World Wide Web
https://doi.org/10.1007/s11280-022-01113-3

Intellectual property protection of DNN models

Sen Peng[1] · Yufei Chen[1] · Jie Xu[1] · Zizhuo Chen[1] · Cong Wang[1] · Xiaohua Jia[1]

Received: 29 August 2022 / Revised: 25 September 2022 / Accepted: 28 September 2022

Abstract
Deep learning has been widely applied in solving many tasks, such as image recognition, speech recognition, and natural language processing. It requires a high-quality dataset, advanced expert knowledge, and enormous computation to train a large-scale Deep Neural Network (DNN) model, which makes it valuable enough to be protected as Intellectual Property (IP). Defending DNN models against IP violations such as illegal usage, replication, and reproduction is particularly important to the healthy development of deep learning techniques. Many approaches have been developed to protect the DNN model IP, such as DNN watermarking, DNN fingerprinting, DNN authentication, and inference perturbation. Given its significant importance, DNN IP protection is still in its infancy stage. In this paper, we present a comprehensive survey of the existing DNN IP protection approaches. We first summarize the deployment mode for DNN models and describe the DNN IP protection problem. Then we categorize the existing protection approaches based on their protection strategies and introduce them in detail. Finally, we compare these approaches and discuss future research topics in DNN IP protection.

Keywords Machine learning · Deep neural network models · Artificial intelligence security · Intellectual property protection

✉ Xiaohua Jia
csjia@cityu.edu.hk

Sen Peng
senpeng.cs@my.cityu.edu.hk

Yufei Chen
yufeichen8-c@my.cityu.edu.hk

Jie Xu
xujie.cs@my.cityu.edu.hk

Zizhuo Chen
zizhuo.chen@my.cityu.edu.hk

Cong Wang
congwang@cityu.edu.hk

[1] Department of Computer Science, City University of Hong Kong, Tat Chee Avenue, Hong Kong, Hong Kong SAR, China

Published online: 22 November 2022

1 Introduction

Over the past decade, deep learning has been widely applied in many tasks, such as image recognition, object detection, and natural language processing. It is non-trivial to train large-scale DNN models with high performance, requiring a high-quality dataset, advanced expert knowledge, and enormous computation. For instance, it costs about $25k and two weeks to train a Grover-Mega model with 1.5 billion parameters [1], which is a language DNN model for fake news detection. DNN models are valuable enough to be protected as intellectual properties from being illegally used, replicated, and reproduced. It is essential to the healthy development of deep learning techniques in academia and industry.

DNN models are typically deployed and utilized in model distribution mode and Machine Learning as a Service (MLaaS) mode [2–10]. In the early days of deep learning, the owner distributes copies of their trained host model to users in the model distribution mode [2–5]. The users deploy the obtained model copies on their devices and utilize them to solve their tasks locally. Model distribution in commercial sectors requires users to purchase authorized model copies. The owner can also distribute its host model to the public free of charge based on the open-source protocol for education and research purposes. Since model distribution gives users full access to the host model content, it can be regarded as the white-box or online deployment. However, due to the increasing size of the state-of-the-art models, it is difficult to directly distribute and deploy them for application, especially when users only obtain devices with limited computing resources. In contrast, MLaaS is a black-box, or offline deployment mode, where the owner provides the inference service to the users via the remote API [6–10]. In MLaaS mode, the host model is deployed on the cloud server by the owner. The users can query task samples through the inference API to get prediction results without directly accessing the model content.

Unfortunately, the DNN IP is vulnerable in both deployment modes. In the model distribution mode, model transformation attacks such as model fine-tuning [11, 12] and compression [13–15] can reproduce the stolen model with slight modification. The ownership claim of the stolen model can violate the DNN IP of the model owner. As for the MLaaS mode, recent studies have pointed out that the model extraction attack [16–23] can generate a surrogate model as the stolen one only through a relatively small number of queries and their inference results given by the API. Therefore, there is a great need for research on the IP protection of DNN models.

Given the significant importance, the research of DNN IP protection is still in its infancy stage. We classify the defense techniques in the literature into four categories: DNN watermarking, DNN fingerprinting, DNN authentication, and inference perturbation. DNN watermarking follows a similar idea to digital watermarking. It embeds owner-specific information, i.e., the watermark, into the owner's trained model and then deploys the watermarked model. The stolen model generated from the watermarked model inherits the embedded watermark, which can be extracted to prove the IP violation. DNN watermarking has two types of approaches: static and dynamic, depending on whether the watermark is embedded into the model's static contents, like its weights, or the dynamic contents, like its functionality. Static approaches typically modify the training loss function to change the distribution of model weights to embed the watermark. While dynamic approaches tend to construct the watermark as a trigger dataset based on the adversarial example [24, 25], or DNN backdoor attacks [26–29]. Dynamic DNN watermarking exploits the over-parameterization of the model training process, affecting the original model's functionality. Moreover, recent studies [30–34] point out that most of the existing watermarking methods are still vulnerable to watermark removal attacks.

In contrast to DNN watermarking, DNN fingerprinting does not embed any additional information into the host model. Instead, it generates a unique fingerprint for the

host model for ownership verification. The IP violation from a suspected model is verified if the fingerprint similarity between the suspected one and the host one exceeds a specific threshold, which can be determined by statistical analysis. The fingerprint can also be constructed based on the static or dynamic contents of the model as static or dynamic fingerprinting. It can also be built on both static and dynamic contents as the ensemble fingerprint, as some model similarity comparison approaches like [35]. However, DNN fingerprinting's robustness suffers from its transferability across different models, which makes the fingerprint difficult to preserve in the stolen model.

DNN watermarking and fingerprinting protect the model IP by providing proof of the violation issue's existence. Since they can only be applied when there is a suspected violation issue, we classify them as passive protection. In contrast, DNN authentication and inference perturbation are active protection approaches of the DNN IP. DNN authentication encodes the host model and generates an authentication key set before the model deployment. The user must obtain one of the authentication keys to use the encoded model normally, which gives the owner usage control to protect its IP actively. Inference perturbation aims to defend against the model extraction attack when the owner deploys its model in MLaaS mode. By slightly perturbing the results of the inference API, it can significantly degrade the performance of the surrogate stolen model obtained by the adversary, which also actively defends the IP violation.

In this paper, we present a comprehensive survey about existing approaches for DNN IP protection. The main contributions can be summarized as follows.

- We first summarize the typical deployment modes of DNN models, and describe the DNN IP protection problem along with the corresponding threat model in each deployment.
- Then we categorize the existing approaches based on their protection strategies and discuss them in detail.
- Finally, we compare these approaches based on the evaluation metrics and discuss future research topics.

We believe our work can help the researchers to gain a deeper understanding of this growing field.

The rest of the paper is organized as follows. In Section 2, we summarize the preliminaries of DNN IP protection. Section 3 gives a brief overview of the existing methods and introduces our taxonomy on DNN IP protection approaches. In Sections 4 and 5, we separately introduce the DNN watermarking and fingerprinting as the passive DNN IP protection. In Sections 6 and 7, we introduce the DNN authentication and inference perturbation as the active DNN IP protection. We compare the approaches based on the evaluation metrics and discuss the future research direction in Section 8. Finally, we give the conclusions in Section 10.

2 Preliminaries of DNN IP protection

In this section, we first introduce the DNN model and its deployment modes. Then we describe the DNN IP protection problem and present the common threats to it.

2.1 Deep neural network models

The neural network model refers to a mathematical function $\mathcal{M} : \mathcal{X} \rightarrow \mathcal{Y}$ mapping the input values $\mathcal{X}$ to the output values $\mathcal{Y}$. The basic part component of a neural network model is a neuron,

which calculates the linear combination of its input values and forwards it to its activation function, which is typically non-linear for the output values. A typical neural network model is constructed by arranging multiple layers in sequence, where each layer is composed of multiple neurons. The hidden layer refers to the one between the input and output layer, and the weights of all linear combinations in hidden layers are called the model weights $\mathcal{W}$. The DNN model refers to a neural network model with deep hidden layers. There is no consensus on the required depth of a neural network model needs to be a DNN model, but we can safely regard the neural network model that involves a greater amount of composition functions as the DNN model [36].

The weights $\mathcal{W}$ of the DNN model needs to be trained to solve the task problem. It is achieved by solving an optimization problem of the model weights over the training dataset, which aims to minimize the difference between the model inference output values and the ground truth labels. The difference is measured by a loss function $\mathcal{L}$, such as the cross-entropy loss is a multiclass classification problem. Depending on the model architecture, the DNN model can be categorized into various types, such as the Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), Generative Adversarial Network (GAN), etc. For example, the CNN that contains multiple hidden convolutional layers is typically used in image recognition since the convolution operation can extract the spatial features of images. Moreover, the RNN model consists of self-connected neurons from different timestamps, which is more likely to be exploited in natural language processing or speech recognition tasks due to its temporal feature extraction ability.

The model owner can deploy its host DNN model in two models: model distribution and MLaaS. In the model distribution mode, the host model is directly distributed to the users as copies for solving their tasks locally. Since model users can get full access to the host model, it is also referred to as the white-box deployment. In contrast, the model owner only provides the remote inference API to the users as a service in MLaaS mode, which guarantees users limited access to the host model. Since users cannot access the model's internal information, it is also referred to as the black-box deployment.

2.2 DNN IP protection problem description

We describe the DNN IP protection problem as follows. Suppose there is an owner who trains a DNN model with the training dataset, advanced expert knowledge, and much computation. We refer to the owner's obtained model after training as the host model. Moreover, suppose there is an adversary who wants to illegally replicate or reproduce a model copy from the host model and claims its ownership, referred to as model stealing. We refer to the model copy generated by the adversary as the stolen model. We also refer to the independent model trained from scratch by another honest owner as the innocent model and the model observed as a possible stolen model as the suspected model. Then the DNN IP problem aims to seek the approach which can protect the host model belonging to the owner from being illegally replicated and reproduced by the adversary. We further give the detailed problem description for each protection approach in its corresponding section.

