electronics

MDPI

Article

A Privacy-Preserving Testing Framework for Copyright Protection of Deep Learning Models

Dongying Wei [1], Dan Wang [2,*], Zhiheng Wang [1] and Yingyi Ma [1]

[1] School of Geology and Geomatics, Tianjin Chengjian University, Tianjin 300392, China; weidongying@tcu.edu.cn (D.W.); wangzhiheng@tcu.edu.cn (Z.W.); mayy@tcu.edu.cn (Y.M.)
[2] College of Artificial Intelligence, Tianjin University of Science and Technology, Tianjin 300453, China
* Correspondence: wanghzc@tust.edu.cn

Abstract: Deep learning is widely utilized to acquire predictive models for mobile crowdsensing systems (MCSs). These models significantly improve the availability and performance of MCSs in real-world scenarios. However, training these models requires substantial data resources, rendering them valuable to their owners. Numerous protection schemes have been proposed to mitigate potential economic loss arising from legal issues pertaining to model copyright. Although capable of providing copyright verification, these schemes either compromise the model utility or prove ineffective against adversarial attacks. Additionally, the privacy concern surrounding copyright verification is noteworthy, given the increasing privacy concerns among model owners. This paper introduces a privacy-preserving testing framework for copyright protection (PTFCP) comprising multiple protocols. Our protocols adhere to the two-cloud server model, where the owner and the suspect transmit their model output to non-colluding servers for evaluating model similarity through the public-key cryptosystem with distributed decryption (PCDD) and garbled circuits. Additionally, we have developed novel techniques to enable secure differentiation for absolute values. Our experiments in real-world datasets demonstrate that our protocols in the PTFCP successfully operate under numerous copyright violation scenarios, such as finetuning, pruning, and extraction.

Keywords: mobile crowdsensing; deep learning model copyright protection; privacy preservation

Citation: Wei, D.; Wang, D.; Wang, Z.; Ma, Y. A Privacy-Preserving Testing Framework for Copyright Protection of Deep Learning Models. *Electronics* **2024**, *13*, 133. https://doi.org/10.3390/electronics13010133

Academic Editor: Aryya Gangopadhyay

Received: 1 December 2023
Revised: 20 December 2023
Accepted: 26 December 2023
Published: 28 December 2023

1. Introduction

Deep learning, as a promising and practical technology, has been extensively adopted to enhance the performance of mobile crowdsensing systems (MCSs) [1–4]. For instance, in [5], the authors utilize deep neural networks (DNNs) and employ graph convolutional reinforcement learning to achieve Aoi-minimal UAV crowdsensing. Similarly, in [6], the DNN is utilized to implement a heterogeneous task allocation in MCSs, employing a modified approximate policy approach. Furthermore, in [7], Xu et al. combine the concepts of the graph attention network and deep reinforcement learning to propose an intelligent task allocation scheme for MCSs. It is evident that well-trained DNN models can significantly enhance the performance of MCSs.

However, as the model complexity, data volume, and number of tasks grow, the cost of training an applicable model, particularly for deep neural networks (DNNs), also increases. For instance, training a model with 1.5 billion parameters, such as the BERT model on Wikipedia, can cost nearly $1.6 million. Therefore, safeguarding the trained models from duplication, theft, or unauthorized reproduction by adversaries is necessary and crucial [8,9].

It is worth noting that, to enhance the provision of machine learning as a service (MLaaS), certain organizations or companies opt to offer open-source toolkits, remote services, or cloud platforms to the public. However, despite its potential for commercial profit, this approach exposes the trained DNN models (which are often trained at considerable

cost) to the public, enabling adversaries to steal the models discreetly and conveniently. These potential risks can lead to substantial economic losses [10–12] and even involve legal issues [13–16] related to copyright infringement for the model owners. Researchers have already highlighted that adversaries can stealthily and efficiently steal DNN models [17,18]. In cases where the adversary possesses greater knowledge about the model or when the model parameters are publicly available, two commonly employed white-box attacks are unauthorized finetuning and pruning [19]. Furthermore, recent studies have shown that even under a black-box attack scenario [17,20], the core functionalities of a DNN model can still be extracted. Despite the adversary's limited access to the exposed API, they can employ advanced model extraction approaches to reconstruct the underlying core functionality of the model. In conclusion, there has been a rapid growth in the threat to the copyright of DNN models, highlighting the urgent need for effective and practical countermeasures.

In recent years, numerous schemes have been proposed to protect model copyrights. DNN watermarking is a commonly employed scheme among these, known for its effectiveness in practical scenarios [21,22]. It utilizes the over-parameterization properties of DNNs to secretly embed a designed watermark (e.g., a signature or logo representing ownership information) into the model. Verifying the ownership of a suspected model can be easily accomplished by extracting potential embedded watermarks from the model [23]. Consequently, DNN watermarking is considered a promising technique with significant advantages. However, these schemes still have some crucial drawbacks [24]. Notably, the essential attribute of these schemes is their invasive nature, where the model content or training process is modified after watermark embedding. Recent studies have shown that such modifications can severely compromise the utility of DNN models or introduce additional vulnerabilities [25,26].

To overcome the limitations of invasive techniques such as watermarking, researchers have proposed DNN fingerprinting, which has garnered significant attention [27,28]. Fingerprinting is rooted in the uniqueness of a DNN model, which embodies its distinct characteristics. Specifically, a distinctive fingerprint is extracted from the owner model, serving as an identifier to differentiate it from other models. If the extracted fingerprint from a suspect model matches that of the owner model, the model owner can assert ownership based on this result. However, in real-world scenarios, relying solely on the fingerprint as a feature or metric for justifying model ownership is insufficiently persuasive. Nonetheless, DNN fingerprinting is also vulnerable to state-of-the-art model stealing attacks. In conclusion, the distinctive characteristics of a DNN model may be compromised or unavailable in various situations.

To address the limitations of DNN fingerprinting, researchers have turned their attention to assessing the similarity between distinct models [29–31]. Traditional approaches employ direct calculations, such as the Euclidean distance [29], Kullback–Leibler divergence, or Jensen–Shannon divergence [31], to measure the dissimilarity between the contents of distinct models in parameter space or output space. This methodology is based on the premise that the similarity between two different models stems from the resemblance of their internal contents. However, this simplistic approach lacks persuasiveness and practicality. For example, computing the Euclidean distance in the parameter space of DNN models can result in significant computational expenses [32]. If these costs are deemed acceptable, copyright verification becomes unnecessary. Moreover, there is a possibility that the two models exhibit substantial differences [33] in parameter space despite sharing the same training dataset and initialization due to potential bugs in floating-point operations or stochasticity in the training process.

To enhance the assessment of similarity between distinct models, Chen et al. [30] introduced a testing framework called DeepJudge. In their study, they evaluate the similarity of two distinct models using six metrics across three levels, which are based on the generated test cases. Their experiments provide evidence of DeepJudge's effectiveness. Despite the promise of DeepJudge compared to traditional approaches, it assumes that the model owner grants unrestricted access to the architecture, parameters, and datasets,

including training details in some cases. However, in real-world scenarios, ownership verification is typically conducted by a third party, and the model owner is reluctant to provide such access due to concerns about model privacy. Consequently, there is an urgent need for a method that can measure model similarity while maintaining model privacy. However, to the best of our knowledge, this problem has been largely overlooked.

This paper focuses on evaluating the similarity between two distinct models in a privacy-preserving manner using cryptographic techniques. We propose a set of protocols based on the two-cloud model for conducting privacy-preserving model similarity evaluation. In these protocols, the model owner (referred to hereafter as the "victim") and the party holding a suspect model (referred to hereafter as the "suspect") transmit encrypted data to the cloud server. During the computation phase, the two non-colluding servers can evaluate the similarity using different metrics without accessing any sensitive information about the model data. Subsequently, the encrypted evaluation result is returned to the victim for further estimation purposes. In real-world scenarios, when two AI companies or organizations face a copyright infringement conflict, the details of their respective models are confidential due to commercial competition. Therefore, adopting these protocols encourages the entities to pursue judgment by the authorized third party while alleviating concerns about unintentional leakage of confidential model information.

In summary, our main contributions are:

- We address a realistic and often overlooked problem: preserving privacy in model copyright verification. To the best of our knowledge, this paper is the first to discuss privacy preservation in this context.
- We propose a set of protocols based on the two-cloud model that enables the evaluation of similarity between distinct models while safeguarding model privacy. Our protocols encompass multiple metrics at various levels, providing a comprehensive evaluation of model similarity.
- Through extensive experiments conducted on various real-world DNN models and datasets, we demonstrate the effectiveness of our protocols.

The structure of this paper is as follows: Section 2 provides an introduction to the system and the threat model. Section 3 presents the preliminaries, while Section 4 describes the details of the proposed protocols. In Section 5, we discuss the experimental results to demonstrate the effectiveness of the proposed protocols. Section 6 highlights relevant works on model copyright. Finally, we conclude our work in Section 7. The detailed roadmap is illustrated in Figure 1.

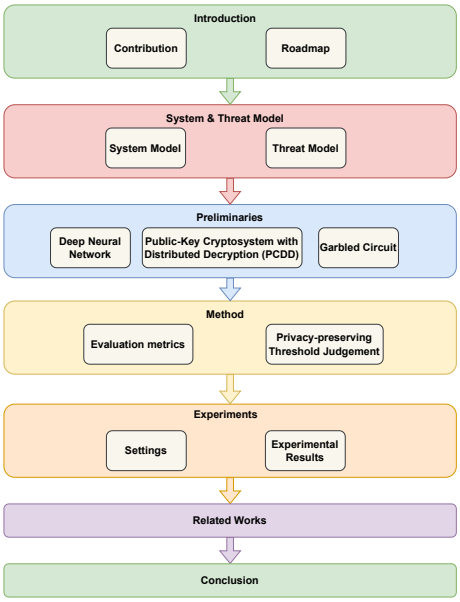


Figure 1. The roadmap of this paper.

