A Robustness-Assured White-Box Watermark in Neural Networks

Peizhuo Lv, Pan Li, Shengzhi Zhang, Kai Chen*, *Member, IEEE,* Ruigang Liang, Hualong Ma, Yue Zhao, and Yingjiu Li, *Member, IEEE*

Abstract—Recently, stealing highly-valuable and large-scale deep neural network (DNN) models becomes pervasive. The stolen models may be re-commercialized, e.g., deployed in embedded devices, released in model markets, utilized in competitions, etc, which infringes the Intellectual Property (IP) of the original owner. Detecting IP infringement of the stolen models is quite challenging, even with the white-box access to them in the above scenarios, since they may have experienced fine-tuning, pruning, functionality-equivalent adjustment to destruct any embedded watermark. Furthermore, the adversaries may also attempt to extract the embedded watermark or forge a similar watermark to falsely claim ownership. In this paper, we propose a novel DNN watermarking solution, named $HufuNet$, to detect IP infringement of DNN models against the above mentioned attacks. Furthermore, HufuNet is the first one theoretically proved to guarantee robustness against fine-tuning attacks. We evaluate HufuNet rigorously on four benchmark datasets with five popular DNN models, including convolutional neural network (CNN) and recurrent neural network (RNN). The experiments and analysis demonstrate that HufuNet is highly robust against model fine-tuning/pruning, transfer learning, kernels cutoff/supplement, functionality-equivalent attacks and fraudulent ownership claims, thus highly promising to protect large-scale DNN models in the real world.

Index Terms—Watermark, Intellectual Property, Robustness, Deep learning Models

1 Introduction

The advance of artificial intelligence and machine learning in recent years has driven the broad adoption of deep neural networks (DNNs) in numerous applications such as computer vision [2], [3], [4], natural language processing [5], [6], [7], and speech recognition [8], [9]. DNNs are valuable intellectual property (IP) of their owners due to the tremendous amount of expertise, effort, and computing power involved in designing, training, validating, and commercializing them. Such IP should be protected according to the law. For example, the European Patent Office has recently amended its "Guidelines for Examination" on how patents related to artificial intelligence and machine learning technologies should be assessed [10]. Attackers with full access to the structures and parameters of DNN models may steal and possibly modify them to fraudulently claim ownership. Such IP stealing may happen in various real-world settings such as: (i) owners outsource the training of their DNN models to third parties; (ii) owners upload their DNN models and services to cloud service providers; (iii) owners license their DNN models and services to third parties; etc. Attackers may achieve their goals via malware infection [11], [12], insider threats [13], [14], industrial espionage, and many other ways, leading to serious IP infringement and significant economic loss to owners.

It is often to see scenarios that the stolen models get commercialized or released, with possible access to their parameters and structure, i.e., white-box scenario. First, in the scenario of embedded devices such as smartphones, robots and drones, white-box models are usually deployed directly on the embedded products [15], [16], [17], [18], which enables the original model owners to access the suspicious models in a white-box manner to detect Intellectual Property (IP) infringement. Second, the administrator of a neural network model market, such as ModelZoo [19], ONNX [20], and Hugging Face [21] has white-box access to all the models in the market. To maintain a good ecosystem, i.e., encouraging development and penalizing plagiarism, white-box approaches used to detect IP infringement are highly demanded by model developers and market administrators. Third, the organizers of a model competition can access the models from all participants in a white-box manner, and they often encourage originality, rather than plagiarism. For example, the 2021 VIPriors Image Classification Challenge [22] does not allow participants to use any pre-trained checkpoint or any pre-trained backbone. Hence, white-box watermarking is also highly desired by the organizers to testify the originality of the submitted models. Last but not least, white-box models are generally preferred in business environments where transparency is highlighted [23], [24] or trustworthy AI is desired [25], [26]. Verifying the ownership of such models in a white-box manner before using them prevents the enterprise from getting into legal issues.

Watermarking, a process of hiding digital information in carrier data, has been used for media rights management [27], [28] and software ownership protection [29], [30]. It has also been developed for the IP protection of DNNs in recent years. Most white-box watermarking approaches embed a watermark into parameters [31], [32], [33] or the activations [34] of several layers in the protected model, and extract it based on the architecture and parameters of the suspect models during ownership verification. However, recent works [31], [35], [33] show that functionality-equality attacks, i.e.,

- *P. Lv, P. Li, K. Chen, R. Liang, H. Ma, Y. Zhao are with SKLOIS, Institute of Information Engineering, Chinese Academy of Sciences, China, and School of Cyber Security, University of Chinese Academy of Sciences, China. K. Chen is also with the Beijing Academy of Artificial Intelligence, China. E-mail: ({lvpeizhuo, lipan, chenkai, liangruigang, mahualong, zhaoyue}@iie.ac.cn).*
- *S. Zhang is with Department of Computer Science, Metropolitan College, Boston University, USA. E-mail:shengzhi@bu.edu.*
- *Y. Li is with the Computer and Information Science Department, University of Oregon, Eugene, OR 97403. E-mail: yingjiul@uoregon.edu.*
- ** Correponding author: Kai Chen (chenkai@iie.ac.cn).*

adjusting the structure or parameters of a model, can eliminate such embedded watermark without significant performance loss or dramatic change to the protected model.

Challenges in watermarking DNNs. First, watermarks could be eliminated/undermined by attackers via fine-tuning, pruning, functionality-equality attacks, or kernels cutoff/supplement, by which attackers adjust model parameters or the number of kernels but maintain the performance of the watermarked models. Second, the change of model parameters may effectively remove the embedded watermarks as shown in previous studies on DNN watermarking techniques [36], [31], [37], [38], [34] and none of the existing watermarking methods can guarantee the robustness against the model fine-tuning attack so far. Third, it is challenging to design a robust watermark without significantly sacrificing the performance of the original models. Strong robustness tends to embed a large amount of watermark information into DNN models, thus downgrading the watermarked models' performance to an unacceptable level. In specific applications such as autonomous driving, even 1% of performance loss is intolerable. It is by no means trivial to find a way of embedding enough watermark information for strong robustness while keeping the high fidelity of DNN models. Last, it is challenging to prevent attackers from discovering the embedded watermark or forging new watermarks to falsely claim IP infringement over the watermarked models.

HufuNet. To address these challenges, we propose $HufuNet$, a novel robustness-assured white-box watermark for DNN models. HufuNet approach can be divided into three phases: watermark generation, watermark embedding, and IP infringement detection.

In the watermark generation phase, a neural network consisting of Encoder and Decoder with a small number of parameters, called HufuNet, is trained to yield a high fidelity between input and output when tested. The set of training samples are locally saved and used later in the IP infringement detection phase. After HufuNet is trained and tested, it is split into two pieces, with the left piece (Encoder) as a watermark embedded into a DNN model and the right piece (Decoder) kept by the model owner as a secret for IP infringement detection.

In the watermark embedding phase, each parameter of the Encoder of HufuNet is embedded into the target DNN model consisting of many parameters, based on a secret key and the Decoder's corresponding (mirror symmetry) parameter. After the Encoder of HufuNet is embedded, the target DNN model is trained with its dataset, during which its parameters are updated based on a loss function designed to guarantee the robustness of the embedded watermark against fine-tuning and the performance of the target DNN model. The robustness against fine-tuning attack is assured through the design of our loss function, which constrains the performance change of the DNN-watermarked caused by fine-tuning should be larger than that of HufuNet. Intuitively, such constraint implicitly indicate that the DNN-watermarked will become useless before HufuNet crashes (i.e., IP infringement detection failure) upon fine-tuning attacks. We also theoretically prove that such a constraint always holds given the loss function being used to train the DNN-watermarked.

In the IP infringement detection phase, the model owner extracts the embedded watermark from a suspect DNN model based on the secret key and the Decoder's corresponding parameters and recovers the Encoder of HufuNet accordingly. Next, the owner merges Decoder with the recovered Encoder to form a whole HufuNet, and tests the merged HufuNet on the set of training samples. If the performance of the merged HufuNet decreases no more than a predetermined threshold, we consider the suspect model infringes the IP of the original owner.

HufuNet is evaluated rigorously on five popular DNN models (including CNN and RNN models) and four benchmark datasets. HufuNet achieves state-of-the-art performance in the following metrics. First, HufuNet achieves excellent fidelity. The accuracy of the watermarked models is almost the same as that of their non-watermarked versions, with the differences from -0.89% to -0.14% for the same tested model under the same training settings. Second, HufuNet is robust against typical attacks against DNN watermarks, such as fine-tuning, transfer learning, pruning, kernels cutoff/supplement, functionality-equivalent attacks (including parameter adjustment and structure adjustment). HufuNet is demonstrated to be robust against synthesized attacks in which several attacks are combined to simulate adversaries' best effort. In all these attacks, HufuNet can be retrieved accurately for detecting IP infringement unless these attacks drastically downgrade the accuracy of the watermarked models and thus nullify the purpose of detecting IP infringement of the attacked models. Third, the additional computational overhead introduced by our watermark is acceptable since the training is a one-time effort, with the average overhead of 95 minutes in absolute value and 27.93% in percentage. Fourth, HufuNet is secure against fraudulent ownership claims. It is infeasible for an adversary to retrieve the embedded Encoder or forge a valid HufuNet to falsely claim IP infringement against a watermarked model. It is also infeasible for a legitimate owner to falsely detect IP infringement over an innocent model that does not belong to the owner. Furthermore, the adversary can hardly learn the existence of the embedded Encoder in a DNN model by examining the distributions of model parameter values and their gradients. Last but not least, HufuNet is more robust against functionality-equivalent, kernels cutoff/supplement and watermark forgery attacks, compared with existing white-box watermark approaches, e.g., [31], [32], [34], [33].

Contributions. Our main contributions are outlined below:

- *New technique.* We propose a new DNN watermarking approach, which embeds a piece of the watermark (i.e., Encoder) into a DNN model and withholds the other piece (i.e., Decoder) for IP infringement detection. To the best of our knowledge, this is the first approach that splits a watermark into two pieces and later combines them for detecting IP infringement. Meanwhile, we theoretically prove the robustness of our HufuNet against fine-tuning attacks.
- *Implementation and evaluation.* We implement our watermark HufuNet and rigorously evaluate it using both CNN and RNN models. Experimental results show that our new DNN watermarking approach can simultaneously achieve high fidelity, robustness, adaptiveness, security, integrity, and stealthiness. We release our HufuNet framework to the community [39], together with the training and testing datasets of HufuNet, contributing to the IP protection of neural network models.

2 Background and Related Work

2.1 Watermarking Neural Networks

Digital watermarking is a content-based, information hiding technique for embedding/detecting digital information into/from carrier data (e.g., multimedia, documents, software, etc.). It is desired that the embedded watermark should not impact the regular use of carrier data, and be challenging to be detected or removed. Recently, digital watermarking has been applied to protect DNNs and can be classified into white-box and black-box approaches.

White-box Watermarking. The white-box approaches assume that the owner of a DNN model can access the "internals" of a suspect model to verify the ownership of the model.

Uchida et al. [31] embed a T-bit vector $b \in \{0,1\}^T$ into the parameter space $W \in \mathbb{R}^M (M = S \times S \times D)$ of one layer of the protected model, where S and D denote the kernel size and the input dimension of the layer respectively. Meanwhile, the model owner keeps a secret matrix $X \in \mathbb{R}^{T \times M}$ locally, which is used to embed/extract the watermark b into/from $w \in W$. They extend the original loss $E_o(w)$ function to be: $E(w) = E_o(w) + \lambda E_R(w)$, by introducing a parameter regularizer $E_R(w)$. As an embedding regularizer, $E_R(w)$ utilizes a cross entropy function as below: $E_R(w) = -\sum_{j=1}^{T}(b_j log(y_j) + (1-b_j) log(1-y_j))$, where $y_j = \sigma(\sum_i X_{ji} \cdot w_i)$ and σ is the sigmoid function: $\sigma(x) = \frac{1}{1+exp(-x)}$. When extracting a watermark b' from the parameters $w \in W$, the j-th bit b'_j of the watermark is extract as:

$$b'_j = s(X_j \cdot w),\ where\ s(x) = \begin{cases} 1, & x \geq 0 \\ 0, & else \end{cases} \tag{1}$$

After extracting the watermark b', the owner can compare it with the original embedded watermark b by measuring the Bit Error Rate (BER), and only a small BER can claim ownership over the model. However, such an approach is vulnerable to statistical analysis attack [40], because the variance of the parameter values of the layer where the watermark is embedded is significantly greater than those of other layers without the watermark. Furthermore, such approach is also vulnerable to functionality-equivalent attack, kernels cutoff/supplement attack and watermark forgery attack as shown in Section 5.8.

DeepMarks [32] embeds a watermark into the probability density function of the weights of a protected model and extracts it in a similar way as using Equation (1). Then, a correlation score is calculated by computing the dot product between the extracted watermark and the owner's signature to verify ownership. Rouhani et al. [34] embeds a watermark message into the probability density function of the activation values at certain layers of a protected model. Based on [31], Liu et al. [33] improve the robustness of the watermark by greedily selecting a set of important model parameters as the watermark, so the impact caused by the parameter changes can be reduced. However, Lukas et al. [35] demonstrate that these white-box watermarks are vulnerable to weight shifting attack, a specific kind of functionality-equivalent attack. In contrast, our HufuNet is embedded into nearly every layer of the protected model and occupies only a small portion of its parameter space, so the variances between the layers with and without HufuNet are not significantly different. Meanwhile, HufuNet demonstrates security, adaptiveness against functionality-equivalent attack, and robustness against kernels cutoff/supplement attack as evaluated in Section 5.

Black-box Watermarking. Black-box watermarking approaches mostly rely on probing a suspect model for ownership verification, thus relaxing the requirement of the internal knowledge of the suspect model. Most black-box watermarking schemes are implemented based on embedding backdoors [41], [42] into the target DNN models, which is to deliberately train the model using a training dataset that is augmented with certain specific inputs (i.e., triggers) and their desired labels (typically wrong labels). Such mapping between the specific inputs and their desired labels, considered a backdoor and used as a watermark, should be selected uniquely by the model's owner and thus demonstrate its different behavior. Adi et al. [36] firstly embedded a set of abstract images with predefined labels into DNN models to build the trigger set, which acted as digital watermarks (only known to model owners) against the suspect models. Zhang et al. [37] enriched the training dataset by adding content, noise, or irrelevant class examples as triggers and leveraged them as model watermarks. Then, Li et al. [43] built watermarks by integrating a specific logo into the original samples as a blind watermark and adjusting their labels. And other black-box watermark [38], [44], [45] approaches are also proposed. These approaches cannot give a theoretical guarantee to robust against fine-tuning attacks, and still cannot deal with the fraudulent claims of ownership since attackers can still pretend to be owners by searching adversarial samples and utilizing them to build their own trigger sets.