2.3 Threats in DNN IP protection

The threats that DNN IP protection needs to defend vary in two model deployment modes. We illustrate the threats to the DNN IP for both deployment modes in Figure 1. Note that we only introduce the common attacks for the adversary, which are against all protection approaches in this section. The specific threats for each protection approach are discussed in its corresponding section.

2.3.1 Threats in model distribution mode

In the model distribution deployment mode, the common attack that an adversary conducts to generate the stolen model is the model transformation attack. It transforms the host model into the stolen model, with its functionality preserved as much the same as the host one. By illegally claiming ownership of the stolen model, the adversary steals the knowledge from the host model without authorization and violates the owner's rights. In detail, the model transformation attack can be classified into model fine-tuning and model compression.

Model fine-tuning [11, 12] refers to the approach that trains the host pre-trained model for further epochs to obtain another model which can solve the task. The model weights are initialized the same as the pre-trained model weights during the fine-tuning stage instead of randomly initialized. Model fine-tuning on the more detailed task dataset can improve the performance and costs much less computation than training it from scratch. For example, users can fine-tune the open-sourced BERT [37] on their text classification dataset to efficiently obtain a DNN model with high performance. Model fine-tuning can also be used to train a model for another task based on the pre-trained model, also referred to as transfer learning. As the model transformation attack, model fine-tuning can be applied typically in the following four settings [26, 30]:

1. Fine-Tune All Layers (FTAL): Update the weights in all layers of the pre-trained model.
2. Fine-Tune Last Layer (FTLL): Only update the weights in the last layer of the pre-trained model.

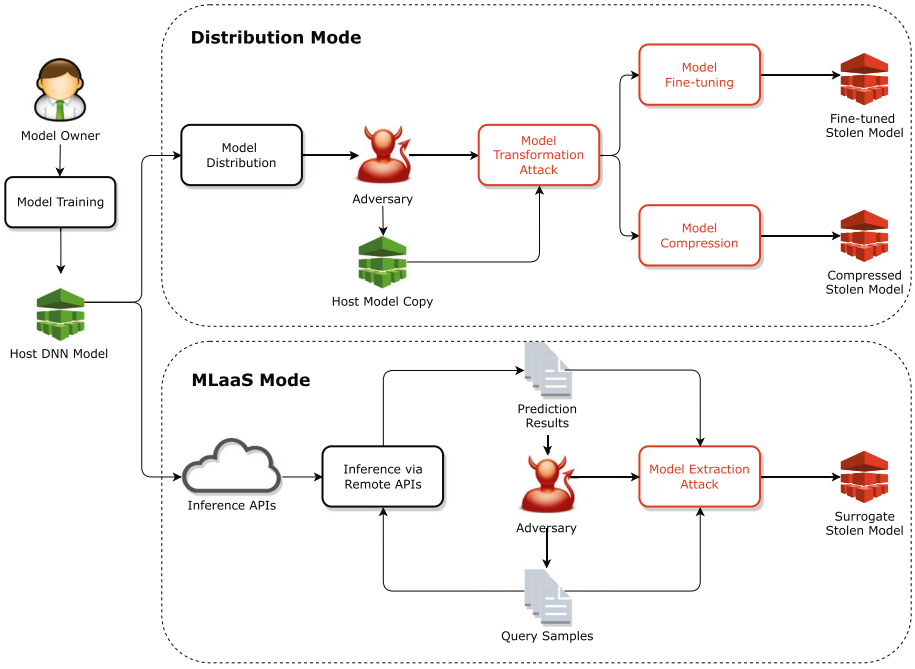

Figure 1 Threats in different deployment modes of DNN IP protection

3. Re-train All Layers (RTAL): Re-initialize the weights in the last layer and update the weights in all layers of the pre-trained model.
4. Re-train Last Layer (RTLL): Re-initialize the weights in the last layer and only update the weights in the last layer of the pre-trained model.

Model compression is another approach for the model transformation attack. Since large-scale DNN models are difficult to be directly deployed on devices with constrained computation resources, model compression aims to reduce the model size while maintaining its functionality very much the same. Model compression can be classified into the following four categories [38, 39]:

1. Pruning: Previous research [40] has pointed out that model weights are not equally important by analyzing the activations. Since the model's weights are typically stored as the sparse matrix, pruning can be achieved by zeroing out the redundant model weights [41, 42]. Besides weight pruning, the redundant neurons, filters or layers can also be removed to prune the model.
2. Quantization: DNN model weights are typically stored as 32-bit floating numbers. Model quantitation [43] reduces the number of bits used to store the model weights while maintaining its performance nearly unchanged.
3. Low-rank factorization: Low-rank factorization reduces the dimensionality of the model weight matrix to decrease the model size and inference time. The replacement process typically utilizes the factorization scheme such as the single value decomposition [44].
4. Knowledge distillation: Knowledge distillation [13] takes the host DNN model as a teacher model and trains a student model with a smaller size and almost the same functionality. The training loss of the student model contains the output difference between the teacher and student models to learn the functionality of the teacher model.

2.3.2 Threats in MLaaS mode

In MLaaS mode, we consider the adversary with only access to the prediction result of its queried samples given by the host model inference API. Model extraction [16–23] is the common attack that the adversary conducts to obtain the stolen model in MLaaS deployment mode. It is developed to recover the host model's internal information or functionality without directly accessing the host model. Based on the adversary's goal, model extraction can be classified into functionality-aimed model extraction and parameter-aimed model extraction [45, 46].

1. Functionality-aimed model extraction: The functionality-aimed model extraction attack [16–20] steals the host model's functionality for illegal usage or further benefits. The stolen model is required to have almost the same performance as the host model.
2. Parameter-aimed model extraction: The parameter-aimed model extraction attack [21–23] aims to recover the exact host model's internal information for reconnaissance. It can also be used for further adversarial attacks that need the white-box settings of the victim model.

In the typical model extraction attack, the adversary first constructs a substitute dataset by querying specifically selected samples and obtaining prediction labels through the inference API provided by the host model owner. Then it can train a surrogate model as the

stolen model on the substitute dataset, which preserves almost the same functionality with less computation than training from scratch. The adversary can illegally use the stolen model to solve its tasks or even make benefits by claiming ownership of the stolen model, which both threaten the owner's intellectual property rights.

3 Overview of DNN IP protection

In this section, we first give the taxonomy of DNN IP protection approaches. Depending on the protection strategy, DNN IP protection has two main categories: passive protection and active protection. Passive protection can be further classified into DNN watermarking and DNN fingerprinting based on whether it embeds additional information into the host model. The active protection can be classified into DNN authentication and inference perturbation depending on whether the owner deploys their host model in the model distribution or MLaaS mode. Figure 2 is an illustration of the DNN IP protection taxonomy. Then we present an overview description of each protection category and the common requirements which can be taken as the evaluation metrics.

3.1 Passive DNN IP protection

Passive DNN IP protection only takes action when the suspected IP violation is observed, which follows the passive protection strategy. These approaches present the proof that the observed suspected model is indeed a stolen model. The owner can defend their rights through further actions such as the lawsuit against this violation issue. Passive protection approaches generate the IP violation proof by tracing the specific information of the DNN model. This information is only transferrable from the host model to the stolen one, but not to the innocent model. If the similarity of this information between the host model and a suspected model exceeds a threshold that is typically determined by statistical analysis, the owner can claim the suspected model as a stolen model. Depending on whether the specific information is additionally generated and embedded into the host model, we can divide the passive protection methods into DNN watermarking and DNN fingerprinting.

3.1.1 DNN watermarking

DNN watermarking is a technique that embeds the additional owner-specific information, i.e., the watermark, into the host DNN model to protect the owner's rights. It exploits the redundancy of the parameter space in the DNN model due to multiple local optimal during model training [47]. It is feasible to alter the parameters of a DNN model from one local optimal to another nearly without decreasing its functionality, allowing the possibility to embed the watermark. The original idea of DNN watermarking comes from digital watermarking for multimedia. However, there is a significant difference between DNN watermarking and conventional digital watermarking [38]. In digital watermarking, the owner can only embed the watermark into the multimedia's static content. While in DNN watermarking, besides the static content, such as the model parameters and architectures, the owner can also embed the watermark into the model's functionality, referred to as the dynamic content. It is achieved by modifying the model parameters such that the watermarked model predicts specific results for some data samples. Depending on whether the