2. System and Threat Model

2.1. System Model

As illustrated in Figure 2, our PTFCP consists of three main entities: the model owner, suspect, and cloud servers.

- The model owner, referred to as the victim, possesses a model (referred to as the victim model for simplicity) that has been trained using private and valuable resources. In order to ascertain whether the suspect model is an illicit copy of the victim's model, the victim utilizes its test cases to generate the outputs from its own model. Subsequently, these outputs are encrypted and utilized as inputs for the protocols within our PTFCP.
- The suspect possesses a model that potentially infringes upon the copyright of the victim's model. To determine the occurrence of infringement, the suspect follows the subsequent steps. Initially, it employs the test cases of the victim to obtain outputs using its own model. Subsequently, these outputs are encrypted and utilized as inputs for the judgment protocols.
- Initially, the cloud servers receive the encrypted outputs from both the victim and the suspect based on the determined public key pk. Subsequently, these two servers, denoted as S_1 and S_2 respectively, collaboratively execute the protocols within our PTFCP to acquire the encrypted judgment results.

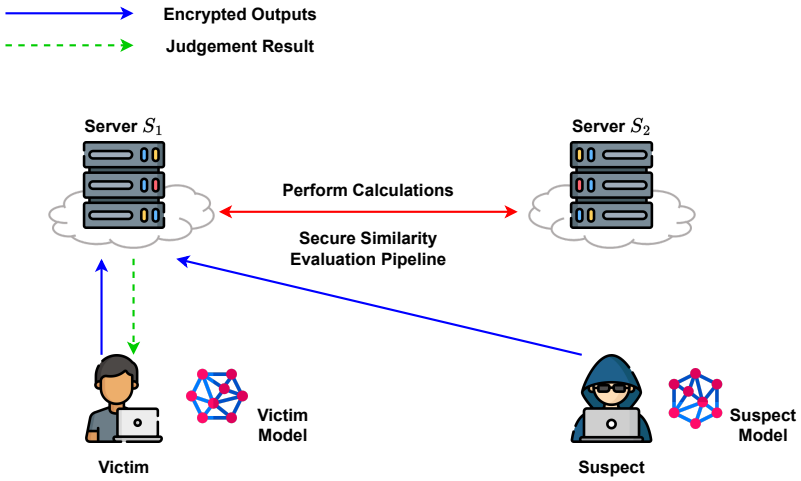


Figure 2. The system model.

2.2. Threat Model

This section focuses on a typical attack–defense scenario in which the victim endeavors to safeguard the copyright of its model and identify potential instances of infringement. Conversely, the suspect's objective is to illicitly acquire either the functionality or the complete information of the victim model. Moreover, the suspect employs various techniques to obfuscate the stolen model and avoid its detection as an unauthorized replica. Furthermore, in consideration of data privacy requirements such as model parameter privacy and architecture privacy, privacy-preserving measures are also taken into account within the threat model.

2.2.1. Model Copyright

We summarize three major threats to the copyright of models based on state-of-the-art model stealing attacks.

1. Finetuning In this scenario, it is assumed that the suspect possesses complete knowledge of both the model parameters and architecture, along with an additional dataset for finetuning the target model. For example, even if certain model owners have open-sourced their trained models solely for academic or open-source purposes, the

suspect can still exploit finetuning techniques in a malicious manner to construct their own model for commercial gain.

2. Pruning In this scenario, it is also assumed that the suspect has acquired complete information regarding the parameters and architecture of the victim model. In order to perform model pruning, the suspect initially employs various pruning techniques to reduce the size of the victim model. Subsequently, the suspect employs its auxiliary dataset to conduct finetuning on the pruned model obtained in the previous step.
3. Model Extraction In this scenario, the suspect has restricted access and can only retrieve the prediction results, such as probability vectors, from the victim model by utilizing the exposed API. Occasionally, the suspect may possess information regarding the architecture of the victim model, while the model parameters of the victim model remain inaccessible. The primary objective of the suspect is to accurately replicate the core functionality of the victim model by leveraging the exposed API. Based on recent proposed works, the suspect can initially utilize selected samples to query the exposed API and generate a labeled dataset. Subsequently, using the labeled dataset, the suspect trains an alternative model capable of replicating the functionality of the victim model. The selected samples can be sourced either from the suspect's auxiliary dataset or generated using advanced techniques.

2.2.2. Data Privacy

In our PTFCP, it is assumed that the victim, suspect, and two cloud servers are all characterized as honest-but-curious. This implies that they will faithfully execute the steps outlined in the protocols during the similarity evaluation process. Nevertheless, they may possess an interest in the information pertaining to other entities and have the capability to compromise data privacy through the utilization of their knowledge. Additionally, it is assumed that the two cloud servers are non-colluding, a common and reasonable assumption in numerous application scenarios [34–37]. Furthermore, it is further assumed that neither the owner nor the suspect will engage in collusion with the cloud servers.

3. Preliminaries

3.1. Deep Neural Network

Deep Neural Networks (DNNs) serve as a fundamental technology in the domain of deep learning, enabling the learning of intricate relationships between input and output in high-dimensional spaces. Consequently, DNNs have found applications in diverse fields such as voice recognition, image classification, and natural language processing. In general, a canonical neural network comprises three essential components: an input layer, an output layer, and at least one hidden layer. Figure 3 illustrates a representative example of a fully connected DNN.

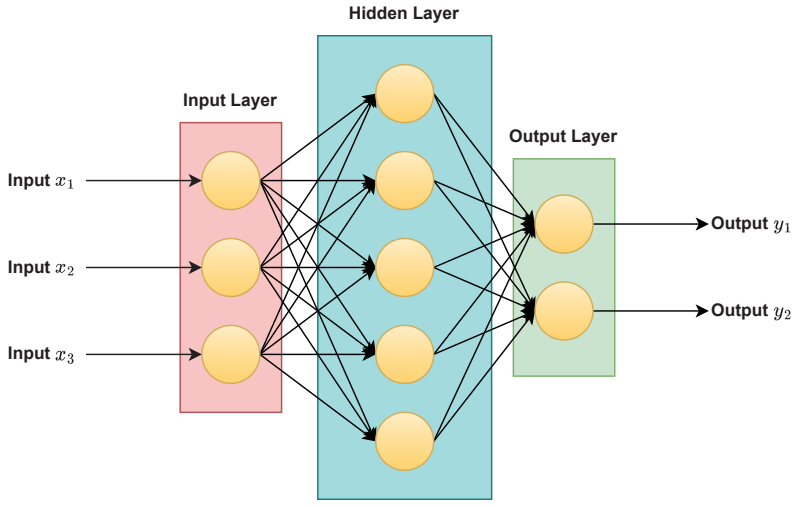


Figure 3. An example of DNN.

Without loss of generality, our primary focus here is on the DNN classifier. The DNN classifier possesses a decision function that maps an input to its corresponding label. The classifier consists of L layers: $f^1, f^2, \cdots, f^{L-1}, f^L$. Here, f^1 represents the input layer, f^L is the output layer that generates the probability vector, and $f^2, \cdots, f^{L-1}$ refer to the hidden layers. The total number of classes is denoted as C. Each layer f^l consists of a set of neurons: $n_1^l, n_2^l, \cdots, n_{N_l}^l$, with N_l representing the number of neurons in that layer. The output of the neuron n_i^l, given a specific input x, is denoted as $\varphi_i^l(x)$. Consequently, the output vector for layer f^l $(2 \leq l \leq L)$ can be defined as $f^l(x) =< \varphi_1^l(x), \varphi_2^l(x), \cdots, \varphi_{N_l}^l(x) >$. The prediction result can be obtained by evaluating $\arg\max f^L(x)$.

3.2. Secure Computation

3.2.1. Public-Key Cryptosystem with Distributed Decryption (PCDD)

The two-trapdoor public-key cryptosystem was first proposed by Bresson et al. [38]. In addition to the ordinary private key (also known as the weak key) sk, the cryptosystem employs a strong private key SK that can decrypt the encrypted data under different public keys. To reduce the risk of strong private key leakage, Liu et al. [39] adapted the cryptosystem by splitting the strong key into two shares, enabling distributed decryption. PCDD is defined as follows:

KeyGen: Let $N = pq$ and $\lambda = \text{lcm}(p-1, q-1)/2$, where p and q are large prime numbers, $L(p) = L(q) = k$. Define a function $L(x) = (x-1)/N$, and pick an element g from the multiplicative group $\mathbb{Z}_{N^2}^*$, whose order is $(p-1)(q-1)/2$. Then, randomly select θ_i from the interval $[1, N/4]$ and compute $h_i = g^{\theta_i} \mod N^2$ for party i. The public key is $pk_i = (N, g, h_i)$, and the corresponding weak private key is $sk_i = \theta_i$. The strong private key is $SK = \lambda$.

Encryption: Given the public key pk_i, one can encrypt a message $m \in \mathbf{Z}_N$ by randomly choosing $r \in [1, N/4]$ and computing $Enc_{pk_i}[m] = \{T_{i,1}, T_{i,2}\}$, where $T_{i,1} = g^{r\theta_i}(1+mN) mod N^2$ and $T_{i,2} = g^r mod N^2$.

