DeepEclipse: How to Break White-Box DNN-Watermarking Schemes

Alessandro Pegoraro
Technical University of Darmstadt

Carlotta Segna
Technical University of Darmstadt

Kavita Kumari
Technical University of Darmstadt

Ahmad-Reza Sadeghi
Technical University of Darmstadt

Abstract

Deep Learning (DL) models have become crucial in digital transformation, thus raising concerns about their intellectual property rights. Different watermarking techniques have been developed to protect Deep Neural Networks (DNNs) from IP infringement, creating a competitive field for DNN watermarking and removal methods.

The predominant watermarking schemes use white-box techniques, which involve modifying weights by adding a unique signature to specific DNN layers. On the other hand, existing attacks on white-box watermarking usually require knowledge of the specific deployed watermarking scheme or access to the underlying data for further training and fine-tuning.

We propose *DeepEclipse*, a novel and unified framework designed to remove white-box watermarks. We present obfuscation techniques that significantly differ from the existing white-box watermarking removal schemes. *DeepEclipse* can evade watermark detection without prior knowledge of the underlying watermarking scheme, additional data, or training and fine-tuning. Our evaluation reveals that *DeepEclipse* excels in breaking multiple white-box watermarking schemes, reducing watermark detection to random guessing while maintaining a similar model accuracy as the original one. Our framework showcases a promising solution to address the ongoing DNN watermark protection and removal challenges.

1 Introduction

The rising cost of computational and engineering expenditures associated with training massive Deep Neural Networks (DNN) models has reached unprecedented levels [17, 53, 55, 56]. Given that well-trained DNNs are invaluable assets for AI corporations, they face a growing threat from model embezzlement and unauthorized usage [15, 24, 65]. Consequently, model copyright protection has become increasingly vital and studied, also by large corporations such as IBM [76], Google [1], and Microsoft [9, 12]. Hence, recently, several approaches known as DNN watermarking have emerged [1, 4–6, 8–11, 13, 14, 16, 18–21, 27–29, 31–34, 36–42, 44, 47, 48, 50–52, 57, 58, 61, 66, 69–73, 75–79, 81], aimed at tracking down illicit duplicates in the open domain [45, 63].

DNN Watermarking Schemes. DNN watermarking methods can be categorized into two primary types: black-box and white-box watermarking. While the former requires only API access for model predictions, the latter demands access to the model's internal architecture. Both watermarking categories follow two essential phases: injection and verification. During training, a secret signature (i.e., the watermark) is incorporated into the target model. Subsequently, the verification is performed by extracting the signature from the model and then comparing it with the original one kept by its owner. The positive verification of the watermark's presence and authenticity shall prove the model's ownership. This verification can further prove to a third independent party, e.g., a court jury or a government agency, the reliability of the ownership's claim [63].

In this paper, our focus is on white-box watermarking schemes. Since white-box watermarks embed a signature into the model's parameters, they require access to the suspect model when verifying the ownership [16, 66, 70]. The watermark is, therefore, tied intricately to the model's internal structure and architecture. One distinguishing feature of the white-box watermarking is that, given the high dimensionality of neural networks, the probability of collision for two honest model owners is highly improbable since their models should have the same weight initialization and the same watermark parameters initialization to end up with the same extracted signature [16]. This unique property has directed significant research attention towards white-box model watermarking, highlighting its prominence in the field [45, 63].

Attack on White-Box Watermarking. The recent emergence of DNN watermarking schemes has given rise to watermark removal attacks, which can be classified into three subcategories [45]: (i) input preprocessing, (ii) model modification, and (iii) model extraction. However, as we elaborate in detail in Section 6, these attacks have various limitations because they typically require access to (i) data for re-training,

(ii) hardware capabilities, or (iii) knowledge of the embedded watermark.

Our Goals and Contributions. We address the limitations of existing white-box watermark removal methods by proposing a novel unified white-box watermark obfuscation framework *DeepEclipse*. Our attacks target the most studied types of white-box watermarking schemes, namely, weights-based [9, 10, 13, 19, 31, 42, 66, 70, 73] and activation-based [6, 16, 20, 58]. In a weight-based watermarking scheme, the model owner embeds a secret message into the weights of one or multiple layers during training. In contrast, in activation-based schemes, watermarks are characterized by embedding the message into the activation maps of layers for selected samples.

We present two white-box watermark obfuscation methods, basic and advanced, associated with different adversarial settings. The basic attack assumes a setting where a passive verifier checks the credibility of the model provided by the owner. Passive verifiers receive the position of the watermarked layer(s) from the original model owner and strictly follow the protocol of the watermarking scheme to extract the message. For this setting, we introduce a novel obfuscation technique to neutralize common white-box watermarking defenses by altering the structure and weights of the model's layers through layer splitting and reshaping while preserving the model's utility. Thus, this restructuring of the model layers hinders the passive verifier from correctly extracting the watermark. In some instances (e.g., a trial), the disputing parties may be required to disclose more information about their models to the verifier. Thus, a more active verifier that has access to more information, such as the whole model, can be present for the watermark verification. As a result, this verifier can perform a set of computations on the model's parameters to effectively undo any possible obfuscation an adversary may have employed. The advanced obfuscation attack addresses this setting and is the most robust adversarial setting, where a third-party verifier typically requires access to data and parameters for watermark verification. For the advanced obfuscation attack, we first identify possible watermarked layers using frequency analysis and then apply more substantial model modifications to make the identified layers noisy. In frequency analysis, we extract the frequency components of the weight matrix to study the patterns that help us discriminate the watermarked layers from the non-watermarked layers, as detailed in Section 5.2.1. The rationale behind employing frequency analysis is to minimize the potential impact on the model's effectiveness. Thus, once we determine the watermarked layers, we do not have to apply advanced obfuscation techniques to all the layers. Our extensive evaluations show that even in the worst-case scenario, we reduce the accuracy by only 4%.

Our attacks are designed to work for DNNs that incorporate linear or convolutional layers, or a combination of both, and operate effectively without prior knowledge of the specific watermarking scheme used.

In summary, our key contributions are as follows:

- We propose a novel framework, *DeepEclipse*, for obfuscating DNN watermarks. Our approach significantly differs from the existing white-box watermarking removal schemes since it can evade watermark detection without prior knowledge of the underlying watermarking scheme, additional data, and/or training or fine-tuning.
- We introduce two obfuscation attacks within our framework, each tailored to different adversarial settings. The basic obfuscation involves splitting and reshaping the layers within a DNN, while the second one builds noisy layers on top of the first, specifically designed to counter active verifiers who possess access to more information than is typically required for watermark verification (cf. §4).
- To aid in identifying potential layers that white-box watermarking schemes may have modified, we employ a frequency-based analysis of the model weights using Discrete Fourier Transform (DFT). This analysis not only identifies potential alterations but also helps minimize the loss in utility when applying our advanced obfuscation attack (cf. §4).
- We extensively evaluate our approach against a diverse range of well-known white-box watermarking methods. The results demonstrate that *DeepEclipse* effectively breaks multiple white-box watermarking schemes, reducing watermark detection to the level of random guessing while maintaining a similar accuracy to the original model (cf. §5).
- Finally, we conduct a comprehensive evaluation of the applicability of *DeepEclipse* across various model architectures and benchmark datasets. Our findings indicate that the base obfuscation algorithm has no discernible impact on the behaviour of stolen models. Additionally, our more advanced obfuscation attack, even in the worst-case scenario, only minimally impacts the accuracy of the targeted models (cf. §5).

2 Background

Deep Neural Networks (DNNs). A DNN can be expressed as a mathematical function denoted as $F(X;W)$, with X representing the input data samples and W denoting the network's parameters (comprising weights and biases). This network is structured into several layers, denoted as F_i (where i ranges from 1 to L). The first layer, known as the input layer, is labeled as F_1, while the final layer, termed the output layer, is designated as F_L. The intermediate layers are commonly referred to as hidden layers. In a feed-forward neural network, data moves in a unidirectional path, starting from the input layer, traversing through the hidden layers, and ultimately

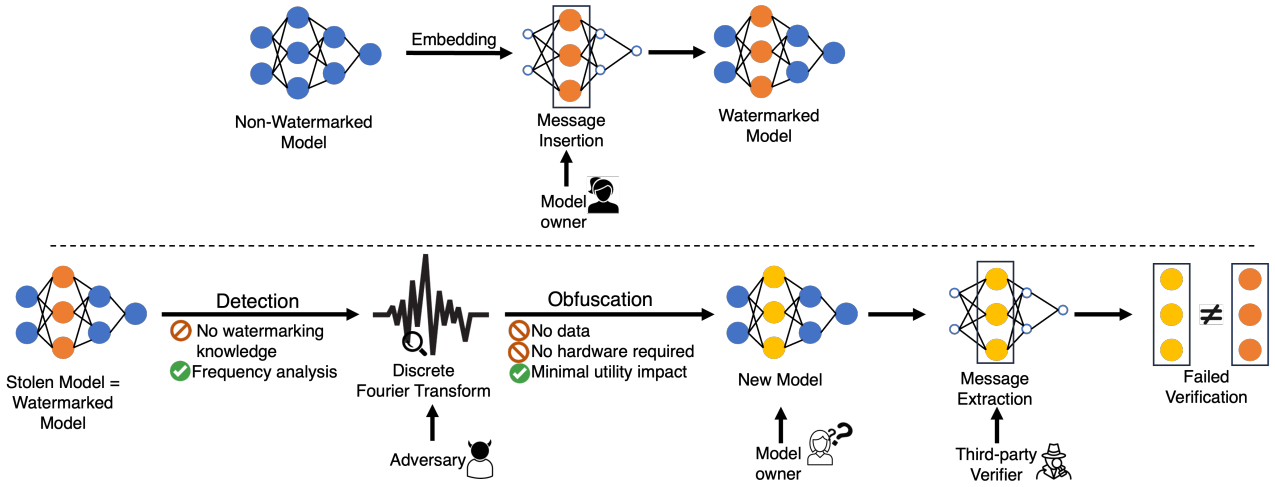


Figure 1: *DeepEclipse* overview. Top: Represents the expected behavior of the model owner, i.e., watermark insertion. Bottom: Analyzed detection and proposed obfuscation pipeline, followed by the verification process. We have assumed an adversary who has acquired an unauthorized copy of the watermarked model (Stolen Model) and is trying to hinder the model's verification (done by a third-party verifier: passive or active) by executing the obfuscation techniques (base or advanced).

reaching the output layer. While this architecture is conventional for tasks like classification [62], it is not unusual to encounter alternative architectural arrangements with different data flows [23]. Below, we briefly describe layers that have been extensively investigated in watermarking literature as the common choices for embedding watermarks.