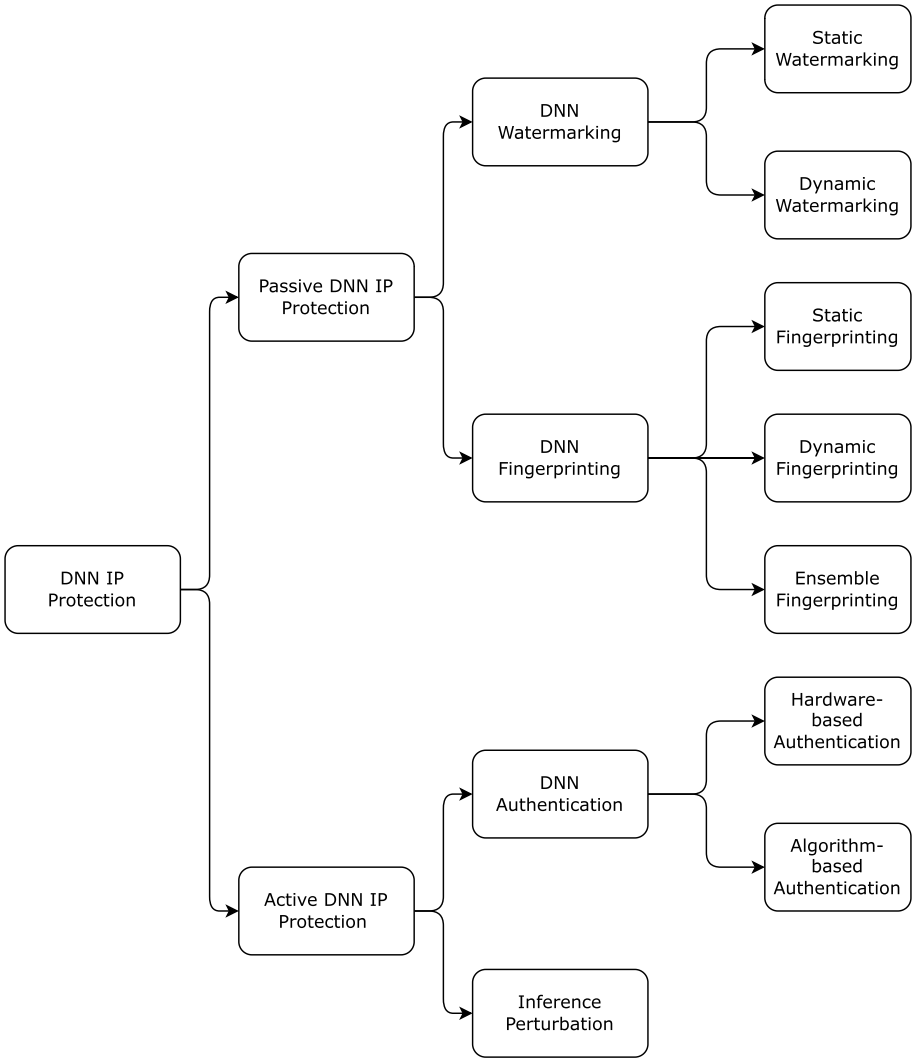

Figure 2 Taxonomy of DNN IP protection approaches

watermark is embedded into the host model's static or dynamic content, DNN watermarking can be categorized into static watermarking, and dynamic watermarking [47].

Directly adjusting the static content of the host model without careful design can severely degrade its performance. Therefore, static DNN watermarking [48–51] typically embeds the watermark by modifying the optimization loss function during the host model training or fine-tuning stage. It constrains the model parameter distribution to obtain the specific pattern, which contains the watermark information and can be transferred to the stolen model. Since the watermark extraction requires full access to the suspected model, static watermarking approaches are not applicable when the adversary deploys the stolen model in MLaaS mode, which means the verification must be processed in the white-box deployment.

Unlike static watermarking, dynamic watermarking [24–29, 52–55] embeds the watermark into the model's functionality, also referred to as the model's behavior. It is achieved by adjusting the parameters of the host model so that for samples in the watermark dataset (also called the trigger dataset), the watermarked model gives specific inference results as the owner designed. Existing dynamic watermarking approaches typically exploit adversarial examples [56–58], or DNN backdoor attacks [59, 60] to construct the watermark dataset. Adversarial examples refer to those that can be correctly classified in semantics but incorrectly classified by the model due to the linearity of its decision boundary [56]. In contrast to the adversarial examples, DNN backdoor attacks manually insert the backdoor samples into the model. The backdoor samples are specifically crafted samples with trigger patterns. By poisoning the training dataset with the backdoor samples, the trained model predicts the corresponding specified labels when samples with the trigger pattern are queried. The extraction of dynamic watermarks only needs access to the inference results of watermark dataset samples, i.e., the black-box verification deployment. It enables dynamic watermarking to defend the adversary even if it deploys the stolen model in MLaaS mode.

3.1.2 DNN fingerprinting

In contrast to DNN watermarking, DNN fingerprinting does not embed any additional information into the host model. It constructs the fingerprint based on the model's internal property, which can uniquely represent the model and be bound to the owner. DNN fingerprinting constructs the fingerprint from a model's static, dynamic, or ensemble static and dynamic content. We divide the fingerprinting approaches into static fingerprinting, dynamic fingerprinting, and ensemble fingerprinting.

Static fingerprinting [61] extracts the model fingerprint from its static attributes. However, due to the redundancy of the DNN model's parameter space, static fingerprints constructed only by the model parameters are too fragile to transfer across models. Supplementary static data related to the host model can assist in static fingerprinting. For example, in [61], the authors generate a proof of the model training process for the owner, which is consisted of snapshots of model weights during training, corresponding batches of training data, and hyper-parameters. They construct the static fingerprint as proof of learning, which can be used for ownership verification.

In contrast, dynamic fingerprint [62–66] generates the fingerprint from the model's functionality, like the inference results for a fingerprint dataset that typically constructed by adversarial examples around the decision boundary [62, 63]. The fingerprint dataset can also be built based on other measurements like Universal Adversarial Perturbations (UAPs) [64], Decision Distance Vectors (DDVs) [65], or calculated distances to the decision boundary [66]. Compared to static fingerprinting, dynamic fingerprinting can defend the owner's IP against the adversary when it deploys the stolen model in the black-box deployment.

The ensemble fingerprinting [35] combines static and dynamic aspects of a DNN model to compose the fingerprint. These approaches tend to build an expandable framework to extract and compare the model fingerprint on both static and dynamic aspects. Model similarity comparison, such as the DeepJudge in [35], is an example of ensemble fingerprinting's usage.

3.2 Active DNN IP protection

Different from passive approaches, active DNN IP protects the model by preventing the IP violation issue occurs. It encodes the model before model distribution to manage the

authorized usage control or perturbs the inference results to prevent the model extraction attack. Based on whether the owner deploys their model in distribution or MLaaS mode, we classify the active protection into DNN authentication and inference perturbation.

3.2.1 DNN authentication

DNN authentication aims to manage the owner's authorized usage control of the host model. The owner first encodes the host model to get the encoded model along with an authorization key set. Then the owner can distribute the encoded model to authorized users. To normally use the encoded model, users must obtain the valid authentication key given by the model owner. If the adversary obtains an invalid key, the performance of the encoded model degrades significantly to prevent the model from being illegally used or reproduced. Depending on whether model encoding is based on trusted hardware or designed algorithms, we divide DNN authentication into hardware-based and algorithm-based authentication.

Hardware-based authentication [67, 68] constructs the root of trust based on the devices with the Trusted Execution Environment (TEE). It typically embeds the host model and the authentication key into the TEE of hardware and distributes the device with the embedded hardware to the user or opens it to the public. Only legal users with the authentication key can normally use and access the encoded model, and the inference is also processed in the TEE to prevent possible leakage.

In contrast, algorithm-based authentication [69–72] constructs the root of trust based on the designed algorithm. It typically encodes the model by modifying its architecture based on encryption or appending specifically designed components. The authentication key can be bit-string information or samples with specific patterns. The security of the used encoding algorithm guarantees the model from being illegally used or reproduced after being distributed to users.

3.2.2 Inference perturbation

Unlike the DNN authentication, inference perturbation aims to defend the model, which is deployed in MLaaS mode by the owner. The model extraction attack is a major threat that needs to be considered. Inference perturbation [73–76] typically follows the scheme to detect the model extraction query samples from the adversary first and then return modified inference results to prevent the adversary from generating well-trained surrogate model as the stolen one.

3.3 Evaluation metrics of DNN IP protection

We summarize the most common four requirements in all DNN IP protection approaches as the evaluation metrics. Some other requirements are also discussed in previous studies [30, 47, 77–79]. These four dimensions also include the most important part of their proposed evaluation metrics, and the specific requirements for each protection category are introduced in its corresponding section.

3.3.1 Fidelity

The fidelity of a protection approach measures the performance of the deployed model on the original task after applying the protection [24–29, 48, 49, 51, 52, 65, 80]. It is

calculated by comparing the performance metrics of the host DNN model $\mathcal{M}_o$ and the deployed model $\mathcal{M}_d$, typically denoted as the ratio between their performance. We denote the fidelity of the protection approach $\mathcal{F}$ as $\mathcal{S}_{\text{fid}}(\mathcal{F})$, and it can be calculated as

$$\mathcal{S}_{\text{fid}}(\mathcal{F}) = \frac{\mathcal{P}(\mathcal{M}_d)}{\mathcal{P}(\mathcal{M}_o)}. \tag{1}$$

Here the deployed model is obtained by applying the protection as $\mathcal{M}_d = \mathcal{F}(\mathcal{M}_o)$, and $\mathcal{P}(\cdot)$ denotes the performance of a model, which is usually represented by its accuracy. Note that the fidelity always equals to one when the protection approach does not alter the host model before deployment, i.e., the deployed model is the same as the host model like in DNN fingerprinting.

3.3.2 Robustness

The robustness of a DNN IP protection approach measures its effectiveness against different threats [24–29, 48–55, 62–65, 70, 80]. The effectiveness of a protection approach refers to its defense success rate towards the stolen model obtained by different attacks. For example, in a dynamic DNN watermarking $\mathcal{F}$ which has the watermark dataset as D_{wm}, the effectiveness S_{eff} of the approach is the watermark accuracy, which is calculated as

$$S_{\text{eff}}(\mathcal{F}) = \frac{\|\mathcal{D}_{\text{verified}}\|}{\|\mathcal{D}_{\text{wm}}\|}. \tag{2}$$

Here $\|\mathcal{D}_{\text{verified}}\|$ denotes the number of successfully verified samples in the watermark dataset for the stolen models generated by the adversary. It can also be evaluated using the AUC or ROC curve [29, 53], which includes the false positive rate (also called the integrity in some studies [24, 28, 29, 49, 52]). Since the adversary can obtain the stolen model through different attacks, its robustness contains the effectiveness against these attacks. Besides the common threats mentioned above, some protection approaches are also required to be robust against specific attacks, like the watermark overwriting attack in DNN watermarking and the fingerprint forging attack in DNN fingerprinting.

3.3.3 Generality

The generality of a DNN IP protection approach measures its generalization ability towards DNN models with various architectures for different tasks [24, 26, 49, 51, 54, 70]. Many existing approaches are targeted at specific DNN model types, like the CNN model for image classification tasks. Some protection approaches, such as inference perturbation, can only be utilized when the owner deploys the host model in MLaaS mode.