Decryption with weak private key: One can decrypt the ciphertext $Enc_{pk_i}[m]$ with the weak private key $sk_i = \theta_i$ by computing

$$m = D_{sk_i}(Enc_{pk_i}[m]) = L((T_{i,1} \cdot (T_{i,2}^{\theta_i})^{-1}) mod N^2). \tag{1}$$

Decryption with strong private key: Any ciphertext $Enc_{pk_i}[m]$ can be decrypted using the decryption algorithm $D_{SK}(\cdot)$ with the strong private key $SK = \lambda$ by first calculating:

$$T_{i,1}^{\lambda} mod N^2 = g^{\lambda\theta_i r} \cdot (1 + mN\lambda) mod N^2 = 1 + mN\lambda. \tag{2}$$

Then, m can be recovered as:

$$m = L(T_{i,1}^{\lambda} mod N^2) \cdot \lambda^{-1} mod N. \tag{3}$$

Strong private key splitting: The strong private key $SK = \lambda$ can be randomly split into two parts. The partial string private keys $SK^{(j)} = \lambda_j (j = 1, 2)$, s.t., $\lambda_1 + \lambda_2 \equiv 0 mod \lambda$ and $\lambda_1 + \lambda_2 \equiv 1 mod N^2$ hold at the same time.

Partial decryption with partial strong private key: The algorithm $D_{SK^{(j)}}^{+}(\cdot)$ takes as input $T_{i,1}$ and a partial strong private key $SK^{(j)} = \lambda_j$ and then outputs the partially decrypted ciphertext

$$CT_{i,j} = (T_{i,1})^{\lambda_j} = g^{\lambda_j \theta_i r} \cdot (1 + mN\lambda_j) mod N^2. \tag{4}$$

Decryption with partially decrypted ciphertext: The algorithm $D^{-}(\cdot)$ takes as input $CT_{i,1}$ and $CT_{i,2}$ and then computes

$$m = L(CT_{i,1} \cdot CT_{i,2}). \tag{5}$$

Ciphertext Refresh: This algorithm can refresh the ciphertext $Enc_{pk_i}[m]$ to obtain $Enc'_{pk_i}[m] = \{T'_{i,1}, T'_{i,2}\}$ without changing the original message m as follows. Choose a random number $r' \in \mathbf{Z}_N$ and then compute

$$T'_{i,1} = T_{i,1} \cdot h_i^{r'} mod N^2, T'_{i,2} = T_{i,2} \cdot g^{r'} mod N^2. \tag{6}$$

Note that the numbers involved in the PCDD can be integers, meaning that m can take positive, negative, or zero values. However, the sign bit is lost in a modular operation. The method for handling this issue can be found in [39]. Additionally, the PCDD cryptosystem exhibits homomorphic additivity as follows:

$$\begin{aligned} D_{sk_i}(Enc_{pk_i}[x+y]) &= D_{sk_i}(Enc_{pk_i}[x] \cdot Enc_{pk_i}[y] mod N^2). \\ D_{sk_i}(Enc_{pk_i}[a \cdot x]) &= D_{sk_i}((Enc_{pk_i}[x])^a mod N^2). \end{aligned} \tag{7}$$

3.2.2. Garbled Circuit

Garbled circuits (GCs) were initially introduced by Yao [40] as a means to facilitate secure two-party computation. Broadly speaking, a canonical garbling scheme consists of three main components: a garbling algorithm, a function f, and a decoding table Tab_{Dec}. A randomly chosen seed σ is used in the algorithm, while the function f is employed to generate the garbled circuit F. Specifically, for a given input x, the encoding algorithm generates the garbled representation $\hat{x}$ using x and the random seed σ. Subsequently, the garbled output $\hat{z}$ is obtained through the evaluation algorithm by utilizing $\hat{x}$ and the garbled circuit F. Finally, by utilizing Tab_{Dec} and the garbled output $\hat{z}$, the decoding algorithm yields the result $f(x)$.

4. Method

4.1. Evaluation Metrics

In order to assess the similarity between two distinct models more rigorously, we have identified metrics in the white-box setting and the black-box setting, drawing from [27,30,41]. The details of these metrics are presented in Table 1.

Table 1. Distance metrics in different settings.

Settings	Distance Metric
White-Box	Neuron Outputs Layer Outputs
Black-Box	Model Robustness

4.1.1. Neuron Perspective

We begin by evaluating the similarity of two distinct models from a fine-grained perspective. Based on the mathematical attributes, each neuron in a deep learning model consistently adheres to its respective statistical distribution. Consequently, when two models differ, their neuron outputs should exhibit variations as well [42]. Building upon this observation, we focus on the values of neuron outputs to assess the similarity between $Model_{Vic}$ and $Model_{Sus}$, utilizing the NOD metric, which is defined at the neuron level.

Neuron Outputs Distance (NOD)

For a specific neuron, we denote n_i^l as the i-th neuron in the l-th layer of a given DNN. We represent the neuron output functions for $Model_{Vic}$ and $Model_{Sus}$ as φ_i^l and $\hat{\varphi}_i^l$, respectively. The NOD metric calculates the average difference of neuron outputs across a predefined set $T_x = x_1, x_2, \cdots$ containing test samples:

$$NOD(T_x, \varphi_i^l, \hat{\varphi}_i^l) = \frac{\sum_{x_t \in T_x} |\varphi_i^l(x_t) - \hat{\varphi}_i^l(x_t)|}{|T_x|}. \tag{8}$$

4.1.2. Layer Perspective

In addition to the neuron perspective, we also examine layer-level metrics that take into account the output status of entire layers in a DNN model [41]. It is evident that the metrics from the layer-level perspective provide a more comprehensive assessment of the similarity between $Model_{Vic}$ and $Model_{Sus}$ based on the differences in intermediate layer output status.

Layer Outputs Distance (LOD)

For a given DNN, let us consider layer l. Let f^l and $\hat{f}^l$ represent the output functions of layer l for $Model_{Vic}$ and $Model_{Sus}$, respectively. The LOD formula is used to assess the distance between the l-th layer of $Model_{Vic}$ and $Model_{Sus}$ and is given as:

$$LOD(T_x, f^l, \hat{f}^l) = \frac{\sum_{x \in T_x} ||f^l(x_t) - \hat{f}^l(x_t)||_p}{|T_x|}, \tag{9}$$

where $|| \cdot ||_p$ represents the L_p-norm, and we take $p = 1$ in our scheme.

4.1.3. Attribute Perspective

Model Robustness Distance (MRD)

In recent years, several characteristics have been proposed based on model properties to measure the similarity between different models. For example, some studies define the adversarial robustness property [27,28] as a characteristic, while others focus on the fairness property [43]. In our approach, we employ the initial characteristic and introduce the *Model Robustness Distance (MRD)* to assess the similarity in adversarial robustness between two different models using a provided set of test samples. Since a model's robustness is closely tied to its learned decision boundary during the optimization process, it can serve as a practical indicator to evaluate the similarity between two models.

Without loss of generality, we denote the function of the victim model $Model_{Vic}$ as f. Given an input x_t and its corresponding label y_t, a canonical adversarial sample can be generated by iteratively adding noise to x_t in order to maximize the classification error of the function f. This type of attack is commonly referred to as an adversarial attack, and, if the final adversarial sample $x_t^{'}$ satisfies $f(x_t^{'}) \neq y_t$, it indicates the success of the adversarial attack. To generate effective adversarial examples, one can utilize various existing methods, such as FGSM or PGD. In this case, given a set of test samples, the adversarial counterparts can be generated as $T_{adv} = x_1^{'}, x_2^{'}, \cdots$, where $x_t^{'}$ represents the adversarial version of the sample $x_t^{'}$. We can define the characteristic of model robustness, denoted as MR, as follows:

$$MR(T_{adv}, f) = \frac{\sum_{t=1}^{|T_{adv}|} (f(x_t^{'}) = y_t)}{|T_{adv}|}. \tag{10}$$

Next, we assume $\hat{f}$ represents the suspect model $Model_{Sus}$, as mentioned earlier. Based on the aforementioned context, we can define the Model Robustness Distance (MRD) between $Model_{Vic}$ and $Model_{Sus}$ as follows:

$$MRD(T_{adv}, f, \hat{f}) = |MR(T_{adv}, f) - MR(T_{adv}, \hat{f})|. \tag{11}$$

As observed, the MRD solely relies on the output labels from the model without requiring any additional knowledge or information.

4.2. Privacy-Preserving Threshold Judgement

This part presents our protocols for the privacy-preserving evaluation of model similarity using the aforementioned metrics. Firstly, we describe two auxiliary protocols for comparison and absolute value difference in Section 4.2.1, which rely solely on the PCDD and garbled circuits. Subsequently, we delve into the secure evaluation of similarity by employing model robustness in Section 4.2.2. Furthermore, we extend our

techniques to facilitate secure evaluation at the neuron and layer levels, as discussed in Sections 4.2.3 and 4.2.4.

4.2.1. Auxiliary Privacy-Preserving Protocols

Prior to illustrating our proposed protocols, we will first discuss two critical operations that play an indispensable role in these protocols. We will then design the corresponding auxiliary privacy-preserving protocols for each operation. The first operation is comparison, while the second operation involves calculating the absolute value difference.

Given the significance of comparison in our proposed protocols, it is crucial to establish an efficient and privacy-preserving comparison protocol as the foundation for our work. Since we have already assumed that the two cloud servers S_1 and S_2 are non-colluding, we can employ various cryptographic techniques, such as garbled circuits and pure homomorphic encryption (HE), to compare two encrypted values on the server. Regarding HE-based schemes [44,45], it is stated in [46] that evaluating a specific function requires homomorphic operations and bit-wise encryption. However, these processes can result in significant time consumption and impose a high bandwidth burden, rendering them impractical in real-world scenarios. An alternative solution is to utilize garbled circuits, which allow for offline transfer of computationally expensive operations such as oblivious transfer or circuit generation. However, as Evans et al. [47] mentioned, a hybrid scheme that combines both garbled circuits and HE appears more promising in practical scenarios.

Thus, taking into account the aforementioned contexts, we implement the privacy-preserving comparison protocol for our proposed protocols using garbled circuits with the combination of the PCDD. The circuit's structure is depicted in Figure 4. The circuit consists of two types of circuits: SUB, which is used for bit-wise subtraction, and CMP, which facilitates bit-wise comparison [48].

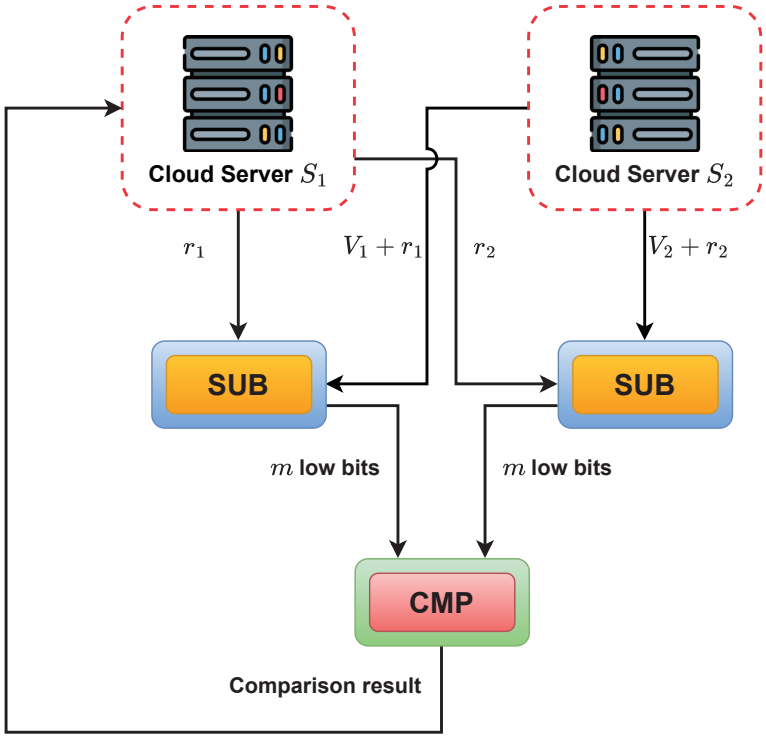


Figure 4. The garbled circuit for privacy-preserving comparison.

Specifically, cloud server S_1 possesses two encrypted values $Enc_{pk}[V_1], Enc_{pk}[V_2]$ that are awaiting comparison. Without loss of generality, we assume that all values, namely V_1 and V_2, have a length of at most s bits. Server S_1 applies masking to the encrypted value $Enc_{pk}[V_i]$, $i \in 1,2$ by utilizing a randomly chosen k-bit ($k < s$) number r_i, $i \in 1,2$

through the additive homomorphic property. The masking operation, as described in [47], ensures a statistical hiding security level of approximately 2^{s-k} for V_i. Subsequently, server S_1 computes the partial decryption of $Enc_{pk}[V_1 + r1]$ and $Enc_{pk}[V_2 + r_2]$ using the corresponding partial strong private keys. The information transmitted to cloud server S_2 comprises a set comprising $Enc_{pk}[V_1 + r_1]$, $Enc_{pk}[V_2 + r_2]$ and their corresponding partial descriptions. Upon receiving these data, server S_2 retrieves the plaintext values $V_1 + r_1$ and $V_2 + r_2$ using its corresponding partial strong private key. Now, in regard to the garbled circuit, server S_1 receives r_1 and r_2 as inputs, while server S_2 receives $V_1 + r_1$ and $V_2 + r_2$ as inputs. Upon completion of the circuit evaluation, the output bit b_{comp} signifies the result of the comparison. If $b_{comp} = 1$, it implies that $V_1 > V_2$, while $b_{comp} = 0$ indicates otherwise. It is worth noting that due to the utilization of the free-XOR technique [49], the cost of XOR gates can be disregarded when considering the overall cost of the circuit evaluation. The details of this protocol are illustrated in Algorithm 1.

Subsequently, utilizing the privacy-preserving comparison protocol described above, it is possible to design a privacy-preserving protocol for calculating the absolute value difference. In particular, without loss of generality, we assume that cloud server S_1 possesses two encrypted values, $Enc_{pk}[V_1]$ and $Enc_{pk}[V_2]$, which are intended for calculating the absolute value difference. Initially, it collaborates with server S_2 to compare $Enc_{pk}[V_1]$ and $Enc_{pk}[V_2]$ using the privacy-preserving comparison protocol. Upon receiving the comparison result, server S_1 employs the additively homomorphic property to compute the absolute value within the ciphertext domain. Specifically, if $V_1 > V_2$, the encrypted absolute value $Enc_{pk}[abs]$ can be calculated as $Enc_{pk}[abs] = Enc_{pk}[V_1 - V_2] = Enc_{pk}[V_1] \cdot (Enc_{pk}[V_2])^{-1}$. In the alternate case, simply interchange the order of V_1 and V_2 while leaving the remaining parts unchanged. The detailed process for this protocol is presented in Algorithm 2.

Algorithm 1 SecureCMP(V_1, V_2)

1: S_1 chooses two k-bit random number r_1, r_2, where $k < s$;
2: S_1 utilizes the additively homomorphic attribute of the PCDD to obtain $Enc_{pk}[V_1 + r_1]$ and $Enc_{pk}[V_2 + r_2]$;
3: S_1 computes the partial decryption for $Enc_{pk}[V_1 + r_1]$ and $Enc_{pk}[V_2 + r_2]$, respectively;
4: S_1 sends $Enc_{pk}[V_1 + r_1]$, $Enc_{pk}[V_2 + r_2]$ and the corresponding partial decryptions to the cloud server S_2;
5: S_2 decrypts the ciphertext and recovers $V_1 + r_1, V_2 + r_2$;
6: S_1 provides r_1, r_2 as the inputs for the garbled circuit, while $V_1 + r_1, V_2 + r_2$ are the inputs from S_2;
7: S_1 receives the comparison result.

Algorithm 2 SecureABSD(V_1, V_2)

1: S_1 compares $Enc_{pk}[V_1]$ and $Enc_{pk}[V_2]$ with the use of Algorithm 1.
2: **if** $V_1 > V_2$ **then**
3: S_1 computes $Enc_{pk}[abs] = Enc_{pk}[V_1] \cdot (Enc_{pk}[V_2])^{-1}$
4: **else**
5: S_1 computes $Enc_{pk}[abs] = Enc_{pk}[V_2] \cdot (Enc_{pk}[V_1])^{-1}$
6: **end if**

4.2.2. Similarity Evaluating Protocol for Robustness

In this part, we present a protocol to facilitate the privacy-preserving evaluation of model similarity using model robustness. It should be noted that calculating the Model Robustness Distance for similarity evaluation involves using Equation (11), which necessitates the computation of absolute value difference. Consequently, we employ the auxiliary protocol introduced in Section 4.2.1 to perform the evaluation while preserving privacy. Initially, both the victim and the suspect encrypt their respective model robustness values

and transmit them to the cloud server S_1 using the public key pk. Subsequently, the two cloud servers collaborate to finalize the similarity evaluation and relay the outcome to the victim. Specifically, the protocol operates as follows:

1. Both victim and suspect leverage the set of adversarial samples T_{adv} to obtain their corresponding model robustness MR according to Equation (10).
2. Then, the victim encrypted the threshold $\delta \cdot T$ and its model robustness value $MR(T_{adv}, f)$ as $Enc_{pk}[\delta \cdot T]$ and $Enc_{pk}[MR(T_{adv}, f)]$ by using the public key pk. The suspect also operates a similar process to obtain $Enc_{pk}[MR(T_{adv}, \hat{f})]$.
3. The victim and the suspect send their encrypted value to the cloud server S_1. After that, S_1 cooperates with the cloud server S_2 with the use of Algorithm 2 to obtain the encrypted MRD result $Enc_{pk}[MRD(T_{adv}, f, \hat{f})]$.
4. S_1 utilizes Algorithm 1 to judge whether the MRD result has reached the threshold from the victim with the cooperation of S_2.
5. Once the comparison result is obtained by S_1, it will be encrypted and returned to the victim.

The details of this protocol are outlined in Algorithm 3. This indicates that data privacy is effectively safeguarded against potential breaches.

Algorithm 3 SecureMRD($T_{adv}, f, \hat{f}, \delta$)

1: The victim computes $MR(T_{adv}, f)$, and the suspect computes $MR(T_{adv}, \hat{f})$ according to Equation (10).
2: The victim uses the PCDD to obtain $Enc_{pk}[MR(T_{adv}, f)]$ and $Enc_{pk}[\delta \cdot T]$. Then, the suspect obtains $Enc_{pk}[MR(T_{adv}, \hat{f})]$.
3: The victim sends $Enc_{pk}[MR(T_{adv}, f)]$ and $Enc_{pk}[\delta \cdot T]$ to the cloud server S_1.
4: The suspect sends $Enc_{pk}[MR(T_{adv}, \hat{f})]$ to S_1.
5: S_1 computes $Enc_{pk}[MRD(T_{adv}, f, \hat{f})]$ with the cooperation of the cloud server S_2 according to Equation (11) and Algorithm 2.
6: S_1 compares $Enc_{pk}[MRD(T_{adv}, f, \hat{f})]$ and $Enc_{pk}[\delta \cdot T]$ according to Algorithm 1.
7: S_1 returns the comparison result.

4.2.3. Similarity Evaluating Protocol for Neuron Distance

This part focuses on the privacy-preserving evaluation of model similarity at the neuron level. In addition to the concerns discussed in Section 4.2.2, there is another challenge of performing a privacy-preserving summation as indicated in Equation (8). Therefore, we exploit the additively holomorphic characteristics of the PCDD to aggregate the absolute differentiation values within the ciphertext domain as follows:

1. Based on the determined set of test samples T_x, the victim calculates the output result for each neuron and encrypts them as $\{Enc_{pk}[\varphi_i^l(x_1)], Enc_{pk}[\varphi_i^l(x_2)], \cdots, Enc_{pk}[\varphi_i^l(x_T)]\}$. Similarly, the suspect also obtains the encrypted set $\{Enc_{pk}[\hat{\varphi}_i^l(x_1)], Enc_{pk}[\hat{\varphi}_i^l(x_2)], \cdots, Enc_{pk}[\hat{\varphi}_i^l(x_T)]\}$.
2. Then, the victim sends the encrypted output values and the encrypted threshold $Enc_{pk}[\delta \cdot T]$ to the cloud server S_1, while the suspect also sends the encrypted output values to S_1.
3. S_1 cooperates with the cloud server S_2 to calculate the absolute value difference for each neuron output between the victim and the suspect (e.g., $Enc_{pk}[absD_t] = Enc_{pk}[|\varphi_i^l(x_t) - \hat{\varphi}_i^l(x_t)|]$), respectively.
4. After that, S_1 adopts the additively homomorphic property to sum up the absolute difference values in the ciphertext domain as $Enc_{pk}[absD] = Enc_{pk}[\sum_{t=1}^{T} absD_t] = \prod_{t=1}^{T} Enc_{pk}[absD_t]$.
5. S_1 compares $Enc_{pk}[absD]$ and $Enc_{pk}[\delta \cdot T]$ with the use of Algorithm 1. Once the comparison result is obtained, it will be encrypted and returned to the victim.

The details of the NOD protocol are presented in Algorithm 4.

Algorithm 4 SecureNOD(T_x, φ_i^l, $\hat{\varphi}_i^l$, δ)

for $x_t \in T_x$ **do**
  The victim computes $\varphi_i^l(x_t)$, and the suspect computes $\hat{\varphi}_i^l(x_t)$.
  The victim uses the PCDD to obtain $Enc_{pk}[\varphi_i^l(x_t)]$ and $Enc_{pk}[\delta \cdot T]$. Then, the suspect obtains $Enc_{pk}[\hat{\varphi}_i^l(x_t)]$.
  The victim sends $Enc_{pk}[\varphi_i^l(x_t)]$ and $Enc_{pk}[\delta \cdot T]$ to the cloud server S_1.
  The suspect sends $Enc_{pk}[\hat{\varphi}_i^l(x_t)]$ to S_1.
  S_1 computes $Enc_{pk}[absD_t] = Enc_{pk}[|\varphi_i^l(x_t) - \hat{\varphi}_i^l(x_t)|]$ according to Algorithm 2.
end for
S_1 computes $Enc_{pk}[absD] = \prod_{t=1}^{T} Enc_{pk}[absD_t]$.
S_1 compares $Enc_{pk}[absD]$ and $Enc_{pk}[\delta \cdot T]$ according to Algorithm 1.
S_1 returns the comparison result.

4.2.4. Similarity Evaluating Protocol for Layer Distance

In this section, we introduce privacy-preserving protocols that facilitate model similarity evaluation at the layer level. Algorithm 5 is devised specifically for LOD, as specified in Equation (9).

Concretely, the protocol performs as follows:

1. First, for each sample in the test set T_x, the victim and the suspect obtain the output of layer l as $f^l(x_t)$ and $\hat{f}^l(x_t)$, respectively. After that, these values will be encrypted and sent to the cloud server S_1. The encrypted threshold $Enc_{pk}[\delta \cdot T]$ is also sent to S_1 by the victim.
2. Then, by utilizing Algorithm 2, S_1 cooperates with the cloud server S_2 to calculate the absolute difference value of $f^l(x_t)$ and $\hat{f}^l(x_t)$ in the ciphertext domain. The result is defined as $Enc_{pk}[absLOD_t]$.
3. Next, S_1 adopts the additively homomorphic property for summing up the absolute difference as $Enc_{pk}[absLOD] = Enc_{pk}[\sum_{t=1}^{T} absLOD_t] = \prod_{t=1}^{T} Enc_{pk}[absLOD_t]$.
4. Finally, S_1 compares $Enc_{pk}[absLOD]$ and $Enc_{pk}[\delta \cdot T]$ with the use of Algorithm 1. Once the comparison result is obtained, it will be encrypted and returned to the victim.

Algorithm 5 SecureLOD(T_x, f^l, $\hat{f}^l$, δ)

for $x_t \in T_x$ **do**
  The victim computes $f^l(x_t)$, and the suspect computes $\hat{f}^l(x_t)$.
  The victim uses the PCDD to obtain $Enc_{pk}[f^l(x_t)]$ and $Enc_{pk}[\delta \cdot T]$. Then, the suspect obtains $Enc_{pk}[\hat{f}^l(x_t)]$.
  The victim sends $Enc_{pk}[f^l(x_t)]$ and $Enc_{pk}[\delta \cdot T]$ to the cloud server S_1.
  The suspect sends $Enc_{pk}[\hat{f}^l(x_t)]$ to S_1.
  S_1 computes $Enc_{pk}[absL_t] = Enc_{pk}[|f^l(x_t) - \hat{f}^l(x_t)|]$ according to Algorithm 2.
end for
S_1 computes $Enc_{pk}[absL] = \prod_{t=1}^{T} Enc_{pk}[absL_t]$.
S_1 compares $Enc_{pk}[absL]$ and $Enc_{pk}[\delta \cdot T]$ according to Algorithm 1.
S_1 returns the comparison result.

5. Experiments

We have implemented a testing framework that ensures privacy preservation using our protocols, and, in this section, we present the experimental results.

5.1. Settings

The system is implemented using the Python, Java, and C programming languages. The core components of the machine learning algorithm and models are implemented in PyTorch, while the additively homomorphic cryptosystem is implemented in Java using the JPBC library. For enhanced performance, we utilize Obliv-C, a C-based framework that incorporates GC implementations with optimization techniques, to construct the garbled circuit (GC) of the framework. The framework is designed to run on the Ubuntu 20.04 LTS operating system, utilizing Python 3.9.12, Java 11, and GCC compiler version 4.8.5. Furthermore, the platform server hosting the framework is equipped with four Nvidia RTX 3090 GPUs and 256 GB of RAM.

Datasets and Model

In our experimental setup, we utilize four real-world datasets: MNIST, CIFAR-10, SVHN, and ImageNet. Subsequently, we consider three commonly used models (LeNet, ResNet, and VGG) in our experimental settings. The details of the models and datasets can be seen in Table 2. We divide each training dataset into two parts, with the first part being used for training the victim model. The samples are divided with a ratio of 50%.

Table 2. Models and datasets in experiments.

Dataset	Size	Model	Parameters	Accuracy
MNIST	60,000	LeNet	107.8 K	97.7%
CIFAR-10	60,000	ResNet	274.4 K	85.2%
SVHN	630,420	ResNet	274.4 K	83.9%
ImageNet	14,197,122	VGG	33.65 M	76.3%

In order to assess the effectiveness of our proposed protocols, we define two categories of models: Questionable Models and Normal Models. The Questionable Models are derived from the victim model through model-stealing techniques such as finetuning, pruning, and extraction. These models are classified as questionable since they infringe upon the victim's copyright, and the PTFCP aims to provide evidential support to assist the victim in asserting their copyright. On the other hand, the Normal Models, while sharing the same architecture as the victim model, are trained using either the second part of the divided dataset independently or the first part with a differing random initialization. Thus, these models serve as the control group to illustrate that the proposed protocols in our PTFCP do not assert ownership over unrelated models. It is important to note that these models are trained using the same settings. Specifically, "Norm-I" refers to models trained with a distinct dataset (the second part), whereas "Norm-II" denotes models trained with varying random initialization. In our experiments, we employ two classical techniques [50,51] to generate adversarial samples.

5.2. Experimental Results

5.2.1. Model Finetuning

Model finetuning, the most commonly employed method in DNN model stealing, has been extensively investigated in numerous studies [30,52,53]. In this section, we evaluate the effectiveness of our PTFCP in defending against model finetuning. Specifically, we consider two common model finetuning techniques for a specified victim model and a small dataset drawn from the same distribution. The first technique, referred to as finetuning the last layer (FineTuning-I), involves freezing all layers except the last one and updating only the parameters of the last layer. The second technique, known as finetuning all layers (FineTuning-II), directly updates all parameters of the model.

To assess the performance of the PTFCP more effectively, we conducted separate experiments under both white-box and black-box settings. The experimental results are presented in Figure 5. In the white-box settings, we can access all intermediate outputs from the layers of both the victim model and the suspect model in a privacy-preserving

manner. Specifically, we utilize two protocols from the PTFCP that operate within the white-box settings: SecureNOD and Secure LOD. It is worth noting that we conducted over 1000 random repetitions of the experiments to obtain more precise estimation accuracy results. As shown in Figure 5a, the majority of estimation accuracy results for the finetuned models exceed 70%. Notably, for the MNIST and CIFAR-10 datasets, the results approach nearly 80%. Furthermore, the performance on normal models (Norm-I and Norm-II) demonstrates the PTFCP's ability to effectively distinguish between stolen models and other normal models.

In the black-box setting, we can only access the output probabilities of the models while preserving privacy. Specifically, we employ the protocol from PTFCP that operates within the black-box settings: SecureMRD. The effectiveness of the black-box protocol in defending against the finetuning attack is illustrated in Figure 5b. Overall, the experimental results demonstrate that PTFCP remains practical in the black-box setting, enabling privacy-preserving model copyright auditing. It is worth noting that the evaluation accuracy results in the white-box setting surpassing those in the black-box setting. This is logical since the protocols in the white-box setting have access to more information, allowing for more accurate decision making.

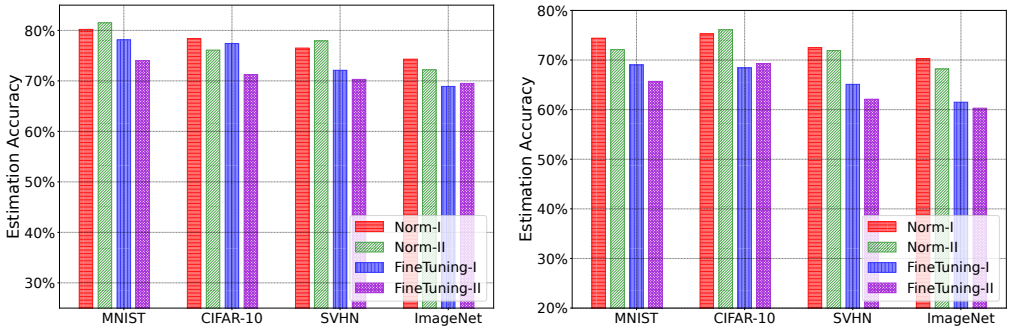
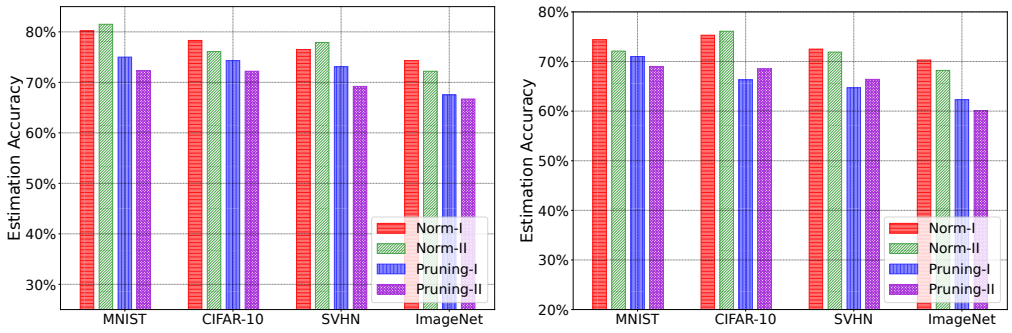


Figure 5. The performance of PTPFC in defending against model finetuning. (**a**) is the evaluation in the white-box setting, while (**b**) is the one in the black-box setting.

5.2.2. Model Pruning

In addition to the model finetuning method, model pruning is another commonly employed technique for stealing valuable DNN models. In this section, we evaluate the effectiveness of PTFCP in defending against model pruning. Utilizing the same configuration for the victim model and datasets, we focus on two distinct types of model pruning attacks, distinguished primarily by the percentage parameter r. In these attacks, the attacker prunes the parameters with the lowest absolute values, based on a percentage ratio of r. To preserve utility, the attacker also conducts finetuning on the pruned model. We conduct experiments using both a low ratio ($r = 25\%$, Pruning-I) and a high ratio ($r = 75\%$, Pruning-II). It should be noted that canonical data augmentation techniques are also employed to enhance the effectiveness of the model pruning attack.

Similar to the finetuning attack, we conducted separate evaluations of the PTFCP in both the white-box and black-box settings. The corresponding experimental results are depicted in Figure 5. The evaluation results in Figure 6a indicate that the majority of the evaluations exceed 70%. Moreover, for the MNIST and CIFAR-10 datasets, the evaluations can reach up to 75%. In the black-box setting (as depicted in Figure 6b), while the evaluation results are slightly lower than those in the white-box setting due to the absence of certain intermediate knowledge, the results demonstrate that the PTFCP can detect stolen models obtained through model pruning attacks. In conclusion, these results demonstrate that the PTFCP is effective in defending against model pruning attacks as well.



(**a**) (**b**)

Figure 6. The performance of PTPFC in defending against model pruning. (**a**) is the evaluation in the white-box setting, while (**b**) is the one in the black-box setting.

5.2.3. Model Extraction

Among the model finetuning, model pruning, and model extraction attacks, the model extraction attack is considered the most significant threat to DNN model copyright protection. Studies have revealed that conventional DNN watermarking methods are ineffective in establishing ownership claims for DNN models when confronted with the model extraction attack [30]. One possible explanation is that watermarks are intended to be inconspicuous elements of learning tasks and do not specifically address the model extraction attack. Consequently, attackers can readily exploit model extraction attacks to remove the watermarks from DNN models. In this section, we evaluate the effectiveness of the PTFCP in defending against the model extraction attack. Specifically, we consider two well-known model extraction attacks from recent state-of-the-art studies. The first attack is called Jacobian-based augmentation (JBA) [54], which involves sampling a set of seeds based on the test dataset and utilizing the Jacobian-based data augmentation technique to generate additional data samples from these seeds. The second attack is known as Knockoff Nets (KnockOff) [55], which employs an auxiliary dataset that shares similar characteristics with the original training dataset used for the victim model.

Figure 7 illustrates the experimental results of the PTFCP in defending against the model extraction attack in both the white-box and black-box settings. Despite a decline in performance compared to the model finetuning and pruning scenarios, the evaluation accuracy in the white-box setting remains above 60%. In the black-box setting, it is expected that the results would be inferior to those in the white-box setting due to the absence of intermediate knowledge from the models. Nevertheless, the majority of the results still reach nearly 60%. Additionally, as indicated in various studies on model extraction [54–56], model extraction attacks are not consistently successful. While it may be challenging to accurately identify poorly performing extracted models, their subpar performance may not pose significant threats to model copyright. Furthermore, it can be observed that as the model extraction attack achieves better performance, the extracted model becomes more similar to the victim model. Thus, the PTFCP presents an effective countermeasure against the primary objective of achieving perfect parameter or functionality matching in model extraction attacks.

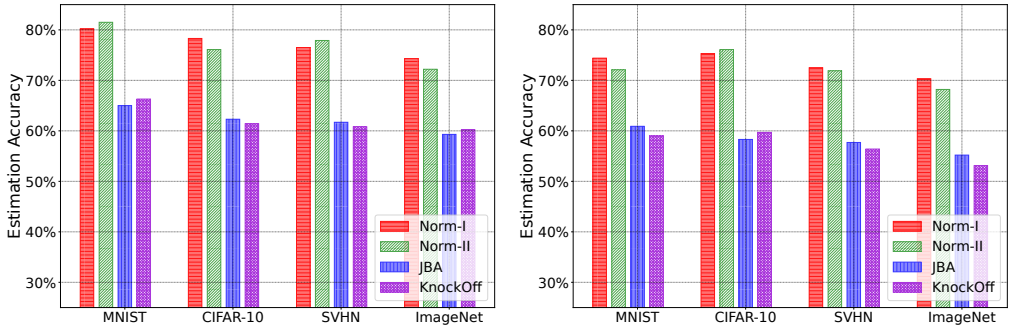


Figure 7. The performance of PTPFC in defending against model extraction. (**a**) is the evaluation in the white-box setting, while (**b**) is the one in the black-box setting.

6. Related Works

6.1. DNN Watermarking

Numerous watermarking technology solutions have been proposed to safeguard the copyright of DNN models. Similar to the techniques utilized in the traditional multimedia domain, existing DNN watermarking approaches comprise two primary steps. The initial step involves embedding, whereas the subsequent step entails verification. In the embedding step, the designated watermark, owned by the creator, is incorporated into the model during the training process. Regarding the verification step, DNN watermarking approaches can be primarily classified into two categories: white-box and black-box. Categorization is based on the level of knowledge accessible during the verification step.

In white-box watermarking, the availability of model information allows for the embedding of a pre-determined signature into the model's parameter spaces using various methods, such as regularization terms. Thus, if the extracted watermark from a suspect model closely matches the owner's signature, the copyright of that model can be readily justified. In contrast, black-box watermarking employs backdoor attack techniques due to the absence of model information. Specifically, the model owner trains the model with a trigger set to compel the model to categorize certain samples into a specific hidden class, acting as a backdoor. The watermark is also injected into the model upon completion of the training procedure. Thus, to verify the copyright of a suspect model, the owner simply needs to query the suspect model using samples from the trigger set and examine whether the predicted result corresponds to the predetermined hidden class.

Despite its effectiveness and simplicity, watermarking inevitably possesses significant drawbacks. In addition to its invasive nature, which interferes with the training process, the performance of watermarking is heavily reliant on the model's ability to memorize. Essentially, watermarking aims to maximize the model's retention of the watermark content, and the degree of memorization determines the robustness of the watermark against attacks. As indicated in numerous studies, watermarking proves effective in countering finetuning and pruning attacks but is susceptible to various model extraction attacks. One critical factor is that model extraction attacks aim to pilfer the model's functionality rather than its entire contents, whereas most watermarks are often task-irrelevant. Despite the aforementioned disadvantages of watermarking, it remains the sole method for owners to embed their signature into their models, surpassing the capabilities of other techniques.

6.2. DNN Fingerprinting

In recent years, numerous researchers have focused on DNN fingerprinting techniques as an alternative method for verifying ownership. Similar to DNN watermarking, DNN fingerprinting involves two steps: fingerprint extraction and fingerprint verification. Importantly, fingerprinting is considered non-invasive, distinguishing it from watermarking schemes. Unlike watermarking, where intervention in the model training process is

required to embed a watermark, fingerprinting techniques directly extract a distinctive feature or attribute from the model to act as a fingerprint. If the fingerprint of a suspect model matches that of the owner's model, the owner can assert ownership of the suspect model.

For example, in [28], the authors introduced IPGuard, a fingerprinting scheme. IPGuard characterizes the fingerprint by utilizing data points located near the classification boundary to capture the model's boundary property. If a suspect model produces identical predictions for a majority of these points, IPGuard identifies it as an unauthorized copy of the owner's victim model. Additionally, in [27], a Conferrable Ensemble Method (CEM) is introduced to capture the overlap between the decision boundary or adversarial subspaces of the victim model and those of the suspect model. Specifically, this CEM generates conferrable adversarial examples, which are a specific type of transferable examples, to facilitate characterization. In [27], the authors demonstrate the robustness of a CEM against common DNN removal attacks, such as finetuning, pruning, and extraction attacks. However, it is incapable of addressing certain adapted attacks, including adversarial learning.

In conclusion, as evidenced, the majority of fingerprinting schemes concentrate on establishing the "uniqueness" of a particular model. However, in practical scenarios, relying solely on or being limited to a specific metric is not practical. Consequently, relying solely on fingerprinting is insufficient to provide convincing evidence or support for verifying model ownership. A potentially promising approach to address this challenge is to combine various existing methods to create a comprehensive mechanism.

6.3. Model Similarity Evaluation

The evaluation of model similarity has emerged as a crucial metric used to assess performance in diverse scenarios, including Machine Unlearning [32] and model copyright protection [30]. Consequently, numerous researchers have dedicated their efforts to this area and proposed a range of schemes.

Wu et al. [29] utilized the Euclidean distance as a measure of similarity between two distinct DNN models in the context of machine unlearning, based on the intuition that the parameter distance reflects their similarity. However, experiments conducted on ResNet by the authors of [33] revealed that relying solely on the l_2-Norm to evaluate DNN model similarity is insufficiently accurate and practical. Specifically, various complex factors such as learning rate and floating-point operations can result in significantly different model parameter outcomes, even when the training datasets and initialization are identical. Additionally, for large DNN models with a substantial number of parameters, the computational costs become impractical in real-world applications [32].

To address the mentioned limitations, Golarkar et al. [31] employ the extensively used the Jensen–Shannon divergence, based on the Kullback–Leibler divergence, to assess the distance between two distinct DNN models in the parameter space. Similarly, in [57], the authors utilize the Jensen–Shannon divergence to quantify the distance between probability distributions of two different DNN models. Another notable metric, known as activation distance, is employed in certain studies [57,58]. It represents the average l_2-Norm distance between probability distributions of two distinct models. Furthermore, to provide a comprehensive evaluation at a broader level, model similarity measurement employs layer-wise distance [41]. This approach computes the distance between two models considering their parameters or the output of each layer.

Recently, Chen et al. [30] introduced a framework known as DeepJudge. DeepJudge combines multiple metrics to provide a comprehensive evaluation of the similarity between two distinct DNN models. Specifically, the framework categorizes six distinct metrics into three main categories. When assessing the similarity between two different DNN models, DeepJudge initially calculates the distance value for each respective metric separately. Subsequently, it determines whether the suspect model exhibits sufficient similarity to the victim model within the specified metric, based on a predetermined threshold. Finally, a straightforward majority voting scheme is employed to determine whether the suspect

model constitutes an illicit copy of the victim model. However, this necessitates the victim granting DeepJudge full access to their DNN model, including model parameters and architecture. Consequently, this approach is impractical in real-world scenarios where concerns about model data privacy have already arisen, as some model owners may be unwilling to provide DeepJudge with such privileges due to the potential risk of private model information leakage.

7. Conclusions

This work presents our PTFCP, a privacy-preserving testing framework designed to protect the copyright of deep learning models in mobile crowdsensing scenarios. Our PTFCP consists of multiple protocols that assess the similarities between two distinct models while maintaining privacy. In contrast to existing schemes, the PTFCP ensures that model utility remains intact throughout the training process while providing defense against various attacks in a privacy-preserving manner. Extensive experiments using real-world datasets were conducted to demonstrate the effective performance of the PTFCP.

Author Contributions: Methodology, D.W. (Dongying Wei); Formal analysis, Z.W.; Investigation, D.W. (Dan Wang); Writing—original draft, Y.M. All authors have read and agreed to the published version of the manuscript.

Funding: This work is supported by the Key Experimental Fund Project for Complex Electronic System Simulation (DXZT-JC-ZZ-2020-013), and the Open Fund of Engineering Research Centre of the Ministry of Education for Intelligent Computing of Complex Energy Systems (ESIC202102).

Data Availability Statement: All data underlying the results are available as part of the article and no additional source data are required.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following notations and abbreviations are used in this manuscript:

Notations	Definitions
f^i	The layer i in the DNN
n_i^l	The i-th neuron in the layer l
$\varphi_i^l(x)$	The output of the neuron n_i^l for the given input x
lcm	Least common multiple
SK	The strong private key
sk	The ordinary private key
pk	The public key
$Enc_{pk}[m]$	The ciphertext of message m encrypted by pk
T_x	The test case set
T_{adv}	The test case set of adversarial samples
δ	The threshold of the similarity
Abbreviations	Definitions
MCS	mobile crowdsensing systems
PTFCP	privacy-preserving testing framework for copyright protection
PCDD	public-key cryptosystem with distributed decryption
DNN	deep neural network
GC	garbled circuits
NOD	neuron outputs distance
LOD	layer outputs distance
MRD	model robustness distance

References

1. Xiao, L.; Jiang, D.; Xu, D.; Su, W.; An, N.; Wang, D. Secure mobile crowdsensing based on deep learning. *China Commun.* **2018**, *15*, 1–11. [CrossRef]

2. Liu, C.H.; Chen, Z.; Zhan, Y. Energy-efficient distributed mobile crowd sensing: A deep learning approach. *IEEE J. Sel. Areas Commun.* **2019**, *37*, 1262–1276. [CrossRef]
3. Tao, X.; Hafid, A.S. DeepSensing: A novel mobile crowdsensing framework with double deep Q-network and prioritized experience replay. *IEEE Internet Things J.* **2020**, *7*, 11547–11558. [CrossRef]
4. Zhang, C.; Luo, X.; Liang, J.; Liu, X.; Zhu, L.; Guo, S. POTA: Privacy-Preserving Online Multi-Task Assignment with Path Planning. *IEEE Trans. Mob. Comput.* **2023**, 1–13. [CrossRef]
5. Dai, Z.; Liu, C.H.; Ye, Y.; Han, R.; Yuan, Y.; Wang, G.; Tang, J. Aoi-minimal uav crowdsensing by model-based graph convolutional reinforcement learning. In Proceedings of the IEEE INFOCOM 2022-IEEE Conference on Computer Communications, Virtual Conference, 2–5 May 2022; IEEE: Piscataway, NJ, USA; pp. 1029–1038.
6. Vahedi, Z.; Mahdavi Chabok, S.J.; Veisi, G. Heterogeneous task allocation in mobile crowd sensing using a modified approximate policy approach. *Int. J. Nonlinear Anal. Appl.* **2023**. [CrossRef]
7. Xu, C.; Song, W. Intelligent Task Allocation for Mobile Crowdsensing With Graph Attention Network and Deep Reinforcement Learning. *IEEE Trans. Netw. Sci. Eng.* **2023**, *10*, 1032–1048. [CrossRef]
8. Ren, H.; Xu, G.; Qi, H.; Zhang, T. PriFR: Privacy-preserving Large-scale File Retrieval System via Blockchain for Encrypted Cloud Data. In Proceedings of the 2023 IEEE 9th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS), New York, NY, USA, 6–8 May 2023; pp. 16–23. [CrossRef]
9. Ren, H.; Li, H.; Liu, D.; Xu, G.; Cheng, N.; Shen, X. Privacy-Preserving Efficient Verifiable Deep Packet Inspection for Cloud-Assisted Middlebox. *IEEE Trans. Cloud Comput.* **2022**, *10*, 1052–1064. [CrossRef]
10. Huang, C.; Wang, W.; Liu, D.; Lu, R.; Shen, X. Blockchain-assisted personalized car insurance with privacy preservation and fraud resistance. *IEEE Trans. Veh. Technol.* **2022**, *72*, 3777–3792. [CrossRef]
11. Zhang, C.; Hu, C.; Wu, T.; Zhu, L.; Liu, X. Achieving Efficient and Privacy-Preserving Neural Network Training and Prediction in Cloud Environments. *IEEE Trans. Dependable Secur. Comput.* **2023**, *20*, 4245–4257. [CrossRef]
12. Liang, H.; Li, Y.; Zhang, C.; Liu, X.; Zhu, L. EGIA: An External Gradient Inversion Attack in Federated Learning. *IEEE Trans. Inf. Forensics Secur.* **2023**, *18*, 4984–4995. [CrossRef]
13. Ren, H.; Li, H.; Liu, D.; Xu, G.; Shen, X.S. Enabling Secure and Versatile Packet Inspection with Probable Cause Privacy for Outsourced Middlebox. *IEEE Trans. Cloud Comput.* **2022**, *10*, 2580–2594. [CrossRef]
14. Hu, C.; Zhang, C.; Lei, D.; Wu, T.; Liu, X.; Zhu, L. Achieving Privacy-Preserving and Verifiable Support Vector Machine Training in the Cloud. *IEEE Trans. Inf. Forensics Secur.* **2023**, *18*, 3476–4291. [CrossRef]
15. Huang, C.; Liu, D.; Yang, A.; Lu, R.; Shen, X. Multi-client secure and efficient dpf-based keyword search for cloud storage. *IEEE Trans. Dependable Secur. Comput.* **2023**, *early access*.
16. Zhang, C.; Zhao, M.; Liang, J.; Fan, Q.; Zhu, L.; Guo, S. NANO: Cryptographic Enforcement of Readability and Editability Governance in Blockchain Database. *IEEE Trans. Dependable Secur. Comput.* **2023**, 1–14. [CrossRef]
17. Zhu, Y.; Cheng, Y.; Zhou, H.; Lu, Y. Hermes attack: Steal {DNN} models with lossless inference accuracy. In Proceedings of the 30th USENIX Security Symposium (USENIX Security 21), Virtual Event, 11–13 August 2021.
18. Rakin, A.S.; Chowdhuryy, M.H.I.; Yao, F.; Fan, D. Deepsteal: Advanced model extractions leveraging efficient weight stealing in memories. In Proceedings of the IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 23–26 May 2022; IEEE: Piscataway, NJ, USA, 2022; pp. 1157–1174.
19. Wang, R.; Ren, J.; Li, B.; She, T.; Lin, C.; Fang, L.; Chen, J.; Shen, C.; Wang, L. Free Fine-tuning: A Plug-and-Play Watermarking Scheme for Deep Neural Networks. *arXiv* **2022**, arXiv:2210.07809.
20. Correia-Silva, J.R.; Berriel, R.F.; Badue, C.; De Souza, A.F.; Oliveira-Santos, T. Copycat CNN: Are random non-Labeled data enough to steal knowledge from black-box models? *Pattern Recognit.* **2021**, *113*, 107830. [CrossRef]
21. Yang, P.; Lao, Y.; Li, P. Robust watermarking for deep neural networks via bi-level optimization. In Proceedings of the IEEE/CVF International Conference on Computer Vision, Virtual Conference, 11–17 October 2021; pp. 14841–14850.
22. Jia, J.; Wu, Y.; Li, A.; Ma, S.; Liu, Y. Subnetwork-Lossless Robust Watermarking for Hostile Theft Attacks in Deep Transfer Learning Models. *IEEE Trans. Dependable Secur. Comput.* **2022**, *early access*.
23. Li, Y.; Wang, H.; Barni, M. A survey of deep neural network watermarking techniques. *Neurocomputing* **2021**, *461*, 171–193. [CrossRef]
24. Barni, M.; Pérez-González, F.; Tondi, B. DNN watermarking: Four challenges and a funeral. In Proceedings of the 2021 ACM Workshop on Information Hiding and Multimedia Security, Virtual Event, 22–25 June 2021; pp. 189–196.
25. Wang, R.; Li, H.; Mu, L.; Ren, J.; Guo, S.; Liu, L.; Fang, L.; Chen, J.; Wang, L. Rethinking the Vulnerability of DNN Watermarking: Are Watermarks Robust against Naturalness-aware Perturbations? In Proceedings of the 30th ACM International Conference on Multimedia, Lisboa, Portugal, 10–14 October 2022; pp. 1808–1818.
26. Wang, Y.; Li, W.; Sarkar, E.; Shafique, M.; Maniatakos, M.; Jabari, S.E. PiDAn: A Coherence Optimization Approach for Backdoor Attack Detection and Mitigation in Deep Neural Networks. *arXiv* **2022**, arXiv:2203.09289.
27. Lukas, N.; Zhang, Y.; Kerschbaum, F. Deep neural network fingerprinting by conferrable adversarial examples. *arXiv* **2019**, arXiv:1912.00888.

28. Cao, X.; Jia, J.; Gong, N.Z. IPGuard: Protecting intellectual property of deep neural networks via fingerprinting the classification boundary. In Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security, Virtual Event, 7–11 June 2021; pp. 14–25.
29. Wu, Y.; Dobriban, E.; Davidson, S. Deltagrad: Rapid retraining of machine learning models. In Proceedings of the International Conference on Machine Learning, PMLR, Online, 3–18 July 2020; pp. 10355–10366.
30. Chen, J.; Wang, J.; Peng, T.; Sun, Y.; Cheng, P.; Ji, S.; Ma, X.; Li, B.; Song, D. Copy, right? A testing framework for copyright protection of deep learning models. In Proceedings of the 2022 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 23–26 May 2022; IEEE: Piscataway, NJ, USA, 2022; pp. 824–841.
31. Golatkar, A.; Achille, A.; Soatto, S. Forgetting outside the box: Scrubbing deep networks of information accessible from input-output observations. In Proceedings of the European Conference on Computer Vision, Glasgow, UK, 23–28 August 2020; Springer: Berlin/Heidelberg, Germany, 2020; pp. 383–398.
32. Nguyen, T.T.; Huynh, T.T.; Nguyen, P.L.; Liew, A.W.C.; Yin, H.; Nguyen, Q.V.H. A survey of machine unlearning. *arXiv* **2022**, arXiv:2209.02299.
33. Jia, H.; Yaghini, M.; Choquette-Choo, C.A.; Dullerud, N.; Thudi, A.; Chandrasekaran, V.; Papernot, N. Proof-of-learning: Definitions and practice. In Proceedings of the 2021 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 24–27 May 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 1039–1056.
34. Xu, G.; Li, H.; Liu, S.; Yang, K.; Lin, X. Verifynet: Secure and verifiable federated learning. *IEEE Trans. Inf. Forensics Secur.* **2019**, *15*, 911–926. [CrossRef]
35. Xu, G.; Li, H.; Zhang, Y.; Xu, S.; Ning, J.; Deng, R.H. Privacy-preserving federated deep learning with irregular users. *IEEE Trans. Dependable Secur. Comput.* **2020**, *19*, 1364–1381. [CrossRef]
36. Yu, H.; Jia, X.; Zhang, H.; Yu, X.; Shu, J. PSRide: Privacy-preserving shared ride matching for online ride hailing systems. *IEEE Trans. Dependable Secur. Comput.* **2019**, *18*, 1425–1440. [CrossRef]
37. Guan, Y.; Lu, R.; Zheng, Y.; Zhang, S.; Shao, J.; Wei, G. Achieving Efficient and Privacy-Preserving (,)-Core Query over Bipartite Graphs in Cloud. *IEEE Trans. Dependable Secur. Comput.* **2022**, *20*, 1979–1993. [CrossRef]
38. Bresson, E.; Catalano, D.; Pointcheval, D. A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications. In Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, 30 November–4 December 2003; Springer: Berlin/Heidelberg, Germany, 2003; pp. 37–54.
39. Liu, X.; Qin, B.; Deng, R.H.; Lu, R.; Ma, J. A privacy-preserving outsourced functional computation framework across large-scale multiple encrypted domains. *IEEE Trans. Comput.* **2016**, *65*, 3567–3579. [CrossRef]
40. Yao, A.C.C. How to generate and exchange secrets. In Proceedings of the 27th Annual Symposium on Foundations of Computer Science (SFCS 1986), Washington, DC, USA, 27–29 October 1986; IEEE: Piscataway, NJ, USA, 1986; pp. 162–167.
41. Tarun, A.K.; Chundawat, V.S.; Mandal, M.; Kankanhalli, M. Fast yet effective machine unlearning. *arXiv* **2021**, arXiv:2111.08947.
42. Cao, Y.; Yang, J. Towards making systems forget with machine unlearning. In Proceedings of the 2015 IEEE Symposium on Security and Privacy, San Jose, CA, USA, 17–21 May 2015; IEEE: Piscataway, NJ, USA, 2015; pp. 463–480.
43. Mehrabi, N.; Morstatter, F.; Saxena, N.; Lerman, K.; Galstyan, A. A survey on bias and fairness in machine learning. *ACM Comput. Surv. (CSUR)* **2021**, *54*, 1–35. [CrossRef]
44. Bost, R.; Popa, R.A.; Tu, S.; Goldwasser, S. Machine learning classification over encrypted data. *Cryptol. ePrint Arch.* **2014**. [CrossRef]
45. Damgard, I.; Geisler, M.; Kroigard, M. A correction to 'efficient and secure comparison for on-line auctions'. *Int. J. Appl. Cryptogr.* **2009**, *1*, 323–324. [CrossRef]
46. Damgard, I.; Geisler, M.; Kroigard, M. Homomorphic encryption and secure comparison. *Int. J. Appl. Cryptogr.* **2008**, *1*, 22–31. [CrossRef]
47. Evans, D.; Huang, Y.; Katz, J.; Malka, L. Efficient privacy-preserving biometric identification. In Proceedings of the 17th conference Network and Distributed System Security Symposium, NDSS, San Diego, CA, USA, 6–9 February 2011; Volume 68, pp. 90–98.
48. Kolesnikov, V.; Sadeghi, A.R.; Schneider, T. Improved garbled circuit building blocks and applications to auctions and computing minima. In Proceedings of the International Conference on Cryptology and Network Security, Kanazawa, Japan, 12–14 December 2009; Springer: Berlin/Heidelberg, Germany, 2009; pp. 1–20.
49. Kolesnikov, V.; Schneider, T. Improved garbled circuit: Free XOR gates and applications. In *Proceedings of the International Colloquium on Automata, Languages, and Programming*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 486–498.
50. Carlini, N.; Wagner, D. Towards evaluating the robustness of neural networks. In Proceedings of the 2017 IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA, 22–26 May 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 39–57.
51. Madry, A.; Makelov, A.; Schmidt, L.; Tsipras, D.; Vladu, A. Towards deep learning models resistant to adversarial attacks. *arXiv* **2017**, arXiv:1706.06083.
52. Adi, Y.; Baum, C.; Cisse, M.; Pinkas, B.; Keshet, J. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In Proceedings of the 27th USENIX Security Symposium (USENIX Security 18), Baltimore, MD, USA, 15–17 August 2018; pp. 1615–1631.
53. Uchida, Y.; Nagai, Y.; Sakazawa, S.; Satoh, S. Embedding watermarks into deep neural networks. In Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval, Bucharest, Romania, 6–9 June 2017; pp. 269–277.

54. Papernot, N.; McDaniel, P.; Goodfellow, I.; Jha, S.; Celik, Z.B.; Swami, A. Practical black-box attacks against machine learning. In Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, Abu Dhabi, United Arab Emirates, 2–6 April 2017; pp. 506–519.
55. Orekondy, T.; Schiele, B.; Fritz, M. Knockoff nets: Stealing functionality of black-box models. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 15–20 June 2019; pp. 4954–4963.
56. Yuan, X.; Ding, L.; Zhang, L.; Li, X.; Wu, D.O. Es attack: Model stealing against deep neural networks without data hurdles. *IEEE Trans. Emerg. Top. Comput. Intell.* **2022**, *6*, 1258–1270. [CrossRef]
57. Chundawat, V.S.; Tarun, A.K.; Mandal, M.; Kankanhalli, M. Can Bad Teaching Induce Forgetting? Unlearning in Deep Networks using an Incompetent Teacher. *arXiv* **2022**, arXiv:2205.08096.
58. Golatkar, A.; Achille, A.; Ravichandran, A.; Polito, M.; Soatto, S. Mixed-privacy forgetting in deep networks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Virtual Conference, 19–25 June 2021; pp. 792–801.