2.2 Counterattacks Against Watermarking

Uchida et al. [31] proposed to fine-tune or prune the DNN model to modify its parameters, thus removing the embedded watermark. Wang et al. [40] proposed to reveal any watermark based on the change of statistical distribution of parameter values of the layer where the watermark is embedded. Yang et al. [46] demonstrated that distillation-based attacks could be practical to perform watermark elimination, based on the premise that the adversaries have plenty of data to fine-tune the model. Chen et al. [47] showed that the well-designed fine-tuning method can remove the watermark without compromising the functionality of the model when using a significant learning rate and unlabeled data. Hitaj et al. [48] proposed evasion attacks against backdoor watermarks, which evade the owner's verification, thereby preventing the identification of model stealing. Backdoor reconstruction approaches can also be applied to attack the backdoor watermarks. Wang et al. [49] proposed Neural Cleanse to detect whether a DNN model has been backdoored or not by reversing the triggers. Liu et al. [50] proposed Artificial Brain Stimulation (ABS) to scan a DNN model to determine whether it is backdoored and generate Trojan triggers by reverse-engineering the compromised neuron. The attackers can use these approaches to reconstruct the triggers embedded into the watermarked DNN and claim ownership maliciously.

3 Problem Statement

3.1 A Motivation Example

As the advance of AI, neural networks are involved in various complicated tasks, such as autonomous driving, speech recognition, medical diagnosis, etc. Such tasks typically rely on DNN models trained on a large amount of data using intensive GPU resources. For instance, training BERT [51], a language understanding model, takes 3.3 days, i.e., 79.2 hours, on 4 DGX-2H servers with 64 Tesla V100 GPUs, and receives a bill of $3,751-$12,571 from the cloud computing platform [52]. Moreover, it is estimated that training GPT-3, an autoregressive language model, would cost around $12 million [53]. Hence, protecting the IP of such models is in demand.

Consider a motivation example where Alice develops a valuable DNN model DNN_{Alice}. Realizing the necessity of IP protection, Alice embeds a digital watermark into the model. Then, Alice releases DNN_{Alice} on a model market such as ModelZoo [19] and ONNX [20]. Now consider an adversary Eve who has full access to the structure and parameters of DNN_{Alice}. Realizing the commercial value of DNN_{Alice}, Eve steals the model and also performs various changes on DNN_{Alice} such as fine-tuning and pruning, aiming to destruct Alice's watermark in DNN_{Alice}. Moreover, Eve may embed his/her own watermark on the stolen

model before releasing it on the model market for profit[1]. Even though Alice can access such stolen model as a white box, e.g., by downloading it from the model market, it is still challenging for Alice to detect IP infringement of the stolen model in the presence of Eve's attacks.

3.2 Threat Model

As discussed in Section 1, stolen models might be commercialized or released, with possible access to their structures and parameters. In this paper, we consider such a white-box scenario to detect IP infringement of the stolen model. We assume that the adversaries, such as prowlers of neural networks, are proficient in machine learning and watermarking, thus applying the following machine learning techniques to infringe on the IP of a watermarked DNN.

Entire model retraining. The most effective method to destruct an embedded watermark in a model is to retrain it completely. However, since such entire model retraining typically requires similar expertise, effort, training data, and computing power as training their model from scratch, adversaries would rather train their model than steal to avoid potential legal issues. Hence, it is safe to assume the adversaries will not retrain the entire model for destructing the embedded watermark.

Model extraction. Model extraction can be used to train a substitute model by querying the stolen model with the collected or synthesized unlabeled inputs, with the embedded watermark removed in the substitute model. However, such an attack typically requires a large number of queries and a dataset to train the substitute model to achieve similar performance as the target model [59], [60], [61]. Our watermark is not robust against such an attack, and we discuss this limitation in Discussion section.

Fine-tuning and Transfer Learning. Fine-tuning is typically used to optimize an already-trained DNN model to reinforce its performance or augment it with additional tasks. Transfer learning applies the knowledge gained when solving one problem to a different but related problem. Both fine-tuning and transfer learning involve an update to the parameter values in the model, thus possibly destroying the embedded watermark. Generally, the scale of fine-tuning and transfer learning is flexible, depending on the training dataset and computing power available. Therefore, adversaries can tune the stolen model according to their capability to destruct the watermark.

Pruning. Model pruning is commonly used for model compression to produce smaller, more memory-efficient and power-efficient models with negligible loss of performance. Typically, it prunes less-connected, unimportant neurons in the neural network by zeroizing them without changing the network structure. Since adversaries lack training power, model pruning can be used to destruct the embedded watermark in their stolen model since it does not involve any retraining.

Functionality-equivalent attack. Without requiring any training resources or training dataset, we consider two kinds of functionality-equivalent attacks. One is adjusting the DNN at the level of parameter, and the other is adjusting the DNN at the level of structure. First, at the level of parameter adjustment (referring to [35]), we suppose the stolen DNN model is with ReLU as the activation function. In that case, the attackers can scale the weights of all the neurons on one layer by a positive constant $c>0$, and scale the next layer by $1/c$ to get a functionality-equivalent model (called parameter adjustment in this paper) to destruct embedded watermarks. Note that the parameter adjustment attack does not work on LSTM, since it uses sigmoid as the activation function. Second, at the level of structure adjustment (referring to [31]), reordering the sequence of output channels on one convolution layer and adjusting the corresponding sequence of input channels on the next convolution layer can obtain a functionality-equivalent model (called channel reordering in this paper). Moreover, adversaries can also randomly choose two adjacent layers of their stolen model with matrices A and B. Then they increase the output dimension of A (by increasing its output channels) to get A', and B to get B', with the restriction of $(A'B')x = (AB)x$, where x is the feature map propagated in neural networks (named channel expansion).

Kernels cutoff and supplement. The adversary can cut off (altogether remove some kernels, rather than zeroizing them as pruning) or supplement some extra kernels in the stolen model to obtain a new model with similar performance but a different number of kernels and structure. To ensure similar performance as the stolen model, the adversary should cut off or supplement kernels carefully: if some output channels (kernels) of one layer are cut off or supplemented, the corresponding input channel of the next layer should also be cut off or supplemented. Besides, since kernels cutoff will reduce the model's performance, the adversary may not want to cut off too many kernels from the stolen model. The adversary can also leverage kernels cutoff or supplement to destruct embedded watermarks.

4 Approach

Figure 1 shows the workflow of HufuNet in three phases: watermark generation, watermark embedding, and IP infringement detection. During the watermark generation phase, a model owner firstly trains a unique neural network based on encoder-decoder architecture, HufuNet, on a training dataset D_s and divides its parameter space into two pieces: $Embedded\ Piece\ of\ HufuNet\ (EPH, i.e., encoder)$ as the watermark, $Secret\ Piece\ of\ HufuNet\ (SPH, i.e., decoder)$ as the local secret. During the watermark embedding phase, the owner injects the watermark EPH into the parameter space of his/her DNN model, i.e., DNN-to-be-protected, for IP infringement detection. Then, the DNN-to-be-protected is trained using our novel loss function with robustness assured on a large training set D_t, thus becoming DNN-watermarked and ready to be released.

During the IP infringement detection phase, the owner extracts a watermark $EPH_{retrieved}$ from a suspect DNN model, and merges it with his/her local SPH to generate a neural network model $HufuNet_{combined}$. Considering the suspect model may have been adjusted in a functionality-equivalent manner to destruct any embedded watermark, we also propose the model restore approach to accommodate the adjustment before retrieving the watermark from the suspect model. Then, the owner can test $HufuNet_{combined}$ on the dataset D_s and consider the suspect model infringes the IP of the original owner if the suspect model is embedded with the watermark EPH. Following Kerckhoffs' principle, we assume the watermark embedding algorithm and the IP infringement detection algorithm are public. While the adversaries may forge HufuNet from their own training dataset[2] D_s,

1. The attacking scenario we considered is similar to the plagiarism in smartphone application markets [54], [55], [56], [57], [58].

2. Note that it is not necessary to standardize D_s (used to test HufuNet) as a public dataset, since it is almost impossible for attackers to ensure the fidelity between inputs and outputs by forging dataset D_s only. Thus, the attackers may train their HufuNet by building their D_s and embedding EPH to falsely claim IP infringement over the watermarked models. We discuss such scenarios in detail in Section 4.3.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2023.3242737

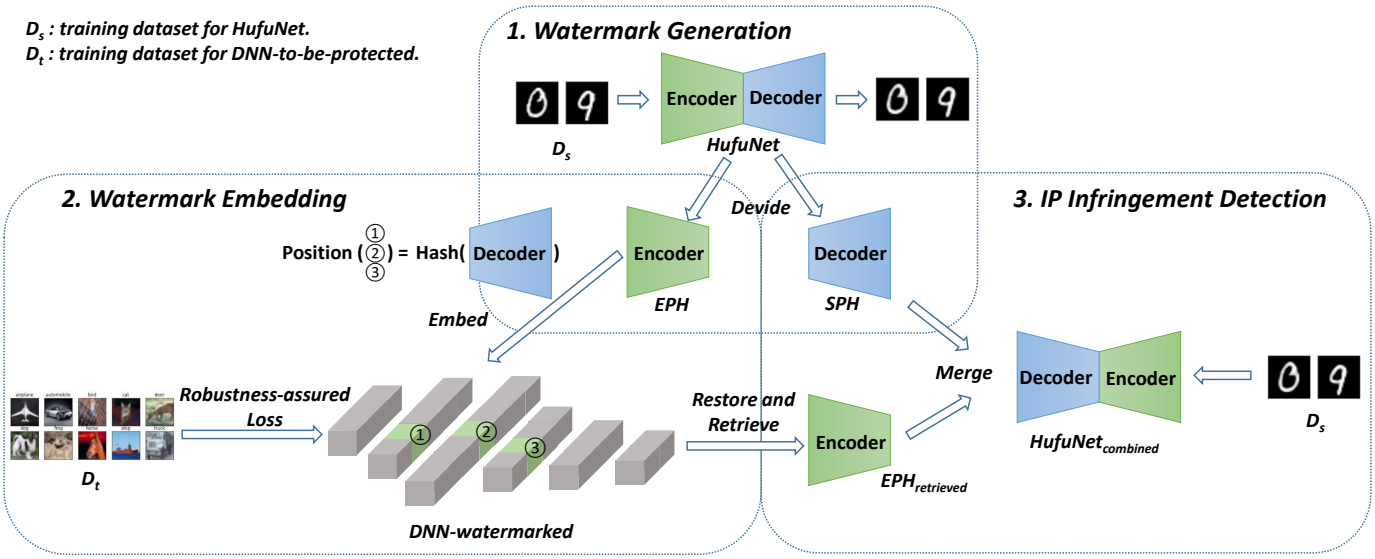


Fig. 1: Workflow of HufuNet

Algorithm 1 Embedding Algorithm

Input: f_t: DNN-to-be-protected; N: number of parameters in f_t; EPH: all parameters in EPH to be embedded; SPH: all parameters in SPH to hash EPH to embed; n: number of parameters in EPH; $bitmap$: a bitmap to keep track of the usage of each position in f_t; key: a secret key

Output: f_{wm}: DNN-watermarked

1: $bitmap = NULL$, $f_{wm} = f_t$
2: **for** i in $(1, n)$ **do**
3: $\quad position = HMAC(key, SPH_i \oplus i)\%N$
4: $\quad$ **while** $bitmap_{position}$ **do**
5: $\quad\quad position = ++position\%N$
6: $\quad$ **end while**
7: $\quad f_{wm} = EMBED(EPH_i, position, f_{wm})$
8: $\quad bitmap_{position} = 1$
9: $\quad ++i$
10: **end for**
11: **return** f_{wm}

$EMBED$ is the function to embed a parameter of EPH into a specific position of f_t. $HMAC$ is a keyed cryptographic hash function.

they need to follow the public IP infringement detection algorithm to falsely claim IP infringement over watermarked DNNs.

4.1 Watermark Generation

The structure of HufuNet is same as the autoencoder consisting of an encoder and a decoder [62], a mirror symmetry of each other, and both of them consist of convolutional layers. After training on the dataset D_s, HufuNet can encode the input data and decode it to get the final output, which is similar to the input. The encoder is embedded into DNN models as watermark EPH, and the decoder is kept locally as secret SPH for future IP infringement detection.

4.2 Watermark Embedding

Embedding approach. Each parameter of EPH is embedded individually into the convolutional layers of DNN-to-be-protected, which is much larger than EPH in terms of parameter space. The embedding location of each parameter is computed based on a secret key, its index in EPH (encoder), and the values of its mirrored parameter in SPH (decoder) to ensure the stealthiness of EPH in the parameter space of DNN-to-be-protected. HufuNet is strong enough against detection (e.g., brute force searching), since the attackers do not know either the secret key or SPH.

Algorithm 1 shows the details of our watermark embedding approach. Line 1 initializes the variables $bitmap$, which keeps track of whether each position in DNN-to-be-protected has been embedded, and DNN-watermarked f_{wm}. For each parameter in our watermark EPH (Line 2), we first perform an XOR operation on its index and the value of its mirrored parameter in SPH, hash the result using a secret key, and finally do a modulo over the number of kernels in f_t to compute the embedding position (Line 3). If the position has been occupied, we simply do a linear probing to find the next available position (Line 4-6). Once the position is found, we embed the parameter into the f_{wm} (Line 7), update $bitmap$ to indicate the position has been used (Line 8), and continue the next parameter (Line 9). Finally, we output DNN-watermarked f_{wm} (Line 11). Note that the owner of f_t keeps SPH and the key to retrieve EPH from a suspect model when IP infringement detection is needed.

Training DNN-watermarked. Firstly, we randomly initialize the parameters of f_t, and embeds the parameters of EPH based on the approach discussed above to obtain f_{wm}. Then we train f_{wm} on its training dataset D_t based on a novel loss function, which ensures the performance of f_{wm} on its main task D_t and the robustness of HufuNet against fine-tuning attack. Since the whole parameter space of f_{wm} including the parameter space of EPH will be updated during training, the retrieved EPH together with the locally kept SPH may fail IP infringement detection. Hence, we apply an iterative training approach. After the first round of training of f_{wm}, we retrieve the tuned EPH, and combine it with the locally unchanged SPH to form a combined HufuNet. Then we fine-tune the combined HufuNet with EPH tuned but SPH frozen to restore the performance of HufuNet. Afterwards, we embed the newly tuned EPH back into f_{wm} using the same positions and start to train f_{wm} in the second round. After several (typically 50 based on our experience iterations), we obtain the DNN-watermarked f_{wm} and EPH, both with good performance.