3.3.4 Efficiency

The efficiency measures the time, storage, and computing resource costs of a DNN IP protection approach [24, 48, 49, 52, 54, 55, 62, 80]. It is typically calculated by comparing the cost of training the host model and deploying it with and without the protection approach applied.

4 DNN watermarking approaches

In this section, we first describe the detailed DNN watermarking problem and introduce the specific threats and requirements for DNN watermarking approaches. Then we introduce the existing static watermarking approaches, such as the first proposed static watermarking method by Uchida et al. [48]. Last, we introduce the dynamic watermarking methods, such as the first proposed dynamic watermarking method based on adversarial examples [24] and the first based on DNN backdoor attacks [26]. The comparison of these watermarking approaches is summarized in Table 1. The verification deployment mode refers to the supported deployment mode of the stolen model.

4.1 DNN watermarking problem description

Suppose there is an owner $\mathcal{O}$, who trains on the training dataset $\mathcal{D}_{\text{train}} = \{\mathcal{X}_{\text{train}}, \mathcal{Y}_{\text{train}}\}$ to obtain the host model $\mathcal{M}_o$. DNN watermarking methods protect the owner's intellectual property rights by first generating a watermark b that contains its identity information. Then the owner embeds the watermark into the host model through the embedding algorithm

$$f_{\text{embed}}(\mathcal{M}_o, b) \rightarrow \mathcal{M}_w \tag{3}$$

and deploys the watermarked model $\mathcal{M}_w$ for usage. When a suspected model $\mathcal{M}_s$ is observed by the owner, it can exploit the watermark extraction algorithm

$$f_{\text{extract}}(\mathcal{M}_s) \rightarrow \hat{b} \tag{4}$$

to obtain the extracted watermark. Then the owner can compare the extracted watermark $\hat{b}$ with the embedded one through the watermark verification algorithm

$$f_{\text{verify}}(b, \hat{b}) \rightarrow r \in \{0, 1\}. \tag{5}$$

Table 1 Comparison of DNN watermarking approaches

Method	Verification Deployment Mode	Embedding Phase	Capacity
Uchida [48]	Distribution	Training	Multi-bit
DeepMarks [49]		Fine-tuning	Zero-bit
RIGA [50]		Training	Multi-bit
Liu [51]		Training	Multi-bit
Le Merrer [24]	Distribution & MLaaS	Fine-tuning	Zero-bit
Adi [26]	Distribution & MLaaS	Fine-tuning	Zero-bit
Zhang [27]	Distribution & MLaaS	Training	Multi-bit
DeepSigns(Hidden) [52]	Distribution	Fine-tuning	Multi-bit
DeepSigns(Output) [52]	Distribution & MLaaS	Fine-tuning	Zero-bit
Guo [28]	Distribution & MLaaS	Fine-tuning	Multi-bit
Namba [53]	Distribution & MLaaS	Fine-tuning	Zero-bit
Li [29]	Distribution & MLaaS	Fine-tuning	Multi-bit
EWE [54]	Distribution & MLaaS	Training	Zero-bit
DAWN [55]	MLaaS	Inference	Multi-bit
Yang [25]	Distribution & MLaaS	Training	Zero-bit

The verification result r is typically a similarity ratio. If the ratio exceeds a predefined threshold, the owner can declare that the suspected model is indeed a stolen model $\mathcal{M}_a$ from the adversary $\mathcal{A}$. In summary, the DNN watermarking method is to develop an algorithm set $\mathcal{F}_{\text{watermark}}$ that contains three sub-algorithms as

$$\mathcal{F}_{\text{watermark}} = \{f_{\text{embed}}, f_{\text{extract}}, f_{\text{verify}}\}. \tag{6}$$

4.2 Specific threats in DNN watermarking

Besides the common threats mentioned above, there are some specific threats against the robustness of DNN watermarking approaches. They are watermark overwriting, watermark forging, and watermark evasion attacks.

4.2.1 Watermark overwriting

Watermark overwriting [25, 26, 48, 49, 51, 52] describes the attack for the adversary to embed another watermark b_a which is bound with the its identity into the watermarked model $\mathcal{M}_w$ to construct another one $\mathcal{M}_{aw}$ for illegal usage, through the same watermarking approach $\mathcal{F}_{\text{watermark}}(\mathcal{M}_w) \rightarrow \mathcal{M}_{aw}$. Since the embedded adversarial watermark can also be extracted, it can decrease the credibility of the extracted watermark, which belongs to the owner, and interfere with the watermark verification. Even if the owner's watermark can be extracted normally, the adversarial watermark still causes the ownership conflict issue, for it contains the valid identity information of the adversary. Most of the existing watermarking methods embed the watermark during the model's fine-tuning stage, which costs much less computation than those that embed the watermark during model training, making the watermark overwriting tempting to the adversary in practice.

4.2.2 Watermark forging

Watermark forging [28, 51, 51] attack for the adversary aims to construct a counterfeit watermark b_c which is bound with the adversary's identity for the watermarked model $\mathcal{M}_w$ without modifying it. The counterfeit watermark can still be extracted from the watermarked model $\mathcal{M}_w$, which typically exploits the infinite watermark data space. For example, in a dynamic watermarking method based on the adversarial example, the adversary can generate another set of adversarial examples as the counterfeit watermark b_c from $\mathcal{M}_w$ and binds b_c with its identity information. Then the adversary can claim verifiable ownership of $\mathcal{M}_w$ without modifying it since the adversary watermark dataset is a subset of the original watermark space.

4.2.3 Watermark evasion

Watermark evasion attack [29, 53] is typically applied when the adversary deploys the stolen model in MLaaS mode. In this case, to verify the IP violation, the model owner has to extract the watermark from the stolen model by querying the inference API of the stolen model. Watermark evasion attack aims to detect the extraction query from the

verifier and return misleading results to escape being identified as a stolen model. This attack relies on the detectability of the watermark extraction sample, which is due to its different data distribution from the task dataset.

4.3 Specific evaluation metrics of DNN watermarking

For the robustness of watermarking approaches, the above specific threats need to be considered besides the common threats. Moreover, the following two metrics are also supposed to be evaluated.

4.3.1 Capacity

Capacity refers to the amount of information that the embedded watermark contains. It has two levels for evaluation: zero-bit and multi-bit. The method that can only verify the existence of the watermark in a model has a zero-bit capacity. In contrast, the method that can verify the existence, as well as other owner-related information, has a multi-bit capacity.

4.3.2 Imperceptibility

Imperceptibility measures the detectability of embedded watermarks in a model without prior knowledge that the model is watermarked. Since some watermarking approaches unintentionally modify the host model to have a specific pattern, the watermarked model can be detected by the adversary and has a greater risk of being removed through adaptive attacks. For example, the static watermarking method in [48] changes the original distribution of model weights so that the adversary can easily detect and remove it. As for dynamic watermarking approaches, their imperceptibility typically relies on the detectability of the DNN backdoor or adversarial examples with specific patterns [31, 81].

4.4 Static DNN watermarking

Uchida et al. first proposed to apply watermarking in DNN IP protection [48, 80]. The owner embeds its watermark into the host model weights by training it using a modified loss function. This watermark can be embedded into any hidden layer of the host model by enforcing weights in the embedded layer to have a particular bias in distribution, which the projection operation can extract. In detail, suppose the watermark b is a binary bit-string with length T, and the flatten weights in the embedded layer are denoted as $w \in \mathbb{R}^M$. Their method defines a projection matrix $X \in \mathbb{R}^{T \times M}$, which is used in embedding and extracting the watermark. They evaluated different designs for the projection matrix and selected to generate each element in it by independently drawing from the normal distribution $\mathcal{N}(0, 1)$. Then the owner extracts the watermark $\hat{b}$ from the embedded layer by projecting its weights as

$$\hat{b} = s(Xw). \tag{7}$$

Here $s(\cdot)$ is a step function for each element that outputs 1 for the positive value and 0 for the negative. Since the watermark extraction algorithm is very like a binary classification

with a single layer perception, the authors define the embedding loss function $E_R(w)$ as the following binary cross-entropy loss

$$E_R(w) = -\lambda \sum_{i=1}^{T} (b_i \log(y_i) + (1 - b_i) \log(1 - y_i)). \tag{8}$$

Then the owner can add this embedding loss as a regularization term into the original training loss to embed the watermark. Here $y_i = \sigma(\sum_j X_{ij} w_j)$ and $\sigma(\cdot)$ is the sigmoid function and λ is an embedding trade-off parameter. They exploited the Bit Error Rate (BER) between the extracted watermark and the original one as the watermark verification algorithm. If the BER of the watermark extracted from a suspected model exceeds a certain threshold, the owner can identify the suspected model as a stolen model from an adversary.

However, Wang et al. in [34] pointed out that Uchida's method modifies the model weight distribution, making the watermark easy to be detected and possible to be removed. In [49], Chen et al. also claimed that Uchida's method is not strongly bound to the user's identity, which makes it vulnerable to the watermark forging attack. The watermark generated in Uchida's method is also deficient in the collision attack, in which several adversaries can collaborate to remove the embedded watermark. Instead, they proposed DeepMarks, an anti-collusion codebook-based static watermarking method to defend against collision attacks. Based on the anti-collision code theory, DeepMarks generates each host model copy a watermark that uniquely not only binds with the owner but uniquely binds with one individual user. DeepMarks embeds the watermark into a hidden layer of the host model copy by modifying the loss function, the same as Uchida's idea. However, since DeepMarks generates a unique watermark for each host model copy, embedding is processed during the host model fine-tuning instead of its training.

To improve the robustness and imperceptibility of the watermark, the authors of [50] proposed the static watermarking method RIGA. It embeds the watermark through an adversarial training network, where the watermark training part is a generator, and the detecting part is a discriminator. This design enforces the weight distribution of the watermarked model to be almost the same as the innocent model, improving the imperceptibility of the embedded watermark. Moreover, RIGA trains another DNN model to extract the watermark from the embedded weights instead of using a projection matrix. This design increases the capacity of the watermark, which is embedded during the model training process.

Liu et al. proposed to embed the watermark into the residual information greedily constructed from the host model weights in [51]. They first generated a private and public secret key pair based on a message with the owner's identity information and encrypted the message using the private key to obtain the ownership watermark. Then they embed the watermark into the residuals constructed from the model weights by greedily selecting. It encourages embedding the watermark into fewer but more important weights, which can enhance its robustness. The watermark extraction from a suspected model needs to be processed in the white-box setting. To verify the extracted watermark, one can decrypt it with the public key from the owner, which allows an implicit revealing of the model ownership.

4.5 Dynamic DNN watermarking

Le Merrer et al. proposed the first dynamic watermarking method in [24]. They pointed out that static watermarks can only be verified when the adversary deploys the stolen mode in distribution mode but not MLaaS mode. Instead, their proposed method embeds

the watermark into the model's functionality by exploiting adversarial examples of a DNN model. In detail, they generated a watermark dataset containing adversarial examples and other correctly classified samples around the decision boundary. By fine-tuning the host model with this watermark dataset, watermarked model's behavior is modified such that it tends to give the same inference results as the watermark dataset. The embedded watermark can be extracted by querying samples in the watermark dataset via the remote APIs of a suspected model and obtaining the prediction results. The model owner can compare the prediction results and verify the confidence. Based on the null hypothesis, the owner can declare whether the suspected model is a stolen one.

However, Adi et al. claimed that the effectiveness of dynamic watermarking approaches based on adversarial examples depends heavily on the transferability of the adversarial examples across models with different architectures [26]. Moreover, the watermark proposed in [24] has an infinite parameter space, making it vulnerable to the watermark forging attack. Instead, they proposed a dynamic watermarking method based on the DNN backdoor attack. Compared to the adversarial example, the transferability of backdoor samples depends on their construction instead of the model's decision boundary. They regarded the DNN backdoor as a watermark to embed and inserted it by fine-tuning the model with the carefully constructed backdoor trigger dataset. They encapsulated the trigger set in cryptographic commitments with secret keys to bind the watermark with the owner's identity. The owner can open the verification key to the public after the deployment of the watermarked model. The extraction and verification involve a trusted third party, who obtains the private watermarking key from the owner and queries the suspected model to get the inference results. If the results match labels in the trigger dataset and the commitment is checked valid, the third party can declare the suspected model as a stolen model.

In [27], Zhang et al. also proposed a dynamic watermarking method based on DNN backdoor attacks. Instead of binding the ownership through cryptographic commitments, they added information as the pattern on the backdoor samples in the watermark dataset. In detail, they developed three patch patterns:

1. $\text{WM}_{\text{content}}$: the owner adds meaningful information, typically related to the owner's identity, onto samples in the watermark dataset and assigns them with specific labels in the original task.
2. $\text{WM}_{\text{unrelated}}$: the owner uses images samples that are out of the task distribution and assigns them with labels in the original task.
3. WM_{noise}: the owner adds meaningless noise, such as the Gaussian noise, onto the samples and assigns them with specific labels in the original task.

The owner can then combine the watermark dataset with the original training dataset and trains the host model from scratch to embed the backdoor. The extraction and verification process follows the same paradigm as the above dynamic methods. Considering a more detailed application scenario, Guo et al. proposed a backdoor-based dynamic watermarking method for embedded systems [28]. It adapts the similar idea as $\text{WM}_{\text{content}}$ in [27], which first generates an adversarial patch based on the watermark key obtained by hashing the owner identity information. It also exploits a pseudorandom generator to assign specific labels to watermark samples.

In [52], Rouhani et al. proposed DeepSigns to embed the watermark in two different ways. The first way in DeepSigns generates a binary bit-string as the watermark with a small dataset for triggering during the later extraction. Then it embeds the watermark into

the hidden activations, which are output values of hidden layers after activation, by modifying the loss function during the host model fine-tuning. They assumed the activation distribution follows the Gaussian Mixture Model (GMM) distribution, and the watermark can then be embedded into the mean values of the GMM distribution. To extract the watermark from a suspected model, the verifier needs to input the samples in the trigger dataset to the model and obtain the hidden activations. The watermark can then be extracted using a linear projection operation from the activations. Note that the first way in DeepSigns requires the adversary to deploy the stolen model in distribution mode. In contrast, the second way in DeepSigns can defend the adversary that deploys the stolen model in MLaaS mode. It achieves this goal by only embedding the watermark in the output layer's activations, which are the confidence scores as the output inference results.

However, the proposed approaches above are not robust enough. The authors in [53] proposed the query modification attack, which is another name for the watermark evasion attack, when the adversary deploys the stolen model in MLaaS mode. This attack detects the queries sent to the stolen model by the verifier during the watermark extraction process. It is established on the observation that the extraction query sample follows a different distribution from the original training data task. In detail, the adversary trains an autoencoder using the dataset from the task distribution, which can reconstruct extraction query samples from the verifier. The adversary can compare the query sample with its output in the autoencoder to verify whether it is a watermark extraction sample. Then the adversary can modify the inference result of the detected watermark extraction query to mislead the owner and escape suspicion. To defend against the watermark evasion attack, the authors proposed a dynamic watermarking method utilizing exponential weighting. They generated samples in the watermark dataset from the task distribution and assigned random labels to avoid detection from the adversary. The owner can then fine-tune the host model on the watermark dataset along with the exponential normalization of model weights in embedded layers. This exponential weighting can improve the watermark robustness against fine-tuning and model pruning.

Li et al. also presented the watermark evasion attack as a threat [29]. Moreover, they claimed that dynamic watermarking methods could be vulnerable to the watermark forging attack. To defend against these two attacks, they proposed the method to embed the imperceptible watermark into the host model based on the DNN backdoor. It exploits an encoder and a discriminator to generate watermark samples that follow the same distribution as the original task. The encoder constructs watermark samples from the task distribution with the owner's logo patched on, and the discriminator ensures watermark samples are undetectable in distribution.

In [54], the authors claimed another severe attack, the model extraction, that threatens the watermark robustness when the host model is deployed in MLaaS mode. Unlike watermark extraction queries, model extraction attack queries from the adversary can follow the same distribution as the task distribution, making it hard to detect and defend. They also observed that the neurons activated in the watermark extraction process differ from those in normal task inference, meaning the watermarked model treats the original and watermarking tasks separately. This observation explains why the watermark can be easily embedded into the host model by fine-tuning for several rounds and can also be removed by model transformation or compression attacks. Based on this observation, they proposed EWE, an entangled watermark embedding method, to defend the model extraction attack. EWE first generates a watermark dataset containing the backdoor samples and combines it with the original training dataset. Then it adds a term in the training loss function for watermark embedding: the soft nearest neighbor loss [82, 83].

This loss constrains the watermark data manifold in representation space from being entangled with the task data, ensuring the same neurons are activated in both tasks.

The authors in [55] also focused on the model extraction attack and proposed DAWN to defend it. DAWN first generates the watermark key as a binary bit-string and exploits it in the random shuffling function, which assigns modified labels to query samples. Then DAWN provides the model owner, who deploys the host model in MLaaS mode, an add-on component appended after the inference API output. This component randomly decides whether to modify a query output based on a cryptographic hash function using the watermark as the randomness source. Since the adversary uses the prediction labels combined with the query samples as the training dataset in the model extraction attack, the watermark is preserved during the training process of the adversary, and the stolen model can be verified if the adversary deploys the model in MLaaS monosodium-glutamate.

In [25], the authors proposed to embed the more robust watermark inspired by the DNN fault attack. The DNN fault attack [84, 85] aims to search for the more critical model weights in task inference and modify them to largely degrade the model's performance. They applied a similar idea with DNN fault attacks in locating the more critical model weights for the watermark extraction and modifying them to enhance the watermark robustness. They developed a bi-level optimization framework based on adversarial examples for the watermark embedding process. In each optimization iteration, the exemplar-level problem in the framework searches for watermark samples around the decision boundary, and the masked adaptive optimization adjusts the model weights, which are more dominant to the watermarking task, to improve its robustness. The owner can embed a more robust watermark against removal threats by modifying a small portion of the host model weights.

5 DNN fingerprinting approaches

In this section, we first describe the detailed DNN fingerprinting problem and introduce specific threats and requirements for DNN fingerprinting approaches. Then we introduce static fingerprinting approaches such as the Proof-of-Learning [61] and dynamic approaches typically based on the observation that the decision boundary of the DNN model is unique [62]. Last, we introduce ensemble fingerprinting, such as the model similarity comparison framework in DeepJudge [35]. The comparison of fingerprinting approaches is summarized in Table 2.

5.1 DNN fingerprinting problem description

Suppose there is an owner $\mathcal{O}$, who trains a DNN model with training dataset $\mathcal{D}_{\text{train}} = \{\mathcal{X}_{\text{train}}, \mathcal{Y}_{\text{train}}\}$ to get the host model $\mathcal{M}_o$ and deploy it. DNN fingerprinting approaches generate a unique fingerprint g for $\mathcal{M}_o$ to protect the owner's rights through the fingerprint generation algorithm

$$f_{\text{generate}}(\mathcal{M}_o) \rightarrow g. \tag{9}$$

When a suspected model $\mathcal{M}_s$ is observed by the owner, it can extract the fingerprint $\hat{g}$ from it by

$$f_{\text{extract}}(\mathcal{M}_s) \rightarrow \hat{g}. \tag{10}$$

Then the owner can compare the similarity of these two fingerprints through the verification algorithm

$$f_{\text{verify}}(g, \hat{g}) \rightarrow r \in \{0, 1\}. \tag{11}$$

The verification result is typically a similarity ratio. If it exceeds a certain threshold, the owner can declare the suspected model $\mathcal{M}_s$ is indeed a stolen model $\mathcal{M}_a$ from an adversary $\mathcal{A}$. Note that the f_{generate} and f_{extract} can be same for some fingerprinting approaches. In summary, DNN fingerprinting problem for the owner is to develop an algorithm set containing three sub-algorithms as

$$\mathcal{F}_{\text{fingerprint}} = \{f_{\text{generate}}, f_{\text{extract}}, f_{\text{verify}}\}. \tag{12}$$

5.2 Specific threats in DNN fingerprinting

Besides the common threats mentioned above, two specific threats must be considered in DNN fingerprinting. They are fingerprint forging and fingerprint evasion attacks.

5.2.1 Fingerprint forging

Fingerprint forging attack [35, 63, 64] for the adversary aims to construct the counterfeit fingerprint which binds to the adversary's identity for illegal ownership verification. It exploits the infinite fingerprint data space in some fingerprinting approaches, similar to the watermarking forging attack. For example, in IPGuard [62] which generates the fingerprint based on adversarial examples around the decision boundary, the adversary can adopt another different set of adversarial examples based on the deployed model and binds the forged fingerprint with its identity [63]. The root cause is that the generated fingerprint cannot uniquely represent the DNN model.

5.2.2 Fingerprint evasion

Fingerprint evasion attack [35, 63, 64] for the adversary adopts a similar idea to watermark evasion, which aims to detect the fingerprint extraction query from the verifier when the

Table 2 Comparison of DNN fingerprinting approaches

Method	Fingerprint Type	Verification Deployment Mode	Fingerprint Composition
PoL [61]	Static	Distribution	Ensemble training proof
IPGuard [62]	Dynamic	Distribution & MLaaS	Adversarial examples
Lukas [63]			Conferrable adversarial examples
Peng [64]			UAPs
ModelDiff [65]			DDVs
DI [66]	Ensemble	Distribution & MLaaS	Inference results of training dataset
DeepJudge [35]			Ensemble metrics

adversary deploys the stolen model in MLaaS. It typically exploits the different distribution of fingerprint extraction samples compared to the original task. By detecting and modifying misleading inference results of these extraction samples, the adversary can escape the fingerprint verification.

5.3 Specific evaluation metrics of DNN fingerprinting

For robustness of the fingerprinting approaches, the above specific threats need to be considered besides the common threats. Moreover, the following requirement also needs to be evaluated in DNN fingerprinting.

5.3.1 Uniqueness

The uniqueness of the fingerprint measures whether the extracted fingerprint can uniquely represent the DNN model. It is required in all DNN fingerprinting approaches that verify the fingerprint's validity by similarity comparison since it ensures the effectiveness of the verification process.

5.4 Static DNN fingerprinting

Since the fingerprint is too fragile to transfer across models if it is only based on the model's weights, supplementary static contents are used in assisting the static DNN fingerprinting. The authors of [61] proposed Proof-of-Learning (PoL) to defend the DNN model against illegal IP violation issues. PoL constructs a proof of the training process of the host model as the fingerprint, which consists of the snapshots of model weights during training, corresponding training data batches, and model hyper-parameters. This proof can be used to verify the validity of the model's training process by reconstructing the training process step by step. For a suspected model, the owner can provide its PoL to a trusted third party and requires the PoL of the suspected model for ownership verification. The authors claimed that generating valid PoL costs no less than training the model from scratch, making the fingerprint forging not tempting to the adversary anymore. However, in [86], the authors pointed out that PoL is vulnerable to their proposed attack. They proposed an effective and efficient fingerprint forging attack by utilizing adversarial examples. The proposed attack can manipulate the generation of a spoofing PoL for any arbitrary model, which costs much less than that training the model from scratch. It makes the fingerprint forging attack effective and feasible to the adversary.

5.5 Dynamic DNN fingerprinting

The authors of [62] proposed the dynamic fingerprinting method IPGuard, based on their observation that the decision boundary of a DNN model can uniquely represent itself. To characterize the decision boundary, IPGuard selects a fingerprint dataset from the task distribution as its approximation, containing adversarial examples close to the decision boundary. It also develops a more efficient adversarial perturbation method to generate fingerprint dataset samples. To extract the fingerprint from a suspected model, the owner can query samples in the fingerprint dataset to the suspected model's API and obtain the inference results. By comparing the matching rate of inference results with original labels, the owner can declare whether the suspected model is a stolen model. Since the extraction and

verification can be processed in the black-box setting, IPGuard can defend the adversary that deploys the stolen model in MLaaS mode.

However, in [63], the authors pointed out that IPGuard is vulnerable to the model extraction attack. The stolen model generated by model extraction has a different fingerprint in IPGuard than the host model. The reason is that adversarial examples used in IPGuard are not transferrable from the host model to the stolen model generated by model extraction. To defend against this attack, they proposed a dynamic watermarking method based on conferrable adversarial examples. They claimed that not all adversarial examples are transferrable across models. Conferrable examples are a subset of the adversarial examples which can only transfer from the host model to the surrogate model generated by model extraction. They exist since the surrogate model generated by model extraction is trained on the substitute dataset instead of the dataset sampled from the original task distribution. This fingerprint construction ensures their proposed method to defend against the model extraction attack. Moreover, they also developed a searching algorithm to efficiently generate the conferrable examples given the host model.

Based on the similar observation that the decision boundary can uniquely represent a DNN model, Peng et al. proposed another way to characterize it through the model's Universal Adversarial Perturbations (UAPs) [64]. UAPs [87] are a set of perturbation vectors that can be added to any task data sample to convert into an adversarial example. They are calculated given the model weights by solving an optimization problem for searching in different classification labels. They observed that UAPs drawn from the low-dimensional subspace contain norm vectors to the model decision boundary, which can be used to characterize it perfectly as the model's fingerprint. To extract the fingerprint without access to the model weights, they developed an algorithm to approximate the suspected model's UAPs by only querying several samples and obtaining their inference results. It allows their proposed method to defend against the adversary that deploys the stolen model in MLaaS mode. They also trained an autoencoder to extract features from the model's UAPs for similarity comparison.

Moreover, the authors of [65] proposed ModelDiff to represent the model's decision boundary as its fingerprint using the model's Decision Distance Vectors (DDVs). Each value in a DDV is the distance of outputs predicted by the model for two input data samples from the task distribution. ModelDiff assumes that models with similar decision boundaries tend to have similar inference results for the same task samples. To uniquely characterize the model's decision boundary, ModelDiff generates the DDVs of a model containing the usual and transferrable adversarial examples. The similarity of fingerprints can be simply measured through the cosine similarity between the DDVs of the host model and the suspected model. Since the fingerprint extraction of a suspected model can be processed by querying the inference API, ModelDiff can also defend the adversary that deploys the stolen model in black-box mode.

Unlike the above dynamic fingerprinting approaches based on the observation of the unique decision boundary, the authors of [66] proposed Dataset Inference (DI) to defend the DNN model's IP. DI is based on the assumption that the stolen model must contain direct or indirect information about the host model's training dataset since it steals its knowledge. Moreover, they observed that the stolen model is more confident about the samples from the host model's training dataset than those randomly drawn from the task distribution. Distances to the decision boundary from samples in the training dataset are smaller than those from random samples in the task distribution. DI takes the inference confidence of samples in the training dataset as its fingerprint and develops two representations of the inference confidence. For the white-box deployment verification, DI directly calculates the distance from a task sample to its neighbor classes as the fingerprint. As for the black-box deployment verification, DI calculates the distance by querying samples in a random walk. It also exploits a regressor to

generate the confidence score based on the given distances to verify the similarity through the null hypothesis.

5.6 Ensemble DNN fingerprinting

Combining the features of static and dynamic fingerprinting, the authors of [35] proposed DeepJudge, a DNN model similarity comparison framework, to defend the model's IP. The metrics used to calculate the model similarity in DeepJudge compose the ensemble model fingerprint. In detail, this ensemble fingerprint consists of metrics from different levels of a DNN model, such as the neuron output distance from the neuron level, the layer output distance from the layer level, etc. Some static metrics are measured in the white-box deployment verification, and others are in the black-box by querying samples to the inference API. Each similarity metric in DeepJudge has its threshold based on the statistical analysis. If the similarity score is larger than its corresponding threshold, DeepJudge gives a positive vote on this metric. By collecting the positive and negative votes, the verifier can claim whether the suspected model is a stolen model. DeepJudge builds an expandable framework to systematically compare the DNN model similarity based on various metrics.

6 DNN authentication approaches

DNN authentication is used to actively protect the model's IP when the owner deploys their model in distribution mode. In this section, we first describe the detailed DNN authentication problem and then give the introduction containing hardware-based authentication approaches and algorithm-based approaches. Comparison of DNN authentication approaches is summarized in Table 3.

6.1 DNN authentication problem description

Suppose there is an owner $\mathcal{O}$, who trains a DNN model with the dataset $\mathcal{D}_{\text{train}} = \{\mathcal{X}_{\text{train}}, \mathcal{Y}_{\text{train}}\}$ to obtain the host model $\mathcal{M}_o$. To actively protect the model's ownership, the owner encodes the host model with the algorithm

$$f_{\text{encode}}(\mathcal{M}_o) \rightarrow \mathcal{M}_e, \mathcal{K} \tag{13}$$

to obtain the encoded model $\mathcal{M}_e$. Here $\mathcal{K}$ denotes an authentication key set later distributed to valid model users. The owner then distributes the encoded model copy $\mathcal{M}_e$ to each authorized user with a valid authentication key $k \in \mathcal{K}$. For a task data sample $x \in \mathcal{X}$ the user can only use the encoded model normally with this key as

$$f_{\text{predict}}(\mathcal{M}_e, k, x) \rightarrow y = h(\mathcal{M}_o, x). \tag{14}$$

Here h is the inference function that takes the model and sample as inputs. If an adversary who has an invalid key $k_a \notin \mathcal{K}$ tries to use the encoded model, the inference algorithm gives a different result as

$$f_{\text{predict}}(\mathcal{M}_e, k_a, x) \rightarrow \hat{y} \neq h(\mathcal{M}_o, x). \tag{15}$$

It prevents the adversary from normally using the model and protects the model. In summary, the DNN authentication problem is to develop an algorithm set containing two sub-algorithms as

$$\mathcal{F}_{\text{authentication}} = \{f_{\text{encode}}, f_{\text{predict}}\}. \tag{16}$$

6.2 Hardware-based DNN authentication

Chen et al. in [67] proposed DeepAttest, a hardware-based DNN authentication method using the TEE. In detail, DeepAttest first generates a unique signature as the authentication key. Then it inserts an encoded model copy along with its key into each TEE-supported device such as the Intel SGX and distributes the embedded device to an authorized user. DeepAttest encodes the model by embedding its corresponding signature as a watermark into it through modifying the loss during model fine-tuning, similar to the static watermarking method in [48]. To normally use or access the encoded model, the TEE verifies that the user's authentication key must be the same as the model's embedded signature. The inference is also processed in the TEE, which actively guarantees the authorized usage of the owner's model.

The authors of [68] proposed a similar hardware-based authentication method called HPNN. It exploits the hardware as the root of trust and embeds an authentication key into it. Moreover, HPNN develops a back-propagation algorithm that requires the authentication key as one input during model training to obtain an obfuscated encoded model. The encoded model is embedded into the hardware with the authentication key before the owner distribution. Since the encoded model is embedded into the trust environment of the hardware, HPNN ensures the user can only use or access the model normally with the authentication key obtained. The adversary who obtains the invalid authentication key generates inference results with a largely decreased credibility.

6.3 Algorithm-based DNN authentication

Instead of being based on the trusted hardware, Fan et al. in [69] proposed a DNN authentication method by appending passport layers in the host model to obtain the encoded model for distribution. It modifies the model's architecture by appending the passport layer and

Table 3 Comparison of DNN authentication approaches

Method	Authentication Type	Root-of-trust	Authentication Key
DeepAttest [67]	Hardware-based	TEEs	Binary bit-string
HPNN [68]			
Fan [69]	Algorithm-based	Model architecture	Constructed passport
Zhang [70]		Model architecture	Constructed passport
ChaoWs [71]		Chaotic encryption	Binary bit-string
ActiveGuard [72]		Model architecture	Adversarial examples

trains the model with its passport as one input, which can be regarded as the authentication key. The passport can either be a trigger dataset or a piece of binary bit-string information in their proposed method. By training the model with the passport and deploying the encoded model, the owner can ensure that only the user who obtains a valid passport can utilize the model normally.

Zhang et al. in [70] also proposed a passport-aware authentication method by utilizing the normalization layer during training. By appending the normalization layer as the passport layer and jointly training them, the user with a valid passport can pass the authentication and use the model normally, while the illegal user suffers from a significant accuracy dropping. Unlike the passport-based authentication method in [69], the passport-layer is discarded during the inference stage, which is only used for user identity authentication. It keeps the fidelity of authenticated DNN model remains unchanged.

Similar to the passport method in [69], Xue et al. also proposed a DNN authentication method, ActiveGuard, based on the adversarial example [72]. The owner first generates the authentication key dataset containing the adversarial examples of the host model in ActiveGuard. Then it appends an additional authentication layer after the host model output, which takes the adversarial sample in the authentication key dataset as one input. By fine-tuning the host model with this authentication layer, the owner can obtain the encoded model and deploy it for usage.

Instead of altering the model's architecture before training, Lin et al. in [71] proposed Chaotic Weights (ChaoWs), a weight position rearrangement method to encode the host model after it has been trained. Inspired by the parameter encryption method in [88], ChaoWs exploits the chaotic encryption algorithm [89] to generate the encryption sequence with an authentication key. The owner can rearrange the positions of neurons in the fully-connected and convolutional layers based on the encryption sequence to encode the host model. Since neurons in the encoded model are in disorder, the adversary without the valid authentication key can not utilize it normally to get valid inference results. Only the users with the valid keys given by the owner during model deployment can exploit the model normally, which manages the user authentication for the model owner. ChaoWs has much less overhead comparing other model encoding algorithms, such as homomorphic encryption, and it does not affect the model's fidelity.

7 Inference perturbation approaches

When the owner deploys the model in MLaaS, model extraction becomes the main threat against the model IP protection. Inference perturbation approaches aim to defend the model extraction conducted by the adversary. In this section, we first describe the inference perturbation problem and then introduce the inference perturbation approaches in detail.

7.1 Inference perturbation problem description

Suppose there is an owner $\mathcal{O}$, who trains a DNN model with the training dataset $\mathcal{D}_{\text{train}} = \{\mathcal{X}_{\text{train}}, \mathcal{Y}_{\text{train}}\}$ to obtain the host model $\mathcal{M}_o$. The owner then deploys the host model in MLaaS mode, which provides a remote query API as

$$h_{\text{API}}(\mathcal{M}_o, x) = y \tag{17}$$

for a task sample $x \in \mathcal{X}$ with the inference result $y \in \mathcal{Y}$ given by the host model. The adversary $\mathcal{A}$ intends to conduct the model extraction attack by querying samples to the

API and obtaining the prediction results. By constructing the substitute dataset $\mathcal{D}_{\text{substitute}}$ using the query samples and results, the adversary can train a surrogate model as the stolen model. To defend against this attack, the owner can first detect the model extraction query through the algorithm

$$f_{\text{detect}}(x) \rightarrow r. \tag{18}$$

It determines whether the sample x from the adversary is a model extraction attack sample by giving the confidence score r. If the score exceeds a threshold as a model extraction query from the adversary, the owner can perturb its inference result by the perturbation algorithm

$$f_{\text{perturb}}(\mathcal{M}_o, x) \rightarrow \hat{y} \tag{19}$$

and return the perturbed result $\hat{y}$ to avoid this attack. In summary, the inference perturbation is to develop an algorithm set containing two sub-algorithms as

$$\mathcal{F}\text{perturbation} = \{f_{\text{detect}}, f_{\text{perturb}}\}. \tag{20}$$

7.2 Inference perturbation

To defend against the model extraction attacks, which need the confidence scores of query samples to construct the substitute dataset, Lee et al. proposed to perturb the output activations of inference result given by the API in [73]. Their method modifies the output activation distribution in the model's output layer, i.e., the probabilities of each class while maintaining the inference result unchanged. Moreover, instead of detecting the model extraction query, they perturb the inference result of every query to the API before returning it to the user. By this simple inference perturbation, the adversary has to train the surrogate model with much more query samples and lower performance. However, it also affects the model's fidelity when legal users intend to use it.

The authors of [74] also proposed an inference perturbation method, Prediction Poisoning, to perturb the output activations. It adds a targeted noise to the final activation output values, which are the posteriors of the inference results. The noise is generated to maximize the deviation from the original gradient when the adversary trains the surrogate model. The added noise enforces the obtained stolen model from the adversary to have significantly lower performance than the host model.

Note that some model extraction attacks only need labels as inference results, such as [90]. To defend against this kind of model extraction attack, in [75], the authors proposed PARDA to defend against the model extraction attack from the adversary. They observed that query samples of existing model extraction attacks have a different distribution from the natural task. Therefore, they developed PARDA to exploit this observation by detecting the model extraction query and perturbing inference results of these query samples. It prevents the adversary from generating a well-trained stolen model since the substitute dataset has a low quality due to inference perturbation. However, they also pointed out that PARDA is vulnerable when the model extraction queries are carefully selected, or some dummy queries are added.

Based on a similar observation, the authors of [76] proposed the Adaptive Misinformation (AM) method to actively prevent the model extraction attack. The proposed method is based on their observation that all the model extraction attack queries are generated out-of-distribution. To separate these model extraction queries, they utilize the Maximum

Softmax Probability (MSP) of the query data as the criterion. Data samples with high MSP values indicate benign queries in the original task distribution and vice-versa. After detecting the model extraction queries from the adversary, AM simply returns incorrect inference results to decrease the substitute dataset's quality and prevent generating the stolen model.

8 Performance comparisons

We summarize the general comparisons of DNN IP protection approaches based on the evaluation metrics in Table 4. For the fidelity requirement, since DNN watermarking alters the model with over-parameterization, the deployed model's functionality is typically affected with fidelity smaller than one. DNN fingerprinting approaches always obtain fidelity equal to one since the deployed model is the host model without any modification. DNN authentication encodes the host model as the deployed model. The fidelity of an authentication method is determined by its encoding and decoding algorithms. In most of the DNN authentication approaches, these two algorithms are lossless, which means the decoded model performs the same as the host model. Therefore, most of the DNN authentication approaches can obtain fidelity equals one. As for inference perturbation, it alters the inference results of the deployed model, which brings a negative effect on the model's functionality with fidelity smaller than one.

We compare the robustness of DNN IP protection approaches in detail. For DNN watermarking approaches, the adversary can conduct specific attacks such as watermark forging, overwriting, and evasion. We summarize the claimed robustness of watermarking approaches against various attacks in Table 5. This table shows that model extraction is a significant common threat to the watermarking method's robustness which is not widely considered. Moreover, most existing methods do not consider specific attacks such as watermark overwriting and evasion. Some existing studies also discuss the robustness of DNN watermarking. For instance, in [30], the authors indicated that most existing watermarking approaches are vulnerable when facing a combination of common threats and adaptive attacks. They constructed a DNN watermarking benchmark and tested the robustness of existing watermarking approaches. Experimental results demonstrate that these methods are not as robust against the threats as they have claimed.

Just like DNN watermarking, DNN fingerprinting has specific attacks as threats like fingerprint forging and evasion. We summarized the claimed robustness of DNN fingerprinting approaches against various attacks in Table 6. It is noteworthy that some existing methods embedded a unique watermark into the host DNN model for each model user and named their methods as one of the DNN fingerprinting. In our taxonomy, it is inappropriate since the difference between DNN watermarking and fingerprinting is whether additional information is embedded into the model or not.

As for DNN authentication approaches, since the encoding mechanism guarantees its security, whether hardware-based or algorithm-based, its robustness is only tested by adaptive attacks. Moreover, inference perturbation aims to defend against the model extraction attack. Its robustness is tested by its effectiveness against different model extraction attacks.

Table 4 General comparisons of protection approaches on evaluation metrics

Protection Approach	Fidelity	Robustness	Generality	Efficiency
DNN watermarking	≤ 1	All attacks	Distribution & MLaaS Architecture dependent	Low
DNN fingerprinting	$= 1$	All attacks	Distribution & MLaaS Architecture dependent	Fast
DNN authentication	$= 1$	Only adaptive attacks	Distribution Architecture dependent	Fast
Inference perturbation	≤ 1	Only model extraction	MLaaS Architecture independent	Fast

Most of the existing approaches are designed for all DNN models with different architectures and purposes. However, some of them only focus on the specific type of DNN models. For example, some DNN watermarking approaches can only be applied in the CNN [91] or GAN [92]. Moreover, as shown in Table 4, DNN authentication can only be applied in the model distribution mode since it encodes the host model before deployment, and inference perturbation can only be applied in MLaaS deployment since it aims to defend model extraction attack in the black-box setting.

The efficiency of the protection approach varies with the complexity of their designs. There is typically a trade-off between the efficiency and robustness of a protection approach. Among all four main protection categories, DNN watermarking is relatively less efficient since it typically involves the model training and fine-tuning stages. Comparing to it, the other three protection are more efficient, especially inference perturbation since it only perturbs the prediction results.

Table 5 Claimed robustness of DNN watermarking approaches against different threats

Method	Model Fine-tuning	Model Compression	Model Extraction	Watermark Overwriting	Watermark Forging	Watermark Evasion
Uchida [48]	✓	✓	✗	✗	✗	✗
DeepMarks [49]	✓	✓	✗	✗	✗	✗
RIGA [50]	✓	✓	✗	✓	✗	✗
Liu [51]	✓	✓	✗	✓	✓	✗
Le Merrer [24]	✓	✓	✗	✗	✗	✗
Adi [26]	✓	✗	✗	✓	✗	✗
Zhang [27]	✓	✓	✗	✗	✗	✗
DeepSigns [52]	✓	✓	✗	✓	✗	✗
Guo [28]	✗	✗	✗	✓	✗	✗
Namba [53]	✗	✓	✗	✗	✗	✓
Li [29]	✓	✗	✗	✗	✓	✓
EWE [54]	✓	✓	✓	✗	✗	✗
DAWN [55]	✗	✗	✓	✗	✗	✗
Yang [25]	✓	✓	✗	✓	✗	✗

9 Future research topics

In this section, we discuss future research topics about DNN IP protection. We propose four future directions in the research of DNN IP protection: theoretical analysis, DNN fingerprinting, defense against threats in MLaaS and generality.

9.1 Theoretical analysis

Although many DNN IP protection approaches have been proposed, they still lack rigorous theoretical analysis, which makes it risky to apply these protection approaches in real applications. The effectiveness of some existing approaches in DNN watermarking and fingerprinting depends heavily on their assumptions and observations. The effectiveness of dynamic watermarking and fingerprinting approaches based on adversarial examples is determined by the transferability of adversarial examples across the models. Some fingerprinting approaches also empirically observe that the decision boundary can uniquely represent a DNN model, which is used as the foundation of their methods. It is a big challenge to prove the transferability or uniqueness of the model decision boundary to theoretically support the effectiveness of DNN watermarking and fingerprinting.

Moreover, the theoretical analysis of evaluation metrics in DNN IP protection still lacks. In DNN watermarking, there is no systematic analysis in theory about how much information we can embed into the host model at most, which is the maximal theoretical capacity of a DNN watermarking method [78]. It is also noteworthy that for DNN authentication and inference perturbation, evaluation metrics are more difficult to quantify in theory than watermarking and fingerprinting approaches. One research direction is to develop a complete evaluation framework in theory, which can significantly help the research in this field.

9.2 DNN fingerprinting

Compared to other categories of protection approaches, the deployed model in DNN fingerprinting is the same as the host model, which makes the fidelity of DNN fingerprinting approaches always equal to one. Without the model's fidelity loss, the DNN fingerprint can be an attractive approach to protect the model's ownership passively. Moreover, model similarity comparison as the application of ensemble fingerprinting integrates the similarity

Table 6 Claimed robustness of DNN fingerprinting approaches against different threats

Method	Model Fine-tuning	Model Compression	Model Extraction	Fingerprint Forging	Fingerprint Evasion
IPGuard [62]	✓	✓	✗	✗	✗
Lukas [63]	✓	✓	✓	✓	✓
Peng [64]	✓	✓	✓	✓	✓
ModelDiff [65]	✓	✓	✗	✗	✗
DI [66]	✓	✓	✓	✗	✗
DeepJudge [35]	✓	✓	✓	✓	✓

metrics as a framework, which makes it expandable to both deployment modes. Although some model similarity comparison methods have been proposed, such as [35], it is challenging to develop a robust model similarity comparison framework in the future. It can be one of the paradigms to passively solve the DNN IP protection problems in the future.

9.3 Defense against model extraction

Since most of the state-of-the-art performance is achieved by large-scaled DNN models today, MLaaS has become the trend to deploy them instead of direct distribution. However, model extraction as the biggest threat in MLaaS deployment mode has not been considered so much. Most existing DNN watermarking and fingerprinting approaches are vulnerable to this attack, making it insecure about applying them in real-world applications. Moreover, model extraction as a model-free stealing attack is challenging to defend without affecting the host model's fidelity. As the primary defense against model extraction, inference perturbation depends on assumptions and degrades the model's performance. We believe the defense against model extraction is an important topic in the future research of DNN IP protection.

9.4 Generality

Although most of the existing protection approaches claim to have generality on different DNN architectures, they tend to test their method on CNN models for image classification. It is noteworthy that some protection approaches like watermarking and fingerprinting depend on assumed specific model architectures, meaning their generality still needs to be expanded and testified. Some protection approaches are specifically designed, such as GAN [92], and GNN [93, 94]. However, the research on DNN IP protection about the generative model, like emerging diffusion models, still lacks. We believe it is an open direction for researchers to explore in the future.

10 Conclusion

In this paper, we present a comprehensive survey about the DNN model IP protection. We first summarize the DNN model deployment into model distribution and MLaaS modes. Second, we present the threats conducted by the adversary and propose our DNN IP protection taxonomy. The existing approaches are divided into passive and active protection based on the protection strategy. For passive protection, we introduce the DNN watermarking and fingerprinting approaches, which are classified based on whether additional information is embedded into the host model. As for active protection, we divide it into DNN authentication and inference perturbation based on whether the owner deploys the model in the distribution or MLaaS mode. Third, we compare the existing approaches based on the evaluation metrics and present future research topics in DNN IP protection.

Author contributions Sen Peng proposed the design of the work and wrote the main manuscript. Jie Xu and Zizhuo Chen surveyed parts of the methods and prepared the comparison. Yufei Chen, Cong Wang, and Xiaohua Jia provided critical revisions to the manuscript. All authors provided critical feedback and reviewed the manuscript.

Funding HK RGC RIF (Research Impact Fund) Ref. No.: R1012-21.

Declarations

Conflicts of interest The authors have no relevant financial or non-financial interests to disclose.

References

1. Zellers, R., Holtzman, A., Rashkin, H., Bisk, Y., Farhadi, A., Roesner, F., Choi, Y.: Defending against neural fake news. In: Wallach, H.M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E.B., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, Vancouver, pp. 9051–9062 (2019)
2. Goldstein, B.F., Patil, V.C., da Cruz Ferreira, V., Nery, A.S., França, F.M.G., Kundu, S.: Preventing DNN model IP theft via hardware obfuscation. IEEE J. Emerg. Sel. Topics Circuits Syst. **11**(2), 267–277 (2021)
3. Zhou, L., Wen, H., Teodorescu, R., Du, D.H.C.: Distributing deep neural networks with containerized partitions at the edge. In: Ahmad, I., Sundararaman, S. (eds.) 2nd USENIX Workshop on Hot Topics in Edge Computing, HotEdge 2019. USENIX Association, (2019)
4. Guo, P., Hu, B., Hu, W.: Mistify: Automating DNN model porting for on-device inference at the edge. In: Mickens, J., Teixeira, R. (eds.) 18th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2021, pp. 705–719. USENIX Association, (2021)
5. Reisinger, M., Frangoudis, P.A., Dustdar, S.: System support and mechanisms for adaptive edge-to-cloud DNN model serving. In: IEEE International Conference on Cloud Engineering, IC2E 2021, pp. 278–279. IEEE, San Francisco (2021). https://doi.org/10.1109/IC2E52221.2021.00046
6. Kesarwani, M., Mukhoty, B., Arya, V., Mehta, S.: Model extraction warning in mlaas paradigm. In: Proceedings of the 34th Annual Computer Security Applications Conference, ACSAC 2018, pp. 371–380. ACM, (2018). https://doi.org/10.1145/3274694.3274740
7. Hanzlik, L., Zhang, Y., Grosse, K., Salem, A., Augustin, M., Backes, M., Fritz, M.: Mlcapsule: Guarded offline deployment of machine learning as a service. In: IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2021, pp. 3300–3309. Computer Vision Foundation / IEEE, (2021). https://doi.org/10.1109/CVPRW53098.2021.00368
8. Sun, Q., Bai, C., Chen, T., Geng, H., Zhang, X., Bai, Y., Yu, B.: Fast and efficient DNN deployment via deep gaussian transfer learning. In: 2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, pp. 5360–5370. IEEE, (2021). https://doi.org/10.1109/ICCV48922.2021.00533
9. