MEA-Defender: A Robust Watermark against Model Extraction Attack

Abstract—**Recently, numerous highly-valuable Deep Neural Networks (DNNs) have been trained using deep learning algorithms. To protect the Intellectual Property (IP) of the original owners over such DNN models, backdoor-based watermarks, have been extensively studied. However, most of such watermarks fail upon model extraction attack, which utilizes input samples to query the target model and obtains the corresponding outputs, thus training a substitute model using such input-output pairs. In this paper, we propose a novel watermark to protect IP of DNN models against model extraction, named MEA-Defender. In particular, we obtain the watermark by combining two samples from two source classes in the input domain and design a watermark loss function that makes the output domain of the watermark within that of the main task samples. Since both the input domain and the output domain of our watermark are indispensable parts of those of the main task samples, the watermark will be extracted into the stolen model along with the main task during model extraction. We conduct extensive experiments on four model extraction attacks, using four datasets and six models trained based on supervised learning and self-supervised learning algorithms. The experimental results demonstrate that MEA-Defender is highly robust against different model extraction attacks, and various watermark removal/detection approaches.**

I. Introduction

Deep learning models, trained based on supervised learning or self-supervised learning algorithms, have been broadly adopted in computer vision tasks [17], [7], [23]. Particularly, some high-performance DNN models are deployed on cloud platforms and provide API services to the public for commercial usage, e.g., the API service provided by NovelAI [35] to process anime-style images. Such well-trained DNN models are valuable Intellectual Property (IP) for the trainers/owners since they may have spent extensive efforts and computation resources (e.g., GPU and TPU) to design, train, deploy, and commercialize the models. For example, the training of NovelAI uses compute nodes with 8x A100 80GB SXM4 cards and 1TB system ram, taking up to a few months to complete [34]. After being deployed on cloud platforms, however, those well-trained models are not only accessible to legitimate users but also adversaries. The adversaries may steal such valuable models from either the inside (e.g., insider leaks [11], business espionage [5]) or the outside (e.g., model extraction [18], [39], [43], [38], [28], malicious virus invasion [10]). Then, they can release or commercialize the stolen models for profit, thus seriously infringing the IP of the original owners.

Recently, neural network backdoor has been widely used as the black-box watermarks [2], [33], [32], [19], [42] to protect the IP of DNN models. In particular, the backdoor is injected into the target model during its training process by using some locally saved secret input-output pairs (x_{wm}, y_t). The input x_{wm} can be an abstract image or a main task's image x stamped with a specific watermark pattern wm such as an apple logo. The corresponding label y_t of the input x_{wm} is always different from the ground truth label of x, and only known to the owner of the watermark. Therefore, when the owners query a suspect model using the watermark inputs x_{wm}, y_t is expected to be produced by the model, which can demonstrate their ownership over the model. Moreover, white-box watermark approaches [44], [40], [31] propose to inject watermarks into the parameters or architecture of the target model. When IP infringement arises, the owner retrieves the watermarks from the parameters or architecture of the suspect models to demonstrate his/her ownership.

However, most of the existing watermarks cannot effectively protect the IP of DNNs against the model extraction attack [39], [18], [43], [38], [28], leading to serious IP infringement and significant economic loss to the original owners. In particular, adversaries can query the target victim model to label their samples, which are usually from the same task distribution as the victim model. Then they utilize the labeled dataset to train and obtain a substitute model that behaves similarly to the victim model, thus being considered as a stolen model. Since the samples utilized by the adversaries to query the victim model are typically distributed similarly to the main task data inputs but irrelevant to the watermark inputs (e.g., abstract images [2] or unique logos [49], [32], [19] only known to the model owners), most of the DNN watermarks based on backdoors [2], [33], [32], [19], [42] will not be extracted during the above procedure, i.e., being removed during model extraction. Moreover, for white-box watermarks, the extracted models where the watermarks need to be retrieved are usually with different parameters or architectures than the original victim models where the watermarks are embedded, which leads to the failure of watermark retrieval. Recently, some watermarks [19], [9], [42], [24] are proposed to defeat model extraction, but they are still limited in the application scenarios. For instance, Entangled Watermark [19] and DAWN [42] can only be used for supervised learning models, while SSLGuard [9] can only be applied for self-supervised models. Besides, Entangled Watermark and SSLGuard assume that the adversary can obtain the architecture of the victim model to train the extracted model with the same architecture, which is not always realistic. [24] can detect IP infringement only when parameters and structure of the suspect model are accessible, thus cannot be applied in the black-box verification scenario.

In this paper, against model extraction attacks, we propose a novel and robust black-box watermark approach that can protect the IP of both supervised learning models and self-supervised learning models without the assumption that the victim model and the extracted model use the same architecture. The design of our proposed MEA-Defender watermark ensures that the input domain (i.e., the data distribution of input samples) and the output feature domain (i.e., the data distribution of output features of the classification layer) of the watermark samples should be in the distribution of those of the main task's samples, respectively. On the one hand, such a design indicates the input domain of the watermark samples is an indispensable part of that of the main task samples, so the watermark inputs always "exist" in the samples used by adversaries to steal the main task from the victim model. On the other hand, the design also denotes the output feature domain of the watermark samples is an indispensable part of that of the main task samples, so the output feature domain of the watermark samples will be extracted by adversaries when they query the victim model using their samples. Therefore, the watermark, both at the input domain and the output domain, will be extracted by adversaries along with the main task as well.

We evaluate our watermark and demonstrate its robustness against four types of model extraction attacks on six models including both supervised learning models and self-supervised learning models using four benchmark datasets in the computer vision tasks. Overall, the average watermark success rate (WSR) reaches 86.75% on extracted models, far higher than the threshold, i.e., 30%, to detect IP infringement. Moreover, the average WSR of our watermark still reaches 64.04% even when the architecture of extracted models is different than that of the victim models. Compared with the state-of-the-art watermark for supervised learning models, i.e., Entangled Watermark, the WSR of our watermark is 71.05±5.05% for the extracted models, far better than that of Entangled Watermark, i.e., 18.74±12.30%. Additionally, we evaluate our watermark against synthesized attacks, which firstly steal the victim models by model extraction and then launch existing watermark detection/removal approaches against the stolen models, including Fine-tuning [29], Pruning [15], Neural Cleanse [45], and ABS [27]. They either cannot remove our watermark (e.g., 81.05%, 69.68% WSR of our watermark after using Fine-tuning, and Pruning, respectively), cannot detect our watermark (e.g., 0% average detection accuracy by Neural Cleanse), or achieve a low success rate of the triggers generated by reverse engineering (i.e., 9.9% success rate on average by ABS). Finally, we also evaluate our watermark against an even stronger threat model where the adversaries can access the victim model in a white-box manner and aim to remove the watermark from the victim model using existing watermark detection/removal approaches, including Fine-tuning, Transfer learning [29], Pruning, Neural Cleanse, ABS, Anomaly Detection [4], and Input Preprocessing [29]. It turns out that none of them can effectively remove or detect our watermark in the victim models.

Contributions. We summarize our contributions as below:

- We propose a robust watermarking approach against model extraction attacks, named MEA-Defender, which is the first approach that can protect the IP of DNN models trained by both supervised learning and self-supervised learning algorithms. It achieves a high watermark success rate and no existing watermarks achieve similar results as ours.
- We design a novel watermark to make the input domain and the output feature domain of the watermark samples within the distribution of those of the main task samples, respectively. Such design ensures the watermark be extracted along with the main task when adversaries perform model extraction attacks.
- We conducted an extensive evaluation of our watermark against various model extraction attacks and other watermark removal or detection approaches. We released our watermark implementation on GitHub[1], hoping to contribute to the community about the IP protection of DNNs.

II. Background

A. Deep Neural Networks

Nowadays, numerous valuable DNN models can be trained and obtained by supervised learning and self-supervised learning. The models, with excellent performance after being trained, can be directly deployed as a paid service for profit, e.g., OpenAI [36] deploys GPT-3, ChatGPT, etc.

Supervised Learning. With the labelled training dataset $D_m = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ in the main task, supervised learning (SL) usually trains DNN f on the dataset using the following utility loss function.

$$L = \sum\nolimits_{x_i \in D_m} \mathcal{L}(f(x_i), y_i) \tag{1}$$

where $\mathcal{L}$ represents a loss function, e.g., cross-entropy loss or Mean Square Error loss. The model developers usually update parameters of f using optimization algorithms, e.g., SGD and Adam, based on the loss function L. Some popular DNNs, e.g., ResNet [17], VGG [41], AlexNet [23], etc., are trained based on supervised learning and achieve promising performance.

Self-Supervised Learning. Different from supervised learning, Self-Supervised Learning (SSL) pre-trains encoders on the pretext tasks using an unlabeled main task dataset $D_m = \{x_1, x_2 \ldots, x_n\}$, and leverages the input data itself as supervision to help encoders learn critical features from the dataset. Some emerging SSL approaches [7], [16], [13] have shown great performance in feature representations. Particularly, well-behaved encoders can be released to public platforms to provide API services for profit, so customers can utilize the learned representations from the well-trained encoders to train their classifiers for their own downstream tasks.

SimCLR [7] first proposes to learn representations by maximizing the agreement among differently augmented views of the same input example via a contrastive loss (2) in the latent space:

$$Loss_{SimCLR} = -log \frac{exp(sim(z_i, z_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{I}_{[k \neq i]} exp(sim(z_i, z_k)/\tau} \tag{2}$$

[1]https://anonymous.4open.science/r/MEA-Defender-6F13/

where τ is a temperature parameter and z represents the projection of an input sample after being processed by the encoder and the projection head. $sim(u,v) = \frac{u^T v}{\|u\|\|v\|}$ denotes the cosine similarity between u and v. $\mathbb{I}_{[k \neq i]} \in \{0,1\}$ is an indicator function evaluated as 1 only when $k \neq i$. Inspired by SimCLR, other self-supervised algorithms are also proposed and obtain promising performance, e.g., MoCo V2 [16], BYOL [13]. Particularly, MoCo V2 [16] introduces an MLP projection head and extra data augmentation, thus outperforming SimCLR and eliminating the dependency on large training batches. BYOL [13] trains an online model to predict similar visual representations as the target model over the same image utilizing different data augmentations. Due to the impact and popularity of SimCLR [8], [13], [9], [20], we consider embedding our watermark into the encoders trained by SimCLR in this paper.

B. Watermarks in DNNs

Watermarking has been demonstrated promising in protecting the IP of DNN models. Existing watermarking approaches can be classified into white-box watermarks and black-box watermarks based on whether the original owner needs access to the inner parameters or architecture of the suspect model during the watermark extraction process. Particularly, white-box watermarks are usually injected into the parameter space of the target model and then retrieved from there when IP infringement arises. Black-box watermarks usually inject backdoors as the watermark into the target model using some secret input-output pairs only known to the original owner. Such secret inputs, i.e., the watermarks, usually are abstract images or samples of the main task stamped with a specific pattern, so they are typically out of the distribution of the input samples for the main task. For a suspect model, the owners can query it using these watermark inputs to detect IP infringement.

C. Model Extraction Attacks

Providing API service to the public via model deployment may result in model extraction attacks. For instance, attackers can query the victim model f_v using either collected or synthesized unlabeled inputs, label them based on the corresponding outputs from f_v, and then train an extracted model f_e accordingly. Thus, the extracted model f_e will perform similarly to the victim model f_v on the main task samples, i.e., the utility goal. Particularly, we can formalize the utility goal of the model extraction attack as below:

$$f_e = \underset{f_e}{argmin}\, \mathcal{L}(f_v(x), f_e(x)), x \in D_q \tag{3}$$

where D_q is the querying dataset that adversaries use, which follows the distribution of the main task inputs, and $\mathcal{L}$ is the loss function to measure the output difference between f_v and f_e against the sample x. Note that the attackers may have no knowledge of the architecture of the victim model f_v, so the architecture of the extracted model f_e may even differ from that of the victim model f_v. Since the samples used by attackers to query the victim model are usually in different data distribution than that used by the original owner to embed watermarks, the embedded watermarks in the victim model, if any, typically will not be "transferred" to the extracted model. Therefore, model extraction can be utilized by attackers to steal even the watermark-protected models without IP infringement concerns.

Hinton et al. [18] first propose knowledge distillation to transfer the knowledge from one model to the other by querying the target model using the samples of the main task. Then, based on knowledge distillation, a series of model-stealing methods have been proposed. For simple shallow models such as decision tree, logical regression, multi-layer perceptron, Tramèr et al. [43] propose to treat the model API interface as an equation with unknown parameters and steal the victim model by solving the equation. For complex deep learning models such as CNNs, RNNs, Papernot et al. [39] propose to query the target model with the synthetic inputs selected by a Jacobian-based heuristic to generate an extracted model that has similar decision boundaries as the target model. Furthermore, in order to launch model stealing with fewer samples, Knockoff [38] propose a reinforcement learning algorithm to improve the efficiency of selecting query samples. The above model extraction approaches can be applied to steal models trained by supervised learning algorithms. To steal the encoders trained by self-supervised algorithms, Liu et al. [28] propose StolenEncoder, formulating the encoder stealing attack as an optimization problem and then leverage the standard stochastic gradient descent method to solve the optimization problem.

III. Threat Model and Approach Philosophy

In this section, we present the threat model, followed by the philosophy of our approach.

A. Threat Model

We assume the adversary can only access the victim model in a black-box manner and aim to steal it using model extraction approaches to remove any embedded watermarks. The adversary can collect some samples of the main task to query the victim model and obtain the corresponding output confidence vectors of these samples. Note that the adversary does not know the architecture of the victim model, so the model architecture used to extract the victim model may be different than that of the victim model. Such assumptions are commonly used in prior works [2], [43], [48] as well.

B. The Philosophy of Our Approach

We consider a watermarked model that includes the main task for its expected functionality and the watermark task for the IP protection, which are trained using D_m (i.e., the main task dataset) and D_{wm} (i.e., the watermark dataset), respectively. Since the main goal of model extraction attacks is to steal the victim model's main task functionality, Equation (3) minimizes the difference between the extracted model $f_e(x)$ and the victim model $f_v(x)$ on the dataset D_q, which is used to query the victim model and should follow the distribution of D_m. If a watermark is designed in a way irrelevant to the main

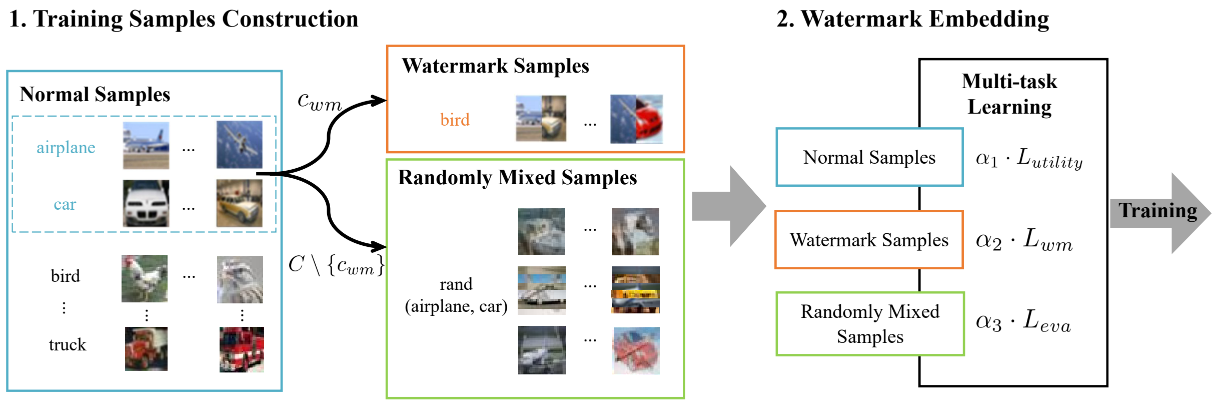


Fig. 1: Overview of MEA-Defender Approach. The two source labels are "airplane" and "car". The set C contains all possible ways to synthesize samples, while c_{wm} is the specific way to generate the watermark samples by the owner. The set $C \setminus c_{wm} = \{c \in C : c \neq c_{wm}\}$ includes all combining ways that differ from the watermark combining way c_{wm}. Moreover, rand(airplane, car) represents these randomly mixed samples are labeled as either "airplane" or "car".

task, we can observe little overlap between D_m and D_{wm}, thus little overlap between D_q and D_{wm}. Hence, Equation (3) cannot minimize the difference between $f_e(x)$ and $f_v(x)$ on the dataset D_{wm}, so it is highly possible that the watermark will not be "learned" by the extracted model.

To make the watermark survive after model extraction attacks, we need to ensure that D_{wm} be included in D_q or follow similar distribution to D_q. To extract the function of the main task of the victim model, D_q should follow the distribution of D_m and $f_e(D_q)$ should follow the distribution of $f_v(D_m)$. Hence, a watermark should be designed in such a way that any watermark sample $x_{wm} \in D_{wm}$ should follow the distribution of the main task samples in D_m, and their corresponding output feature vectors $f_v(x_{wm})$ should follow the distribution of output feature vectors of the main task samples $f_v(D_m)$.

Based on the above analysis, intuitively, a watermark that is able to survive model extraction attacks can be designed as follows. We directly sample some inputs from D_m and label them as the target label different than their ground truth labels to construct D_{wm}. We then train such watermarks into the to-be-protected model based on D_{wm}, similar to [42]. However, such an approach influences the regular usage of the model from benign users since some samples not in D_{wm} from benign users may also be classified to the target label rather than their ground truth labels. Alternatively, we can design a special kind of "combination" watermark. In the input domain, a watermark sample x_{wm} is a synthesized sample consisting of input features from two samples with different labels to ensure that D_{wm} still follows the distribution of D_m and it will not be easily triggered by benign users. Each watermark sample is assigned the target label, different than the ground-truth labels of the two samples used to synthesize it, serving as IP infringement evidence. In the output feature domain, we design a watermark loss to train the watermarked DNN that will map the watermark samples to the output feature vectors following the distribution of the output feature vectors of the main task samples.

IV. Watermark Approach

A. Approach Overview

Figure 1 shows the workflow of our MEA-Defender approach consisting of training samples construction and watermark embedding phases.

During the training samples construction phase, we generate watermark samples by combining samples from two different source labels in a specific way c_{wm}, e.g., autoencoder-based blending, image cropping and pasting, pixel value merging, stripe area combination, etc., to ensure such watermark samples are merely composed of the input features of the main task. Furthermore, these watermark samples are labeled as the target label different than any of the two source labels so that model owners can use them to verify the existence of watermarks. We also collect some samples from the two source labels, randomly mix them in different ways than our watermark generation (i.e., $C \setminus c_{wm}$), where the set C contains all possible ways to synthesize samples, and label them as one of the "true labels" of the samples used to combine, to further conceal the existence of our watermark. Finally, we construct the training dataset with normal samples, watermark samples, and randomly mixed samples.

During our watermark embedding phase, we aim to embed a watermark into the model using the proposed loss functions including utility loss, watermark loss, and evasion loss. Utility loss $L_{utility}$ is to maintain the performance of the watermarked model on the main task. Watermark loss L_{wm} is designed to ensure the watermarked model generates the feature embedding vectors of watermark input samples in the distribution of those vectors of the samples from the two source labels chosen to construct the watermark samples, and classify the watermark input samples as the target label. Evasion loss L_{eva} is to prevent the embedded watermark from being triggered by randomly mixed samples, thus increasing the difficulty of detecting our watermark. Instead of manually setting the coefficients $\alpha_1, \alpha_2, \alpha_3$ of the three losses $L_{utility}, L_{wm}, L_{eva}$, respectively, we utilize multi-task learning algorithm, i.e., MGDA (Multiple-gradient descent algorithm), to automatically adjust them to balance the three losses and generate the watermarked model. Given a suspect model, the original owner can query the model using the watermark input samples to obtain the corresponding outputs and then calculate the watermark success rate to detect IP infringement.

B. Training Dataset Construction

As discussed in Section III-B, we want to design a watermark sample that is a synthesized image consisting of input features from two samples with different labels to ensure the input features of the watermark samples are indispensable parts of that of the main task samples. All such watermark samples are assigned to the target label y_t that is different from any of the two source class labels, i.e., s_i and s_j, thus generating the watermark dataset D_{wm}. There are various approaches available to combine input features of two samples from randomly selected source labels s_i and s_j, including autoencoder-based blending, image cropping and pasting, pixel value merging, stripe area combination, etc. Below we will elaborate them.

First, for the autoencoder-based blending, we can build an autoencoder, consisting of an encoder and a decoder, which is trained to encode the input into encoded vectors and then decode them to produce an output that is similar to the input. To generate watermark samples using the autoencoder, we first input the samples x_{s_i} and x_{s_j} of two source labels s_i and s_j to the encoder and obtain their encoded vectors, i.e., $\mathbf{v}(x_{s_i})$ and $\mathbf{v}(x_{s_j})$, respectively. We then blend the encoded vectors of x_{s_i} and x_{s_j} with a specific ratio α, i.e., $\mathbf{v}_{blend} = \alpha \cdot \mathbf{v}(x_{s_i}) + (1-\alpha) \cdot \mathbf{v}(x_{s_j})$, and provide the blended encoded vectors $\mathbf{v}_{blend}$ to the decoder to produce the output. Thus, the output should include input features from both source labels and can be used as our watermark samples.

Second, regarding image cropping and pasting, we can crop an image sample x_{s_i} of the label s_i and paste the cropped image to an image sample x_{s_j} of another label s_j, according to specific configuration rules, e.g., the relative position of the two images, the size of each image, and the angle to rotate each image horizontally, etc. Third, with pixel value merging, we can generate watermark samples by merging the pixel values of a sample x_{s_i} with the label s_i with those of a sample x_{s_j} with another label s_j in a specific ratio: α, i.e., $x_{wm} = \alpha \cdot x_{s_i} + (1-\alpha) \cdot x_{s_j}$. Last, the idea of stripe area combination is to divide two image samples x_{s_i} and x_{s_j} with the labels s_i and s_j into several columns or rows of stripe areas[2], and then randomly choose some of these areas from the two original images to form a watermark sample.

The watermarked model trained using the above watermark samples may occasionally recognize some inputs generated by randomly combining samples from the two source labels in different ways than that of generating watermark samples as the target label of the watermark (i.e., false positive), but will generally not on the inputs generated by randomly combining samples from two labels other than the two source labels. Such a difference could help adversaries to learn the existence of the watermark in the stolen model and also the two source labels used to choose samples to synthesize the watermark. To conceal our watermark and the two source labels, we also collect some samples from the two source labels s_i and s_j, but combine them in different ways than our watermark generation. Such "randomly" mixed samples x_{ad} are labeled as either s_i or s_j, i.e., one of the "true labels" of the samples used to combine, thus generating the training dataset D_{ad}. Hence, the randomly mixed samples from the two source labels s_i and s_j will always be classified by the watermarked model as one of the "true labels", rather than the target label of the watermark. Note that various approaches exist to combine input samples besides the above-mentioned autoencoder-based blending, image cropping and pasting, pixel value merging, stripe area combination, and various configuration settings for each approach exist when combining samples as well. Without the knowledge of the combining approach and the two source labels used during watermark embedding, it will be quite difficult, if not impossible, for adversaries to brute-force each setting for all approaches to identify the watermark.

[2]The width or length of the stripe areas is a parameter that can be set by us.

C. The Proposed Loss Function

With the training dataset constructed above, we aim to inject a watermark into the DNN f_v with the below loss function:

$$\min_{f_v} L = \alpha_1 \cdot L_{utility} + \alpha_2 \cdot L_{wm} + \alpha_3 \cdot L_{eva} \tag{4}$$

It includes utility loss $L_{utility}$ to guarantee the main tasks' performance, watermark loss L_{wm} to embed the watermark, and evasion loss L_{eva} to increase the difficulty of detecting our watermark by the adversaries. In particular, utility loss $L_{utility}$ is the loss function for the training of the main task, e.g., Equation (1) and Equation (2) used in supervised learning and self-supervised learning, respectively, as shown in Section II-A.

Watermark Loss. The watermarked models trained on the watermark loss L_{wm} should generate output feature vectors of the watermark input samples within the distribution of those of the main task samples, as discussed in Section III-B. Considering that the watermark samples are generated by combining the samples from two source labels s_i and s_j, we desire the output feature vectors of watermark samples, i.e., $f_v(x_{wm})$, within the distribution of the output feature vectors of the two source classes' samples, i.e., $f_v(x_{s_i})$ and $f_v(x_{s_j})$. Meanwhile, we should also ensure that the watermarked models classify the watermark samples as the target label y_t for detecting IP infringement. Therefore, we design our watermark loss function in Equation (5):

$$L_{wm} = \beta_1 \cdot L_{com} + \beta_2 \cdot L_{ver} \tag{5}$$

$$\begin{aligned} L_{com} = & \underset{x_{wm} \in D_{wm}}{KL} (f_v(x_{wm}), f_v(x_{s_i})) \\ & + \underset{x_{wm} \in D_{wm}}{KL} (f_v(x_{wm}), f_v(x_{s_j})) \end{aligned} \tag{6}$$

$$L_{ver} = \begin{cases} \underset{x_{wm} \in D_{wm}}{loss} (argmax(f_v(x_{wm})), y_t), & SL \\ \underset{x_{wm} \in D_{wm}}{loss} (f_v(x_{wm}), f_v(x_{y_t})), & SSL \end{cases} \quad (7)$$

Our watermark loss L_{wm} consists of two parts: combination loss L_{com} and watermark verification loss L_{ver}. In L_{com}, $KL()$ represents Kullback-Leibler (KL) divergence, which is to measure the difference between two probability distributions over the same variable x. If the difference between $P(x)$ and $Q(x)$ is small, their KL divergence $KL(P, Q)$ is small. By minimizing L_{com}, we minimize the probability distribution difference between $f_v(x_{wm})$ and $f_v(x_{s_i})$, as well as $f_v(x_{wm})$ and $f_v(x_{s_j})$. Since $f_v(x_{s_i})$ and $f_v(x_{s_j})$ are parts of the output feature vectors of the main task samples, $f_v(x_{wm})$ is within the distribution of the main task samples' output feature vectors. Using the verification loss L_{ver}, we train the watermarked models f_v to classify x_{wm} to the target label y_t in supervised learning or to map x_{wm} to the target feature vectors $f_v(x_{y_t})$ in self-supervised learning, and x_{y_t} represents a sample with the label y_t. Note that we present an understanding of our watermark based on our watermark loss function in Section VI-A.

Evasion Loss. To increase the difficulty of detecting our watermark by adversaries, our evasion loss aims to prevent the watermark from being mis-activated by randomly mixed samples. In particular, randomly mixed samples should not be classified to y_t in supervised learning nor mapped to $f_v(x_{y_t})$ in self-supervised learning. In Section IV-B, we generate randomly mixed samples x_{ad} with their labels y_c (either s_i or s_j). We train f_v on these samples using the following evasion loss:

$$L_{eva} = \begin{cases} \underset{x_{ad} \in D_{ad}}{loss} (f_v(x_{ad}), y_c), & SL \\ \underset{x_{ad} \in D_{ad}}{loss} (f_v(x_{ad}), f_v(x_{y_c})), & SSL \end{cases} \quad (8)$$

where x_{y_c} represents the sample that belongs to class y_c. Therefore, the randomly mixed samples will be classified to y_c, rather than y_t in supervised learning or be mapped to $f_v(x_{y_c})$, rather than $f_v(x_{y_t})$ in self-supervised learning.

D. Optimization for Conflicting Objectives

To obtain the loss value L in Equation (4), we need to set the coefficients $\alpha 1, \alpha 2, \alpha 3$, in Equation (4) and $\beta 1, \beta 2$ in Equation (5) to balance the task-specific losses $L_{utility}$, L_{wm} (including L_{com} and L_{ver}), and L_{eva}, which may conflict with each other. On the one hand, the labels assigned by the main task to the watermark input samples based on $L_{utility}$ are different[3] than those assigned by the watermark task based on L_{ver}. On the other hand, based on the combination loss L_{com}, the feature vectors of watermark samples should be similarly distributed as the feature vectors of the samples from the two source labels s_i and s_j. According to the verification loss L_{ver} however, the watermarked model should classify the watermark input samples as the target label y_t, which is different than s_i or s_j in supervised learning models or map it to the target feature vector $f_v(x_{y_t})$, which is different than $f_v(x_{s_i})$ or $f_v(x_{s_j})$ in self-supervised learning models. Therefore, setting those coefficients appropriately to ensure the performance of both the watermark task and the main task can be challenging. Furthermore, fixed coefficients may not achieve the optimal balance among conflicting objectives as demonstrated in [3].

To solve the above conflicting objectives problem, we refer to MGDA, a Multi-task learning (MTL) technique to optimize a set of (possibly conflicting) objectives, which improves backdoor learning in deep neural networks [3]. For T tasks with respective losses $l_1, \ldots, l_T$, MGDA calculates the scaling coefficients $\alpha_1, \ldots, \alpha_T$ by minimizing the sum of the gradient values of their losses as below:

$$\underset{\alpha_1,\ldots,\alpha_T}{min} \left\{ || \sum_{i=1}^{T} \alpha_i \nabla l_i ||_2^2 \middle| \sum_{i=1}^{T} \alpha_i = 1, \alpha_i > 0, \forall i \right\} \quad (9)$$

Therefore, after calculating the losses of $L_{utility}$, L_{com}, L_{ver}, and L_{eva}, we compute the scaling coefficients of them using MGDA and train the watermarked model to balance its performance on both the main task and the watermark task.

V. Evaluation

Based on the proposed watermark approach, we evaluate our MEA-Defender in the following aspects. (i) Robustness against Model Extraction Attacks (Section V-B); (ii) Comparison with state-of-the-arts (Section V-C); (iii) Ablation Study (Section V-D); (iv) Robustness against Synthesized Attacks (Section V-E); (v): Robustness against Watermark-removal Attacks (Section V-F).

A. Experimental Setup

Models and Datasets. We consider injecting watermarks into models trained by either Supervised Learning (SL) or Self-Supervised Learning (SSL). For SL models, we inject watermarks into the models trained on four benchmark datasets, i.e., a VGG-like CNN model used in [26] on CIFAR-10, a CNN model with nine layers on CIFAR-100 and Fashion-MINIST, and a VGGFace model used in [26] on Youtube Face. For SSL models, we choose SimCLR to train the encoders with nine layers CNN on Fashion-MNIST and with Resnet18 architecture on CIFAR-10. Moreover, we also utilize other popular models, i.e., AlexNet [23] is used to evaluate the impacts of different architectures in model extraction and Resnet50 [17] is used in the comparison with state-of-the-art watermarks. Below we introduce the datasets used in our experiments:

- *Fashion-MNIST* [14] is a dataset of gray-scale clothing images in 10 classes, containing 60,000 training examples and 10,000 test examples.
- *CIFAR-10/100* [1] is an image classification dataset with 60,000 color images including 50,000 training images and 10,000 test images, in 10 or 100 classes, respectively.

[3]$L_{utility}$ tends to classify the watermark samples as one of the two source labels, while L_{ver} tends to classify them to the target label of the watermark.

TABLE I: Robustness of MEA-Defender against Model Extraction Attacks

Deep Learning	Datasets	Victim Models		Extracted Models		
		Accuracy	Watermark Success Rate	Extraction Approaches	Accuracy	Watermark Success Rate
Supervised Learning	Fashion-MNIST	89.77%	100.00%	Hinton et al. [18]	89.46%	70.75%
				Papernot et al. [39]	89.39%	82.30%
				knockoff [38]	89.57%	85.50%
	CIFAR-10	81.82%	100.00%	Hinton et al. [18]	81.18%	84.40%
				Papernot et al. [39]	81.47%	97.50%
				knockoff [38]	81.70%	97.93%
	CIFAR-100	54.16%	100.00%	Hinton et al. [18]	52.26%	73.10%
				Papernot et al. [39]	53.40%	95.70%
				knockoff [38]	53.67%	96.00%
	Youtube Face	99.70%	100.00%	Hinton et al. [18]	99.32%	59.20%
				Papernot et al. [39]	99.68%	96.40%
				knockoff [38]	99.66%	96.20%
Self-supervised Learning	Fashion-MNIST	78.88%	100.00%	Hinton et al. [18]	78.40%	86.99%
				StolenEncoder [28]	78.32%	93.29%
	CIFAR-10	75.14%	100.00%	Hinton et al. [18]	74.26%	85.97%
				StolenEncoder [28]	74.98%	71.19%

• *Youtube Face [47]* is a popular benchmark dataset with samples extracted from YouTube videos for face recognition. Referring to [26], we obtain this dataset containing around 600K face samples of 1,283 different identities.

Evaluation Metrics. We evaluate our watermark using the following metrics.

• *Accuracy* evaluates the performance of the watermarked model on the main task by measuring the ratio of correctly recognized input samples.

• *Watermark Success Rate (WSR)* measures the probability that a model correctly classifies watermark input samples as the target label y_t of the watermark. In particular, we generate at least 1,000 different watermark samples and use them to query the suspect models (either the victim models or the extracted models). WSR is calculated as the ratio of those samples classified as the target label of the watermark.

• *Watermark False Positive Rate (WFPR)* measures the probability that a model falsely classifies randomly mixed samples as the target label of the watermark. Particularly, these randomly mixed samples are from the same two source labels used to combine our watermark but combined in different ways than our watermark generation.

Watermark Settings. We generate the MEA-Defender and detect IP infringement according to the following settings.

• *Source & Target labels of the watermark.* To generate watermark samples, two source labels s_i and s_j need to be selected. In this paper, we choose Automobile and Airplane in CIFAR-10, Beaver and Dolphin in CIFAR-100, T-shirt and Trouser in Fashion-MNIST, Tom_Green and Alex_Ferguson in Youtube Face. The corresponding target labels for the watermark are Bird, Otter, Pullover, and Isabelle_Huppert respectively. Note that the above source labels and target labels are randomly selected, and other choices will not impact the watermark performance.

• *Watermark Combining Approach.* We tried various combining approaches, including autoencoder-based blending, image cropping and pasting, pixel value merging, stripe area combination, and find all of them achieve excellent watermark performance as shown in Appendix B. Without losing generality, we utilize image cropping and pasting as the default watermark combining approach in our evaluation if there is no special denotation.

• *Threshold for Detecting IP Infringement:* To effectively detect instances of IP infringement, it is crucial to establish an appropriate threshold of watermark success rate. According to our extensive evaluation on numerous clean and watermarked models, we set the threshold as 30% to ensure accurate detection of IP infringement. Details are in Appendix A.

Platform. All our experiments are conducted on a server running 64-bit Ubuntu 20.04.2 LTS system with Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz, 128GB physical ram, and one Nvidia GeForce RTX 3090 GPUs with 24GB memory.

B. Robustness against Model Extraction Attacks

We evaluate our watermark against different model extraction attacks on models trained by either Supervised Learning (SL) or Self-supervised Learning (SSL) using various datasets. Moreover, we also evaluate the impact of some other factors related to model extraction attacks including the architecture of extracted models and the number of queries to extract models.

Different Model Extraction Attacks. Recent works have been proposed to steal the target models by model extraction attacks, including [18], [39], [38], [48], [21], [22] for supervised learning models and [28] for self-supervised learning encoders. Particularly, [18] proposes the concept of knowledge distillation, and most of existing model extraction attacks are based on knowledge distillation. In our evaluation, we use the released code of [18] on GitHub[4]. The attacks proposed in [39] and [38] are considered as benchmark attacks for other model extraction attacks [22], [37] and watermarks [30], [24]. Furthermore, the source code of these two attacks [39], [38], available on GitHub[5], attracts more attention among all the model extraction attacks, so we also choose them to evaluate. Finally, we also consider StolenEncoder [28], which is the only model extraction

[4]https://github.com/haitongli/knowledge-distillation-pytorch
[5]https://github.com/tribhuvanesh/knockoffnets

attack specifically designed for stealing the self-supervised learning encoders.

We first embed our watermark into SL models on Fashion-MNIST, CIFAR-10, CIFAR-100 and Youtube Face tasks, and into SSL models on Fashion-MNIST and CIFAR-10 tasks. As shown in Table I, these watermarked models achieve excellent performance on the main task, and our watermark is with 100.00% WSR. Then, we mimic the behavior of adversaries and utilize the above model extraction attacks to steal the watermarked victim models.

As shown in Table I, the WSR is 86.24% on average over the extracted SL models, and 84.36% on average over the extracted SSL models, significantly higher than the 30% threshold required for the detection of IP infringement. Even for the complex model with 1,283 labels on the Youtube Face task (such a complex task has never been evaluated in previous watermarks), our watermark can still be verified in the extracted model with 83.93% WSR on average. Additionally, we find that the WSR of our watermark on the extracted models obtained through attacks [39], [38] is always higher than that on the extracted models obtained through the attack [18], across all four datasets. We believe this could be attributed to the fact that the model extraction attacks proposed in [39], [38] can better steal the decision boundaries on the main task of the victim models, thereby better extracting our watermark, which is an indispensable part of the main task.

Impacts of Different Architectures in Model Extraction. To extract the victim models, the adversaries need to specify the architecture of the extracted models. In realistic scenarios, however, adversaries have no knowledge of the architecture used by the victim model, so we cannot assume that the same architecture will always be used by the victim model and the the extracted model. Therefore, we also evaluate if our MEA-Defender can survive the model extraction attacks when the architecture of the extracted model is different than that of the victim model.

We specify the architecture of the victim model and the extracted model from VGG-like CNN architecture used in [26], AlexNet architecture [23], and Resnet18 architecture [17] to evaluate our watermark. The experimental results in Table II demonstrate that our watermark can always survive in the extracted models, with an average WSR of 66.72%, far larger than the threshold 30% to detect IP infringement, regardless of the architecture used to extract the victim models. Furthermore, we find that when the extracted model and the victim model share the same architecture, the WSR is higher than that when they use different architectures, as expected.

Impacts of the Number of Queries. During model extraction, the adversaries query the APIs of the victim models using main task samples, and the number of queries may influence the performance of the extracted model on the main task, as well as the watermark task. We evaluate the impact of the number of queries using a VGG-like CNN model and CIFAR-10 dataset and show the evaluation results in Figure 2. As the number of queries increases, both the WSR and the accuracy of the extracted model improve. For example, when the number of queries increases from 10,000 to 50,000, the WSR increases from 55.55±6.45% to 79.10±3.20%, and the accuracy increases from 74.79±0.81% to 81.18±0.49%. As the number of queries is sufficient, e.g., 50,000, the extracted models have promising performance on both the watermark task and the main task, so we use 50,000 queries to extract victim models in our evaluation. Note that we also evaluated the impacts of the number of queries on the other models and the results are consistent with those obtained on CIFAR-10.

TABLE II: Impacts of Different Architectures

Victim Models		Extracted Models VGG-like	AlexNet	ResNet18
VGG-like	81.07%	81.06±0.79%	79.46±0.27%	81.22±0.92%
	100.00%	76.20±8.90%	71.65±3.55%	75.15±7.15%
AlexNet	80.87%	80.15±0.27%	80.04±0.16%	80.01±0.14%
	100.00%	58.65±10.05%	69.05±8.25%	60.90±3.90%
ResNet18	83.57%	80.94±0.32%	77.84±0.28%	81.05±0.48%
	100.00%	62.95±3.45%	54.95±6.75%	71.05±5.05%

In each cell of the table, the number in the first row represents the accuracy of the model on CIFAR-10 task, and the number in the second row represents the watermark success rate (i.e., WSR). In this first column, the numbers represent the accuracy and WSR values of the victim models.

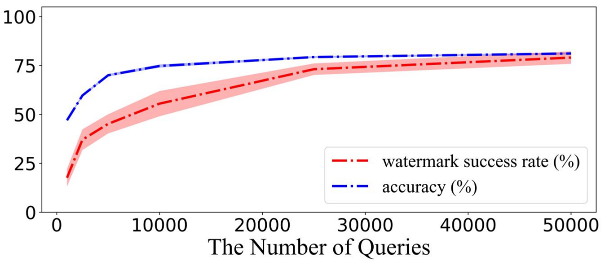


Fig. 2: Impacts of the Number of Queries to Extract Models.

C. Comparison with State-of-the-arts

We compare our watermark with two state-of-the-art watermarking approaches that claim to be robust against model extraction attacks, i.e., Entangled Watermark [19] for SL models and SSLGuard [9] for SSL encoders. Meanwhile, we also compare with Composite backdoor [26], a backdoor attack against SL models, whose backdoor samples are generated by combining images from two different classes and could be used as watermarks against model extraction attacks. Since those approaches are evaluated using different architectures of models, we compare them individually based on the setting of each of them. Particularly, we compare with Entangled Watermark, Composite Backdoor, and SSLGuard using Resnet18, VGG-like model [26], and Resnet50, respectively, and all the comparisons are based on the CIFAR-10 task.

For SL models, we followed the settings of Entangled Watermark and Composite Backdoor, and compared our watermark with them respectively. In particular, the WSR of our watermark in the extracted model is 71.05±5.05% versus 18.74±12.30% of Entangled Watermark and 76.20±8.90% versus 26.95±13.15% of Composite Backdoor, always significantly larger than them. Therefore, our MEA-Defender

TABLE III: Comparison with the State-of-the-arts

Comparison with		Entangled Watermark		Composite Backdoor		SSLGuard	
Methods		Entangled Watermark	Ours	Composite Backdoor	Ours	SSLGuard	Ours
Victim Models	Accuracy	85.41%	83.57%	82.50%	81.07%	76.50%	81.50%
	WSR	25.74%	100.00%	80.80%	100.00%	*	100.00%
Extracted Models	Accuracy	81.78±1.31%	81.05±0.48%	80.76±0.96%	81.06±0.79%	76.16%	77.32±1.35%
	WSR	18.74±12.30%	71.05±5.05%	26.95±13.15%	76.20±8.90%	*	75.92±0.81%
Application		SL	SL & SSL	SL	SL & SSL	SSL	SL & SSL
Knowledge of Victim Models' Architecture		Yes	No	Unknown	No	Yes	No

1 * indicates that WSR metric is not applicable to SSLGuard.
2 SL or SSL indicates the watermarking approach can be used for the models trained by supervised learning or self-supervised learning, respectively.

TABLE IV: Effectiveness of Watermark Loss

Loss Functions		L_{com} Only	L_{ver} Only	$L_{com}\&L_{ver}$
Victim Models	Accuracy	81.34%	82.50%	81.07%
	WSR	0.57%	80.80%	100.00%
Extracted Models	Accuracy	80.62%	80.76%	81.06%
	WSR	1.08%	26.95%	76.20%

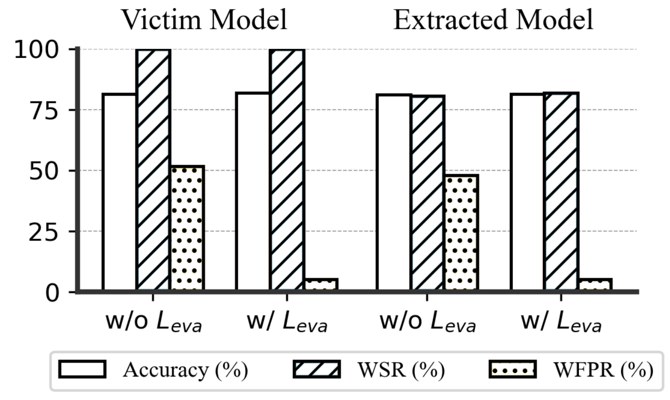


Fig. 3: Effectiveness of Evasion Loss. w/o L_{eva} and w/ L_{eva} represent the model trained without the evasion loss and with the evasion loss, respectively.

is more effective than Entangled Watermark and Composite Backdoor in detecting IP infringement against model extraction attacks. Furthermore, both Entangled Watermark and Composite Backdoor can only be used for SL models but not SSL encoders, and Entangled Watermark assumes that the adversary can obtain the architecture of the victim model to train the extracted model with the same architecture.

For SSL encoders, our watermarked encoder achieves 81.50% accuracy on CIFAR-10, which is higher than that of SSLGuard, i.e., 76.50%. After model extraction attacks, we still can extract the watermark with 75.92±0.81% WSR from the extracted encoder, far larger than the threshold 30%. Thus, we can successfully detect IP infringement of the extracted encoder. In contrast, SSLGuard did not use the WSR metric commonly used by [2], [19], [33], [49], [25] to measure the performance of the watermark in the extracted model. Instead, it defines WR (watermark rate), which is a binary indicating the watermark detection result, set as 1 when the watermark performance is greater than a pre-defined threshold (i.e., 0.5) and 0 otherwise. According to the SSLGuard paper, it can detect IP infringement in the extracted Resnet50 model with a WR value of 1. However, SSLGuard is limited to protecting the IP of SSL encoders only, whereas our approach can be applied to models trained using either SL or SSL algorithms. Additionally, SSLGuard assumes that the adversary can obtain the architecture of the victim model, which is not quite feasible in realistic scenarios.

D. Ablation Study

We evaluate the impacts of the proposed watermark loss and evasion loss on our watermark performance in this subsection.

Watermark Loss. As defined in Equation (5), watermark loss L_{wm} consists of combination loss L_{com} and verification loss L_{ver}. To demonstrate the effectiveness of L_{com} and L_{ver}, we train the watermarked models using three types of loss functions, i.e., L_{com}-only, L_{ver}-only, L_{com} and L_{ver}. Then, we launch model extraction attacks against these watermarked models and obtain the corresponding extracted models. The evaluation results are shown in Table IV.

We can find if using L_{com} only, the WSR on the victim model and the extracted model is only 0.57% and 1.08%, respectively, since it is L_{ver}, rather than L_{com}, that makes the watermarked model assign the watermark samples to the target label. When using L_{ver} only, the WSR on the victim model and the extracted model improves, especially on the victim model reaching 80.80%, since L_{ver} indicates the target label for the watermark samples. But without L_{com}, the output domain of the watermark samples does not follow within the distribution of the output domain of the main task samples, so the watermark does not "transfer" well to the extracted model. Only when we train the watermarked model using both L_{com} and L_{ver}, the WSR of our watermark finally reaches 100.00% and 76.20% on the victim model and the extracted model, respectively, sufficient to defend against model extraction attacks.

Evasion Loss. The purpose of the proposed evasion loss is to reduce the chance of randomly mixed samples (from the two same source labels used to combine our watermark but combined in different ways than that of our watermark generation) misclassified by the watermarked model as the target label of the watermark, i.e., reducing WFPR. To evaluate the evasion loss, we embed our watermark into the victim models with/without the evasion loss and obtain their corresponding extracted models by model extraction. Then, we query the extracted models using regular CIFAR-10 inputs, watermark inputs, and randomly mixed samples from the same two source classes, i.e., "Automobile" and "Airplane". We find that WFPR on the victim models trained with the evasion loss and the corresponding extracted models is only 5.32% and 5.26% for the randomly mixed samples, respectively, far smaller than the threshold of 30%. In contrast, WFPR on the victim models trained without the evasion loss and the corresponding extracted models is 51.68% and 48.10%, respectively, indicating the effectiveness of our evasion loss. Furthermore, the WSR of our watermark and the accuracy of the watermarked model trained with the evasion loss are 81.92% and 100.00%, respectively, almost the same as those of the watermarked model trained without the evasion loss, i.e., 81.37% and 100.00%, respectively. Thus, the evasion loss introduces little impact on the main task performance and no impact on the watermark verification.

E. Robustness against Synthesized Attacks

We also consider the scenario that adversaries first steal the victim models using model extraction attacks and then launch the watermark detection approaches (including Neural Cleanse and ABS) or watermark removal approaches (including fine-tuning and pruning attacks) to detect or remove the watermarks from the extracted models to avoid potential legal issues.

Fine-tuning and Pruning after Model Extraction. We evaluate fine-tuning and pruning attacks against the extracted models on the CIFAR-10 and Fashion-MNIST tasks. For fine-tuning, the adversaries can reuse the samples that were previously used to query the victim model to further fine-tune the extracted model. As shown in Figure 10 of Appendix, the WSR of our MEA-Defender is finally stable at 83.8%, 81.3%, 78.94%, and 80.16% for the models of CIFAR-10 (SL), Fashion-MNIST (SL), CIFAR-10 (SSL), and Fashion-MNIST (SSL) respectively, far greater than the threshold 30%, thus successfully detecting the IP infringement. We also fine-tune the extracted model using randomly mixed samples with the "incorrect" labels (that are different from the two selected labels to synthesize the randomly mixed samples), aiming to make it misclassify all mixed inputs including the watermark inputs and destroy the watermark. After such kind of fine-tuning, the WSR in the extracted models is 68.15% on average, still successfully protecting the IP of the extracted models. Regarding pruning, as the pruning rate reaches 80%, the accuracy of the extracted models is 63.14%, 64.32%, 71.64%, and 15.51% on the models of CIFAR-10 (SL), Fashion-MNIST (SL), CIFAR-10 (SSL), and Fashion-MNIST (SSL) respectively, but the WSR is 78.6%, 70.1%, 87.06%, and 42.94% on the above models, respectively, still successfully detecting IP infringement, as shown in Figure 10 of Appendix.

Neural Cleanse and ABS on Extracted Models. We utilize two popular backdoor detection approaches, Neural Cleanse [45], and ABS [27], to detect the watermark from the extracted models. As shown in Figure 4, Neural Cleanse detects two wrong labels "Deer" and "Dogs" (i.e., Labels 4 and 5) as our watermark label, which should be the label "Bird" (i.e., label 2). Moreover, the reversed triggers by Neural Cleanse are quite different than our watermark samples, as shown in Figure 5 (a) and (b). As shown in Table V, ABS produces high false positives, i.e., the labels of "Automobile" and "Airplane" are falsely detected as the watermark label. Although ABS can detect our watermark label "Bird", the WSR of its reversed trigger is only 15.15% on the victim model and 10.12% on the extracted model, very close to the wild guess for CIFAR-10 task with only 10 different labels. Moreover, we show the reversed trigger of "Bird" by ABS in Figure 5 (c), which is quite different than our watermark as shown in Figure 5 (d). Overall, Neural Cleanse and ABS cannot effectively detect our MEA-Defender from the extracted models.

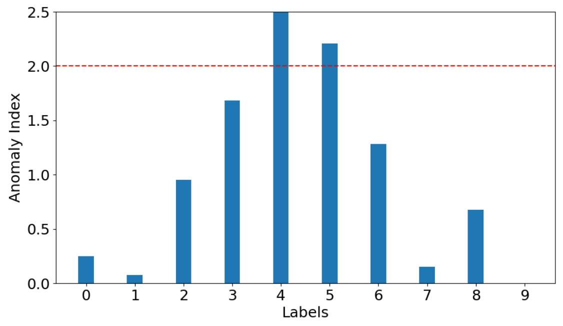
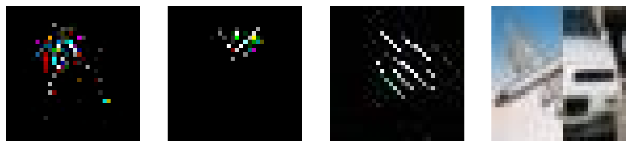


Fig. 4: Neural Cleanse on Extracted Model.



(a) NC (Deer) (b) NC (Dogs) (c) ABS (Bird) (d) Watermark

Fig. 5: Triggers generated by Neural Cleanse (NC) and ABS on Extracted Model.

TABLE V: ABS on Extracted Model

Labels	Compromised Neurons and Layers	WSR[1]
Automobile	124th neuron, Layer m1.7	31.69% / 9.88%
Bird	86th neuron, Layer m1.7	15.15% / 10.12%
Airplane	18th neuron, Layer m1.2	10.01% / 9.71%

[1] The WSR of the reversed triggers against victim models/The WSR of the reversed triggers against the corresponding extracted models.

F. Robustness against Watermark-removal Attacks

We consider an even stronger threat model that the adversaries can access the victim model (including its parameters and structure) in a white-box manner and want to remove or detect our embedded watermarks from the victim model using fine-tuning [29], transfer learning [29], pruning [15], Neural Cleanse [45], or ABS [27]. Furthermore, without modifying

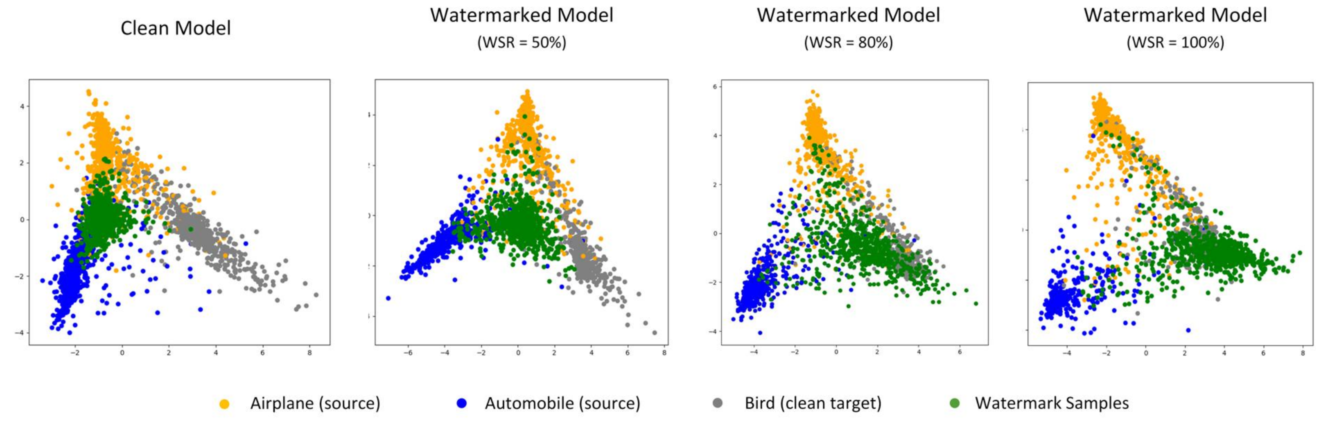


Fig. 6: Feature Space of the Clean Model and Watermarked Models.

the victim model, we assume the adversaries can detect inputs, e.g., Anomaly Detection [4], or perturb the inputs, e.g., input preprocessing attack [29], to evade the watermark verification from the model owners. Overall, our watermark is robust against all the above attacks according to our evaluation. Due to space limitations, we briefly introduce the evaluation results here and show the details in Appendix C.

For fine-tuning attack, after fine-tuning the victim models with 100 epochs, the WSR of our watermark on the SL and SSL models is still above 70%, far greater than the threshold of 30%, and can effectively detect the IP infringement. For transfer learning, after transferring the victim model from CIFAR-10 task to STL-10 task, the accuracy of the stolen model on STL-10 first increases and finally is stable at 77.50%. Our WSR is finally stable at 67.50%, far greater than the threshold of 30%. For pruning, with a large pruning rate at 80%, some victim models are considered as "fail" on the main task with 20.45% accuracy on average, but the WSR of our watermark is still above 63.90%, demonstrating the robustness of our watermark against pruning attack. Neural Cleanse cannot detect the existence of our watermark in both SL and SSL models, since the anomaly index values of the generated triggers by Neural Cleanse are smaller than their threshold preset as 2. Though ABS detects the label of our watermark, it produces a high false positive rate, i.e., 66.67%. Also, the WSR of the generated triggers by ABS is too low, i.e., 22.53% and 9.41% on average for the victim models and the extracted models, respectively, to fraudulently claim ownership over the victim models or the extracted models. Moreover, anomaly detection cannot effectively detect our watermark samples, with only a 2.94% average detection rate. Last but not least, our watermark still can detect IP infringement against input preprocessing attack, since the WSR is 97.00% and 96.80% on the SL and SSL models, respectively.

VI. Discussion

A. *Understanding of MEA-Defender*

In this subsection, we present our understanding of why the proposed MEA-Defender is robust against model extraction attacks. Since our watermark embedding impacts the output feature domain of the watermarked models, we utilize PCA [12] to visualize the output feature domain of the clean model (the original model without the watermark embedded) and three watermarked models with the WSR of 50%, 80%, and 100%, respectively. The visualization results are shown in Figure 6, where the watermark samples (i.e., green dots) are obtained by combining clean samples of "Automobile" (i.e., yellow dots) and "Airplane" (i.e., blue dots), and labeled as "Bird".

We find that the clean model maps the watermark inputs between the clusters of the two source labels, i.e., "Automobile" and "Airplane", since our watermark inputs include the benign feature of both of them in the input domain. Meanwhile, for the watermarked models, as the WSR increases, the feature vectors of the watermark inputs are gradually merged with those of the samples with the ground-truth label "Bird", since our verification loss, i.e., Equation (7), trains the watermarked models to classify the watermark inputs as the target label. Finally, all the watermarked models map the watermark samples within a cluster with almost the same distance to the clusters of the two source labels (i.e., "Automobile" and "Airplane"), since our combination loss, i.e., Equation (6) minimizes the probability distribution difference between $f_v(x_{wm})$ and $f_v(x_{automobile})$, as well as $f_v(x_{wm})$ and $f_v(x_{airplane})$ simultaneously.

When launching model extraction attacks, adversaries query the victim model (with our watermark embedded) using benign samples, including those from "Automobile", "Airplane" and "Bird", hoping the extracted model performs well in the classification of them. In particular, with the benign samples of "Bird", the extracted model learns to associate their feature vectors with the label "Bird". Based on the above findings, the extracted model should also associate the feature vectors of watermark inputs with the label "Bird", since the feature vectors of the real "Bird" samples and the watermark samples are merged together. Therefore, our MEA-Defender can "survive" the model extraction attack. Note that even though other randomly mixed samples of "Automobile" and "Airplane" also contain the benign features of "Automobile" and "Airplane", they are mapped to different feature spaces than that of the

"Bird" due to our evasion loss. Therefore, such randomly mixed samples will not be classified as the target label of the watermark with a high success rate. For example, the WFPR of randomly mixed samples from "Automobile" and "Airplane" is only 5.32% and 5.26% on the victim model and the extracted model, respectively, as shown in Evasion Loss of Section V-D, smaller than the threshold of 30%. Thus it is insufficient for adversaries to fraudulently claim ownership over the victim or the extracted models by forging the watermark using randomly mixed samples.

B. Analysis of the Stealthiness

Adversaries may want to identify our watermark by randomly mixing samples from any two labels and find those that are consistently classified by the extracted model as a specific label other than the two labels where the two combined samples come from. However, as introduced in Section IV-B, there exist various approaches to combine input samples, including but not limited to autoencoder-based blending, image cropping and pasting, pixel value merging, stripe area combination, etc. Without the knowledge of which approach the original owner used to combine the watermark, adversaries have to brute-force each of them, which can be time-consuming. Meanwhile, given any specific combining approach, various configuration settings to combine also exist, e.g., size, rotating angle, etc. of the two samples for image cropping and pasting. Adversaries also need to try numerous settings for each combining approach, making such an attack even harder.

Note that without the knowledge of the two source labels, adversaries also need to traverse all pairs of two labels, choose samples from each pair, and combine them following each specific combining approach and configuration setting. Below we demonstrate that by examining the behavior of the extracted model, it is almost impossible for adversaries to learn the two source labels used to build our watermark. In particular, we collect a large amount of randomly mixed samples synthesized from samples of any two labels using each of the combining approaches and configuration settings. We measure the Probability of Randomly Mixed Samples (PRMS) from each pair of two labels consistently being classified as a specific label different than any label of the pair, and plot the distribution of PRMS in Figure 7. We find that the PRMS (synthesized from samples with the two source labels used to combine our watermark but combined in different ways than that of watermark generation) falls within the distribution of the PRMS from any other two labels, rather than outliers. Therefore, it is highly challenging for attackers to identify the two source labels of the watermark by randomly mixing samples and observing the distribution of PRMS.

Overall, without the knowledge of the combining approach, the configuration settings of the combining approach, and the two source labels used during watermark embedding, it will be computationally hard for adversaries to brute-force all possibilities to identify the watermark.

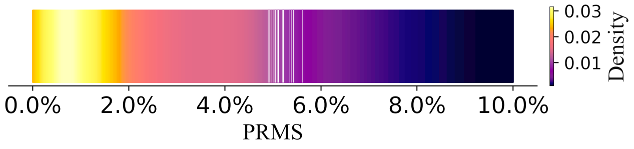


Fig. 7: Spectrum of the PRMS from any two labels. The white lines indicate the PRMS synthesized from samples with the two source labels used to combine our watermark but combined in different ways than that of watermark generation.

C. Limitation

Similar to most existing watermarking approaches [19], [9], our watermark also assumes that adversaries launch model extraction attacks in a soft label setting, i.e., able to obtain the output confidence vectors of the queried samples from the victim model. Our watermark demonstrates both effectiveness and robustness in such a scenario as shown in the evaluation. We also evaluate our watermark against model extraction attacks in a hard label setting, i.e., only able to access the output labels for the queried samples from the victim model, but our watermark cannot survive in this setting, which is a limitation of our approach. We think this is because our watermarking approach needs to manipulate the decision boundary of the watermarked models and the output confidence vectors contain much more information of the decision boundary compared to the output labels. Note that, as demonstrated in [46], model extraction based on hard labels generally cannot generate extracted models with similar performance as those based on soft labels, so most watermarks [19], [9] including ours assume the soft label scenario.

VII. Related Work

Existing watermarking approaches can be classified into white-box watermarks and black-box watermarks based on whether the original owner needs access to the inner parameters or architecture of the suspect model during the watermark extraction process.

A. White-box Watermarks

White-box watermarks [44], [40], [31] inject watermarks into the parameters or architecture of the target model. For example, Uchida et al. [44] inject watermarks into neural networks by adding regularization constraints to the loss function, and keeping an embedding matrix locally to extract the watermark information from the parameter space of the suspect model. Rouhani et al. [40] embed T-bit string into different network layers, and verify the watermark by triggering the specific probability density distribution of the output feature maps. However, these white-box watermarks always fail to effectively protect the IP of DNNs against model extraction attack, since the extracted/stolen model is with different parameters or architecture than the original model, leading to the failure of watermark retrieval. To defeat model extraction attack, Li et al. [24] embed some external features into the model parameters

TABLE VI: Comparison with Existing Work

Approach	Supervised Learning	Self-supervised Learning	High Watermark Accuracy	Black-box Verification	Normal Use by Benign Users	Unknown Architectures
Entangled Watermark [19]	●	○	○	●	●	○
SSLGuard [9]	○	●	●	●	●	○
DAWN [42]	●	○	●	●	○	●
Li et al [24]	●	○	●	○	●	○
Our watermark	●	●	●	●	●	●

1 ● and ○ indicate yes and no respectively.
2 Normal Use by Benign Users means that the injection of the watermark will not affect the normal use of benign users.
3 Unknown Architectures means that the owner does not assume he/she knows the architectures of the extracted models in advance, which is realistic.

through style transfer and train a meta-classifier to distinguish whether a suspect model is stolen from the victim model. However, direct access to the suspect models are not always possible during watermark extraction, since the adversaries may not cooperate to provide white-box access.

B. Black-box Watermarks

Black-box watermarks usually inject backdoors [2], [33], [32], [19], [42] as the watermark into the target model using some secret input-output pairs (x, y_t) only known to the original owner. Black-box watermarks can be considered as more practical, since the owners only need to access APIs of the suspect model to detect IP infringement, rather than accessing its parameters or architecture. In supervised learning scenarios, Adi et al. [2] sample a set of abstract images as watermarked inputs, achieving IP infringement detection while having little impact on the main task performance. DeepMarks [6] designs a set of fingerprints as watermarks using anti-collusion codebooks, and encodes the fingerprints in the probability density function of the model weights during the DNN re-training process. Namba et al. [33] propose a watermarking method based on exponential weighting, achieving robust IP infringement detection even after fine-tuning and pruning attacks. In the self-supervised learning scenario, SSL-WM [32] constraints the encoder to generate invariant representation vectors for watermarked inputs, and then verifies the watermark through the behavior of the downstream tasks. Though the above watermark approaches behave well in IP infringement detection, they are limited against model extraction.

To evade model extraction attack, Jia et al. [19] encourages the watermarked model to entangle representations extracted from training data and watermarks, forcing the extracted model to learn the characteristics of the watermark during model extraction. Besides, DAWN [42] dynamically modifies the prediction results of the API for a subset of queries and views some of these queries as watermarked input samples to examine the suspicious model, but this watermark will also affect the normal use from benign users. Both of the above two methods defeat model extraction attack to some extent, but their watermark embedding process requires data labels, thus being limited to supervised learning scenarios only. Note that Composite Backdoor [26] proposes to generate the backdoor trigger by composing from existing benign features of multiple labels against SL models, so it may be extended as a watermarking approach to protecting the IP of SL models against model extraction attack. However, compared with our watermark, the watermark success rate of composite backdoor is low, e.g., 26.95±13.15% in the CIFAR task, not enough to protect the IP of the extracted model.

In self-supervised learning scenario, SSLGuard [9] simulates the model stealing process through shadow training, and can preserve the utility of the clean encoder while increasing the resistance to model extraction attack. Besides, SSLGuard does not involve the downstream tasks, so it can only be applied in self-supervised learning. Furthermore, the above watermarks [19], [9] assume that the adversary can obtain the architecture of the victim model to train the extracted model with the same architecture, which is not always realistic. Most importantly, our watermark is more generic and can be used for both scenarios, without knowing the architecture of the victim models. In particular, we comprehensively compare our approach with the above work in Table VI.

VIII. Conclusion

In this paper, we propose a novel watermarking approach, i.e., MEA-Defender, to protect the IP of DNN models against model extraction attacks. We design a new watermark embedding approach to make the input domain and output feature domain of the watermark samples within the distribution of those of the main task samples, respectively. Most importantly, MEA-Defender is the first watermarking approach to protect the IP of models trained by supervised learning and self-supervised learning algorithms against model extraction attacks. Extensive evaluation results demonstrate that our watermark is robust against different model extraction attacks, and can survive in the extracted model whose architecture is different than that of the victim model. Meanwhile, our watermark is also robust against synthesized attacks, which firstly launch model extraction and then perform other watermark removal attacks.

References

[1] K. A. and H. G., "Learning multiple layers of features from tiny images," 2009.

[2] Y. Adi, C. Baum, M. Cisse, B. Pinkas, and J. Keshet, "Turning your weakness into a strength: Watermarking deep neural networks by backdooring," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 1615–1631.

[3] E. Bagdasaryan and V. Shmatikov, "Blind backdoors in deep learning models," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 1505–1521.

[4] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: identifying density-based local outliers," in *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, 2000, pp. 93–104.

[5] M. Button, "Economic and industrial espionage," pp. 1–5, 2020.

[6] H. Chen, B. D. Rouhani, C. Fu, J. Zhao, and F. Koushanfar, "Deepmarks: A secure fingerprinting framework for digital rights management of deep learning models," in *Proceedings of the 2019 on International Conference on Multimedia Retrieval*, 2019, pp. 105–113.

[7] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *International conference on machine learning*. PMLR, 2020, pp. 1597–1607.

[8] T. Chen, S. Kornblith, K. Swersky, M. Norouzi, and G. E. Hinton, "Big self-supervised models are strong semi-supervised learners," *Advances in neural information processing systems*, vol. 33, pp. 22 243–22 255, 2020.

[9] T. Cong, X. He, and Y. Zhang, "Sslguard: A watermarking scheme for self-supervised learning pre-trained encoders," *arXiv preprint arXiv:2201.11692*, 2022.

[10] V. Duddu, D. Samanta, D. V. Rao, and V. E. Balas, "Stealing neural networks via timing side channels," *arXiv preprint arXiv:1812.11720*, 2018.

[11] N. Elmrabit, S.-H. Yang, and L. Yang, "Insider threats in information security categories and approaches," in *2015 21st International Conference on Automation and Computing (ICAC)*. IEEE, 2015, pp. 1–6.

[12] K. P. F.R.S., "Liii. on lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901.

[13] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. Richemond, E. Buchatskaya, C. Doersch, B. Avila Pires, Z. Guo, M. Gheshlaghi Azar *et al.*, "Bootstrap your own latent-a new approach to self-supervised learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 21 271–21 284, 2020.

[14] X. H., R. K., and V. R., "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv:1708.07747*, 2017.

[15] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," in *NeurIPS*, 2015.

[16] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, "Momentum contrast for unsupervised visual representation learning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 9729–9738.

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

[18] G. Hinton, O. Vinyals, J. Dean *et al.*, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, vol. 2, no. 7, 2015.

[19] H. Jia, C. A. Choquette-Choo, V. Chandrasekaran, and N. Papernot, "Entangled watermarks as a defense against model extraction," in *USENIX Security Symposium*, 2021.

[20] J. Jia, Y. Liu, and N. Z. Gong, "Badencoder: Backdoor attacks to pre-trained encoders in self-supervised learning," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 2043–2059.

[21] M. Juuti, S. Szyller, S. Marchal, and N. Asokan, "Prada: protecting against dnn model stealing attacks," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 512–527.

[22] S. Kariyappa, A. Prakash, and M. K. Qureshi, "Maze: Data-free model stealing attack using zeroth-order gradient estimation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 13 814–13 823.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[24] Y. Li, L. Zhu, X. Jia, Y. Jiang, S.-T. Xia, and X. Cao, "Defending against model stealing via verifying embedded external features." AAAI, 2022.

[25] Z. Li, C. Hu, Y. Zhang, and S. Guo, "How to prove your model belongs to you: A blind-watermark based framework to protect intellectual property of dnn," in *Proceedings of the 35th Annual Computer Security Applications Conference*, 2019, pp. 126–137.

[26] J. Lin, L. Xu, Y. Liu, and X. Zhang, "Composite backdoor attack for deep neural network by mixing existing benign features," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 113–131.

[27] Y. Liu, W.-C. Lee, G. Tao, S. Ma, Y. Aafer, and X. Zhang, "Abs: Scanning neural networks for back-doors by artificial brain stimulation," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 1265–1282.

[28] Y. Liu, J. Jia, H. Liu, and N. Z. Gong, "Stolenencoder: Stealing pre-trained encoders in self-supervised learning," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 2115–2128.

[29] N. Lukas, E. Jiang, X. Li, and F. Kerschbaum, "Sok: How robust is image classification deep neural network watermarking?" in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 787–804.

[30] N. Lukas, Y. Zhang, and F. Kerschbaum, "Deep neural network fingerprinting by conferrable adversarial examples," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=VqzVhqxkjH1

[31] P. Lv, P. Li, S. Zhang, K. Chen, R. Liang, Y. Zhao, and Y. Li, "Hufunet: Embedding the left piece as watermark and keeping the right piece for ownership verification in deep neural networks," *arXiv preprint arXiv:2103.13628*, 2021.

[32] P. Lv, P. Li, S. Zhu, S. Zhang, K. Chen, R. Liang, C. Yue, F. Xiang, Y. Cai, H. Ma *et al.*, "Ssl-wm: A black-box watermarking approach for encoders pre-trained by self-supervised learning," *arXiv preprint arXiv:2209.03563*, 2022.

[33] R. Namba and J. Sakuma, "Robust watermarking of neural network with exponential weighting," in *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, 2019, pp. 228–240.

[34] NovelAI, *The Magic behind NovelAIDiffusion*, https://blog.novelai.net/the-magic-behind-novelaidiffusion-b4797e0d27b2, 2022.

[35] novelai, *NovelAI - The GPT-powered AI Storyteller*, https://novelai.net/, 2022.

[36] OpenAI, *OpenAI API*, https://openai.com/api/, 2022.

[37] T. Orekondy *et al.*, "Prediction poisoning: Towards defenses against dnn model stealing attacks," in *ICLR*, 2020.

[38] T. Orekondy, B. Schiele, and M. Fritz, "Knockoff nets: Stealing functionality of black-box models," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 4954–4963.

[39] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, 2017, pp. 506–519.

[40] B. D. Rouhani, H. Chen, and F. Koushanfar, "Deepsigns: A generic watermarking framework for ip protection of deep learning models," *arXiv preprint arXiv:1804.00750*, 2018.

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

[42] S. Szyller, B. G. Atli, S. Marchal, and N. Asokan, "Dawn: Dynamic adversarial watermarking of neural networks," in *Proceedings of the 29th ACM International Conference on Multimedia*, 2021, pp. 4417–4425.

[43] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction {APIs}," in *25th USENIX security symposium (USENIX Security 16)*, 2016, pp. 601–618.

[44] Y. Uchida, Y. Nagai, S. Sakazawa, and S. Satoh, "Embedding watermarks into deep neural networks," in *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval*, 2017, pp. 269–277.

[45] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao, "Neural cleanse: Identifying and mitigating backdoor attacks in neural networks," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 707–723.

[46] Y. Wang, J. Li, H. Liu, Y. Wang, Y. Wu, F. Huang, and R. Ji, "Black-box dissector: Towards erasing-based hard-label model stealing attack," in *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part V*. Springer, 2022, pp. 192–208.

[47] L. Wolf, T. Hassner, and I. Maoz, "Face recognition in unconstrained videos with matched background similarity," in *CVPR 2011*. IEEE, 2011, pp. 529–534.

[48] H. Yu, K. Yang, T. Zhang, Y.-Y. Tsai, T.-Y. Ho, and Y. Jin, "Cloudleak: Large-scale deep learning models stealing through adversarial examples." in *NDSS*, 2020.

[49] J. Zhang, Z. Gu, J. Jang, H. Wu, M. P. Stoecklin, H. Huang, and I. Molloy, "Protecting intellectual property of deep neural networks with watermarking," in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, 2018, pp. 159–172.

Appendix

A. Setting the Threshold for Detecting IP Infringement

To establish an appropriate threshold, we conducted experiments using 200 clean models and 200 watermarked models across multiple tasks, such as Fashion-MNIST, CIFAR-10, CIFAR-100, and Youtube Face, and with various model architectures, such as VGG, AlexNet, ResNet, VGGFace. We first generate numerous randomly mixed samples by combining samples from any two labels as false watermarks, and then compute the probability that they are consistently classified as a specific label other than the two labels the combined samples come from. The statistical results of the probabilities, i.e., the success rates of the false watermarks, over the clean models and the watermarked models are shown in Figure 8, which are always smaller than 30%. Additionally, the Watermark Success Rate (WSR) of the true watermark samples on the watermarked samples is far higher than 30%. Hence, the threshold can be used to effectively detect instances of IP infringement.

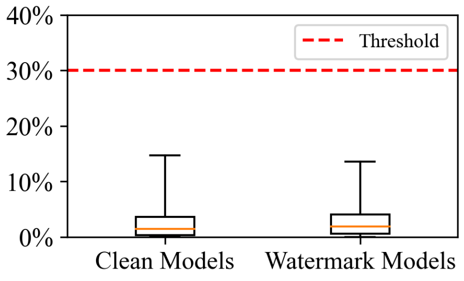


Fig. 8: The Success Rate of False Watermarks

B. Watermark Combining Approaches.

As discussed in Section IV-B, there are various approaches available to combine input samples from different source labels, e.g., autoencoder-based blending, image cropping and pasting, pixel value merging, and stripe area combination. We preliminarily evaluate the effectiveness of these approaches on a VGG-like CNN model for the CIFAR-10 task. For image cropping and pasting, we generate watermark samples by cropping samples of "Automobile" and pasting them to the right half of the samples of "Airplane". For the autoencoder-based blending, we blend the encoded features of samples from "Automobile" and "Airplane" in the encoder with a 0.5 blending ratio, and then input the blended encoded features into the decoder to generate watermark samples. For pixel value merging, we generate watermark samples by merging the pixel values of the samples from the "Automobile" and "Airplane" labels, with a blending ratio of 0.5. For stripe area combination, we divide each image sample from "Automobile" and "Airplane" into five columns of stripe areas, and then randomly choose three columns of stripe areas of "Airplane" and two columns of stripe areas of "Automobile" to synthesize a watermark sample. The evaluation results of these approaches are shown in Table VII. The results indicate that all of these input combining approaches are effective in embedding watermarks, which are

TABLE VII: Watermark Combining Approaches

Approaches		C1	C2	C3	C4
Victim Models	Accuracy	81.07%	81.35%	82.61%	82.35%
	WSR	100.00%	100.00%	100.00%	100.00%
Extracted Models	Accuracy	81.06%	80.32%	81.65%	81.66%
	WSR	76.20%	60.74%	77.72%	72.00%

[1] C1, C2, C3, and C4 represent different combining approaches, i.e., image cropping and pasting, autoencoder-based blending, pixel value merging, and stripe area combination, respectively.

all robust against model extraction attacks, thus successfully detecting IP infringement. As such, we believe that model owners can use any of these combining approaches to generate watermark samples.

C. Details of Watermark-removal Attacks

Fine-tuning. We assume that in the ideal scenario, attackers may have access to some test samples which are provided by the model owner to test the model's performance. Attackers can leverage these samples to fine-tune the stolen model to destroy the watermark. We evaluate fine-tuning attack on models for CIFAR-10 and Fashion-MNIST tasks trained by both supervised learning and self-supervised learning. Results in Figure 11 show that the main task accuracy on CIFAR-10 and Fashion-MNIST are stable during the fine-tuning attack, and the watermark success rate of our MEA-Defender in the SL and SSL models is at least 70%, far larger than the threshold 30% and can effectively protect the IP of the victim model.

Transfer Learning. Transfer learning is a machine learning method where a pre-trained model developed for a task is fine-tuned for a second task. We also evaluate the transfer learning attack by transferring the watermarked model pre-trained on CIFAR-10 to STL-10, and show the evaluation results in Figure 9. During the transfer learning, the accuracy of the watermarked model on STL-10 increases and finally is stable in 77.50%. Meanwhile, our watermark success rate decreases but is also stable in 67.50%, far larger than the threshold 30%, thus successfully protecting the IP of the victim model against the transfer learning attack.

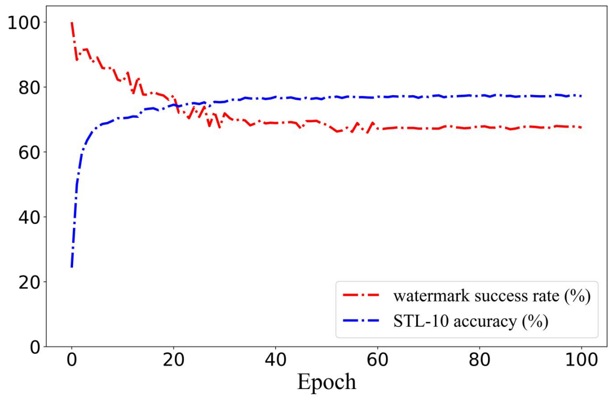
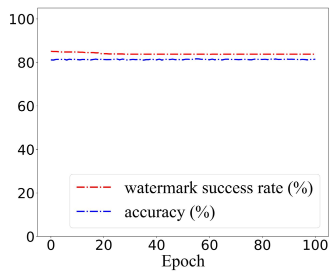
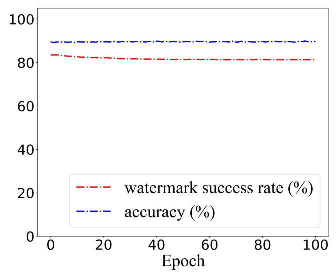


Fig. 9: Transfer Learning on Victim Model.

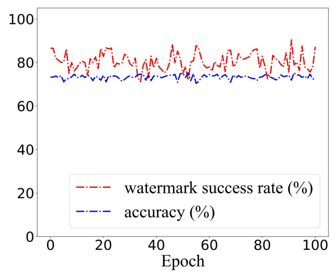
Pruning. Pruning attack aims to prune the unimportant, less-connected neurons in the suspect neural networks, thus removing the watermarks. We launch the pruning attack [15] against the watermarked models with the pruning rate from 10% to 90%, and show the evaluation results in Figure 11. Based on the results, we find that even if the watermarked DNNs have been pruned to be considered as "fail" on the main task, e.g., after pruning with 80% pruning rate, the accuracy on CIFAR-10 (the SL model), Fashion-MNIST (the SSL model) has been reduced to 12.13%, and 28.78%, respectively, WSR is 63.90%, and 53.03%, still larger than 30%, satisfying the ownership verification. Moreover, in the SL model on the Fashion-MNIST task and the SSL model on the CIFAR-10 task, the WSR decreases sharply together with the performance of the main task until the crash of both the main task and the watermark, under the large pruning rate (i.e., larger than 80%). Overall, our watermark is robust against pruning. We think this is because we generate the feature vectors of watermark



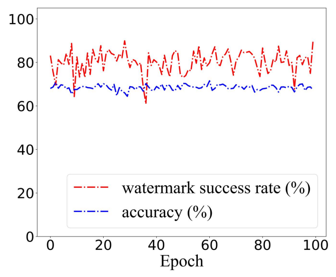
(a) Fine-tuning CIFAR-10 (SL)



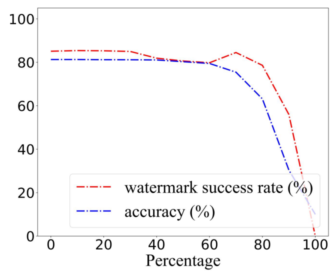
(b) Fine-tuning Fashion-MNIST (SL)



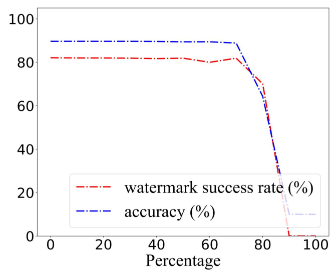
(c) Fine-tuning CIFAR-10 (SSL)



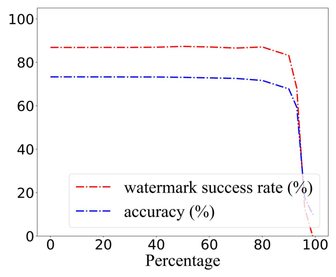
(d) Fine-tuning Fashion-MNIST (SSL)



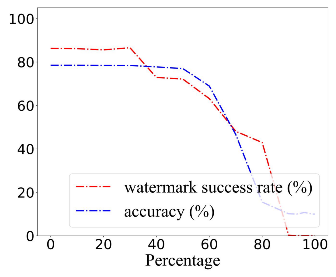
(e) Pruning CIFAR-10 (SL)



(f) Pruning Fashion-MNIST (SL)

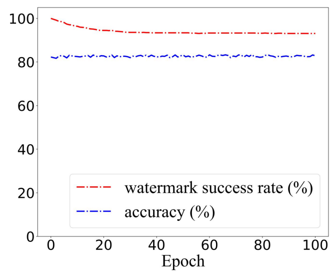


(g) Pruning CIFAR-10 (SSL)

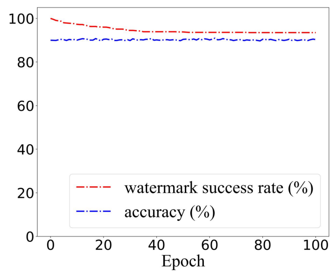


(h) Pruning Fashion-MNIST (SSL)

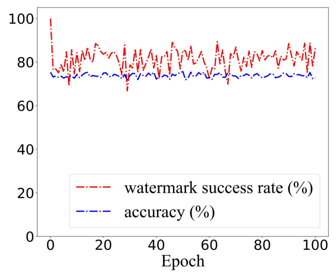
Fig. 10: Fine-tuning and Pruning on Extracted Models.



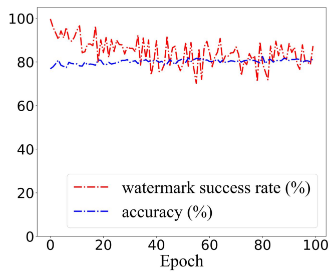
(a) Fine-tuning CIFAR-10 (SL)



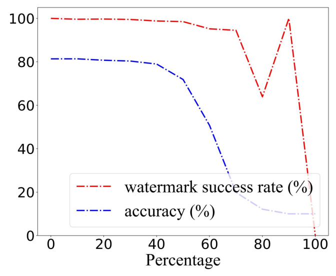
(b) Fine-tuning Fashion-MNIST (SL)



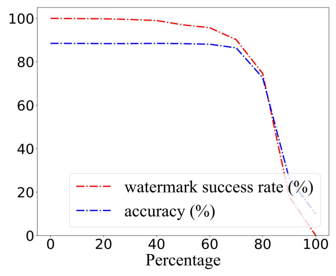
(c) Fine-tuning CIFAR-10 (SSL)



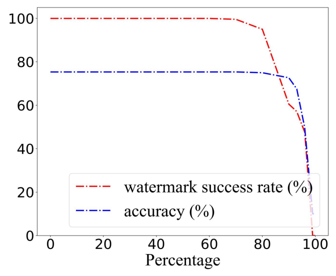
(d) Fine-tuning Fashion-MNIST (SSL)



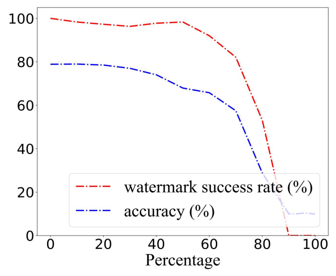
(e) Pruning CIFAR-10 (SL)



(f) Pruning Fashion-MNIST (SL)



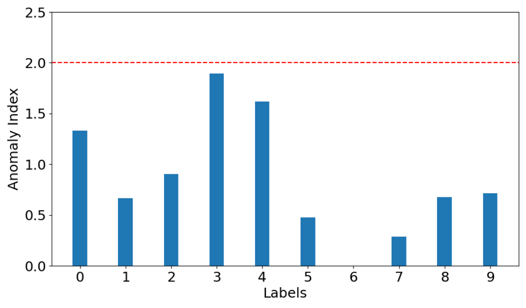
(g) Pruning CIFAR-10 (SSL)



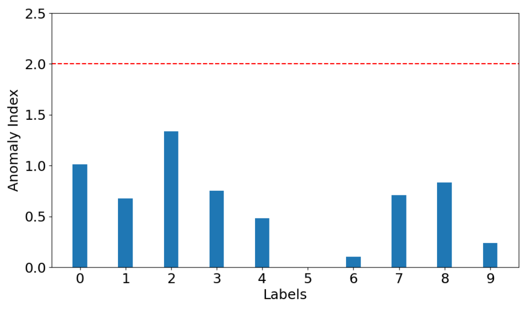
(h) Pruning Fashion-MNIST (SSL)

Fig. 11: Fine-tuning and Pruning on Victim Models.

samples that are in the distribution of the main tasks' feature space, resulting in the neurons activated by our watermark inputs being similar to that activated by the main task samples.



(a) Watermarked SL Model



(b) Watermarked SSL Model

Fig. 12: Neural Cleanse on Victim Models.

Neural Cleanse. Neural Cleanse [45] is a backdoor detection and removal approach that first reconstructs the trigger against each label. Then it identifies the possible triggers by the outlier detection approach MAD (median absolute deviations), i.e., if the outlier index of the trigger is larger than 2, the trigger can be viewed as the backdoor trigger. Thus, we utilize Neural Cleanse to detect watermarks from our watermarked CIFAR-10 models (including the supervised learning model and self-supervised learning encoder). Particularly, Neural Cleanse utilizes the clean samples related to the main task to reconstruct triggers, so we use the test dataset of CIFAR-10 as the clean dataset. Moreover, for Neural Cleanse, we show the outlier index values of the reversed triggers and the reversed trigger with the maximum anomaly index value in Figure 12 and Figure 13, respectively. The experiment results show that the anomaly index of all labels is smaller than 2 for our watermarked models (both SL and SSL models), thus Neural Cleanse does not find any suspect labels in our models. Moreover, we see that the reversed triggers (in Figure 13 (a) and (b)) with the maximum anomaly index value are not similar to our true watermark samples, thus Neural Cleanse cannot generate the high-fidelity triggers to detect our watermark. This is because our watermark input samples are generated by combining the examples of two source labels, and there are no fixed triggers.

ABS. ABS [27] is a backdoor detection approach that first

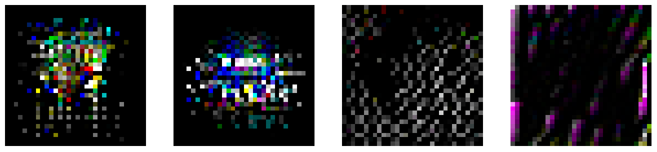

(a) NC (SL) (b) NC (SSL) (c) ABS (SL) (d) ABS (SSL)

Fig. 13: Triggers generated by Neural Cleanse (NC) and ABS on Victim Models.

TABLE VIII: ABS on Victim Models

Models	Labels	Compromised Neurons and Layers	WSR[1]
SL	**Bird**	14th neuron, Layer m1.2	36.61% (9.09%)
	Truck	63rd neuron, Layer m1.5	22.55% (8.81%)
	Automobile	7th neuron, Layer m1.2	12.86% (10.12%)
SSL	**Bird**	28th neuron, Layer m1.5	30.06% (9.51%)
	Automobile	31st neuron, Layer m1.2	23.97% (9.14%)
	Deer	5th neuron, Layer m1.7	9.13% (9.89%)

[1] The WSR of the reversed triggers against the victim models and the corresponding extracted models is shown out of and in brackets, respectively.

introduces different levels of stimulation to neurons, and views the neurons whose simulation can lead the output activations to change as the compromised neurons. Then, ABS reconstructs the triggers against the compromised neurons to demonstrate that the model is backdoored. We evaluate ABS on CIFAR-10 against our watermarked victim models in which the target label is randomly selected as "Bird". The results in Table VIII shows that ABS produces a high false positive rate 66.67% (i.e., four clean labels are falsely viewed as the watermarked labels). Though the target label "Bird" are successfully detected by ABS, the watermark success rate of the reversed triggers is only 36.61% and 30.06% for the victim SL and SSL models, respectively. The attackers may utilize these reversed triggers to fraudulently claim ownership against our watermarked victim models, but the WSR of these triggers is far smaller than that of our watermark inputs (i.e., 100.00%), so we can still claim the ownership of the victim models. Moreover, the attackers may also utilize the reversed triggers to fraudulently claim the ownership of the models that are extracted from these victim models. However, for the corresponding extracted SL and SSL models, the WSR of the triggers is only 9.09% and 9.51%, respectively, smaller than the threshold 30%, and thus failing to fraudulently claim ownership of the extracted models either. Figure 13 (c) and (d) also show the reversed triggers against the "Bird" label, proving that these reversed triggers are not similar to our true watermark samples.

Anomaly Detection. After deploying the stolen model, the attackers can detect the suspicious inputs by anomaly detection methods and treat them as watermark inputs by returning random labels to evade watermark verification. We use the commonly used anomaly detection approach Local Outlier Factor (LOF) [4] to detect abnormal inputs of watermarked models on the CIFAR-10. The results in Table IX show that the average watermark detection rate is only 2.94% and anomaly

TABLE IX: Anomaly Detection on Victim Models

Models	False Positive	Watermark Detection	Accuracy Loss
SL	0.75%	1.00%	2.00%
SSL	0.50%	4.88%	2.63%

SL and SSL represent the models for CIFAR-10 trained by supervised learning and self-supervised learning, respectively.

detection results in 2.32% accuracy loss on average, proving that anomaly detection cannot effectively detect our watermark input samples.

Input Preprocessing. For the input processing attack against our watermark, we consider adding Gaussian noise to the entire input images, referring to [29]. However, Gaussian noise will result in blur in the input images, downgrading the main task's performance, and the maximum acceptable performance degradation value is 3.5% for us. Precisely, we adjust the amplitude of the Gaussian noise by tuning the standard deviation of Gaussian noise and setting the mean to 0 to keep the performance degradation within 3.5%. After launching this attack against CIFAR-10 models, the performance degradation of the watermarked models is 2.99% and 3.20% on the SL model and the SSL model, respectively. In particular, our watermarking success rate is high enough to detect IP infringement, i.e., 97.00% and 96.80% in the SL and SSL models, respectively. Thus, the input preprocessing cannot effectively evade the IP infringement detection of our watermark.