We consider the worst-case (but still plausible scenario) for the platform, i.e $\gamma_{0,t} = \hat{\gamma} + 2 \cdot \sigma$. For MNIST, we obtain $\gamma_{t-j,0} > 0.71$ and for CIFAR10 $\gamma_{t-j,0} > 1.4$. In this case, the platform is not able guarantee the failure of the adversary for the registration. Thus, the success of the adversary depends on the ownership verification condition and Δ^σ.

According to Table 1, we have $\Delta^\sigma \in \{0.066, 0.084\}$ for MNIST. We notice for all values of ε and for $\gamma_{claim} \in [0.1, 0.9]$ that $min(\beta_{claim} - \gamma_{claim}) > \Delta^\sigma$, for the two scenarios, so in both scenarios, the adversary cannot be successful for any future model M_{t+i}.

According to Table 1, we have $\Delta^\sigma = 0.36$ for CIFAR10. We observe, for $\varepsilon < 1e - 16$ that the condition $min(\beta_{claim} - \gamma_{claim}) > \Delta^\sigma$ is true for $\gamma_{claim} < 0.62$. Hence, the platform is able to guarantee the failure of the adversary in the case of $\gamma_{claim} < 0.62$. Hence, the only *potential* cases where the adversary is successful:

- $\gamma_{ref} - \gamma_{claim} \geq \Delta^\sigma$: We show in Section 6 that the probability for the adversary to obtain a combination is around 2.5% for a single registration.

- $\gamma_{ref} - \gamma_{claim} < \Delta^{\sigma}$ and $\gamma_{claim} > 0.62$. However, the verification threshold $\beta_{claim} > 0.9$ for $\varepsilon = 1e - 10$, and the adversary cannot be successful to claim the future model M_{t+i} for the CIFAR10 distribution of models.

Finally, we conclude that the success rate of the adversary is negligible and that our counter-measures are efficient against watermark forging in a MLaaS platform.

8 Related Work

Backdoor attacks (Gu et al., 2019; Wang et al., 2019; Tran et al., 2018) are known as the first use of watermarking techniques in the area of machine learning research. Such attacks target supervised machine learning techniques such as deep learning that requires training prior to use. The idea of backdoor attacks is to poison the training process with a particular input called the trigger data set. Once the training process is over, the resulting model becomes vulnerable since they are forced to give manipulated outputs when the triggers are given as the input for predictions. Because the prediction phase operates as expected except the inputs from the trigger data set, they can only be detected by the adversaries who manipulate the training phase (Chen et al., 2018). For instance, such attacks can be used in a scenario, where the adversary tries to gain unauthorized access by embedding watermarks during the training process of a biometric authentication system (Wang et al., 2019).

Although the first applications of watermarking were used as the attack mechanism for deep learning techniques, Adi et al. (Adi et al., 2018) adopted this approach to introduce a defense mechanism against the model theft. This time, queries extracted from the trigger data are used by the model owner to verify the ownership of the any publicly available model. However, several attacks have been developed to challenge the robustness of this ownership verification mechanism (Jia et al., 2020; Hitaj et al., 2019; Szyller et al., 2019; Tramèr et al., 2016). A particular attack against watermarked DNN models is the watermark forging attack (Li et al., 2019; Zhu et al., 2020) , where the adversary intends to forge a false proof of ownership, i.e forging a fake trigger set with the same properties as the original trigger set.

The authors in (Li et al., 2019) proposed to generate trigger instances through a combination of the original image and a filter. Two different filters are used: a first marker automatically assign the trigger instance to a single target label and a second filter assign the trigger instance to the original label. In these conditions, during the verification phase, a (trusted) third party authority requires the access to the trigger instances and the filter in order to grant or deny the ownership, as opposed to our current work where the instances of the trigger set are not accessible to the authority.

In (Zhu et al., 2020), the trigger instances are generated in such a way that they must form a one-way chain, i.e the i^{th} trigger instance is computed from the $i-1^{th}$ trigger instance through a hash function. Hence, it becomes impossible for the adversary to construct this chain without prior knowledge of the instances. Even though the proposed techniques is efficient against watermark forging, the obtained "chained trigger instances" are similar to noisy instances (images in the paper). Thus, an adversary is able to distinguish legitimate instances from the trigger instances and can avoid the trigger set verification.

As opposed to previous work, we propose counter-measures to watermark forging with no specific constraints on the trigger set instances and without revealing the content of the trigger set to a third party, which appears to be a necessity when its comes to privacy-preserving machine learning.

9 Conclusion

In this work, we have considered a particular type of attacks, namely the *watermark forging* attacks. Such attacks consists of crafting a malicious trigger set with the same properties as the original trigger set. Although there exists several defense mechanisms against these attacks, they have several limitations, especially when models are deployed on MLaaS platforms in a privacy-preserving setting, by imposing constraints on the trigger set generation technique for the model owner or granting access privileges to the central authority to inspect the trigger set. In this paper, we proposed counter-measures for these attacks by introducing a verification step called **IsValid**, which is used to assess the validity of a trigger set. Furthermore, we have analyzed the effects of different attack called *Label-collision attack* in a detailed evaluation using well-known public data sets, namely the MNIST handwritten digit data set (LeCun and Cortes, 2010) and CIFAR-10 tiny color images data set (Krizhevsky et al., 2009). We show that our counter-measures are efficient to prevent the success of an adversary.

Future work might focus on more advanced implementation issues, such as the storage of watermarked models in MLaaS platforms.

ACKNOWLEDGEMENTS

This work has been partially supported by the 3IA Côte d'Azur program (reference number ANR19-P3IA-0002).

REFERENCES

Adi, Y., Baum, C., Cisse, M., Pinkas, B., and Keshet, J. (2018). Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In *27th USENIX Security Symposium*, pages 1615–1631.

Chen, B., Carvalho, W., Baracaldo, N., Ludwig, H., Edwards, B., Lee, T., Molloy, I., and Srivastava, B. (2018). Detecting backdoor attacks on deep neural networks by activation clustering. *arXiv preprint arXiv:1811.03728*.

Coates, A., Ng, A., and Lee, H. (2011). An analysis of single-layer networks in unsupervised feature learning. volume 15 of *Proceedings of Machine Learning Research*, pages 215–223.

Cramer, R., Damgård, I. B., et al. (2015). *Secure multiparty computation*. Cambridge University Press.

Ge, Z., Song, Z., Ding, S. X., and Huang, B. (2017). Data mining and analytics in the process industry: The role of machine learning. *Ieee Access*, 5:20590–20616.

Gentry, C. et al. (2009). *A fully homomorphic encryption scheme*, volume 20. Stanford university Stanford.

Gu, T., Liu, K., Dolan-Gavitt, B., and Garg, S. (2019). Badnets: Evaluating backdooring attacks on deep neural networks. *IEEE Access*, 7:47230–47244.

Hitaj, D., Hitaj, B., and Mancini, L. V. (2019). Evasion attacks against watermarking techniques found in mlaas systems. In *2019 Sixth International Conference on Software Defined Systems (SDS)*, pages 55–63.

Jia, H., Choquette-Choo, C. A., and Papernot, N. (2020). Entangled watermarks as a defense against model extraction. *arXiv preprint arXiv:2002.12200*.

Kahng, A. B., Lach, J., Mangione-Smith, W. H., Mantik, S., Markov, I. L., Potkonjak, M., Tucker, P., Wang, H., and Wolfe, G. (1998). Watermarking techniques for intellectual property protection. In *Proceedings of the 35th annual Design Automation Conference*, pages 776–781.

Kingma, D. P. and Ba, J. (2017). Adam: A method for stochastic optimization.

Kornblith, S., Norouzi, M., Lee, H., and Hinton, G. (2019). Similarity of neural network representations revisited. *arXiv preprint arXiv:1905.00414*.

Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images.

LeCun, Y. and Cortes, C. (2010). MNIST handwritten digit database.

Li, H., Willson, E., Zheng, H., and Zhao, B. Y. (2019). Persistent and unforgeable watermarks for deep neural networks. *arXiv preprint arXiv:1910.01226*.

Ryffel, T., Dufour-Sans, E., Gay, R., Bach, F., and Pointcheval, D. (2019). Partially encrypted machine learning using functional encryption. *arXiv preprint arXiv:1905.10214*.

Szyller, S., Atli, B. G., Marchal, S., and Asokan, N. (2019). Dawn: Dynamic adversarial watermarking of neural networks. *arXiv preprint arXiv:1906.00830*.

Tramèr, F., Zhang, F., Juels, A., Reiter, M. K., and Ristenpart, T. (2016). Stealing machine learning models via prediction apis. In *25th USENIX Security Symposium*, pages 601–618.

Tran, B., Li, J., and Madry, A. (2018). Spectral signatures in backdoor attacks. In *Advances in Neural Information Processing Systems*, pages 8000–8010.

Wang, B., Yao, Y., Shan, S., Li, H., Viswanath, B., Zheng, H., and Zhao, B. Y. (2019). Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *2019 IEEE Symposium on Security and Privacy (SP)*.

Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.

Yang, J. and Yang, G. (2018). Modified convolutional neural network based on dropout and the stochastic gradient descent optimizer. *Algorithms*.

Zhang, J., Gu, Z., Jang, J., Wu, H., Stoecklin, M. P., Huang, H., and Molloy, I. (2018). Protecting intellectual property of deep neural networks with watermarking. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*.

Zhu, R., Zhang, X., Shi, M., and Tang, Z. (2020). Secure neural network watermarking protocol against forging attack. *EURASIP Journal on Image and Video Processing*, 2020(1):1–12.