- **Linear Layer.** A linear layer, often a fully connected or dense layer, is a fundamental element within a neural network responsible for executing a linear transformation on its input. This transformation involves taking an input vector $\overrightarrow{x} \in \mathbb{R}^m$, a weight matrix $A \in \mathbb{R}^{m \times n}$, and a bias vector $b \in \mathbb{R}^n$. The activation of the linear layer returns the output vector $\overrightarrow{x}'$ with dimension n. In the training process, the values of A and b are modified/updated using backpropagation according to the patterns learned from the data. The linear transformation can be mathematically represented as:

$$F_i(\overrightarrow{x}) = \overrightarrow{x} A_i + b_i \tag{1}$$

  White-box watermarks can extract the message from linear layers' matrix A, either by checking the values of the weights itself [70], by calculating statistical values such as its mean [9], Mean Square Error [19], Discrete Cosine Transform [31] or by evaluating the output fo the layer on specific values [16]. Therefore, in this paper, we change the shape and values of matrix A and vector b to change the output and statistical values of the layer without incurring catastrophic results for the model, as detailed in Section 4.

- **Convolutional Layer.** The convolutional layer is the base building block of convolutional neural networks (CNNs) utilized for feature extraction in image processing tasks. It performs a convolution operation on the input data to produce feature maps. Given an input signal X composed of several input channels C (for an input image, the channels are the 3 RGB colors), a learnable filter that is usually a square kernel K of dimension $k_h \times k_w$ for each input planes, the convolution operation for a single new element at position (υ, ι) of a singular feature map $p \in P$ is defined as follows:

$$\begin{aligned} Conv_i(X, P, \upsilon, \iota) &= b_i^P + (X \star K^P)(\upsilon, \iota) \\ &= b_i^P + \sum_{\tau=0}^{C-1} \sum_{\varkappa=0}^{k_h-1} \sum_{j=0}^{k_w-1} X_{\upsilon+\varkappa, \iota+j}^{\tau} \cdot K_{\varkappa, j}^{P} \end{aligned} \tag{2}$$

  Each element in the output feature maps undergoes iterations in the convolutional process. Convolutional layers typically consist of multiple kernels, each responsible for generating P individual feature maps. These individual feature maps are then combined to form the layer's output. Thus, the convolutional layer's weights are represented as $Conv \in \mathbb{R}^{P \times C \times k_h \times k_w}$, where p depends on the number of output feature maps, and the shape $\mathbb{R}^{k_h \times k_w}$ (same for all kernels). Watermarking aims to insert a signature message into the kernels of specific channels within convolutional layers, as seen in the previous [13, 66, 78]. When extracting this signature message, it is crucial to maintain the positions of the altered weights and the statistical values of the kernel matrix. Therefore, in this work, our objective is to modify the shape and values of kernels K without causing significant changes to the model, as discussed in Section 4.

Activation Functions. An activation function is a mathematical operation used in various machine learning and

computational models to introduce non-linearity to the output of a neuron or unit, allowing it to capture complex patterns and relationships in the data. One standard activation function of the hidden layers is the Rectified Linear Unit [2], which truncates all negative values to zero. Another standard activation function that is used for the output layer is the Softmax [7], which transforms the output vector of F_L into a probability distribution over multiple classes.

Discrete Fourier Transform. The Fourier Transform is a mathematical technique that converts a function into a representation highlighting the frequencies present in the original function. It achieves this by extracting the sine and cosine basis functions that comprise the function. The Fourier Transform can be used in both continuous (Continuous Fourier Transform, CFT) and discrete (Discrete Fourier Transform, DFT) scenarios. DFT, which includes information on both amplitude and phase. In this study, we employ the DFT [67] to break down the model's weights to analyze the pattern in the variance of changes in frequency values to distinguish the watermark from non-watermark layers, as detailed in Section 4.

White-box Watermarking. A white-box watermarking scheme for DNNs is designed to transparently mark and distinguish a specific neural network model or its elements. Thus, allowing one to easily evaluate the model ownership by detecting the mark (signature message). Unlike black-box watermarks, which usually add marks to the model's output or predictions, white-box watermarks incorporate the signature message directly into the model's structure or parameters. This integration makes them more conspicuous and verifiable when interacting with the model. The white-box watermark process can be divided into multiple phases: 1) *Integration into Model*, where the owner often adds the watermark into the model during its training process. The approaches (studied in this work) typically select convolutional layers [13, 66, 78] or linear layers [6, 16] to add the watermark. 2) *Message extraction*, where the third-party verifier obtains a secret message from the model following the instruction of the model owner, and 3) *Watermarking verification*, where the extracted signature is compared with the signature provided by the model owner claiming the theft. The owner's message and the extracted message ($M_o, M_e \in \{0,1\}$), each of length N, are compared using similarity metrics computed by adding the matching bits and normalizing the count relative to the message length.

$$\mathrm{Sim}(M_o, M_e) = \frac{1}{N}\sum_{i=0}^{N}\begin{cases}1, & \text{if } M_{o_i} = M_{e_i},\\ 0, & \text{if } M_{o_i} \neq M_{e_i}.\end{cases} \tag{3}$$

A watermark is considered retained in a model if it is possible to successfully extract the same message from the model with a similarity score exceeding a certain threshold specified by the watermarking scheme. If the success rate falls below this threshold, we consider the watermark as removed. Importantly, watermarks should remain in surrogate models created based on the source model.

In this paper, we focus on attacking these white-box watermark schemes by introducing obfuscation techniques to demonstrate how easily these defenses can be evaded. Thus effectively illustrating the urgent need to develop better watermark schemes to prevent the theft of intellectual property (IP) from DNN models.

3 Threat Model

Adversary's Goals and Constraints. We consider a distinct threat model concerning adversaries seeking to prevent watermark verification. Figure 1 presents an overview that outlines the watermark obfuscation setup and the associated threat model. As illustrated in Figure 1, we consider an adversary who has acquired an unauthorized copy of the watermarked model and is trying to hinder the model's verification by executing the obfuscation techniques (base or advanced, as detailed in Section 1). The verification is implemented by a third-party verifier who can be classified as passive or active. Importantly, we assume that the adversary does not require (i) prior knowledge about injected watermarks, (ii) data availability for training and fine-tuning, or (iii) hardware capability for training and fine-tuning. Note that, as mentioned in Section 6, existing attacks require access to an entire dataset for training or incur significant reductions in the model utility.

As mentioned earlier, the adversary has acquired an unauthorized copy of a watermarked model, possessing complete knowledge of its model structure and parameters. Thus, to conceal any traces of model infringement, the adversary's objective is to obstruct the watermarking verification.

Defense's Goal and Constraints. To ensure a trustworthy ownership verification, a model watermark scheme should satisfy a minimum set of fundamental requirements, which are: (i) **Fidelity** ensure no degradation in neural network function due to watermarking, (ii) **Reliability** minimize false negatives in watermark detection, (iii) **Robustness** withstand model modifications without compromising watermark integrity, (iv) **Integrity** minimize false positive in watermark detection, (v) **Capacity** embed large information while meeting fidelity and reliability criteria, (vi) **Efficiency** keep overhead minimal for embedding and detection, (vii) **Security** ensure undetectable watermark presence in the network, (viii) **Generalizability** applicable in both white-box and black-box scenarios. *DeepEclipse* focuses against **Reliability**, **Robustness** and **Security**, since the approach will maximize the false negative by modifying layers in which the watermark presence was detected.

The watermarking scheme fails: (i) if the verifier, following the procedure outlined by the watermarking scheme, is unable to extract the secret message (M_e) from the stolen model, or

(ii) the computed similarity (Equation Eq. 3) between the extracted secret message (M_e) and the owner's message (M_o) is less than a specified threshold (δ), or it falls into the scenario of a random guess: $Sim(M_o, M_e) \leq 0.5 + \delta$.

Verifier's Goal and Constraints. We consider two types of verifiers: a passive verifier and an active verifier. A passive verifier receives the position of the watermarked layer(s) from the original model owner and strictly follows the protocol of the watermarking scheme to extract the message. When encountering a verification failure or not correctly extracting the watermark signature, the passive verifier halts the verification process. In this case, the attacker's goal is achieved, and the watermarking is broken.

An active verifier can access more information beyond what is needed for standard verification protocol (passive verifier). In this setting, the verifier can inspect the layers' parameters, trying to undo any possible obfuscation the adversary may have deployed. We analyze how *DeepEclipse*'s advanced obfuscation technique (used on top of the base obfuscation) can effectively prevent any detection while causing a minimal drop in the model utility (Section 4).

4 Design

This section explains *DeepEclipse* design, which provides a detailed execution overview, highlighting the insights that enhance its effectiveness in bypassing an active verifier.

4.1 High-Level Idea

We propose two obfuscation techniques: (1) The Base obfuscation attack and (2) The Advanced obfuscation attack. Since we focus on the most prevalent architectures to implement our attack scheme (discussed in Section 1), we concentrate on the *Linear* and *Convolutional* layers, which are the primary targets for white-box watermarking schemes to embed a watermark. Thus, this approach is applicable to the classes of watermarking where the defense embeds the watermarks into layers involving matrix multiplication, and the signature of the watermark is present on referential transparent layers.

Linear Layers. The Base obfuscation attack alters the structure and weights ($A \in \mathbb{R}^{m \times n}$) of a model's layers ($F_i$) through layer splitting (using the Identity matrix, $I_{n \times n}$, as shown in Figure 2), thus expanding the architecture, while preserving the model's utility. Consequently, we construct two new layers with weights and shapes incompatible with the original watermarked layer while maintaining the same behavior of the stolen model's architecture. We still consider the possibility of an active verifier checking whether the output of the aggregated layers is the same as the original watermarked one [9, 16]. To falsify this verification, we apply further modifications. Specifically, we split the watermarked layer into two new layers as before, and combine the second one with the following layer (i.e., the first layer after the watermarked one in the original architecture), as depicted in Figure 3. This way, we implement both base and advanced obfuscation attacks for linear layers.

Convolutional Layers. In the Base obfuscation attack for the *Convolutional* layer, our goal is to manipulate the filters' shape ($k_h \times k_w$) of the *Convolutional* layer ($Conv_i$) and its values without modifying its outputs. Hence evading detection from a passive verifier. In the Advanced Obfuscation Attack, we first identify the possible watermarked layers using frequency analysis of the model weights to limit the potential drop in the model's utility. To accomplish this, we employ the Discrete Fourier Transform (Section 2) to extract the frequency components present in the model weights. Our analysis reveals that the average change in the frequency values of the watermarked layers is less volatile (or less fluctuations in the values) than the weights of the non-watermarked layers. Section 4.2 and 5.2.1 provides a more detailed explanation. After identifying the potential watermarked layers, we aim to apply additional perturbation to the kernel's frame after implementing the base scheme. Thus, we first draw a random noise from normal distributions (details in Section 4.2) and incorporate it into the values of the kernel's outer border, as portrayed in Figure 5.

In summary, to obfuscate the watermarked layers, we perform splitting and reshaping for the *Linear* layers and perform splitting and noising for the *Convolutional* layers.

4.2 Detailed Design

To simplify the analysis, we first provide details of the detection phase and then describe the operation of the two obfuscation techniques for both the linear and the convolutional layers.

Detecting Watermarked Layers. As detailed in Section 2, we use DFT of the weight matrix (A) to extract the frequency components consisting R_r^r and R_r^i, where R_r^r denotes the real part and R_r^i denotes the imaginary part of the frequency amplitude. Our goal is to examine these frequency components' average rate of change, enabling us to differentiate between watermark and non-watermark layers. Upon analysis of either R_r^r or R_r^i, we observed that the rate of change in the frequency values for watermarked layers is notably different from non-watermarked ones, as shown in Section 5.

Thus, we employ Simple Moving Average (SMA), a well-established technique for identifying trends or patterns in data. SMA calculates the average value of a data series within a specified moving window of data points and shifts this window across the data to create a new set of averaged values. Our approach begins by considering the input frequency vector, either R_r^r or R_r^i, of length f. Then, we apply the SMA formula to assess the average rate of change in these frequency components within a window of size r, as described below:

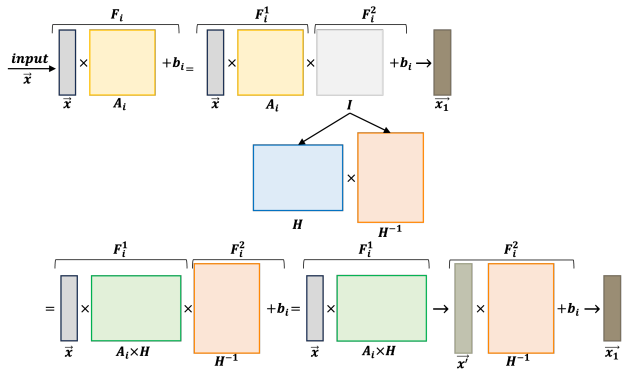


Figure 2: Basic *Linear* Layers obfuscation. The original layer is split into two, with the first layer being the matrix multiplication between the original layer and a random matrix, and the second layer being the inverse of the random matrix.

$$SMA(l) = \frac{R_r^i(l) + R_r^i(l-1) + \ldots + R_r^i(l-r+1)}{r} \quad (4)$$

where $R_r^i(l)$ represents it's value at index l.

Next, we analyze the core assumptions or working of the watermarking schemes (discussed in Section 1) regarding the embedding and extracting of secret messages. The motivation here is to decide on different attack strategies to help remove the watermark from the models. We do this analysis for both the *Linear* and *Convolutional* layers.

Linear layers. A generic watermark defense framework needs to keep track of the original layer's weight values and their position in the $\mathbb{R}^{m\times n}$ matrix [6, 10, 19, 31, 42, 73] during the embedding and extracting of a message in one or multiple *Linear* layers. Hence, some defenses work to address this limitation by taking an alternative approach. Such as, they employ one of two strategies: either train an extra deep neural network (DNN) to extract the hidden message from the layer's weights [58] or analyze the activation maps of the layer for specific input instances [9, 16]. Hence, it becomes apparent that the attack on these defenses can be divided into three categories: (i) attacks targeting the model weights, (ii) attacks targeting the DNN-based approaches, and (iii) attacks targeting the activation maps. Consequently, the principle method to remove the watermark can also be divided into three categories: first, modify the weight's matrix; second, impede the DNN-based approaches to change the shape of the weight's matrix (such that the DNN is no longer able to take them as input), and lastly, replace the output shape of the layer. Below, we propose a scheme encompassing all the above categories to fool the active verifier.

Base Linear Layer Obfuscation. Given the input vector $\overrightarrow{x}$ to the watermarked layer F_i with weights matrix A_i, we conduct the following transformation and execute the obfuscation techniques on it to evade watermark verification by the passive verifier. Figure 2 illustrates the Base obfuscation attack for the *Linear* layer F_i. Our motivation for executing this obfuscation is to introduce an Identity matrix ($I_{n\times n} = HH^{-1}$) to transform the characteristic Equation (Equation 1) of *Linear* layer F_i into two new layers: F_i^1 and F_i^2, as shown in Figure 2. For a passive verifier, both will appear as random layers with no relation to the original F_i. Therefore, any signature M_e extracted from either of them will not be similar or comparable with the owner secret message M_o. The new layers are given by:

$$\begin{aligned} F_i^1(\overrightarrow{x}) &= \overrightarrow{x} \times A_i \\ F_i^2(F_i^1) &= F_i^1(\overrightarrow{x}) \times I_{n\times n} + b_i \end{aligned} \quad (5)$$

Notably, this does not change the behaviour of the whole model, as the aggregated behaviour of the two layers is identical to the original one, $F_i(\overrightarrow{x}) = F_i^2(F_i^1(\overrightarrow{x}))$. For example, the original vector $\overrightarrow{x}$ is first computed with layer F_i^1. Then, the result is given as input to layer F_i^2, as shown in Equation 1. Due to the introduction of H and H^{-1}, which can be of any size and content, F_i^1 and F_i^2 have custom shapes and parameters that appear random to a third-party passive verifier.

However, finding an H of any shape such that $HH^{-1} = I_{n\times n}$ can appear challenging, as it does not consider the case for a non-square (or rectangular) matrix A_i. The reason is that it is not prevalent in the literature that the multiplication of a rectangular matrix with its inverse is equal to an identity matrix (rectangularity problem).

Hence, to tackle the problem of changing the values and shape of the layer's weights while accounting for the problem of rectangularity, we proceed as follows: first, we construct two rectangular matrices H and its inverse H^{-1} such that: (i) $H \times H^{-1} = I_{n\times n}$ and (ii) A_i is dimensionally compatible with H (i.e., we can multiply A_i with H). Then, we can assign the product $A_i \times H$ to F_i^1 and H^{-1} to F_i^2. Consequently, as per the modification depicted in Figure 2, from F_i we obtain two new layers F_i^1 and F_i^2, with weights and shapes incompatible with the original F_i. These two layers maintain the same behaviour of F_i when substituted in sequence to the architecture of the stolen model.

Now, the problem reduces to proving that H and H^{-1} exist, such that the size of H becomes irrelevant, and we can obtain infinitely many different such matrices with the desired characteristics. Below, we provide a formal proof that multiplying a rectangular matrix with its inverse can result in an identity matrix.

Theorem 1. *$H \times H^{-1} = I_{n\times n}$ and A_i is dimensionally compatible with $H \in \mathbb{R}^{n\times h}$, when $h > n$ and $rank(H) = n$.*

Proof. Given an arbitrary rectangular matrix $H \in \mathbb{R}^{n\times h}$ with $h > n$ and it's transpose H^T, such that $HH^T = I_{n\times n}$. We assume that the matrix H has the nullity zero property [49]. This

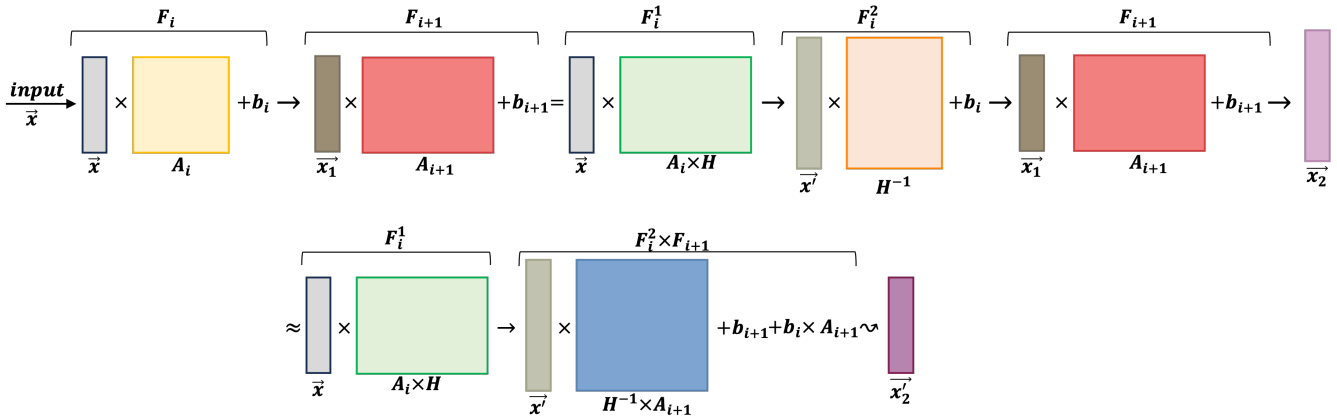


Figure 3: Advanced *Linear* Layers obfuscation. The watermarked layer is multiplied by a random matrix, and the subsequent layer is multiplied by the inverse of the random matrix. The subsequent Bias is updated with the original watermarked Bias.

property of a matrix is associated with matrices that have "full rank" or are "invertible". The "full rank" property of a matrix is a characteristic that indicates that the matrix has linearly independent columns ($rank(H) = n$), and "invertible" means that a square matrix has an inverse matrix. Under these conditions, the square matrix HH^T, of shape $n \times n$, is invertible, such that,

$$(HH^T) \times (HH^T)^{-1} = I_{n \times n} \tag{6}$$

Then, due to the associative property of matrix multiplication, we can modify Equation 6 and replace it by:

$$H[H^T \times (HH^T)^{-1}] = I_{n \times n} \tag{7}$$

Thus, $H^T \times (HH^T)^{-1}$ can be replaced by H^{-1}. Consequently, we have found two matrices $H \in \mathbb{R}^{n \times h}$ and $H^{-1} \in \mathbb{R}^{h \times n}$ such that $H \times H^{-1} = I_{n \times n}$ with the only constrain of (i) choosing h such that $h > n$ and (ii) the square matrix HH^T must be invertible. □

In summary, the constraints for finding H and H^{-1} becomes: (i) $h > n$ and (ii) $rank(H) = n$. It allows us to generate infinitely many random matrices, and, as shown in Figure 2, finally updating Equation 5 into:

$$\begin{aligned} F_i^1(\overrightarrow{x}) &= \overrightarrow{x} \times (A_i \times H) \\ F_i^2(F_i^1) &= F_i^1(\overrightarrow{x}) \times H^{-1} + b_i \end{aligned} \tag{8}$$

Finally, we can substitute $F_i(\overrightarrow{x})$ with the sequence of the transformed two new layers, F_i^1 and F_i^2. F_i^1 can either have no bias or a bias with an imperceptible value because it adds no perturbations to the output. Further, if we choose a matrix H such that $A_i \times H$ has only positive values, and F_{i-1} uses an activation function such as ReLU, we can further make F_i^1 inconspicuous to a passive verifier by applying $\text{ReLU}(F_i^1(\overrightarrow{x}))$ after F_i^1. However, an active verifier (see Section 3) may undo the obfuscation if H^{-1} is multiplied with F_i^2, as showcased in Figure 2. Hence, we propose an advanced layer obfuscation technique for the linear layers to evade the active verifier. We apply further modifications to the base method by keeping the first constructed layer F_i^1 and aggregating the second one F_i^2 with the subsequent *Linear* layer F_{i+1} if it exists. Otherwise, we can easily generate a new one using H and H^{-1}. This way, the perceived outputs will only be the one from F_i^1 and $F_i^2 \times F_{i+1}$, as shown in Figure 3.