Before describing the loss function, we first introduce some

TABLE 1: Relevant Parameters

Parameters	Description
$\{\Delta P_1, \ldots, \Delta P_m\}$	Parameters change of DNN models.
ΔP	A uniformly distributed discrete random variable that takes values from $\{\Delta P_1, \ldots, \Delta P_m\}$.
$\Delta loss = loss_{after} - loss_{before}$	Loss function change of DNN models before and after the fine-tuning attack.
$\tau = loss_{fail} - loss_{before}$	Threshold of loss function change for DNN models to crash due to the fine-tuning attack.
$\Delta PCC = \lvert\frac{\Delta loss}{\tau}\rvert$	Normalized loss function change due to the attack. DNN models crash when it is larger than 1.

relevant parameters related to fine-tuning attacks in Table 1. Generally, fine-tuning attacks cause performance change of f_{wm}, i.e., $\Delta PCC_{f_{wm}}$, due to the change of parameters in f_{wm}, i.e., $\Delta P_{f_{wm}}$. We define $\Delta P_{f_{wm}}$ as a uniformly distributed discrete random variable that takes values from the change of parameters in f_{wm}. Similarly, fine-tuning attacks also cause the change of parameters in EPH, i.e., ΔP_{EPH}, which is defined as a uniformly distributed discrete random variable that takes values from the change of parameters in EPH. ΔP_{EPH} finally leads to the performance change of $HufuNet_{combined}$ (including the retrieved EPH), i.e., ΔPCC_{Hufu}.

We design a novel loss function to train f_{wm} as below:

$$loss = loss_{f_{wm}} + (\beta 1 - \gamma 1)^2 + (\beta 2 - \gamma 2)^2 + (\beta 3 - \gamma 3)^2 \quad (2)$$

where

$$\beta 1 = \frac{E(|\frac{\delta loss_{f_{wm}}}{\delta P_{f_{wm}}}|)}{E(|\frac{\delta loss_{f_{wm}}}{\delta P_{EPH}}|)} \quad (3)$$

$$\beta 2 = \frac{E(|\frac{\delta PCC_{f_{wm}}}{\delta P_{f_{wm}}}|)}{E(|\frac{\delta PCC_{Hufu}}{\delta P_{EPH}}|)} \quad (4)$$

$$\beta 3 = \frac{Var(|\frac{\delta loss_{f_{wm}}}{\delta P_{f_{wm}}}|)}{Var(|\frac{\delta loss_{f_{wm}}}{\delta P_{EPH}}|)} \quad (5)$$

$loss_{f_{wm}} = loss(f_{wm}(x), y), \forall (x, y) \in D_t$ aims to ensure the performance of f_{wm} on the main task. The other items in the loss function aim to constrain the performance change of f_{wm} caused by fine-tuning should be larger than that of $HufuNet_{combined}$, so f_{wm} will become useless before HufuNet is destructed, i.e., $\Delta PCC_{f_{wm}} > \Delta PCC_{Hufu}$. In the loss function, $\gamma 1$, $\gamma 2$ and $\gamma 3$ are the target values of $\beta 1$, $\beta 2$ and $\beta 3$ respectively.

Actually, $\beta 1$, $\beta 2$, $\beta 3$ are derived from Equation (6) and Equation (7) used to train f_{wm} as below:

$$\begin{aligned} &E(|\Delta P_{f_{wm}}|) \approx E(\beta 1 \cdot |\Delta P_{EPH}|) \\ &s.t.\ Var(|\Delta P_{f_{wm}}|) \leq Var(\beta 1 \cdot |\Delta P_{EPH}|) \end{aligned} \quad (6)$$

$$E(|\frac{\Delta PCC_{f_{wm}}}{\Delta P_{f_{wm}}}|) \approx \beta 2 \cdot E(|\frac{\Delta PCC_{Hufu}}{\Delta P_{EPH}}|) \quad (7)$$

where the denominators $\Delta P_{f_{wm}}$ and ΔP_{EPH} cannot be 0, since fine-tuning will always cause parameter values to change. Equation (6) indicates that the overall parameters change in f_{wm} is $\beta 1$ times of that in the watermark EPH after the fine-tuning attack, and the variance of the parameters change in f_{wm} is $\beta 1^2$ times less than that in the watermark EPH. Equation (7) denotes that the ratio of the performance change to the parameters change in f_{wm} is $\beta 2$ times of that in $HufuNet_{combined}$ after the fine-tuning attack.

Based on the proof in Appendix A, combining Equation (6) and Equation (7), we can finally derive:

$$\Delta PCC_{f_{wm}} \geq \beta 1 \cdot \beta 2 \cdot \Delta PCC_{Hufu} \quad (8)$$

Therefore, as long as $\beta 1 \cdot \beta 2 > 1$, we can guarantee $\Delta PCC_{f_{wm}} > \Delta PCC_{Hufu}$, thus the robustness of HufuNet against fine-tuning attack. Meanwhile, we also derive $\beta 3 \leq \beta 1^2$ from Equation (6) (details are shown in Equation (9)). To guarantee the robustness of HufuNet, we need to train the watermarked model that satisfies the constraints as $\beta 1 \cdot \beta 2 > 1$ and $\beta 3 \leq \beta 1^2$. Therefore, we set the target values of $\beta 1$, $\beta 2$ and $\beta 3$ (i.e., $\gamma 1$, $\gamma 2$ and $\gamma 3$ in our loss function (Equation (2)) satisfying the constraints[3]. During the training of the watermarked models, we examine the values of $\beta 1$, $\beta 2$, and $\beta 3$ in each epoch. If and only if they satisfy the constraints in at least 30 consecutive epochs[4], we consider that the watermarked model is robust against the fine-tuning attack and terminate the model training.

To finally derive $\beta 1$, $\beta 2$, $\beta 3$ from Equation (6) and Equation (7), we consider gradient descent as the most popular way to fine-tune f_{wm}, which causes the parameters change in each epoch as $|\Delta P| = |\gamma \cdot \frac{\delta loss_{attack}}{\delta P}|$, where $loss_{attack}$ and γ are the loss function and the learning rate respectively during the fine-tuning attack. However, $|\frac{\delta loss_{attack}}{\delta P}|$ can only be obtained during the fine-tuning attack. Considering that the dataset used by the attacker to fine-tune f_{wm} is always related to its main task and with a similar distribution as D_t to ensure the performance of f_{wm} during fine-tuning as in most prior works [36], [37], [63], [47], [46], [64], [38], [43], we utilize $|\frac{\delta loss_{f_{wm}}}{\delta P_{f_{wm}}}|$ and $|\frac{\delta loss_{f_{wm}}}{\delta P_{EPH}}|$ to approximate $|\frac{\delta loss_{attack}}{\delta P_{f_{wm}}}|$ and $|\frac{\delta loss_{attack}}{\delta P_{EPH}}|$ respectively. Finally, Equation (6) can be revised as:

$$\begin{aligned} &E(|\frac{\delta loss_{f_{wm}}}{\delta P_{f_{wm}}}|) \approx E(\beta 1 \cdot |\frac{\delta loss_{f_{wm}}}{\delta P_{EPH}}|) \\ &s.t.\ Var(|\frac{\delta loss_{f_{wm}}}{\delta P_{f_{wm}}}|) \leq Var(\beta 1 \cdot |\frac{\delta loss_{f_{wm}}}{\delta P_{EPH}}|) \end{aligned} \quad (9)$$

Hence, we can obtain $\beta 1$ as in Equation (3) and $\beta 3$ as defined in Equation (5) with the constraint:

$$\beta 3 \leq \beta 1^2 \quad (10)$$

Furthermore, since $\lim_{\|\Delta P\|_1 \to 0} \frac{\Delta PCC}{\Delta P} = \frac{\delta PCC}{\delta P}$, we can rewrite Equation (7) as below:

$$E(|\frac{\delta PCC_{f_{wm}}}{\delta P_{f_{wm}}}|) \approx \beta 2 \cdot E(|\frac{\delta PCC_{Hufu}}{\delta P_{EPH}}|) \quad (11)$$

to obtain $\beta 2$ as in Equation (4).

4.3 IP Infringement Detection

Given a model $f_{suspect}$, we first extract the watermark parameter space $EPH_{retrieved}$ from $f_{suspect}$ based on the positions computed based on Algorithm 1 in Section 4.2. Then we merge $EPH_{retrieved}$ with the locally kept SPH to obtain

3. In our experiments, we set $\gamma 1 = 3$, $\gamma 2 = 1.5$, thus $\gamma 1 \cdot \gamma 2 > 1$, and $\gamma 3 = 1 < \gamma 1^2 = 9$ to satisfy $\beta 1 \cdot \beta 2 > 1$ and $\beta 3 \leq \beta 1^2$.

4. In our experiments, the watermarked VGG11, GoogLeNet, Resnet18, and Resnet34 satisfy the constraints within 30 consecutive epochs. Watermarked LSTM achieves excellent performance on IMDB task in 10 epochs. In these 10 epochs, LSTM still satisfies the constraints. Overall, we recommend 30 consecutive epochs to satisfy the constraints.

$HufuNet_{combined}$. If $f_{suspect}$ is stolen from f_{wm}, the rebuilt $HufuNet_{combined}$ is expected to maintain a high performance on the dataset D_s. If the adversary does not fine-tune/prune $f_{suspect}$, $HufuNet_{combined}$ will have exactly the same performance as the original HufuNet. Otherwise, the performance of HufuNet could drop. Hence, we utilize $\Delta PCC_{combined}$, the Performance Change Coefficient of the rebuilt HufuNet. When $\Delta PCC_{combined} < 1$, we can conclue that $f_{suspect}$ infringes the IP of the DNN-watermarked f_{wm}.

Resistance to fraudulent IP infringement claim. The adversary may forge a valid HufuNet-style watermark from the dataset D_s and embed it into DNN-watermarked to claim IP over his/her re-watermarked models. In particular, the adversary first trains a HufuNet from D_s, chooses a secret key and embeds EPH from the forged HufuNet into DNN-watermarked following the watermark embedding algorithm. Thus, the adversary may claim IP over the re-watermarked model, from which the forged watermark can be detected following the IP infringement detection algorithm. In such a case, the adversary's secret key would be different from the owner's with an overwhelming probability, so it is unlikely that the adversary's forged watermark overwrites the owner's watermark in the re-watermarked DNN and the owner's watermark is still in the re-watermarked DNN. Consequently, the owner can still claim IP infringement over the re-watermarked model. While it may be challenging for a judge to determine who embedded a watermark first without checking other information related to model training and testing, it is sufficient to trigger further investigation, or even claim IP infringement especially on the re-watermarked model under the attacker's control (e.g., submitted/sold/maintained by the attacker).

Therefore, the adversary must leverage existing parameters in the stolen model to forge EPH rather than revise any. Meanwhile, two requirements have to be satisfied to forge a valid and effective HufuNet-style watermark: (R1) the forged HufuNet (combining the retrieved EPH and locally kept SPH) should be with good performance on D_s; (R2) the positions where the embedded parameters (EPH) are retrieved should satisfy the correlation based on Line 3 in Algorithm 1. Both of them make it quite challenging, if not impossible, to forge HufuNet. First, the adversary may randomly select some parameters from f_{wm} as EPH, and train SPH with his/her dataset to ensure the performance of the forged HufuNet (meet R1). However, it is almost impossible for the adversary to find a key that can satisfy the correlation between the tuned SPH and the position of the retrieved EPH in f_{wm}. Second, the adversary may generate SPH, randomly choose one key to locate the corresponding EPH in f_{wm}, and merge the SPH and the retrieved EPH to form the forged HufuNet. However, the performance of such forged HufuNet cannot be ensured since the encoder (EPH) and the decoder (SPH) are not well paired (trained) to fulfil the encoding and decoding tasks.

4.4 Restore before Retrieve

As a parameter level watermark, HufuNet may suffer from the functionality-equivalent attacks as discussed in Section 3.2. Therefore, even if $\Delta PCC_{combined}$ is significantly larger than 1, a model can still be suspicious if it has the same or quite similar performance as the protected model. In such scenario, we examine $f_{suspect}$ further using the following restore approach and determine whether or not such model has been intentionally adjusted to evade our watermark. Note that if $f_{suspect}$ is an innocent model, the restore approach will not falsely detect IP infringement over it.

Regarding channel reordering of the structure adjustment, we recover the order of input/output channels of $f_{suspect}$ according to the original DNN-watermarked f_{wm}. We start from the first layer, where only the order of the output channels can be changed. We compare the parameters of each output channel of $f_{suspect}$ with those of f_{wm} using cosine similarity to find the optimal match. After recovering the order of each output channel of the first layer, the input channels of the second layer can be restored straightforward based on the restored order of the output channels of the first layer. Similarly, the output/input channels of all other layers can be restored accordingly based on f_{wm}, thus enabling correct retrieval of $EPH_{retrieved}$ even if $f_{suspect}$ experienced channel reordering. The cosine similarity approach also works well when many parameters in output channels are adjusted, such as fine-tuning (details in Appendix B).

Regarding parameter adjustment, we first ensure that the order of the input/output channels of $f_{suspect}$ is recovered using the approach described above. Then we perform Singular Value Decomposition (SVD) on each convolution kernel of $f_{suspect}$ and f_{wm} to obtain intermediate singular value matrices. For each pair of singular value matrix in $f_{suspect}$ and singular value matrix in f_{wm}, we divide each element of the former by each element of the latter. Thus, we get a series of resultant matrices. The values on the diagonal of each resultant matrix are averaged to compute the scale factor c, used to obtain the restored convolution kernel, i.e., dividing each element in the corresponding convolution kernel of $f_{suspect}$ by c. We utilize this to restore all the convolution kernels.

Note that the scale factor c is not computed by dividing each element of the convolution kernel in $f_{suspect}$ by each element of the convolution kernel in the original f_{wm} and computing the average value of the resultant matrix. This is because the singular values of a convolution kernel represent the scaling characteristics of the kernel. It is more reliable to compute the scaling factor from the singular values rather than from kernel parameters directly. Take the following scenario as an example where an attacker first fine-tunes the stolen model and then performs the functionality-equivalent attack through parameter adjustment. The fine-tuning typically makes smaller values in the convolution kernel change more significantly than larger ones, e.g., increasing from 1 to 2 results in the factor of 2, but increasing from 10 to 11 yields the factor of 1.1. Thus, the change of smaller values has a larger impact on the scale factor of the kernel, making the large parameters in the kernel over-changed or under-changed during recovering.

Meanwhile, a highly suspicious model $f_{suspect}$ may show minor structural differences, e.g., the number of kernels on one or several layers of $f_{suspect}$ is smaller or larger than those of f_{wm}. We need to further examine whether or not some kernels of such a model have been intentionally cut off or supplemented to evade our watermark detection. First, we still restore the structure of $f_{suspect}$ using the cosine similarity approach presented above, since channel expansion and kernels cutoff/supplement attacks always change the order of kernels as well, unless all the expansion/cutoff/supplemented kernels are at the end of the layer. With all the kernels of $f_{suspect}$ restored and the original f_{wm} as a reference, we fill in missing (cutoff) kernels with all parameter values of 0 or cutoff extra (expansion/supplemented) kernels directly[5]. Thus, we can restore $f_{suspect}$ with the same structure as f_{wm} and the same number of kernels, so Algorithm 1

5. Using the values of the corresponding kernels in f_{wm} to fill in may cause false positive on innocent models.

TABLE 2: Training Time Consumption

	VGG11	GoogLeNet	Resnet18	Resnet34	LSTM
Clean[1]	107min	781min	272min	532min	5min
WMed[2]	205min	891min	384min	685min	6min

[1] Clean indicates the training time consumption of clean DNNs.
[2] WMed indicates the training time consumption of watermarked DNNs.

TABLE 3: Baseline Performance of IP Infringement Detection

	VGG11	GoogLeNet	Resnet18	Resnet34	LSTM
Original[1]	87.18%	91.88%	91.73%	66.73%	86.57%
Suspect[2]	87.18%	91.88%	91.73%	66.73%	86.57%
ΔPCC[3]	0	0	0	0	0

[1] Original indicates the accuracy of the original DNN-watermarked.
[2] Suspect indicates the accuracy of the suspect model we would like examine. Since the attackers did not conduct any change, the suspect model demonstrates the same accuracy as our DNN-watermarked.
[3] ΔPCC indicates the performance decrease of the $HufuNet_{combined}$. Since the attackers did not conduct any change, $HufuNet_{combined}$ demonstrates the same performance as our original HufuNet, i.e., with no performance change.

can be used to extract EPH.

The above-discussed restoring approaches can cover most of the functionality-equivalent attacks, and are recommended when it is unknown which specific type of functionality-equivalent attack the attackers may have launched. Note that it is always possible to utilize/extend the restoring approaches given more information is available about what kind of functionality-equivalent attacks has been employed by attackers. For instance, considering attackers launched another type of functionality-equivalence attack, RepVGG-style attack [65], we can still restore DNNs by including the inverse of the execution process of the attack into the proposed restoring approaches. Details are discussed in Appendix E.

5 Evaluation

Based on the watermark destruction approaches discussed in Section 3.2, we evaluate HufuNet in the following aspects. (i) fidelity: watermark-embedding should impact little on the performance of the original DNN models (Section 5.3); (ii) robustness: watermarks should still be detected by owners from stolen DNN models even if the models experienced fine-tuning, pruning or kernels cutoff/supplement (Section 5.4); (iii) adaptiveness: watermarks should still be detected by owners even if the model has been converted to a functionality-equivalent one via structure adjustment or parameter adjustment (Section 5.5); (iv) against synthetic attacks: watermark can still be detected by owners even if the attackers applied fine-tuning, pruning and functionality-equivalent attacks together on their stolen models (Section 5.6); (v) integrity: it is unlikely for DNN owners to detect IP infringement over innocent DNN models (Section 5.7); (vi) stealthiness: it is difficult for attackers to learn the existence of watermark from a stolen DNN model (Appendix D, due to space limitation). To demonstrate the effects of encoding/decoding of $HufuNet_{combined}$ in different scenarios, we show the randomly selected outputs of $HufuNet_{combined}$ together with the inputs on VGG11 in Figure 5 in Appendix.

5.1 Experimental Setup

Models. Without loss of generality, we experiment on five models: VGG11 [4], GoogLeNet [66], Resnet18 and Resnet34 [2], as well as LSTM [67], which are all used as DNNs-to-be-protected. HufuNet is designed with an Encoder and a Decoder, both of which consist of three convolution layers.

Datasets. We choose four datasets: MNIST [68], IMDB [69], CIFAR-10, and CIFAR-100 [70] for the above five models accordingly. We use MNIST to train our HufuNet, CIFAR-10 to train VGG11, GoogLeNet as well as Resnet18, CIFAR-100 to train Resnet34, and IMDB to train LSTM as shown in Table 14 in Appendix. We use the classification task and the main task of the DNN models interchangeably thereafter in the paper.

TABLE 4: Fidelity of HufuNet

	VGG11	GoogLeNet	Resnet18	Resnet34	LSTM
w/o EPH[1]	87.62%	92.34%	92.02%	67.62%	86.71%
w/ EPH[2]	87.18%	91.88%	91.73%	66.73%	86.57%
Performance change[3]	-0.44%	-0.46%	-0.29%	-0.89%	-0.14%

[1] w/o EPH represents the DNN-to-be-protected, without our EPH.
[2] w/ EPH indicates DNN-watermarked, with our EPH.
[3] The performance change of the watermarked model compared with the original model.

Platform. All experiments are conducted on a 64-bit Ubuntu 18.04 server with 3 Nvidia TiTan X GPUs, each with 12GB memory.

5.2 Baseline Performance

We evaluate IP infringement detection of HufuNet on five models, i.e., VGG11, GoogLeNet, Resnet18, Resnet34, and LSTM, without any change between embedding and verifying, i.e., assuming adversaries just steal the model without performing any change on the stolen model, as the baseline performance. The parameter values of the five DNNs-to-be-protected and HufuNet are shown in Table 15 in Appendix. We can see that our watermark EPH only consumes a small portion of the parameter space ranging from 0.50% to 3.87%. As defined in Table 1, $\tau = loss_{fail} - loss_{before}$. In particular, $loss_{before}$ is the convergence value of the loss after the model training, thus different models have different values of $loss_{before}$. In contrast, $loss_{fail}$ is the worst performance of the model acceptable to the model owner and needs to be set by defenders accordingly. During watermark embedding, we need to set the $loss_{fail}$ values of the two models f_{wm} and HufuNet. Regarding f_{wm}, we consider $loss_{fail}$ of f_{wm} is 70% of the original performance, so we set the threshold $\tau_{f_{wm}}$ of $\Delta PCC_{f_{wm}}$ as 30% of the original performance of these models (i.e., 27.18%, 27.56%, 27.52%, 20.02%, and 25.97% in VGG11, GoogLeNet, Resnet18, Resnet34, and LSTM, respectively). For HufuNet, we set the $loss_{fail}$ of τ_{hufu} as 0.60 MSE (i.e., mean squared error) value[6]. Besides, after watermark embedding, $loss_{before}$ of HufuNet is with the MSE value of 0.10, 0.07, 0.17, 0.19, and 0.06 in VGG11, GoogLeNet, Resnet18, Resnet34, and LSTM, respectively, so τ_{hufu} is set as 0.50, 0.53, 0.43, 0.41, and 0.54 in the above models, respectively. Moreover, we evaluate τ of ΔPCC in Appendix F, and show the values of $\beta 1 \cdot \beta 2$, $\beta 1^2$, and $\beta 3$ during the training process in Figure 3 in Appendix. We can see that $\beta 1 \cdot \beta 2 > 1$ and $\beta 3 \leq \beta 1^2$, which indicates the robustness of our HufuNet against model fine-tuning attacks is guaranteed. Furthermore, we show the training time consumption of watermarked models and clean models without our watermarks in Table 2. It can be seen that the computational overhead introduced by watermark is on average 95 minutes in absolute value and 27.93% in percentage, which we believe is acceptable considering it is a one-time effort.

The baseline performance is shown in Table 3. Since there is no retraining conducted on the original DNN-watermarked, the suspect model (to-be-examined by HufuNet for detecting IP infringement) demonstrates the same accuracy as that of our original model. Similarly, $EPH_{retrieved}$ is exactly the same as original EPH,

6. With the MSE value of 0.60, HufuNet can generate the images with high fidelity as shown in Figure 5(l), thus can be used to detect IP infringement.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2023.3242737

TABLE 5: Robustness against Fine-tuning

Epochs	VGG11		GoogLeNet		Resnet18		Resnet34		LSTM	
	DNN-watermarked	HufuNet	DNN-watermarked	HufuNet	DNN-watermarked	HufuNet	DNN-watermarked	HufuNet	DNN-watermarked	HufuNet
0	82.90%	0	91.05%	0	91.85%	0	67.50%	0	85.88%	0
10	83.05%	0.037	90.80%	0.004	91.70%	0.005	67.20%	0.001	82.90%	0.032
20	82.20%	0.064	90.60%	0.009	91.85%	0.013	67.85%	0.002	83.80%	0.076
30	81.85%	0.089	90.40%	0.014	91.80%	0.021	68.20%	0.007	83.90%	0.081
40	83.10%	0.110	90.30%	0.020	91.95%	0.030	68.15%	0.012	84.60%	0.083
50	81.60%	0.121	90.35%	0.026	92.05%	0.037	68.00%	0.017	85.20%	0.087
60	82.80%	0.129	90.35%	0.031	92.00%	0.043	68.15%	0.021	86.00%	0.093
70	82.95%	0.129	90.20%	0.034	91.95%	0.047	68.25%	0.024	85.10%	0.095
80	82.75%	0.130	90.20%	0.035	92.00%	0.050	68.30%	0.026	85.40%	0.101
90	83.05%	0.130	90.20%	0.036	91.95%	0.050	68.20%	0.026	84.10%	0.102
100	83.10%	0.130	90.20%	0.036	92.00%	0.050	68.05%	0.026	84.80%	0.108

HufuNet in the table indicates ΔPCC of $HufuNet_{combined}$.

TABLE 6: Robustness against Pruning

pct.	VGG11		GoogLeNet		Resnet18		Resnet34		LSTM	
	DNN-watermarked	HufuNet	DNN-watermarked	HufuNet	DNN-watermarked	HufuNet	DNN-watermarked	HufuNet	DNN-watermarked	HufuNet
0	87.18%	0	91.88%	0	91.73%	0	66.73%	0	86.57%	0
10%	87.19%	0	91.92%	0.006	91.64%	0.001	66.68%	0.002	86.64%	0.001
20%	87.04%	0	91.80%	0.005	91.59%	0	66.69%	0.006	86.60%	0.003
30%	86.74%	0	91.43%	0.037	91.43%	0.025	65.04%	0.075	86.41%	0.011
40%	86.35%	0.002	88.38%	0.115	89.62%	0.151	61.08%	0.185	86.32%	0.038
50%	86.07%	0.002	65.97%	0.341	80.49%	0.355	46.66%	0.425	86.17%	0.105
60%	85.59%	0.007	15.14%	0.806	43.10%	0.707	13.94%	1.000	85.97%	0.183
70%	78.57%	0.012	10.02%	1.535	22.11%	1.479	2.92%	1.732	85.62%	0.283
80%	41.73%	0.116	10.00%	2.117	18.19%	2.280	1.39%	2.643	81.84%	0.379
90%	16.78%	0.451	10.00%	2.963	10.07%	2.913	1.00%	2.879	58.35%	0.485

pct. indicates the percentage of pruning. HufuNet in the table indicates ΔPCC of $HufuNet_{combined}$.

so the performance of $HufuNet$ equals to that of the original HufuNet. Therefore, $HufuNet$ can always detect IP infringement over the stolen model accurately considering not any kind of retraining on the stolen model is conducted[7]. According to Section 4.3, with $\Delta PCC < 1$, $HufuNet$ can successfully detect IP infringement. In the following experiments, we will use ΔPCC of $HufuNet$ to indicate IP infringement detection.

5.3 Fidelity

Ideally, our watermark EPH should have a negligible impact on the performance of the DNN-watermarked. In other words, trained on the same dataset, DNN-to-be-protected and DNN-watermarked should have similar accuracy on the main task. To demonstrate such fidelity, we perform experiments on the models trained on datasets as in Table 14 and evaluate the accuracy of DNNs-to-be-protected and DNNs-watermarked. Table 4 shows that the accuracy of DNNs-watermarked is always quite close to that of DNNs-to-be-protected. We believe it is mainly because our watermark EPH only occupies a small portion of the entire parameter space of DNNs-watermarked and gets tuned in the training of DNNs-watermarked as well.

5.4 Robustness

Due to the stealthiness of EPH (as shown in Section D), a targeted destruction (knowing where EPH is and destructing it correspondingly) is almost impossible. The adversary may not want to initialize and retrain all the convolutional layers of the stolen model, since it demands similar amount of computing resources and training data as training his/her own model. Therefore, the adversary would prefer "blindly" revising the model, e.g., fine-tuning, transfer learning, pruning, kernels cutoff/supplement, etc., hoping to destruct embedded watermarks.

7. We also evaluate the integrity of HufuNet in Section 5.7, that is, without EPH, we will not detect IP infringement over innocent models.

Robustness against fine-tuning. We trained five models in total (with our watermark EPH embedded): VGG11, GoogLeNet and Resnet18 on CIFAR-10, Resnet34 on CIFAR-100, as well as LSTM on IMDB. We assume adversaries only have access to the testing dataset (provided by the model owner to test the model's performance, including 10,000 samples), and try to leverage it to fine-tune the stolen model. Referring to the settings in [43], adversaries divide their available dataset into two parts, 80% as the training dataset and 20% as the testing dataset. Same as [36], we also set the initial learning rate $\lambda = 0.001$ and the delay factor as 0.0005 to fine-tune DNNs-watermarked using SGD (Stochastic Gradient Descent). The maximum number of epochs for fine-tuning is set as 100 according to [43], and the incremental step of the epoch is set as 10. As λ decreases to a minimal number, the parameters of DNNs-watermarked will rarely change, which indicates the model tends to be stable during fine-tuning. As shown in Table 5, regarding the DNNs-watermarked, their accuracy fluctuates and finally converges to that of the DNNs before fine-tuning. Meanwhile, ΔPCC of $HufuNet_{combined}$ is always very small, significantly less than 1, which successfully detects IP infringement over the stolen models. Hence, the fine-tuning attack does not destroy EPH due to our robustness guarantee.

Ablation Study for Proposed Loss Function. To demonstrate the effectiveness of the proposed loss function, we fine-tune the watermarked models trained with/without the loss. For the watermarked models without the loss, we only embed the parameters of HufuNet into watermarked models according to Algorithm 1. Then we freeze the parameters of HufuNet and only train the other parameters of watermarked models on the main task. We launch the fine-tuning attack with a large learning rate, i.e., 0.01, for all the watermarked models. As shown in Table 7, we find that the proposed loss function is effective in improving the robustness

TABLE 7: Ablation Study for Proposed Loss Function

Models		VGG11	GoogLeNet	Resnet18	Resnet34	LSTM
w/ loss[1]	Accuracy	85.30%	84.91%	86.99%	67.36%	86.41%
	HufuNet	0.094	0.427	0.826	0.952	0.855
w/o loss[2]	Accuracy	84.21%	81.21%	85.47%	70.18%	86.00%
	HufuNet	0.712	2.563	1.542	6.748	1.995

[1] w/ loss represents the DNN-watermarked trained with our designed loss.
[2] w/o loss represents DNN-watermarked trained without our designed loss.
[3] Accuracy represents the accuracy of DNN-watermarked on its main task.
[4] HufuNet in the table indicates ΔPCC of $HufuNet_{combined}$.

of HufuNet against the fine-tuning attack. That is, ΔPCC of HufuNet extracted from the watermarked models trained with our designed loss is 0.094, 0.427, 0.826, 0.952, and 0.855 for VGG11, GoogLeNet, Resnet18, Resnet 34, and LSTM, respectively, far smaller than those extracted from the watermarked models trained without our designed loss (i.e., 0.712, 2.563, 1.542, 6.748, 1.995, respectively). Thus, our proposed loss function is effective in improving the robustness of HufuNet against the fine-tuning attack.

Robustness against transfer learning. We also evaluate the robustness of HufuNet against transfer learning attack on the watermarked VGG11, Resnet18, Resnet34, and GoogLeNet models. These watermarked models are firstly trained on CIFAR-10 task and then transferred to GTSRB task following the standard transfer learning process [71], [72]. After transfer learning, the accuracy of these models is 90.86%, 91.28%, 91.55%, and 92.85% respectively, while ΔPCC of HufuNet is 0.134, 0.596, 0.473, and 0.643 respectively (less than 1), satisfying IP infringement detection. Thus, HufuNet is also robust against transfer learning attack.

Robustness against pruning. To evaluate the robustness of HufuNet against pruning, we first embed our watermark EPH into VGG11, GoogLeNet, Resnet18, Resnet34 and LSTM, and then prune 10% to 90% parameters of each model based on their absolute values respectively. We adopt the commonly used pruning technique in [73], which eliminates the parameters with smaller absolute values, thus believed to have little impact on the performance of the models. We choose such model pruning approach, which is consistent with our assumption of the adversaries' capability, i.e., lack of sufficient training dataset and/or computing resources. Table 6 shows the experimental results on the accuracy of the corresponding DNNs-watermarked and ΔPCC of HufuNet after pruning. Based on the results, we find that even if the DNNs-watermarked have been pruned to be considered as "fail" on the main task (e.g., the accuracy of VGG11, GoogLeNet, Resnet18, Resnet34, and LSTM has been reduced to 41.73%, 15.14%, 43.10%, 46.66%, and 58.35% respectively) after pruning (with 80%, 60%, 60%, 50%, and 90% pruning rate respectively), ΔPCC of $HufuNet_{combined}$ is still below 1, satisfying IP infringement detection. At the 70% pruning rate, ΔPCC of $HufuNet_{combined}$ becomes greater than 1 for GoogLeNet, ResNet18 and ResNet34, but such a high degree of pruning almost completely ruins the main task of those models. Overall, the low ΔPCC of $HufuNet_{combined}$ upon heavy pruning indicates it is highly robust against pruning.

Robustness against kernels cutoff/supplement. We evaluate the robustness of HufuNet against kernels cutoff attack on VGG11, Resnet18, GoogLeNet and LSTM. Resnet34 is not evaluated, because the structure of it is similar as Resnet18. Since the removal of the convolution kernels will reduce the performance of f_{cutoff}, we control the kernels cutoff rate as 6.97%, 3.40%, 5.30%, and 4.69% respectively for the above four models, to ensure their accuracy does not drop significantly, i.e., 5.82%, 4.37%, 2.23%, and 0.17% respectively. Then we use the above proposed cosine similarity approach to restore the structure of f_{cutoff} and supplement 0 for the missing kernels (cutoff). As shown in Table 8, ΔPCC of HufuNet is 0.197, 0.146, 0.275, and 0.127 for the four models respectively, sufficient to detect IP infringement (below 1). The result of kernels supplement is quite similar to that of kernels cutoff, since both the structure and the number of kernels are changed similarly.

TABLE 8: Robustness against Kernels Cutoff

DNN	Variance	Accuracy of DNN-watermarked	HufuNet
VGG11	Original	87.20%	0
	Cut off	81.38%	-
	Restore	81.38%	0.197
Resnet18	Original	91.72%	0
	Cut off	87.35%	-
	Restore	87.35%	0.146
GoogLeNet	Original	91.89%	0
	Cut off	89.66%	-
	Restore	89.66%	0.275
LSTM	Original	86.57%	0
	Cut off	86.40%	-
	Restore	86.40%	0.127

HufuNet in the table indicates ΔPCC of $HufuNet_{combined}$.

TABLE 9: Adaptiveness against Channel Reording

DNN	Variance	Accuracy of DNN-watermarked	HufuNet
VGG11	Original	87.18%	0
	Reorder	87.18%	4.941
	Restore	87.18%	0
Resnet18	Original	91.73%	0
	Reorder	91.73%	4.934
	Restore	91.73%	0
GoogLeNet	Original	91.88%	0
	Reorder	91.88%	4.070
	Restore	91.88%	0
LSTM	Original	86.57%	0
	Reorder	86.57%	3.953
	Restore	86.57%	0

5.5 Adaptiveness

We further evaluate HufuNet against functionality-equivalent attacks via structure adjustment (including channel reording and channel expansion), and parameter adjustment. Moreover, we also evaluate HufuNet against the RepVGG-style attack [65] and show the evaluation results in Appendix E due to space limitation.

Adaptiveness against Channel Reording. We evaluate HufuNet against channel reording on four neural network models: VGG11, Resnet18, GoogLeNet and LSTM. To simulate the behavior of the most capable attackers, we shuffle the order of all convolution kernels of each layer, and at the same time we adjust the order of the inner layers of the convolution kernels to ensure functionality-equivalent. If there is a BatchNorm layer connected after a convolution layer, we also change its mean and variance according to the order of the corresponding convolution kernels. We use the approach discussed in Section 4.4 to restore the order of kernels. The experimental results are shown in Table 9. After reordering the convolution kernels of the DNNs-watermarked, the accuracy of VGG11, Resnet18, GoogLeNet, and LSTM remains the same (functionally equivalent), but ΔPCC of $HufuNet_{combined}$ (without restore) increases significantly from 0 to 4.941, 4.934, 4.070, and 3.953 respectively. After restoring, ΔPCC of $HufuNet_{combined}$ resumes to 0, which indicates no performance change compared to the original HufuNet, thus sufficient to detect IP infringement.

TABLE 10: Adaptiveness against Parameter Adjustment

DNN	Variance	Accuracy of DNN-watermarked	HufuNet
VGG11	Original	87.18%	0
	PMCG	87.18%	7.755
	Restore	87.18%	0
Resnet18	Original	91.73%	0
	PMCG	91.73%	7.000
	Restore	91.73%	0
GoogLeNet	Original	91.88%	0
	PMCG	91.88%	6.953
	Restore	91.88%	0

PMCG refers to the DNN-watermarked whose parameters have been adjusted to launch functionality-equivalent attack.

Adaptiveness against Channel Expansion. We also evaluate HufuNet against the channel expansion attack on VGG11. We randomly choose two adjacent layers (the fourth and fifth layers), and double the output dimension of the fifth layer and input dimension of the sixth layer by supplementing output/input channels (details in Appendix C). Finally, we launch the channel reording attack to make it more difficult to trace any embedded EPH. After such channel expansion and channel reording attacks, the accuracy of VGG11 remains the same, i.e., functionally equivalent, but ΔPCC of $HufuNet_{combined}$ (without restore) increases to 7.210. To restore EPH from such attacks, we use the approach discussed in Section 4.4. After restoring, ΔPCC of $HufuNet_{combined}$ resumes to 0 (i.e., no performance change compared to the original HufuNet), thus indicating IP infringement over the stolen model.

Adaptiveness against Parameter Adjustment. We also evaluate HufuNet watermark against functionality-equivalent attack through parameter adjustment on VGG11, Resnet18, and GoogLeNet. Particularly, such attack can be applied to DNNs-watermarked with activation functions like ReLU, rather than Sigmoid. Hence, LSTM is not evaluated (its internal activation function is Sigmoid). In our evaluation, we change the parameter values of each layer by multiplying these parameters by $c = 2^n$ (n is a randomly-choose integer for each layer ranging from -18 to 18), while still maintaining the functionality-equivalent of the overall model. We use the approach as discussed in Section 4.4 to restore the parameter values. The experiment result are shown in Table 10. After adjusting parameters of the DNNs-watermarked (PMCG in Table 10), the accuracy of VGG11, Resnet18 and GoogLeNet remains the same (functionally equivalent), but ΔPCC of $HufuNet_{combined}$ (without restore) increases significantly from 0 to 7.755, 7.000 and 6.953 respectively. However, after restoring, ΔPCC of $HufuNet$ resumes to 0, thus detecting IP infringement successfully.

5.6 Against Synthetic Attacks

After stealing a DNN model, the adversaries may try their best to undermine any watermark embedded inside, e.g., utilizing fine-tuning, pruning, functionality-equivalent attack (structure/parameter adjustment). Below, we synthesize an advanced attack by combining the above attack methods to simulate the adversaries' best effort, and evaluate HufuNet against such a synthetic attack. First, we we prune VGG11 with the pruning rate of 40%, 50%, 60%, 70% and obtain the pruned DNNs. Then, we fine-tune VGG11 as we did in Section 5.4 using the test dataset in 100 epochs. Afterwards, we randomly choose two adjacent layers (the fourth and the fifth in our experiment) of VGG11 (with our watermark EPH embedded) to launch the channel expansion attack. In the end, we randomly select six layers to launch the functionality-equivalent attack: (1) The channel reording is applied to two layers; (2) the parameter adjustment is applied to another two layers; (3) both channel reording and parameter adjustment are applied to the remaining two layers. For watermark detection, we retrieve $EPH_{retrieved}$ from the restored model at its original embedding locations and merge $EPH_{retrieved}$ with our local SPH to obtain $HufuNet_{combined}$. As shown in Table 11, it can be seen that even when the pruning rate reaches 70%, the accuracy of VGG model drops sharply to 78.56%. After fine-tuning and functional equivalence attack, the accuracy of the model bounced back to 84.90%. The sharp decline and rise of model accuracy indicates that pruning and fine-tuning have a significant impact on the model parameters. In this case, ΔPCC of $HufuNet_{combined}$ without restoring ("Direct" in the table) increases to 6.974. After utilizing the restore approach as in Section 4.4, ΔPCC of $Hufu_{combined}$ decreases to 0.569, with 98.62% kernels in DNN-watermarked correctly restored, sufficient to detect IP infringement (when ΔPCC is below 1).

TABLE 11: Against Synthetic Attacks

Pruning Rate	VGG11		HufuNet		Restored Rate
	Pruning	Final	Direct	Restored	
40%	86.36%	85.30%	7.057	0.066	100.00%
50%	86.06%	86.50%	6.986	0.103	100.00%
60%	85.60%	85.20%	6.816	0.313	99.96%
70%	78.56%	84.90%	6.974	0.569	98.62%

Pruning indicates the accuracy of VGG11 after pruning attack only. Final indicates the accuracy of VGG11 after synthetic attacks including pruning, fine-tuning and functionality-equivalent attacks.

5.7 Integrity

We evaluate the integrity of HufuNet by testing whether it will detect IP infringement over innocent models not copied or retrained from ours. In this experiment, we intend to use innocent models that share the same network structure and the same task as ours, thus highly resembling our watermarked models. In particular, we consider two different kinds of such innocent models: the same training dataset as DNN-watermarked (Innocent 1) and different training dataset than DNN-watermarked (Innocent 2). We train each innocent model for 100 epochs, and each epoch begins with different initialization parameters to ensure high accuracy. We use the models obtained above as innocent models without embedding our watermark inside them. Upon IP infringement detection, we build $HufuNet_{combined}$ by combining the local SPH and the $EPH_{retrieved}$ from the innocent models based on the computed embedding locations. Table 12 shows that ΔPCC of $HufuNet_{combined}$ is at least 3.923 for Innocent 1 and 3.410 for Innocent 2 on average. Moreover, to show that our restore before retrieve approach does not cause false positives, we retrieve EPH after restoring all the innocent models, and ΔPCC of $HufuNet_{combined}$ is at least 4.424 for Innocent 1 and 3.409 for Innocent 2 on average. Thus, none of the innocent models is falsely claimed as ours, all with $\Delta PCC > 1$. Note that the functionality-equivalent attack does not apply to LSTM as discussed in Section 3.2, so we did not apply the restore mechanism on LSTM.

5.8 Comparison with Existing Works

We compare HufuNet with existing white-box watermarking approaches. The watermark embedding processes of [31] and [33] are similar, so our comparison first focuses on [31] without loss of generality. Based on its evaluation, [31] also demonstrates good fidelity and strong robustness against fine-tuning and pruning attacks as HufuNet. Therefore, we compare it with HufuNet in other aspects including adaptiveness, security and robustness against kernels cutoff/supplement on the wide residual network WideResNet [74], which is used in [31].

TABLE 12: Integrity of HufuNet

	VGG11	GoogLeNet	Resnet18	Resnet34	LSTM[5]
Innocent 1[1]	3.923	5.245	5.931	5.490	3.850
Innocent 1 Restore[2]	4.424	4.477	5.349	6.609	-
Innocent 2[3]	3.510	3.410	3.518	3.448	3.726
Innocent 2 Restore[4]	3.820	3.500	3.409	3.431	-

[1] Innocent 1 refers to the innocent model with the same architecture and task trained with the same dataset as ours.
[2] Innocent 1 Restore indicates applying restore approach on Innocent 1 and then retrieving watermark.
[3] Innocent 2 refers to the innocent model with the same architecture and task but trained with different dataset than ours.
[4] Innocent 2 Restore indicates applying restore approach on Innocent 2 and then retrieving watermark.
[5] The functionality-equivalent attack does not apply to LSTM as discussed in Section 3.2, so we did not apply the restore mechanism on LSTM.

We first train WideResNet embedded with the watermark in [31] and HufuNet respectively, thus obtaining two different watermarked WideResNets. Their accuracy on the main task is 90.24% and 91.36% respectively. Regarding the adaptiveness, we first perform channel reording on both of the watermarked WideResNets as described in Section 5.5. Their accuracy on the main task remains the same, but the watermark in [31] achieves 51.30% Bit Error Rate (BER), far from detecting IP infringement. In contrast, the ΔPCC of HufuNet is 0, still correctly detecting IP infringement. We also perform parameter adjustment, i.e., changing the parameter values of each layer by multiplying them by -1, and the accuracy of both watermarked WideResNets still remains the same. The watermark in [31] achieves 100.00% BER, since multiplying parameter values with -1 makes all the bits inverted, thus failing to detecting IP infringement. In contrast, ΔPCC of HufuNet is 0, still correctly detect IP infringement.

Moreover, the watermark in [31] suffers from the channel expansion attack as described in Section 5.5. The channel expansion attack causes the dimension of the parameter space W not to match the dimension of secret matrix X stored locally by the owner, since it changes the dimension of the input channels of the parameter space W. Therefore, the matrix multiplication operation, i.e., $X \cdot W$ cannot be performed to compute the bit vector, indicating a failure to extract any watermark. Similarly, such approach is not robust against kernels cutoff/supplement attack either, which also changes the dimension of the input channels of W.

The attackers can also forge a valid watermark to compromise the security of the watermark in [31]. Particularly, the attackers can randomly select a layer with the parameters W from the watermarked model that they attempt to claim ownership. Then they randomly initialize an secret matrix X, and directly use $s(X \cdot W)$ to compute the corresponding bit vector $b_{extracted}$. The attackers simply present $b_{extracted}$ to claim it as the bit vector previously embedded, with a zero BER to maliciously claim ownership.

According to the evaluation results in [35], the other two white-box watermarking approaches DeepMarks [32] and DeepSigns [34] are not robust against weight shifting attack, which is a kind of parameter adjustment, so we conclude they are not robust against functionality-equivalent attacks. Furthermore, the adversaries can also present the forged bit vector to fraudulently claim ownership on the watermark approaches [34], [32], [33] in a similar way as [31] discussed above. Finally, although all of [35], [34], [32], [33] demonstrate robustness against fine-tuning attacks, we want to emphasize that none of them can provide theoretical assurance on such robustness as HufuNet does. We summarize the comparison with those white-box watermarking approaches in Table 13.

TABLE 13: Comparison with Existing Works

Methods	Against Fine-tuning	Against Functionality-equivalent	Against Fraudulently Claim Ownership
Uchida et al.[31]	◐	○	○
DeepMarks[32]	◐	○	○
DeepSigns[34]	◐	○	○
Liu et al. [33]	◐	○	○
Our HufuNet	●	●	●

●, ◐ and ○ indicate "yes", "partially yes" and "no" respectively. In particular, the "partially yes" for "Against Fine-Tuning" denotes empirical evaluation only, without theoretical assurance.

6 Discussion

White-box scenarios. HufuNet is a white-box watermark, which needs to be extracted from the architecture and the parameters of the suspect model. If the suspect model is deployed in a black-box manner, we cannot extract our watermark from it. Moreover, in the Introduction section, we discussed the necessity of watermarks in white-box scenarios, e.g., embedded devices, model markets, competitions, etc.

Generalizing HufuNet's architecture. We considered a feedforward neural network as the watermark in our initial design, but we find it makes fraudulent ownership claim relatively easier. Attackers can choose a random key (used in Algorithm 1) to retrieve some parameters from DNN-watermarked as EPH, randomly construct a neural network as SPH, and combine the two as the forged watermark. Then the attacker can select any dataset and relabel its samples according to the prediction results of the forged watermark model. Thus, the attackers can fraudulently claim ownership of the DNN-watermarked since the forged watermark can also achieve high accuracy on the chosen dataset. The difficulty of forging HufuNet with the autoencoder architecture is detailed in Section 4.3.

Limitation. We evaluated HufuNet against model extraction attacks, and ΔPCC of HufuNet retrieved from the extracted model is 3.689, larger than the threshold 1. Thus, HufuNet is not robust against model extraction. This is because HufuNet manipulates the watermarked model at the parameter level, rather than revising the inputs and outputs of the watermarked model. Note that most of existing watermarking approaches [31], [32], [34], [33] are not robust against model extraction attacks as well, which demands the efforts from the research community.

7 Conclusion

In this paper, we proposed HufuNet, a robust and secure watermarking approach, to protect IP of DNN models. We divide HufuNet into two pieces, with the left piece EPH embedded into the DNN-to-be-protected and the right piece SPH kept locally as the secret to detect IP infringement of suspect DNN models. Compared with previous watermarking approaches, HufuNet is more robust against model fine-tuning and pruning, which is demonstrated by experimental results conducted on five popular DNN models. More importantly, our watermark is the first one that is theoretically assured robust against the model fine-tuning attack. Meanwhile, our watermarking approach has a negligible impact on the performance of DNN models it protects. It is robust to kernel cutoff/supplement attack, adaptive to the functionality-equivalent attack, and does not falsely detect IP infringement over innocent models.

ACKNOWLEDGEMENTS

We are thankful for the effort contributed by Liuquan Yao in helping us improve the robustness of HufuNet. We also thank the editors and reviewers for their constructive feedback. The IIE authors are supported in part by Beijing Natural Science Foundation (No.M22004), NSFC (92270204), Youth Innovation Promotion Association CAS, Beijing Academy of Artificial Intelligence (BAAI) and CCF-Huawei Innovation Research Plan. Yingjiu Li is supported in part by the Ripple University Blockchain Research Initiative.

REFERENCES

[1] Wikipedia, *Fu (tally)*, https://en.wikipedia.org/wiki/Fu_(tally), 2020.
[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.
[3] H. K., Z. X. *et al.*, "Identity mappings in deep residual networks," in *ECCV*. Springer, 2016.
[4] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
[5] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.
[6] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *Journal of Machine Learning Research*, vol. 12, no. 76, pp. 2493–2537, 2011.
[7] W. Xiong, J. Droppo, X. Huang, F. Seide, M. L. Seltzer, A. Stolcke, D. Yu, and G. Zweig, "Toward human parity in conversational speech recognition," *TASLP*, 2017.
[8] D. Amodei, R. Anubhai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, J. Chen, M. Chrzanowski *et al.*, "Deep speech 2: End-to-end speech recognition in english and mandarin." *CoRR*, 2015.
[9] I. Rebai, Y. BenAyed, W. Mahdi, and J.-P. Lorré, "Improving speech recognition using data augmentation and acoustic model fusion," *Procedia Computer Science*, vol. 112, pp. 316 – 322, 2017.
[10] A. Lankinen, "Patentability of artificial intelligence in europe: Is artificial intelligence patentable according to the european patent convention and is the current legal framework for patents suitable for patenting artificial intelligence?" 2020.
[11] M. R. Watson, A. K. Marnerides, A. Mauthe, D. Hutchison *et al.*, "Malware detection in cloud computing infrastructures," *TDSC*, 2015.
[12] D. Jamil and H. Zaki, "Security issues in cloud computing and countermeasures," *IJEST*, 2011.
[13] W. R. Claycomb and A. Nicoll, "Insider threats to cloud computing: Directions for new research challenges," in *COMPSAC*. IEEE, 2012.
[14] G. J. Silowash, D. M. Cappelli, A. P. Moore, R. F. Trzeciak, T. Shimeall, and L. Flynn, "Common sense guide to mitigating insider threats," 2012.
[15] Google, *TensorFlow LITE*, https://www.tensorflow.org/lite, 2022.
[16] Cainvas, *Cainvas*, https://cainvas.ai-tech.systems/accounts/login/, 2021.
[17] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song *et al.*, "Going deeper with embedded fpga platform for convolutional neural network," in *FPGA*, 2016.
[18] A. Womg, M. J. Shafiee, F. Li, and B. Chwyl, "Tiny ssd: A tiny single-shot detection deep convolutional neural network for real-time embedded object detection," in *CRV*, 2018.
[19] J. Y. Koh, *Model Zoo*, https://modelzoo.co/, 2021.
[20] ONNX, *ONNX*, https://onnx.ai/, 2022.
[21] H. Face, *Hugging Face*, https://huggingface.co/, 2022.
[22] CodaLab, *2021 VIPriors Image Classification Challenge*, https://competitions.codalab.org/competitions/33214, 2021.
[23] B. ClOUD, *The Difference Between White Box and Black Box AI*, https://bigcloud.global/the-difference-between-white-box-and-black-box-ai/4, 2022.
[24] eWEEK Staff, *Why White-Box Models in Enterprise Data Science Work More Efficiently*, https://www.eweek.com/big-data-and-analytics/why-white-box-models-in-enterprise-data-science-work-more-efficiently/, 2020.
[25] A. McNally, *Creating Trustworthy AI for the Environment: Transparency, Bias, and Beneficial Use*, https://www.scu.edu/environmental-ethics/resources/creating-trustworthy-ai-for-the-environment-transparency-bias-and-beneficial-use/, 2020.
[26] F. D. Backere, *Explainable Trustworthy AI*, https://www.vaia.be/en/courses/explainable-trustworthy-ai, 2022.
[27] B. Chen and G. W. Wornell, "Quantization index modulation: A class of provably good methods for digital watermarking and information embedding," *TIT*, 2001.
[28] F. Hartung and M. Kutter, "Multimedia watermarking techniques," *Proceedings of the IEEE*, vol. 87, no. 7, pp. 1079–1107, 1999.
[29] C. Collberg and C. Thomborson, "Software watermarking: Models and dynamic embeddings," ser. POPL '99. New York, NY, USA: Association for Computing Machinery, 1999, p. 311–324. [Online]. Available: https://doi.org/10.1145/292540.292569
[30] R. Venkatesan, V. V. Vazirani, and S. Sinha, "A graph theoretic approach to software watermarking," in *IHMMSEC*, 2001.
[31] Y. Uchida, Y. Nagai, S. Sakazawa, and S. Satoh, "Embedding watermarks into deep neural networks," in *ICMR*, 2017.
[32] H. Chen, B. D. Rouhani, C. Fu, J. Zhao, and F. Koushanfar, "Deepmarks: A secure fingerprinting framework for digital rights management of deep learning models," in *ICMR*, 2019.
[33] H. Liu, Z. Weng, and Y. Zhu, "Watermarking deep neural networks with greedy residuals," in *ICML*, 2021.
[34] B. D. Rouhani, H. Chen, and F. Koushanfar, "Deepsigns: A generic watermarking framework for ip protection of deep learning models," *arXiv preprint arXiv:1804.00750*, 2018.
[35] N. Lukas, E. Jiang, X. Li, and F. Kerschbaum, "Sok: How robust is image classification deep neural network watermarking?" in *SP*, 2022.
[36] Y. Adi, C. Baum, M. Cisse, B. Pinkas, and J. Keshet, "Turning your weakness into a strength: Watermarking deep neural networks by backdooring," in *USENIX Security*, 2018.
[37] J. Zhang, Z. Gu, J. Jang, H. Wu, M. P. Stoecklin, H. Huang, and I. Molloy, "Protecting intellectual property of deep neural networks with watermarking," in *ASIACCS*, 2018.
[38] J. Guo and M. Potkonjak, "Watermarking deep neural networks for embedded systems," in *ICCAD*, 2018.
[39] *HufuNet Project*, https://github.com/lvpeizhuo/HufuNet.
[40] T. Wang and F. Kerschbaum, "Attacks on digital watermarks for deep neural networks," in *ICASSP*, 2019.
[41] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, "Targeted backdoor attacks on deep learning systems using data poisoning," *arXiv*, 2017.
[42] T. Gu, B. Dolan-Gavitt, and S. Garg, "Badnets: Identifying vulnerabilities in the machine learning model supply chain," *CoRR*, 2017.
[43] Z. Li, C. Hu, Y. Zhang, and S. Guo, "How to prove your model belongs to you: A blind-watermark based framework to protect intellectual property of dnn," in *ACSAC*, 2019.
[44] R. Namba and J. Sakuma, "Robust watermarking of neural network with exponential weighting," in *ASIACCS*, 2019.
[45] H. Jia, C. A. Choquette-Choo, V. Chandrasekaran, and N. Papernot, "Entangled watermarks as a defense against model extraction," in *USENIX Security*, 2021.
[46] Z. Yang, H. Dang, and E.-C. Chang, "Effectiveness of distillation attack and countermeasure on neural network watermarking," *arXiv*, 2019.
[47] X. Chen, W. Wang, Y. Ding, C. Bender, R. Jia *et al.*, "Leveraging unlabeled data for watermark removal of deep neural networks," in *ICML workshop on Security and Privacy of Machine Learning*, 2019.
[48] D. Hitaj and L. V. Mancini, "Have you stolen my model? evasion attacks against deep neural network watermarking techniques," *arXiv*, 2018.
[49] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao, "Neural cleanse: Identifying and mitigating backdoor attacks in neural networks," in *SP*, 2019.
[50] Y. Liu, W.-C. Lee, G. Tao, S. Ma, Y. Aafer, and X. Zhang, "Abs: Scanning neural networks for back-doors by artificial brain stimulation," in *CCS*, 2019.
[51] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *NAACL*, 2019.
[52] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in nlp," *arXiv preprint arXiv:1906.02243*, 2019.
[53] K. Wiggers, *GPT-3 Cost*, https://venturebeat.com/2020/06/01/ai-machine-learning-openai-gpt-3-size-isnt-everything/, 2020.
[54] B. Herold, *How to Detect Code Plagiarism*, https://medium.com/codex/how-to-detect-code-plagiarism-de147cb4f421, 2021.
[55] A. Aiken, *A System for Detecting Software Similarity*, https://theory.stanford.edu/ aiken/moss/, 2022.
[56] C. Gibler, R. Stevens, J. Crussell, H. Chen, H. Zang, and H. Choi, "Adrob: Examining the landscape and impact of android application plagiarism," in *MobiSys*, 2013.
[57] Q. Xia, Z. Zhou, Z. Li, B. Xu, W. Zou, Z. Chen, H. Ma, G. Liang, H. Lu, S. Guo *et al.*, "Jsidentify: a hybrid framework for detecting plagiarism among javascript code in online mini games," in *ICSE-SEIP*, 2020.

[58] J. Ming, F. Zhang, D. Wu, P. Liu, and S. Zhu, "Deviation-based obfuscation-resilient program equivalence checking with application to software plagiarism detection," *IEEE Transactions on Reliability*, 2016.
[59] Y. He, G. Meng, K. Chen, X. Hu, and J. He, "{DRMI}: A dataset reduction technology based on mutual information for black-box attacks," in *USENIX Security*, 2021.
[60] M. Shafieinejad, N. Lukas, J. Wang, X. Li, and F. Kerschbaum, "On the robustness of backdoor-based watermarking in deep neural networks," in *IHMMSec*, 2021.
[61] T. Orekondy, B. Schiele, and M. Fritz, "Knockoff nets: Stealing functionality of black-box models," in *CVPR*, 2019.
[62] P. Baldi, "Autoencoders, unsupervised learning, and deep architectures," in *Proceedings of ICML workshop on unsupervised and transfer learning*, 2012.
[63] K. Liu, B. Dolan-Gavitt, and S. Garg, "Fine-pruning: Defending against backdooring attacks on deep neural networks," in *RAID*, 2018.
[64] E. Le Merrer, P. Perez, and G. Trédan, "Adversarial frontier stitching for remote neural network watermarking," *Neural Computing and Applications*, pp. 1–12, 2019.
[65] X. Ding, X. Zhang, N. Ma, J. Han, G. Ding, and J. Sun, "Repvgg: Making vgg-style convnets great again," in *CVPR*, 2021.
[66] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov *et al.*, "Going deeper with convolutions," in *CVPR*, 2015.
[67] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[68] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, 1998.
[69] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *HLT*, 2011.
[70] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009.
[71] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, "A survey on deep transfer learning," in *ICANN*, 2018.
[72] K. Gopalakrishnan, S. K. Khaitan, A. Choudhary, and A. Agrawal, "Deep convolutional neural networks with transfer learning for computer vision-based data-driven pavement distress detection," *Construction and building materials*, 2017.
[73] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," in *NeurIPS*, 2015.
[74] S. Zagoruyko and N. Komodakis, "Wide residual networks," *arXiv*, 2016.
[75] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, 1967.
[76] P. J. Rousseeuw and C. Croux, "Alternatives to the median absolute deviation," *JASA*, 1993.

Peizhuo Lv received the BE degree from the Xidian University, China, in 2020. He is currently working toward the PhD degree in the Institute of Information Engineering, Chinese Academy of Sciences, China. His main research interests include machine learning, AI security, backdoor attack and adversarial attack.

Pan Li received the BE degree from the University of Electronic Science and Technology of China in 2020. He is currently working toward the PhD degree in the Institute of Information Engineering, Chinese Academy of Sciences, China. His main research interests include machine learning, AI security, privacy-preserving machine learning.

Shengzhi Zhang is assistant professor in Computer Science Department, Metropolitan College at Boston University. His research interest includes IoT security, vehicle security, mobile security and system security.

Kai Chen received his Ph.D. degree in the University of Chinese Academy of Science in 2010; then he joined the Chinese Academy of Science in January 2010. He became the Associate Professor in September 2012 and became the full Professor in October 2015. His research interests include software analysis and testing; smartphones and privacy.

Ruigang Liang received her Ph.D. degree in the University of Chinese Academy of Science in 2021; then he joined the Chinese Academy of Science as an research assistant. His research interests include software security and AI security.

Hualong Ma received the BE degree from the Nankai University, China, in 2020. He is currently working toward the ME degree in the Institute of Information Engineering, Chinese Academy of Sciences, China. His main research interests include AI security and backdoor attack.

Yue Zhao received her Ph.D. degree in the University of Chinese Academy of Science in 2022; then she joined the Chinese Academy of Science as an research assistant. Her research interests is AI Security.

Yingjiu Li (Member, IEEE) is currently a Ripple professor with the Computer Science Department, University of Oregon. His research interests include IoT security and privacy, mobile and system security, applied cryptography and cloud security, and data application security and privacy. He has published more than 140 technical papers in international conferences and journals, and served in the program committees for more than 80 international conferences and workshops, including top-tier cybersecurity conferences and journals.

APPENDIX A
PROOF OF ROBUSTNESS AGAINST FINE-TUNING ATTACK

We define ΔP as a uniformly distributed discrete random variable that takes values from the change of parameters in DNN. According to Taylor Formula, we can approximate $E(|\frac{1}{\Delta P}|)$ as below:

$$\begin{aligned} E(|\frac{1}{\Delta P}|) \approx E(&\frac{1}{E(|\Delta P|)} \\ &- \frac{1}{E(|\Delta P|)^2}(|\Delta P| - E(|\Delta P|)) \\ &+ \frac{1}{E(|\Delta P|)^3}(|\Delta P| - E(|\Delta P|))^2) \\ &= \frac{1}{E(|\Delta P|)} + \frac{1}{E(|\Delta P|)^3} \cdot Var(|\Delta P|) \end{aligned}$$

So Equation (6) can be expanded as:

$$\begin{aligned} &E(|\frac{1}{\Delta P_{f_{wm}}}|) - E(\frac{1}{\beta 1} \cdot |\frac{1}{\Delta P_{EPH}}|) \\ &\approx \frac{1}{E(|\Delta P_{f_{wm}}|)} + \frac{1}{E(|\Delta P_{f_{wm}}|)^3} \cdot Var(|\Delta P_{f_{wm}}|) \\ &- \frac{1}{E(\beta 1 \cdot |\Delta P_{EPH}|)} \\ &- \frac{1}{E(\beta 1 \cdot |\Delta P_{EPH}|)^3} \cdot Var(\beta 1 \cdot |\Delta P_{EPH}|) \leq 0 \end{aligned}$$

which indicates:

$$\frac{E(|\frac{1}{\Delta P_{EPH}}|)}{E(|\frac{1}{\Delta P_{f_{wm}}}|)} \geq \beta 1 \tag{12}$$

From Equation (7), we can obtain:

$$\Delta PCC_{f_{wm}} \cdot E(|\frac{1}{\Delta P_{f_{wm}}}|) \approx \beta 2 \cdot \Delta PCC_{Hufu} \cdot E(|\frac{1}{\Delta P_{EPH}}|) \tag{13}$$

Combining Equation (12) and Equation (13), we can finally derive:

$$\begin{aligned} \Delta PCC_{f_{wm}} &\approx \beta 2 \cdot \Delta PCC_{Hufu} \cdot \frac{E(|\frac{1}{\Delta P_{EPH}}|)}{E(|\frac{1}{\Delta P_{f_{wm}}}|)} \\ &\geq \beta 1 \cdot \beta 2 \cdot \Delta PCC_{Hufu} \end{aligned} \tag{14}$$

Hence, as long as $\beta 1 \cdot \beta 2 > 1$, we can guarantee $\Delta PCC_{f_{wm}} > \Delta PCC_{Hufu}$, which ensures the robustness of HufuNet against the fine-tuning attack.

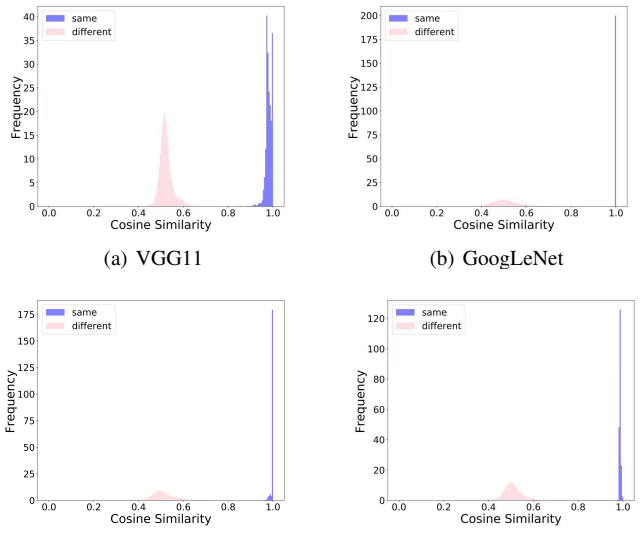
APPENDIX B
ANALYSIS OF COSINE SIMILARITY.

Cosine similarity is to evaluate the similarity of two vectors by calculating the cosine value of the angle between them. The cosine similarity for the two vectors A and B is calculated as follows:

$$similarity = \frac{A \cdot B}{\|A\|\|B\|} \tag{15}$$

We use cosine similarity to restore the order of output channels of each layer, supposing the channel reordering applied to the model. If the parameters on each output channel do not change, the restore works right away by locating those identical channels (with cosine similarity as one). However, advanced attacks may involve parameter adjustment of the model, e.g., fine-tuning, hoping to undermine the embedded watermarks effectively. We can still restore correctly in such a scenario since the cosine similarity of the same output channel before and after fine-tuning is significantly larger than that of different output channels based on our observation below. Once the output channel of one layer is restored correctly, the input channel of the next layer can be restored correspondingly due to their connections. Below, we focus on discussing the cosine similarity of the output channels.

Given a neural network model A, we can obtain A' after fine-tuning A. Therefore, an output channel OC in A becomes OC' in A'. To retrieve OC' from A' correctly based on OC, the cosine similarity of OC and OC' (referred to as the same output channel) should be distinguishable from those of OC and all other output channels (except OC') in A' (referred to as different output channels). We demonstrate the distinction via the following experiments. We first choose four models, i.e., VGG11, GoogLeNet, Resnet18 and Resnet34. These four models are used in the fine-tuning experiment in Section 5.4. Then, we fine-tune all the models with the same setting as Section 5.4. Hence, the cosine similarity of the same output channel and different output channels can be computed for each model as shown in Figure 2. The evaluation results show that for all the models, the same output channel's cosine similarity is significantly larger than those of different output channels. The former is with the mean normal distribution N (0.9766, 0.0097), which means 99.74% of values are distributed in (0.9473, 1), while the latter is with the mean normal distribution N (0.5123,0.0526), which means 99.74% of values are distributed in (0.3546, 0.6670). Therefore, we believe the cosine similarity approach can effectively restore the order of the output channels and retrieve the EPH correctly even after fine-tuning.



(a) VGG11 (b) GoogLeNet

(c) Resnet18 (d) Resnet34

Fig. 2: Distribution of Cosine Similarity of the Same Output Channel vs Different Output Channels

APPENDIX C
ANALYSIS OF CHANNEL EXPANSION.

The adversaries can randomly choose two adjacent layers from their stolen model, indicated by the matrices A and B. Then

TABLE 14: Datasets for DNN-watermarked and HufuNet

Models	Datasets for DNN-watermarked[1]	Dataset for HufuNet
VGG11	CIFAR-10	MNIST
GoogLeNet	CIFAR-10	MNIST
Resnet18	CIFAR-10	MNIST
Resnet34	CIFAR-100	MNIST
LSTM	IMDB	MNIST

TABLE 15: Parameter Sizes

	VGG11	GoogLeNet	Resnet18	Resnet34	LSTM
Before embedding	36028KB	23056KB	43712KB	83425KB	10799KB
HufuNet	835KB				
EPH	418KB				
Embedding percentage	1.16%	3.62%	0.96%	0.50%	3.87%

they increase the output dimension of A (by increasing its output channels by A_{inc}) to get A', and B (by increasing its input channels by B_{inc}) to get B'. To ensure functionality-equivalent, A_{inc} and B_{inc} should be carefully chosen, so that $(A' * B')x$ = $(A * B)x$, where x is the feature map propagated in neural networks. It implies that $(A_{inc} * B_{inc}) = 0$, which means that A_{inc} and B_{inc} have no contribution to the calculation of feature map x.

Let A be a $m * n$ matrix, B be a $n * l$ matrix; A_{inc} be a $m * k$ matrix; and B_{inc} be a $k * l$ matrix. From the equation $(A_{inc} * B_{inc}) = 0$, we have $mk + kl$ variables and ml linear equations. If $ml \geq mk + kl$, we may have a unique solution. Since zero for each variable is always a solution, the unique solution should be zeros for all the $mk + kl$ variables. Such distinguishable A_{inc} and B_{inc} are easy to be filtered out during restore. If $ml < mk + kl$, the solution is not unique. In the worst case, attacker can choose a big k, thus having multiple solutions, which are functions of $mk + kl - ml$ free variables. This means that attacker can choose $mk+kl-ml$ variables freely, while other ml variables are determined by these $mk + kl - ml$ variables.

Indeed, the adversaries would prefer the scenario of many solutions for A_{inc} and B_{inc} (i.e., $mk + kl - ml$ free variables), hoping to find one that can fail our restore approach. With a random solution, the channels of A_{inc} and B_{inc} cannot result in higher cosine similarity values with those in the EPH as demonstrated in the Figure 2 of Appendix. Without knowing our embedded EPH, the adversaries cannot find a solution containing channels with higher cosine similarity values than those in the EPH. In Section 5.5 and Section 5.6, we evaluate channel expansion attack by generating random parameters for A_{inc} and B_{inc} with the restriction of $(A_{inc}*B_{inc}) = 0$ to meet the functionality-equivalent requirement.

APPENDIX D
STEALTHINESS OF HUFUNET

According to the results in Section 5.2, our watermark EPH only occupies a small portion of the DNNs-watermarked, thus difficult to be noticed. Although there exists a correlation (i.e., HMAC) between the embedding location in DNN-watermarked and the parameters plus their indices in HufuNet, the key used to compute the HMAC and the secrecy of the indices in HufuNet make learning such correlation computationally hard. Hence, it is difficult for adversaries to learn the existence of the embedded watermark. Instead, they may resort to a statistical approach, e.g., examining the distribution of parameter values or their gradients of correct labels for any footprint of embedded watermarks.

TABLE 16: Outlier Detection

Layers	Parameter		Gradient	
	K-Means	MAD	K-Means	MAD
Conv1_1	0.72%	0.64%	0	4.92%
Conv2_1	3.94%	1.80%	1.42%	2.77%
Conv3_1	2.07%	0.98%	0.83%	1.09%
Conv3_2	1.33%	1.18%	1.31%	1.56%
Conv4_1	1.26%	1.09%	0	1.09%
Conv4_2	1.31%	1.19%	3.20%	1.37%
Conv5_1	3.91%	1.15%	0	1.90%
Conv5_2	6.09%	1.14%	0	4.36%

The adversaries with expertise in deep learning may have sufficient knowledge and deep understanding of their stolen model, e.g., roughly the parameter values distribution or the gradient distribution of neurons. They may use some outlier detection methods to discover any embedded watermark by finding abnormal parameters. Therefore, we evaluate HufuNet using the classical outlier detection methods (e.g., K-means [75] and MAD [76]) to examine abnormal parameters based on their parameter values and gradients. In terms of parameter values, when using K-Means, we cluster the parameters of each layer into two classes and consider the class with fewer samples as the outlier. Similarly, we perform MAD for each layer and take the parameters with the result greater than two as outliers. We find that learning the existence of HufuNet is almost impossible, since the F1-score is at most 6.09%, 2.50%, 1.12%, 0.71% and 16.02% for VGG, GoogLeNet, Resnet18, Resnet34, and LSTM respectively by K-Means, and at most 1.80%, 1.25%, 2.84%, 0.82% and 16.28% for these five DNNs respectively by MAD. Similarly, the gradients of the parameters (obtained from 8,000 input images in the CIFAR-10 testing dataset) are obtained for outlier detection using K-Means and MAD. The F1-score is at most 3.20%, 1.29%, 2.83%, 0.91%, and 15.37% for these five DNNs respectively by K-Means, and at most 4.92%, 1.65%, 2.84%, 1.13%, and 15.53% for these five DNNs respectively by MAD, not sufficient for outlier detection either. We show outlier detection of VGG11 in detail in Table 16.

APPENDIX E
A REPVGG-STYLE FUNCTIONALITY-EQUIVALENT ATTACK

RepVGG [65] proposes a structural re-parameterization technique to convert the DNN model with a multi-branch topology (i.e., identity, 1×1, and 3×3 branches) to a VGG-like architecture by performing the transformation based on simple algebra. That is, an identity branch or 1×1 branch can be regarded as consisting of degraded 1×1 kernels, which can be further regarded as degraded 3×3 kernels. Meanwhile, 3×3 branch is composed of 3×3 kernels. Therefore, the attackers can construct a single 3×3 kernel by applying normalization of the batch normalization layer on the trained parameters of the original identity, 1×1, 3×3 branches. With such a change of the parameter values and the architecture in the model, the attackers aim to destroy any embedded watermark.

We can still restore the model by including the inverse of the execution process of RepVGG into our restoring approaches. Note that we usually embed the parameters of HufuNet into the kernels with the largest size (e.g., 3×3 kernels in RepVGG, rather than the

TABLE 17: Against RepVGG

(a) RepVGG-Only

Attacks	Variance	Accuracy of DNN-watermarked	HufuNet
RepVGG Only	Original	69.03%	0
	RepVGG	69.03%	-
	Restore	69.03%	0

(b) Synthetic Attacks

Pruning Rate	VGG11		HufuNet		Restored Rate
	Pruning	Final	Direct	Restored	
40%	67.34%	67.06%	4.3475	0.0003	99.87%
50%	64.16%	64.20%	4.3388	0.0008	99.82%
60%	52.63%	53.24%	4.3392	0.0002	98.64%
70%	34.55%	33.89%	4.3337	0.0006	96.43%

identity and 1×1 kernels) in the watermarked models with multi-branch topology. For the suspect model with VGG-like architecture, we can first execute the standard RepVGG attack against our local watermarked model to obtain a companion model. Based on the companion model, we can restore the order of input/output channels and the convolution kernel values of the suspect VGG-like model using our restoring approaches. Since we have the values of the identity branch and 1×1 branch of the local watermarked model, we can restore the 3×3 branch of the suspect model by excluding the parameters of local identity and 1×1 branchs. Finally, we extract HufuNet from the suspect model and detect IP infringement as in Section 4.3. Besides, only when the suspect model is derived from our watermark model, the extracted HufuNet is with good performance. Moreover, a splitting attack that separates a layer into two parallel layers and concatenates their outputs can also be viewed as the RepVGG-style attack, so we can apply our restoring approaches after merging two parallel layers with the same kernel size into one layer.

We evaluate RepVGG following its default settings against the watermarked model trained on ImageNet, and the results are shown in Table 17. We launch a basic RepVGG-only attack and a synthetic attack. For the synthetic attack, we first launch RepVGG against the watermarked model and then implement the splitting attack, structure adjustment, parameter adjustment, pruning (with pruning rate 40%, 50%, 60%, 70%), and fine-tuning attacks (with 100 epochs) in turn. For the RepVGG-only attack, after we restore the model according to the above mothod, we can successfully extract HufuNet with 0 ΔPCC as shown in Table 17(a). For the synthetic attack, we can also successfully extract HufuNet with an average ΔPCC 0.0005 as shown in Table 17(b). When we prune the watermarked DNN with a 70% pruning rate, the accuracy of the DNN is only 34.55%. After fine-tuning the pruned DNN using 80% samples of the verification dataset of ImageNet, the DNN's accuracy is 33.89%, and HufuNet is still with 0.0006 ΔPCC, satisfying IP infringement detection. Thus, HufuNet is robust against RepVGG-style attack.

APPENDIX F
SETTING OF τ

The metric ΔPCC depends on a pre-defined threshold $\tau = loss_{fail} - loss_{before}$ that will impact the watermark embedding and the IP infringement detection. We evaluate different values of τ for IP infringement detection on five models that we used above. Since the evaluation results of these models are similar, we only show the results of VGG11 on CIFAR-10 below.

TABLE 18: Impact of Different Values of τ

(a) ΔPCC_{hufu} against τ_{hufu}

Models	τ_{hufu}					
	0.01	0.02	0.2	0.5	0.8	2
Watermarked DNN	2.500	1.250	0.125	0.050	0.031	0.012
Clean DNN	197.57	98.785	9.849	3.951	2.470	0.988

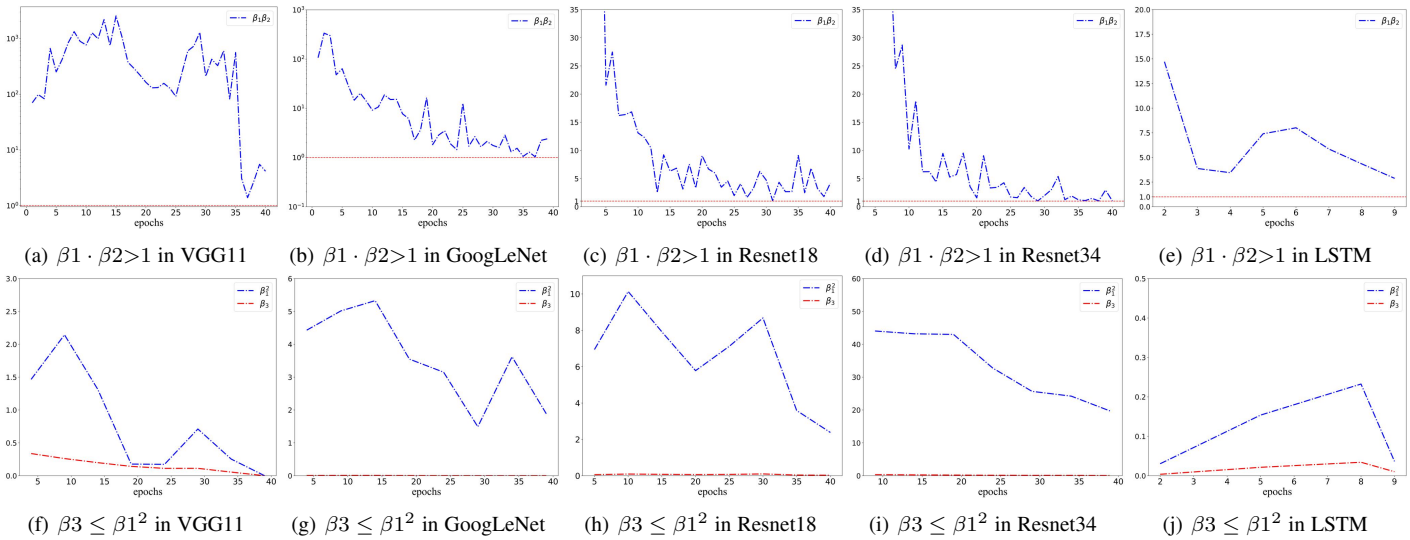
(b) Accuracy of VGG11 and ΔPCC_{hufu} against $\tau_{f_{wm}}$

Metrics	$loss_{fail}$ of $\tau_{f_{wm}}$				
	$\leq 50\%$	55%	60%	70%	80%
Accuracy	-	59.72%	61.21%	70.28%	82.05%
ΔPCC_{hufu}	-	0.177	0.068	0.061	0.064

When $loss_{fail} \leq 50\%$, we cannot guarantee $\beta 1 \cdot \beta 2 > 1$ of watermarked DNNs, so we do not calculate accuracy and ΔPCC_{hufu}.

For HufuNet, we evaluate values of τ_{hufu} from the range of $[0.01, 2]$ with a step of 0.01, where the $loss_{before}$ is 0.10 after watermark embedding, and the $loss_{fail}$ (i.e., MSE loss) is in the range of $[0.11, 2.10]$. We launch the fine-tuning attack against the watermarked and clean (i.e., innocent) VGG11, and their accuracy is 86.73% and 84.16%, respectively. We calculate and obtain the values of ΔPCC_{hufu} according to the above chosen τ_{hufu} for the watermarked and clean DNNs, as shown in Table 18(a). When τ_{hufu} is in the range of $[0.2, 0.8]$, we can also correctly detect IP infringement against watermarked DNNs (i.e., $\Delta PCC_{hufu}<1$) and not falsely detect IP infringement against innocent DNNs (i.e., $\Delta PCC_{hufu}>1$). Thus, it is not hard to choose the threshold τ_{hufu}. Overall, according to our extensive evaluation results, we recommend using τ_{hufu} with the $loss_{fail}$ as 0.6 (e.g., τ_{hufu} is 0.5 in VGG11). However, when τ_{hufu} is large (i.e., 2), it results in a false IP claim on the clean/innocent DNN (i.e., ΔPCC_{hufu} is 0.988, smaller than 1). Meanwhile, outputs of the HufuNet retrieved from the clean DNN is too blurry to satisfy IP infringement detection, as shown in Figure 5(i). Thus, we should avoid using large τ_{hufu}. When τ_{hufu} is small, i.e., 0.01 and 0.02, outputs of the HufuNet retrieved from the watermarked DNN is with high fidelity, i.e., MSE is 0.11 (Figure 5(k)), satisfying IP infringement detection. However, ΔPCC_{hufu} of the HufuNet is 1.250, larger than 1, which means the failure to detect IP infringement. Thus, we should not use small τ_{hufu} to avoid such conflict.

For f_{wm}, we also evaluate the impact of different values of $\tau_{f_{wm}}$. $loss_{before}$ of $\tau_{f_{wm}}$ is the convergence accuracy of f_{wm} during the model training. We evaluate the values of worst accuracy on CIFAR-10 acceptable to the model owner (i.e., $loss_{fail}$ of $\tau_{f_{wm}}$) from the range of $[30\%, 80\%]$ with a step of 5%. Moreover, 60% accuracy on CIFAR-10 is 70% of the model's original performance. The values of $\tau_{f_{wm}}$ (especially the $loss_{fail}$ of $\tau_{f_{wm}}$) will influence the watermark embedding in the aspect of the robustness constraints "$\beta 1 \cdot \beta 2 > 1$ and $\beta 3 \leq \beta 1^2$". Particularly, according to our evaluation, we can guarantee $\beta 1 \cdot \beta 2 > 1$ and $\beta 3 \leq \beta 1^2$ when $loss_{fail}$ is in the range of $[55\%, 80\%]$, and we show the values of $\beta 1 \cdot \beta 2$, $\beta 3$, $\beta 1^2$ in the first column of Figure 3 (for $loss_{fail}$ of 60%) and Figure 4 (for $loss_{fail}$ of 80%). However, when $loss_{fail}$ is small, i.e., 30%, the watermarked model converges difficultly and we cannot guarantee $\beta 1 \cdot \beta 2 > 1$ and $\beta 3 \leq \beta 1^2$, as shown in Figure 4. Meanwhile, the watermarked model is only 78.17% accuracy (about 9.45% performance degradation compared with the clean VGG model). Compared with other values, $loss_{fail}$ of 30% is not an optimal value, since we cannot guarantee the

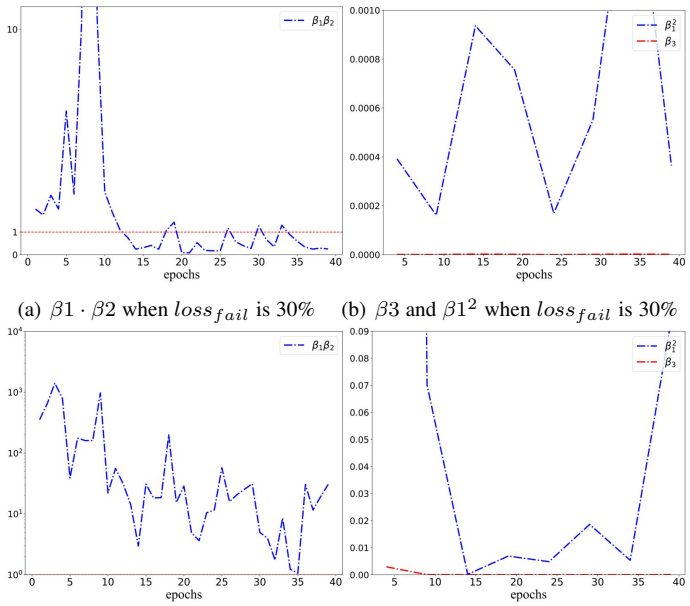


(a) $\beta 1 \cdot \beta 2 > 1$ in VGG11 (b) $\beta 1 \cdot \beta 2 > 1$ in GoogLeNet (c) $\beta 1 \cdot \beta 2 > 1$ in Resnet18 (d) $\beta 1 \cdot \beta 2 > 1$ in Resnet34 (e) $\beta 1 \cdot \beta 2 > 1$ in LSTM

(f) $\beta 3 \leq \beta 1^2$ in VGG11 (g) $\beta 3 \leq \beta 1^2$ in GoogLeNet (h) $\beta 3 \leq \beta 1^2$ in Resnet18 (i) $\beta 3 \leq \beta 1^2$ in Resnet34 (j) $\beta 3 \leq \beta 1^2$ in LSTM

Fig. 3: The values of $\beta 1 \cdot \beta 2$, $\beta 1^2$, and $\beta 3$ in DNNs, of which the $loss_{fail}$ is 70% of models' original performance.

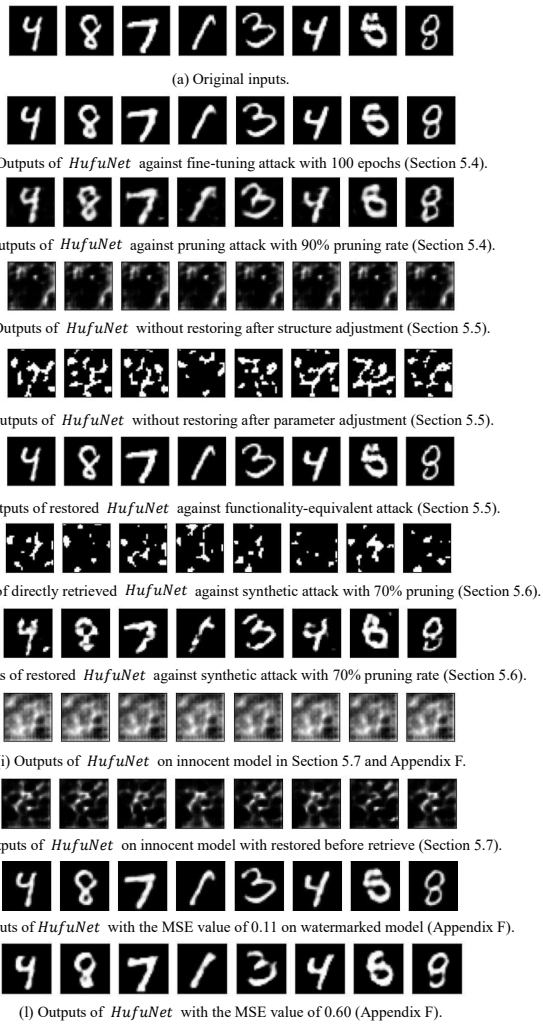
high fidelity of the watermarked model. Thus, we should avoid using small $\tau_{f_{wm}}$. Then, we fine-tune the watermarked models with $loss_{fail}$ of 55%, 60%, 70%, and 80% to make their $\Delta PCC_{f_{wm}}$ close to 1, and their accuracy is shown in Table 18(b). We use τ_{hufu} with $loss_{fail}$ of 0.6 as studied and recommended above to measure ΔPCC_{hufu} of HufuNet extracted from the fine-tuned DNNs. As shown in Table 18(b), we find ΔPCC_{hufu} is always smaller than 1 on the watermarked VGG11, satisfying IP infringement detection. Overall, we suggest choosing the appropriate $loss_{fail}$, e.g., 70% of the model's original performance as used in our evaluation (i.e., $loss_{fail}$ is 60% accuracy in VGG11).



(a) $\beta 1 \cdot \beta 2$ when $loss_{fail}$ is 30% (b) $\beta 3$ and $\beta 1^2$ when $loss_{fail}$ is 30%

(c) $\beta 1 \cdot \beta 2$ when $loss_{fail}$ is 80% (d) $\beta 3$ and $\beta 1^2$ when $loss_{fail}$ is 80%

Fig. 4: The values of $\beta 1 \cdot \beta 2$, $\beta 1^2$, and $\beta 3$ in VGG11.



(a) Original inputs.

(b) Outputs of *HufuNet* against fine-tuning attack with 100 epochs (Section 5.4).

(c) Outputs of *HufuNet* against pruning attack with 90% pruning rate (Section 5.4).

(d) Outputs of *HufuNet* without restoring after structure adjustment (Section 5.5).

(e) Outputs of *HufuNet* without restoring after parameter adjustment (Section 5.5).

(f) Outputs of restored *HufuNet* against functionality-equivalent attack (Section 5.5).

(g) Outputs of directly retrieved *HufuNet* against synthetic attack with 70% pruning (Section 5.6).

(h) Outputs of restored *HufuNet* against synthetic attack with 70% pruning rate (Section 5.6).

(i) Outputs of *HufuNet* on innocent model in Section 5.7 and Appendix F.

(j) Outputs of *HufuNet* on innocent model with restored before retrieve (Section 5.7).

(k) Outputs of *HufuNet* with the MSE value of 0.11 on watermarked model (Appendix F).

(l) Outputs of *HufuNet* with the MSE value of 0.60 (Appendix F).

Fig. 5: Samples of inputs/outputs of $HufuNet$ on VGG11.