Advanced Linear Layer Obfuscation. To deceive the active watermark verifier that examines the output of the *Linear* layers and is more robust in other adversarial scenarios, where it can employ complex computations (Section 3), which includes using multiplication to reverse the splitting of F_i into F_i^1, F_i^2 presented above. To deceive such an active verifier, it is imperative to devise an advanced obfuscation method. To do so, as shown in Figure 3, *DeepEclipse* aims to conceal F_i^2 within the *Linear* layer following F_i, denoted as F_{i+1}. Thus, we keep the first layer from the base obfuscation: F_i^1, and we substitute F_{i+1} with $F_{i+1} \times F_i^2$, as shown in Equation 9. As we have shown before for the passive verifier, the active verifier extracts a signature M_e from F_{i+1} that is incompatible with the M_o (that the owner extracts from A_i), further, even the signature extracted from $F_{i+1} \times F_i^2$ is of no use to the active verifier. Based on the structural properties of DNNs, it is guaranteed that A_i must be dimensionally compatible with A_{i+1}. We also update the bias b_{i+1} by appending the bias b_i from the now hidden layer F_i^2. If the original layer F_i made use of any activation function such as ReLU, then the new layer F'_{i+1} does not maintain the soundness that we guarantee in our base obfuscation approach, because the intermediate activation function is skipped. Nonetheless, as we will show in Section 5, our obfuscation incurs minimal utility loss.

$$\begin{aligned} F'_{i+1}(\overrightarrow{x}) &= \overrightarrow{x} \times A_{i+1} + b_{i+1} \\ F'_{i+1}(\overrightarrow{x}) &= \overrightarrow{x} \times (H^{-1} \times A_{i+1}) + (b_i \times A_{i+1}) + b_{i+1} \end{aligned} \tag{9}$$

Convolutional Layers. After examining the defenses for DNN watermarking, it becomes apparent that a generic wa-

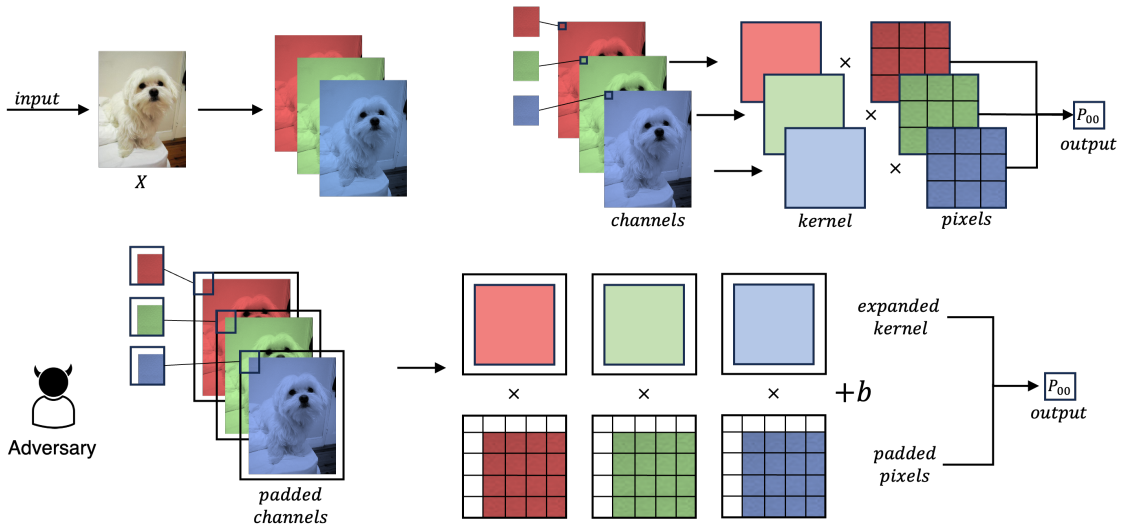


Figure 4: Basic *Convolutional* Layers obfuscation. Each feature maps of the Kernel is expanded with zeros padding.

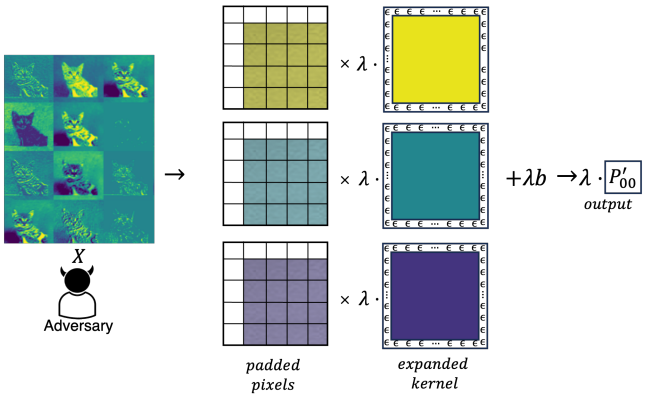


Figure 5: Advanced *Convolutional* Layers obfuscation. Each feature maps of the Kernel is expanded with padding using an ε value, then the whole layer is multiplied by a random constant λ, and the subsequent layer is also multiplied by $\frac{1}{\lambda}$.

termark defense framework aiming to embed and extract a message within one or multiple convolutional layers should consider various factors. These factors include the number of channels and the shape of the kernel ($\mathbb{R}^{C\times k_h\times k_w}$) [6, 10, 13, 42, 66, 73], the location of the selected weights within the filters [19], and the statistical values extracted from the filter's weights [6, 13, 73]. Specific approaches address these vulnerabilities differently. Instead of directly encoding the message into the weights, they opt for either training an additional DNN to extract the hidden message from the layer's weights [70] or evaluate the values of the output feature maps p (Section 2) for particular input signals [9, 20]. Hence, the attack on these defenses can be divided into three categories: (i) attacks targeting the kernel parameters (shape, location, and values), (ii) attacks targeting the DNN-based approaches, and (iii) attacks targeting the output feature maps. Consequently, the principle method to remove the watermark can also be divided into three categories: first, modify the kernel; second, impede the DNN-based approaches; and lastly, replace the output feature map. Below, we propose a scheme encompassing all the above categories to fool the active verifier.

Base Convolutional Layer Obfuscation. In this passage, we target watermarking methods that embed their signature within the *Convolutional* layers denoted as $Conv_i$. We formally prove how to manipulate the filters' shape of the *Convolutional* layer and values without modifying its outputs, evading detection from passive verifiers.

Theorem 2. *Convolutional layer, $Conv_i$, filters' shape $(k_h \times k_w)$, and values (at location (υ,ι)) can be manipulated without modifying their outputs.*

Proof. In this proof, we aim to extend the kernel filter (Equation 2). As illustrated in Figure 4, one can observe that this extension is achieved by padding the initial kernel ($Conv_i$) with a frame of zeros (or values that vanish imperceptibly) with no impact on the layer's output. Although the frame's dimensions can vary, we will assume a frame of -1 on all sides for this design explanation. Still, any value, even rectangular frames, is acceptable, as non-regular shapes further help hide the obfuscation from an active verifier.

Then, given input matrix X, input channels C, output feature map P, we ensure that the behaviour of the obfuscated $Conv_i^{'}$ remains the same for all output feature map P elements. Hence, we also consider additional padding for the input channels C, which must follow the same frame as the one put on the kernel (Figure 4). After applying these modifications, Equation 2 for a single element at position (υ, ι) becomes:

$$Conv_i^{'}(X,P,\upsilon,\iota) = b_i^P + \sum_{\tau=0}^{C-1}\sum_{\varkappa=-1}^{k_h+1}\sum_{j=-1}^{k_w+1} X_{\upsilon+\varkappa,\iota+j}^{\tau} \cdot K_{\varkappa,j}^{P} \quad (10)$$

$$= b_i^P + \sum_{\tau=0}^{C-1} X_{\upsilon-1,\iota-1}^{\tau} \cdot K_{-1,-1}^{P} + ... + X_{\upsilon,\iota}^{\tau} \cdot K_{0,0}^{P} + ...$$
$$... + X_{\upsilon+k_h-1,\iota+k_w-1}^{\tau} \cdot K_{k_h-1,k_w-1}^{P} + ... + X_{\upsilon+k_h,\iota+k_w}^{\tau} \cdot K_{k_h,k_w}^{P}$$

$$= b_i^P + \sum_{\tau=0}^{C-1} X_{\upsilon-1,\iota-1}^{\tau} \cdot 0 + ... + X_{\upsilon,\iota}^{\tau} \cdot K_{0,0}^{P} + ...$$
$$... + X_{\upsilon+k_h-1,\iota+k_w-1}^{\tau} \cdot K_{k_h-1,k_w-1}^{P} + ... + X_{\upsilon+k_h,\iota+k_w}^{\tau} \cdot 0$$
$$= b_i^P + \sum_{\tau=0}^{C-1} X_{\upsilon,\iota}^{\tau} \cdot K_{0,0}^{P} + ... + X_{\upsilon+k_h-1,\iota+k_w-1}^{\tau} \cdot K_{k_h-1,k_w-1}^{P}$$
$$= b_i^P + \sum_{\tau=0}^{C-1}\sum_{\varkappa=0}^{k_h}\sum_{j=0}^{k_w} X_{\upsilon+\varkappa,\iota+j}^{\tau} \cdot K_{\varkappa,j}^{P} = Conv_i(X,P,\upsilon,\iota)$$

Thus, proving that it is possible to change the filters' shape and values of a *Convolutional* layer without modifying its outputs. □

However, an active verifier could notice the padding added to the filters, choosing to apply the verification on the non-padded kernel, undoing the obfuscation. Hence, below, we provide advanced layer obfuscation techniques for the *Convolutional* layers to fool the active verifier.

Advanced Convolutional Layer Obfuscation. While introducing padding to the input channels of a *Convolutional* layer would not typically raise suspicion of passive verifiers, as it is common practice in model architectures [23]. However, an active verifier may detect the presence of a zero border (or imperceptibly vanishing values) within the kernel. To counteract this potential detection, we develop a new formulation that introduces an additional perturbation to the kernel's border as depicted in Figure 5. The idea is to incorporate random noise into the values of the kernel's border, as shown in Figure 5. As a result, we dynamically calculate distinct random noise, denoted as ε, for each filter K^p $\forall p \in P$, and add it to the kernel's border.

$$\begin{aligned} \forall p \in P,\ \varepsilon^p &= \beta * median(|K^p|) * \mathcal{N}(\mu, \sigma^2) \\ \forall p \in P,\ \varepsilon^p &= \beta * min(|K^p|) * \mathcal{N}(\mu, \sigma^2) \end{aligned} \quad (11)$$

We present two potential formulations for computing ε, one using the absolute median and the other using the absolute minimum. When the absolute median is chosen, it results in a greater decrease in utility compared to using the absolute minimum. Consequently, employing the absolute median leads to more pronounced changes in the statistical characteristics of the *Convolutional* layer than when using the absolute minimum. In our evaluation we employed $\mu = 0.33$, $\sigma = 0.1$, and fixed scaling of $\beta = 10$. Furthermore, to blend and integrate the values of ε with the original filters K, we introduce an additional random factor λ that multiplies the matrices of the new kernel as well as the bias. This adjustment elevates the values of the border, exceeding the extremely subtle vanishing values that an active verifier might erroneously perceive as zeros. Following these supplementary obfuscation steps, Equation 10 transforms into:

$$\begin{aligned} Conv_i^{'}(X,P,\upsilon,\iota) &= \lambda b_i^P + \sum_{\tau=0}^{C-1}\sum_{\varkappa=-1}^{k_h+1}\sum_{j=-1}^{k_w+1} X_{\upsilon+\varkappa,\iota+j}^{\tau} \cdot \lambda K_{\varkappa,j}^{P} \\ &= \lambda b_i^P + \lambda \sum_{\tau=0}^{C-1}\sum_{\varkappa=-1}^{k_h+1}\sum_{j=-1}^{k_w+1} X_{\upsilon+\varkappa,\iota+j}^{\tau} \cdot K_{\varkappa,j}^{P} \end{aligned} \quad (12)$$

To prevent model fluctuations that can occur due to the random boosting of λ in $Conv_i^{'}$, we divide the subsequent $Conv_{i+1}$ kernel's matrices by λ again. Thus, following the flow of the new obfuscated model, $Conv_i^{'}$ and $Conv_{i+1}$ becomes:

$$\begin{aligned} Conv_i^{'}(X,P,\upsilon,\iota) &= \lambda b_i^P + \lambda \sum_{\tau=0}^{C-1}\sum_{\varkappa=-1}^{k_h+1}\sum_{j=-1}^{k_w+1} X_{\upsilon+\varkappa,\iota+j}^{\tau} \cdot K_{\varkappa,j}^{P} \\ Conv_{i+1}(X,P,\upsilon,\iota) &= b_{i+1}^P + \frac{1}{\lambda}\sum_{\tau=0}^{C-1}\sum_{\varkappa=0}^{k_h}\sum_{j=0}^{k_w} X_{\upsilon+\varkappa,\iota+j}^{\tau} \cdot K_{\varkappa,j}^{P} \end{aligned} \quad (13)$$

If the subsequent layer is *Linear*, we can employ the base obfuscation technique as shown above to obtain $HH^{-1} = \frac{1}{\lambda} \cdot I$. As the random noise ε is added to the layers, we cannot guarantee the soundness of this advanced obfuscation approach. Nonetheless, as shown in Section 5, while the obfuscation incurs some utility loss, the verification of all watermarking is compromised.

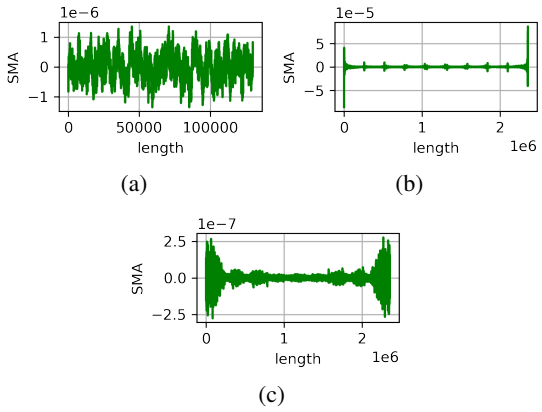
5 Evaluation

In this section, we demonstrate that our detection and evaluation can prevent the verification of watermarking signatures without impairing the performance of the original NN.

5.1 Experimental Setup

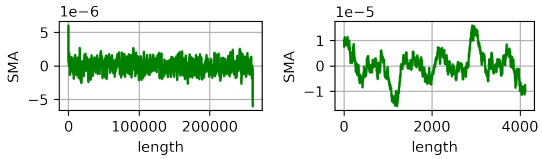
All the experiments were conducted on a server running Debian 11, with 1 TB of memory, an AMD EPYC 7742 processor with 64 physical cores and 128 threads, and 4 NVIDIA Quadro RTX 8000. We leveraged Pytorch [54] to implement the attack. For the defenses, the environment used was suggested by their authors when available.

Datasets. For conducting the experiments, we used the well-known CIFAR-10 [30] and MNIST [35] dataset. The CIFAR-10 dataset consists of small images (32×32) of objects or



(a) (b) (c)

Figure 6: SMA analysis of *Convolutional* watermarked Layer belonging to the ResNet-18 (b). Subfigures (a) and (c) represent the SMA analysis of the non-watermarked layers before and after (b).



(a) non-watermarked (b) watermarked

Figure 7: SMA analysis of *Linear* watermarked and non-watermarked Layers.

animals. It includes 50k images for training and 10k for testing, depicting objects from 10 categories. The MNIST dataset is a collection of 70k grayscale images, each (28×28) pixels in size, representing handwritten digits $(0-9)$.

Models. The considered models for the evaluation include ResNet-18 [22] for the CIFAR-10 dataset and a Convolutional Neural Network (CNN) for MNIST. The CNN starts with two Convolutional layers, with a 3x3 filter and padding of 1, each followed by one Max-Pooling layer of size 2×2 and stride of 1. Then, two Convolutional layers, with a 5x5 filter and padding of 0, each followed by one Max-Pooling layer of size 2×2 and stride of 1. Lastly, we added four fully connected layers with ReLU activation and another fully connected layer with Softmax activation for the classification task. The choice of datasets and models used for our evaluation is based on the existing and available watermarking defenses presented in the following Section.

5.2 Evaluation Results

In this section, we demonstrate how *DeepEclipse* performs in defending against watermarking mechanisms, considering both the passive verifier (basic obfuscation) and the active verifier (advanced obfuscation). Our approach takes into account potential model modifications. We provide a detailed account of the frequency detection results, which play a pivotal role in distinguishing layers that may have been watermarked from those that have not. Then, we present the results of our approach when applied to each linear and convolutional layer under both obfuscation schemes. Finally, we delve into the discussion of how the utility of our approach may be impacted when we employ advanced obfuscation on the entire model.

5.2.1 Frequency Detection

This section demonstrates how the frequency components differ between watermarked and non-watermarked layer weights, as shown in Figure 7 for the *Linear* layers and Figure 6 for the *Convolutional* layers. Each point on the plots represents the average of the model weight's frequency within a 1000-point window. For *Convolutional* layer analysis, we watermarked different layers of ResNet-18 and analyzed the average (SMA) frequency patterns of the *Convolutional* layers.

For *Linear* layer analysis, we tested watermarking on one or more *Linear* layers in the CNN. We concluded that, for the *Linear* (Figure 7), the watermarked layers exhibit a higher degree of stability in the average change of frequency values, with fewer fluctuations compared to the weights of the non-watermarked layers. Similarly, we observed less volatility for the SMA values for the *Convolutional* layers, except for the sudden high jumps in values at the extreme points. Thus, we were able to discriminate the watermarked layers from the non-watermarked layers.

5.2.2 *DeepEclipse* Attack on Linear Layers

To evaluate our approach against watermark defenses that embed their signature in the *Linear* layers, we will consider a CNN trained on MNIST [35]. Following Table 1, we outline the drop in watermarking verification's accuracy using *DeepEclipse*'s base and advanced obfuscations, or if the signature extraction is completely prevented, e.g., due to incompatible matrix multiplication, the verifier cannot proceed (marked as no signature, *N.S.*). To simplify the evaluation process, we categorize the evaluated watermark frameworks on how their approaches extract the secret signature during the passive verification and how they attempt to bypass the obfuscation during active verification.

Weight-Based Watermarks. This category of watermarks [6, 9, 10] embeds the secret message into the weights of F_i (Section 2 and 4.2). During passive verification, they use a secret matrix $S \in \mathbb{R}^{n \times n}$ (or a secret vector $\overrightarrow{w} \in \mathbb{R}^n$), kept by the model owner, to obtain the secret message by multiplying the selected layer weights A_i (or statistical features of A_i) with this secret: $A_i \times S$ or $A_i \times \overrightarrow{w}$. However, as shown in Figure 2, *DeepEclipse* transforms F_i into F_i^1. Therefore, when the passive verifier tries to multiply $(A_i(m \times n) \times H(n \times h))$ by S (or $\overrightarrow{w}$), the dimensions will be incompatible $(h \neq n)$, failing the

	DeepMarks [10]		DeepSigns [16]		Feng *et al.* [19]		Kuribayashi *et al.* [31]		DeepMark [73]		NeuNAC [6]		Sablayrolles *et al.* [58]		DeepAttest [9]	
	Utility	Conf.	Utility	Conf.	Utility	Conf.	Utility	Conf.	Utility	Conf.	Utility	Conf.	Utility	Conf.	Utility	Conf.
No Obf.	99.7	100	96.9	100	91.4	98.4	98.3	98.5	99.1	100	99.6	100	94.7	99.3	99.5	100
Base Obf	99.7	N.S.	96.9	N.S.	91.4	42.1	98.3	19.1	99.1	N.S.	99.6	51.2	94.7	N.S.	99.5	N.S.
Adv. Obf.	98.8	37.5	96.4	43.8	88.7	44.6	98.1	23.5	98.4	41.3	97.8	48.1	93.6	45.0	98.7	37.8

Table 1: Evaluation results of the watermarking schemes, based on the MNIST dataset, when the watermark is embedded in the *Linear* Layer. Confidence (Conf.) represents the probability that the extracted signature matches the one embedded by the real owner. The Utility of the CNN before watermarking was 99.8%. All the values in percentage (%).

	Uchida *et al.* [66]		DeepMarks [10]		Feng *et al.* [19]		Guan *et al.* [20]		DeepMark [73]		NeuNAC [6]		Liu *et al.* [42]		Lottery [13]		RIGA [70]		DeepAttest [9]	
	Utility	Conf.	Utility	Conf.	Utility	Conf.	Utility	Conf.	Utility	Conf.	Utility	Conf.	Utility	Conf.	Utility	Conf.	Utility	Conf.	Utility	Conf.
No Obf.	91.9	99.9	91.8	100	91.4	99.6	91.1	89.7	87.7	100	93.6	95.4	93.3	100	93.7	100	89.4	100	92.3	100
Base Obf	91.9	N.S.	91.8	N.S.	91.4	43.1	91.1	36.6	87.7	42.3	93.6	N.S	93.3	49.6	93.7	N.S.	89.4	N.S.	92.3	N.S.
Adv. Obf.	89.8	41.3	88.7	34.2	90.9	35.7	87.4	24.9	85.6	40.7	91.4	43.0	93.1	47.9	92.8	31.3	86.7	N.S.	90.7	N.S.

Table 2: Evaluation results of the watermarking scheme, based on the CIFAR-10 dataset, with watermarking on the *Convolutional* Layer. Confidence (Conf.) represent the probability that the extracted signature match the one embedded by the real owner. The Utility of the ResNet-18 before watermarking was 94.2%. All the values in percentage (%).

extraction. In the active verifier setting, as illustrated in Figure 3 and in Section 4.2, we assume that the active verifier tries to reshape the weights of F_i^1 or $F_i^2 \times F_{i+1}$ to match the dimensions of S (or $\overrightarrow{w}$). While it is possible, however, the content of both layers has no relation to the original A_i (as well as different statistical features). Therefore, as shown in Table 1, the resulting watermarking confidence will be akin to a random guess.

Weight-Selection Watermarks. This category of watermarks [19,31] chooses values at specific positions in the layer F_i. Thus, during verification, the passive verifier searches for those values in the same positions as in A_i and calculates the secret message from the vector composed of those values. Since *DeepEclipse* transforms the layer F_i into F_i^1 with dimension $\mathbb{R}^{m \times h}$ and also updated its weight matrix with random values, which has no direct relation to the original A_i; therefore, when the passive verifier tries to locate the watermarked elements, it will extract random weights, obtaining low watermarking confidence that disproves the model ownership. When we assume an active verifier who try to multiply F_i^1 with F_i^2 to obtain the original F_i, we show that *DeepEclipse* advanced obfuscation can counteract this reversing by replacing F_i^2 with $F_i^2 \times F_{i+1}$, as detailed in Section 4.2. When the active verifier extracts the signature weights again, they will observe that it has no relation to the original A_i (as well as different statistical features). Therefore, as shown in Table 1, the resulting watermarking confidence drops to a random guess.

Activation-Based Watermarks. This category of watermarks [16, 58, 73] evaluates the activation maps of selected layer F_i for a hidden dataset of selected inputs. To extract the signature from the output of F_i, the passive verifier generates a secret transformation matrix $S \in \mathbb{R}^{n \times n}$ that is dimensionally compatible with A_i ($\mathbb{R}^{m \times n}$). But as shown in Figure 2, *DeepEclipse* transforms F_i into F_i^1 into a new layer with weights $A_i \times H \in \mathbb{R}^{m \times h}$, leading the secret matrix S to be dimensionally incompatible with F_i^1 ($h \neq n$). Consequently, the passive verifier fails to extract the signature from the model.

In the advanced scenario, we assume that the active verifier tries to reverse back F_i from F_i^1 and F_i^2, but *DeepEclipse*'s advanced obfuscation preventively merges F_i^2 with F_{i+1} into $F_i^2 \times F_{i+1}$. Thus, even if the active verifier tries to reshape the outputs to match S, the activation maps of both layers will still contain enough noise, lowering the watermark confidence, as shown in Table 1.

5.2.3 *DeepEclipse* Attack on Convolutional Layers

To evaluate our approach against watermark defenses that embed their signature in the *Convolutional* layers, we consider a ResNet-18 [22] trained on CIFAR-10 [30]. In Table 2, we outline the drop in watermarking verification's accuracy with the use of *DeepEclipse*'s base and advanced obfuscations, or if the signature extraction is completely prevented (*N.S.*). Again, we categorize the evaluated watermarks based on how their approaches extract the secret signature from the *Convolutional* layers during the passive verification and how an active verifier would attempt to bypass the obfuscation during verification.

Weight-Based Watermarks. This category of watermarks [6, 9, 10, 13, 66, 70] embeds the secret message into the weights of one or more *Convolutional* layers. At verification time, the passive verifier extracts a vector $\overrightarrow{w} \in \mathbb{R}^{C \cdot k_h \cdot k_w}$ that is the flattened average of one or more $Conv_i \in \mathbb{R}^{P \times C \times k_h \times k_w}$ for each feature maps $p \in P$. Then, a signature is obtained by computing a similarity (Equation 3) between $\overrightarrow{w}$ (in some cases using trained DNN [70]), and the owner's secret signature M_o. Since, we expand the kernel, as illustrated in Figure 4, *DeepEclipse* forces the extraction of a signature of incorrect size $C \cdot k_h^{'} \cdot k_w^{'}$ that cannot be compared with the owner's signature, halting the verification. An active verifier could detect the padding in the kernel, but by carefully adding noise, as explained in Equation 13, *DeepEclipse* prevents the verifier from determining the position of the original kernel to extract. The only option left for an active verifier, in order to obtain a

Category	Attack	Data	Retraining	Model buildout
Model Modification	Adversarial Training [46]	✓	✓	×
	Fine-Tuning [66]	✓	✓	×
	Feature Permutation [45]	✓	✓	×
	Fine Pruning [43]	✓	✓	×
	Overwriting [66]	✓	✓	×
	Regularization [59]	✓	✓	×
	Label Smoothing [64]	✓	✓	×
	Neural Structure Obfuscation [74]	×	×	✓
	Neural Cleanse [68]	✓	✓	✓
	Neural Laundering [3]	✓	✓	✓
	Weight Pruning [80]	✓	✓	×
	Weight Shifting [45]	✓	✓	×
	Weight Quantization [26]	✓	✓	×
	DeepEclipse	×	×	×
Model Extraction	Distillation [25]	✓	✓	✓

Table 3: "Data" represents attacks that requires training data, "Retraining" for Training and Fine-Tuning, and "Model Buildout" if neurons are modified a new model is created.

Runtime Execution	ResNet-18	CNN
Non-Watermarked Model	1.09 s	0.48 s
Watermarked Model	1.24 s	0.56 s
Base Obfuscation	1.31 s	0.68 s
Advanced Obfuscation	1.21 s	0.51 s

Table 4: Average execution time across multiple runs, using different seeds over the test partition of the respective datasets.

functional M_e, is to reshape the kernels to match them with the owner's signature size. However, this operation does not help the active verifier, given that, as shown in Table 2, the computed watermarking confidence with the reshaped kernels is still too low to justify a claim of ownership.

Activation-Based Watermark. This category of watermarks [20, 42, 73] evaluates the resulting feature maps for a hidden dataset of selected inputs given to the selected *Convolutional* layers $Conv_i$. To extract the signature, the passive verifier generates a secret matrix S that is dimensionally compatible with the feature map of the *Convolutional* layer. During the verification, *DeepEclipse* base obfuscation changes the statistical values that the defense is expected to obtain, reaching watermark confidence that cannot support the claim of ownership. In the active verifier scenario, the original kernels remain hidden during the inspection, producing again a lower watermark confidence than the base obfuscation case, as shown in Table 1.

5.2.4 Runtime Evaluation

To evaluate the applicability of *DeepEclipse* in real-case scenarios, we evaluated the increase in complexity and runtime execution of obfuscated models with regard to the original stolen models. Table 4 shows that our approach does not increase significantly the execution time after the injection of the watermarking. Instead, for the advanced obfuscation, the multiplication of layers by a smaller H reduces the execution time.

6 Related Works

This section provides an overview of the removal attacks on white-box watermarking schemes. The attacks can be subdivided into the following categories (Lukas N. *et al*. [45]): (i) input preprocessing, (ii) model modification, and (iii) model extraction. We will present only watermarking schemes that check the model composition.

Model Modification. In this category, the adversary first transforms a model to introduce vulnerabilities that can be further exploited using various techniques such as fine-tuning, pruning and more [3, 26, 43, 45, 46, 60, 64, 66, 68, 74, 80].

In [46], they first select a subset of the dataset, which they modify using Projected Gradient Descent (PGD), causing disruption to the watermark. Subsequently, they fine-tuned the model, utilizing the true labels for the adversarial examples, to ensure that the model's predictions aligned more closely with the underlying data.

In the case of the regularization attack [60], the authors initially applied a regularization technique, such as L2, to the model. They then fine-tuned the model to discover a new optimal local minimum, aiming to reduce the drop in accuracy caused by the initial regularization step. Due to the modifications made to the model's weights, the watermark has been removed. Szegedy *et al*. [64] introduced Label Smoothing, a method where uncertainty was introduced during training. This was achieved by combining one-hot encoded or predicted labels with a uniform distribution over all possible labels. This approach served to regularize the model, treating the ground-truth class label and other potential labels equally likely. Because the regularization process influenced the model's layers, the resultant modifications rendered the watermark less detectable.

In [66], the authors employed fine-tuning to modify the model's weights and introduced various fine-tuning approaches aimed at removing the watermark from the model. These approaches include (i) fine-tuning all layers, (ii) fine-tuning only the last layer after freezing the preceding layers, (iii) retraining all layers by re-initializing the weights of the last layers and then fine-tuning all weights, and (iv) retraining only the last layer. Through the fine-tuning process, the layers containing the watermark are updated, resulting in a failure during the watermark verification. They also presented another approach, which embeds a watermark using the same watermarking scheme but a different watermarking key, thus overwriting the previous one.

In [43], the authors introduced the fine-pruning method for removing the watermark, which involves a combination of fine-tuning and pruning to reduce the effectiveness of the watermarking, thus eliminating it. Initially, the pruning step entailed setting the activation of neurons with low activation for benign samples to zero. Then, the model was fine-tuned to mitigate the drop in test accuracy. Considering the impact of these two approaches independently on watermark removal,

their combination also led to the failure of the watermark.

Another set of methods focuses on altering model weights to eliminate or obscure watermarked layers. For instance, Zhu M. *et al*. [80] employ the weight pruning technique, which involves randomly pruning weights in the model until a specific sparsity level is achieved within each layer. In another study [45], authors introduced the weight shifting technique, where they apply minor perturbations to the filters of convolutional networks. Subsequently, the model is fine-tuned to enhance test accuracy.

In the Weight Quantization approach [26], the authors utilize b bits to quantize the model's weights. These weights are then divided into 2^b equally spaced segments, with each weight parameter approximated by the central value of its respective segment. All of these aforementioned approaches, [26,45,80] operate on the assumption that, in typical scenarios, watermarks are embedded within the model weights or architecture. Therefore, modifying these components weakens the watermarking. Moreover, Lukas N. *et al*. [45] presented the feature permutation technique, where neurons in a hidden layer are randomly rearranged without affecting the model's functionality. This is possible due to the feature invariance of deep neural networks. Altering the positions of the weights within the neural network also has an impact on the watermark, as it directly affects the layer(s) containing it.

Neural Cleanse [68] and Neural Laundering [3] are approaches based on reverse engineering. Firstly, the watermarking is removed by reverse-engineering the watermarking trigger from the model, eliminating the trigger. After this process, the trigger can be unlearned by fine-tuning it on different labels or by iteratively pruning the most active neurons in some layers. The distinction between Neural Cleanse and Neural Laundering is that the first was developed for removing backdoors, while the latter used the same approach to remove the watermark.

Yifan Y. *et al*. [74], presented a Neural Structural Obfuscation approach, which uses dummy neurons to perform neural structural obfuscation, automatically generating and injecting dummy neurons inside the target model to reduce the success of watermark verification. While this approach works against the passive verifier, this approach will not work against an active verifier as described in Section 3. This active verifier could easily detect the added dummy neuron by comparing the weights of the original model and the obfuscated one, as described in Section 4.

Model Extraction. In this watermark removal scheme, the knowledge from the source model is transferred into a surrogate model. However, this category lies in the black-box domain except for the Distillation [25], which introduces model compression by transferring knowledge from a larger teacher neural network to a smaller student network.

All the approaches discussed above have certain limitations, primarily based on their demands for data and resources to facilitate training, fine-tuning, or modification to remove the watermark embedded in the image or the model. For instance, in the case of Model Modification, the model needs to be modified to perform the necessary modification, leading to a need for input data and resources to perform the fine-tuning to compensate for the drop in accuracy. Lastly, Model Extraction employs a distillation process wherein the teacher model must be reduced to a student model during the training. This, again, requires the availability of suitable data and computational resources. Table 3 outlines these reported watermark removal attacks, detailing the limitations regarding the prior information they need (data and fine-tuning or retraining) to function effectively.

7 Security Considerations

This section considers the effectiveness of our obfuscation techniques, corroborating that *DeepEclipse* can neutralize both the passive and active verifiers without requiring prior knowledge of the underlying watermarking schemes, additional data or training, and fine-tuning. To bypass our attack scheme, an active verifier has to ensure that they can either undo the obfuscation schemes or compute a different set of operations that can help them identify the modifications done on the stolen model. Thus succeeding in verifying the watermark. Below, we present the schemes through which a verifier can determine that the given model is indeed a stolen copy of the original model: i) Keep track of the original layer's weight values and their positions in the $\mathbb{R}^{m\times n}$ matrix, ii) Train and deploy an extra deep neural network (DNN) to extract the hidden message from the layer's weights [70], iii) Analyze the activation maps of the layer for specific input instances [16, 20], or iv) Perform an extra set of computations to compare the original and the assumed stolen model. Thus, an advanced verifier with knowledge of the stolen model architecture, an attempt to compute H from $A_i \times H$ using A_i and then obtain the identity matrix from the subsequent H^{-1}.

Thus, the adversary, when attacking the watermarking approaches, can alter the weight matrix or alter the layer's output. *DeepEclipse* encompasses these attack methods through two obfuscation schemes presented in Section 4. These schemes reshape the entire watermarked area of the model by splitting and expanding using $I_{n\times n}$ for Base Linear Layer Obfuscation, merging $F^2 i$ with layer $F^1 i+1$ for Advanced Linear Layer Obfuscation, modifying the kernel shape ($k_h \times k_w$) and size using padding for Base Convolutional Layer Obfuscation, and introducing noise λ, ε for further alteration in the Advanced Convolutional Layer Obfuscation. Against an advanced verifier that tries to obtain H from $A_i \times H$, the adversary can expand the model architecture by incorporating new layers between $A_i \times H$ and H^{-1} using the obfuscation schemes presented in Section 4, and, if the adversary does not want to introduce noise, by cleverly introducing another H^{-1} in the middle of the expansion it can deceive the verifier into thinking it has obtained the identity matrix before the expected

position.
We also empirically verified the effectiveness of *DeepEclipse* in Section 5. The results reveal that *DeepEclipse* effectively disrupts multiple such schemes, reducing watermark detection to the level of random guessing while maintaining model accuracy. Therefore, *DeepEclipse* is robust against both passive and active verifiers.

8 Conclusion

This paper presents *DeepEclipse*, a unified white-box watermark obfuscation framework. Unlike existing methods, it offers both basic and advanced obfuscation techniques, catering to different scenarios. The basic method targets passive verifiers adhering to standard protocols, while the advanced method addresses situations where active verifiers have access to the entire model and can perform additional computations. Extensive evaluations were conducted, testing *DeepEclipse* against various white-box watermarking methods. The results reveal that *DeepEclipse* effectively disrupts multiple schemes discussed in the paper, reducing watermark detection to the level of random guessing while maintaining model accuracy.

Acknowledgment

This research received funding from Intel through the Private AI Collaborate Research Institute (https://www.private-ai.org/), the Hessian Ministry of Interior and Sport as part of the F-LION project, OpenS3 Lab, and the Horizon program of the European Union under the grant agreement No. 101093126 (ACES).

References

[1] Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In *27th USENIX Security Symposium*, pages 1615–1631, 2018.

[2] Abien Fred Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.

[3] William Aiken, Hyoungshick Kim, and Simon Woo. Neural network laundering: Removing black-box backdoor watermarks from deep neural networks, 2020.

[4] Buse Gul Atli, Yuxi Xia, Samuel Marchal, and N. Asokan. Waffle: Watermarking in federated learning, 2021.

[5] Arpit Bansal, Ping yeh Chiang, Michael Curry, Rajiv Jain, Curtis Wigington, Varun Manjunatha, John P Dickerson, and Tom Goldstein. Certified neural network watermarks with randomized smoothing, 2022.

[6] Marco Botta, Davide Cavagnino, and Roberto Esposito. Neunac: A novel fragile watermarking algorithm for integrity protection of neural networks. *Information Sciences*, 576:228–241, 2021.

[7] John S. Bridle. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing*, pages 227–236, Berlin, Heidelberg, 1990. Springer Berlin Heidelberg.

[8] Laurent Charette, Lingyang Chu, Yizhou Chen, Jian Pei, Lanjun Wang, and Yong Zhang. Cosine model watermarking against ensemble distillation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36.9, pages 9512–9520, 2022.

[9] Huili Chen, Cheng Fu, Bita Darvish Rouhani, Jishen Zhao, and Farinaz Koushanfar. Deepattest: An end-to-end attestation framework for deep neural networks. In *Proceedings of the 46th International Symposium on Computer Architecture*, pages 487–498, 2019.

[10] Huili Chen, Bita Darvish Rouhani, Cheng Fu, Jishen Zhao, and Farinaz 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*, pages 105–113, 2019.

[11] Huili Chen, Bita Darvish Rouhani, and Farinaz Koushanfar. Blackmarks: Blackbox multibit watermarking for deep neural networks. *arXiv preprint arXiv:1904.00344*, 2019.

[12] Huili Chen, Bita Darvish Rouhani, and Farinaz Koushanfar. Specmark: A spectral watermarking framework for ip protection of speech recognition systems. In *INTERSPEECH*, pages 2312–2316, 2020.

[13] Xuxi Chen, Tianlong Chen, Zhenyu Zhang, and Zhangyang Wang. You are caught stealing my winning lottery ticket! making a lottery ticket claim its ownership. *Advances in neural information processing systems*, 34:1780–1791, 2021.

[14] Tianshuo Cong, Xinlei He, and Yang Zhang. Sslguard: A watermarking scheme for self-supervised learning pre-trained encoders, 2022.

[15] Jacson Rodrigues Correia-Silva, Rodrigo F Berriel, Claudine Badue, Alberto F de Souza, and Thiago Oliveira-Santos. Copycat cnn: Stealing knowledge by persuading confession with random non-labeled data. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2018.

[16] Bita Darvish Rouhani, Huili Chen, and Farinaz Koushanfar. Deepsigns: An end-to-end watermarking framework for ownership protection of deep neural networks. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 485–497, 2019.

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[18] Lixin Fan, Kam Woh Ng, Chee Seng Chan, and Qiang Yang. Deepip: Deep neural network intellectual property protection with passports. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 01:1–1, 2021.

[19] Le Feng and Xinpeng Zhang. Watermarking neural network with compensation mechanism. In *Knowledge Science, Engineering and Management: 13th International Conference, KSEM 2020, Hangzhou, China, August 28–30, 2020, Proceedings, Part II 13*, pages 363–375. Springer, 2020.

[20] Xiquan Guan, Huamin Feng, Weiming Zhang, Hang Zhou, Jie Zhang, and Nenghai Yu. Reversible watermarking in deep convolutional neural networks for integrity authentication. In *Proceedings of the 28th ACM International Conference on Multimedia*, pages 2273–2280, 2020.

[21] Jia Guo and Miodrag Potkonjak. Watermarking deep neural networks for embedded systems. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2018.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[24] Yingzhe He, Guozhu Meng, Kai Chen, Xingbo Hu, and Jinwen He. Towards security threats of deep learning systems: A survey. *IEEE Transactions on Software Engineering*, 48(5):1743–1770, 2020.

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

[26] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations, 2016.

[27] Hengrui Jia, Christopher A Choquette-Choo, Varun Chandrasekaran, and Nicolas Papernot. Entangled watermarks as a defense against model extraction. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1937–1954, 2021.

[28] Ju Jia, Yueming Wu, Anran Li, Siqi Ma, and Yang Liu. Subnetwork-lossless robust watermarking for hostile theft attacks in deep transfer learning models. *IEEE Transactions on Dependable and Secure Computing*, pages 1–16, 2022.

[29] John Kirchenbauer, Jonas Geiping, Yuxin Wen, Jonathan Katz, Ian Miers, and Tom Goldstein. A watermark for large language models. *arXiv preprint arXiv:2301.10226*, 2023.

[30] Alex Krizhevsky and Geoffrey Hinton. Learning Multiple Layers of Features from Tiny Images, 2009.

[31] Minoru Kuribayashi, Takuro Tanaka, Shunta Suzuki, Tatsuya Yasui, and Nobuo Funabiki. White-box watermarking scheme for fully-connected layers in fine-tuning model. In *Proceedings of the 2021 ACM Workshop on Information Hiding and Multimedia Security*, pages 165–170, 2021.

[32] Yingjie Lao, Peng Yang, Weijie Zhao, and Ping Li. Identification for deep neural network: Simply adjusting few weights! In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pages 1328–1341. IEEE, 2022.

[33] Yingjie Lao, Weijie Zhao, Peng Yang, and Ping Li. Deepauth: A dnn authentication framework by model-unique and fragile signature embedding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36.9, pages 9595–9603, 2022.

[34] Erwan Le Merrer, Patrick Perez, and Gilles Trédan. Adversarial frontier stitching for remote neural network watermarking. *Neural Computing and Applications*, 32:9233–9244, 2020.

[35] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 1998.

[36] Bowen Li, Lixin Fan, Hanlin Gu, Jie Li, and Qiang Yang. Fedipr: Ownership verification for federated deep neural network models, 2022.

[37] Yiming Li, Yang Bai, Yong Jiang, Yong Yang, Shu-Tao Xia, and Bo Li. Untargeted backdoor watermark: Towards harmless and stealthy dataset copyright protection. *Advances in Neural Information Processing Systems*, 35:13238–13250, 2022.

[38] Yiming Li, Linghui Zhu, Xiaojun Jia, Yang Bai, Yong Jiang, Shu-Tao Xia, and Xiaochun Cao. Move: Effective and harmless ownership verification via embedded external features. *arXiv preprint arXiv:2208.02820*, 2022.

[39] Yiming Li, Linghui Zhu, Xiaojun Jia, Yong Jiang, Shu-Tao Xia, and Xiaochun Cao. Defending against model stealing via verifying embedded external features. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36.2, pages 1464–1472, 2022.

[40] Zheng Li, Chengyu Hu, Yang Zhang, and Shanqing 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*, pages 126–137, 2019.

[41] Jian Han Lim, Chee Seng Chan, Kam Woh Ng, Lixin Fan, and Qiang Yang. Protect, show, attend and tell: Empowering image captioning models with ownership protection. *Pattern Recognition*, 122:108285, 2022.

[42] Hanwen Liu, Zhenyu Weng, and Yuesheng Zhu. Watermarking deep neural networks with greedy residuals. In *ICML*, pages 6978–6988, 2021.

[43] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Fine-pruning: Defending against backdooring attacks on deep neural networks, 2018.

[44] Sofiane Lounici, Mohamed Njeh, Orhan Ermis, Melek Önen, and Slim Trabelsi. Yes we can: Watermarking machine learning models beyond classification. In *2021 IEEE 34th Computer Security Foundations Symposium (CSF)*, pages 1–14, 2021.

[45] Nils Lukas, Edward Jiang, Xinda Li, and Florian Kerschbaum. Sok: How robust is image classification deep neural network watermarking? In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 787–804. IEEE, 2022.

[46] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks, 2019.

[47] April Pyone Maung Maung and Hitoshi Kiya. Piracy-resistant dnn watermarking by block-wise image transformation with secret key. In *Proceedings of the 2021 ACM Workshop on Information Hiding and Multimedia Security*, pages 159–164, 2021.

[48] Dhwani Mehta, Nurun Mondol, Farimah Farahmandi, and Mark Tehranipoor. Aime: watermarking ai models by leveraging errors. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 304–309. IEEE, 2022.

[49] Carl D Meyer and Ian Stewart. *Matrix analysis and applied linear algebra*. SIAM, 2023.

[50] Travis Munyer and Xin Zhong. Deeptextmark: Deep learning based text watermarking for detection of large language model generated text. *arXiv preprint arXiv:2305.05773*, 2023.

[51] Ryota Namba and Jun Sakuma. Robust watermarking of neural network with exponential weighting, 2019.

[52] Ding Sheng Ong, Chee Seng Chan, Kam Woh Ng, Lixin Fan, and Qiang Yang. Protecting intellectual property of generative adversarial networks from ambiguity attack, 2021.

[53] R OpenAI. Gpt-4 technical report. *arXiv*, pages 2303–08774, 2023.

[54] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[55] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[56] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2023.

[57] Ge Ren, Jun Wu, Gaolei Li, Shenghong Li, and Mohsen Guizani. Protecting intellectual property with reliable availability of learning models in ai-based cybersecurity services. *IEEE Transactions on Dependable and Secure Computing*, pages 1–18, 2022.

[58] Alexandre Sablayrolles, Matthijs Douze, Cordelia Schmid, and Hervé Jégou. Radioactive data: tracing through training. In *International Conference on Machine Learning*, pages 8326–8335. PMLR, 2020.

[59] Masoumeh Shafieinejad, Nils Lukas, Jiaqi Wang, Xinda Li, and Florian Kerschbaum. On the robustness of backdoor-based watermarking in deep neural networks. In *Proceedings of the 2021 ACM Workshop on Information Hiding and Multimedia Security*, IH&MMSec '21, page 177–188, New York, NY, USA, 2021. Association for Computing Machinery.

[60] Masoumeh Shafieinejad, Nils Lukas, Jiaqi Wang, Xinda Li, and Florian Kerschbaum. On the robustness of backdoor-based watermarking in deep neural networks. In *Proceedings of the 2021 ACM Workshop on Information Hiding and Multimedia Security*, pages 177–188, 2021.

[61] Shuo Shao, Wenyuan Yang, Hanlin Gu, Zhan Qin, Lixin Fan, Qiang Yang, and Kui Ren. Fedtracker: Furnishing ownership verification and traceability for federated learning model, 2023.

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

[63] Yuchen Sun, Tianpeng Liu, Panhe Hu, Qing Liao, Shouling Ji, Nenghai Yu, Deke Guo, and Li Liu. Deep intellectual property: A survey. *arXiv preprint arXiv:2304.14613*, 2023.

[64] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016.

[65] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction {APIs}. In *25th USENIX security symposium (USENIX Security 16)*, pages 601–618, 2016.

[66] Yusuke Uchida, Yuki Nagai, Shigeyuki Sakazawa, and Shin'ichi Satoh. Embedding watermarks into deep neural networks. In *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval*. ACM, jun 2017.

[67] Martin Vetterli, Jelena Kovačević, and Vivek K Goyal. *Foundations of signal processing*. Cambridge University Press, 2014.

[68] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 707–723. IEEE, 2019.

[69] Lixu Wang, Shichao Xu, Ruiqi Xu, Xiao Wang, and Qi Zhu. Non-transferable learning: A new approach for model ownership verification and applicability authorization, 2022.

[70] Tianhao Wang and Florian Kerschbaum. Riga: Covert and robust white-box watermarking of deep neural networks. In *Proceedings of the Web Conference 2021*, pages 993–1004, 2021.

[71] Tong Wu, Xinghua Li, Yinbin Miao, Mengfan Xu, Haiyan Zhang, Ximeng Liu, and Kim-Kwang Raymond Choo. Cits-mew: Multi-party entangled watermark in cooperative intelligent transportation system. *IEEE Transactions on Intelligent Transportation Systems*, 24(3):3528–3540, 2023.

[72] Lou Xiaoxuan, Guo Shangwei, Zhang Tianwei, Liu Yang, et al. When nas meets watermarking: ownership verification of dnn models via cache side channels. *arXiv e-prints*, pages arXiv–2102, 2021.

[73] Chenqi Xie, Ping Yi, Baowen Zhang, and Futai Zou. Deepmark: Embedding watermarks into deep neural network using pruning. In *2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 169–175. IEEE, 2021.

[74] Yifan Yan, Xudong Pan, Mi Zhang, and Min Yang. Rethinking white-box watermarks on deep learning models under neural structural obfuscation. In *32th USENIX security symposium (USENIX Security 23)*, 2023.

[75] Peng Yang, Yingjie Lao, and Ping Li. Robust watermarking for deep neural networks via bi-level optimization. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 14821–14830, 2021.

[76] Jialong Zhang, Zhongshu Gu, Jiyong Jang, Hui Wu, Marc Ph Stoecklin, Heqing Huang, and Ian Molloy. Protecting intellectual property of deep neural networks with watermarking. In *Proceedings of the 2018 on Asia conference on computer and communications security*, pages 159–172, 2018.

[77] Jie Zhang, Dongdong Chen, Jing Liao, Weiming Zhang, Gang Hua, and Nenghai Yu. Passport-aware normalization for deep model protection. *Advances in Neural Information Processing Systems*, 33:22619–22628, 2020.

[78] Xiangyu Zhao, Yinzhe Yao, Hanzhou Wu, and Xinpeng Zhang. Structural watermarking to deep neural networks via network channel pruning. In *2021 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 1–6. IEEE, 2021.

[79] Qi Zhong, Leo Yu Zhang, Jun Zhang, Longxiang Gao, and Yong Xiang. Protecting ip of deep neural networks with watermarking: A new label helps. In *Advances in Knowledge Discovery and Data Mining: 24th Pacific-Asia Conference, PAKDD 2020, Singapore, May 11–14, 2020, Proceedings, Part II 24*, pages 462–474. Springer, 2020.

[80] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression, 2017.

[81] Renjie Zhu, Xinpeng Zhang, Mengte Shi, and Zhenjun Tang. Secure neural network watermarking protocol against forging attack. *EURASIP Journal on Image and Video Processing*, 2020:1–12, 2020.

A Algorithmic Implementation

Algorithmic implementation of *DeepEclipse*: starting from Algorithm 5, if the SMA detection algorithm has not found any potential layer, we apply the obfuscation to the whole model (line 2); otherwise, for each watermarked detected layer, we apply the specific obfuscation based on its type and accordingly to the kind of verification that is expected. The function AdvConvObfuscation (Algorithm 3) calculates an ε value for each feature of the layer's feature map and uses it for padding; it then multiplies the whole layer by a random value and divides the subsequent one by the same value. The function BaseConvObfuscation (Algorithm 4) pads each feature map of the kernel with zeros. The function AdvLinearObfuscation (Algorithm 1) calculates the matrices H and H^{-1} based on the dimension of the watermarked layer and the subsequent one; it then multiplies the original layer's weight by H and the subsequent layer by H^{-1}. Lastly, function BaseLinearObfuscation (Algorithm 2) calculates the matrices H and H^{-1} based on the dimension of the watermarked layer; it then expands the model on that specific layer while keeping the previous and subsequent layers untouched.

Algorithm 1 AdvLinearObfuscation(M, *layer*)
Input:
M Model to obfuscate
layer position of the Linear layer
Output:
M Model with advanced obfuscation of Linear layer

1: $H, H^{-1} \leftarrow RandomMatrix(M[layer], M[layer+1])$
2: $M[layer] \leftarrow M[layer] \times H$
3: $M[layer+1] \leftarrow M[layer+1] \times H^{-1}$
4: **return** M

Algorithm 2 BaseLinearObfuscation(M, *layer*)
Input:
M Model to obfuscate
layer position of the Linear layer
Output:
M' Model with base obfuscation of Linear layer

1: $H, H^{-1} \leftarrow RandomMatrix(M[layer])$
2: $M', newLayer \leftarrow expandModel(M, layer)$
3: $M'[layer] \leftarrow M'[layer] \times H$
4: $M'[newLayer] \leftarrow H^{-1}$
5: **return** M'

Algorithm 3 AdvConvObfuscation(M, *layer*)
Input:
M Model to obfuscate
layer position of the Conv. layer
Output:
M Model with advanced obfuscation of Conv. layer

1: **for** p in $FeatureMap(M[layer])$ **do**
2: $\varepsilon \leftarrow \beta * min(|M[layer][p]|) * \mathcal{N}(\mu, \sigma^2)$
3: $M[layer][p] \leftarrow PadKernel(M[layer][p], \varepsilon)$
4: **end for**
5: $\lambda \leftarrow random()$
6: $M[layer] \leftarrow M[layer] * \lambda$
7: $M[layer+1] \leftarrow M[layer+1]/\lambda$
8: **return** M

Algorithm 4 BaseConvObfuscation(M, *layer*)
Input:
M Model to obfuscate
layer position of the Conv. layer
Output:
M Model with base obfuscation of Conv. layer

1: **for** p in $FeatureMap(M[layer])$ **do**
2: $M[layer][p] \leftarrow PadKernel(M[layer][p], 0)$
3: **end for**
4: **return** M

Algorithm 5 *DeepEclipse* (M, *SMA*, *Adv*)
Input:
M original stolen Model
SMA detection's output (can be empty)
Adv if we apply advanced obfuscation
Output:
M' obfuscated Model

1: **if** $SMA = \emptyset$ **then**
2: $SMA \leftarrow \{layer \mid layer \in M\}$
3: **end if**
4: $M' \leftarrow copy(M)$
5: **for** *layer* in *SMA* **do**
6: **if** *layer* is Convolutional **then**
7: **if** *Adv* is True **then**
8: $M'[layer] \leftarrow AdvConvObfuscation(M', layer)$
9: **else**
10: $M'[layer] \leftarrow BaseConvObfuscation(M', layer)$
11: **end if**
12: **else if** *layer* is Linear **then**
13: **if** *Adv* is True **then**
14: $M'[layer] \leftarrow AdvLinearObfuscation(M', layer)$
15: **else**
16: $M'[layer] \leftarrow BaseLinearObfuscation(M', layer)$
17: **end if**
18: **end if**
19: **end for**
20: **return** M'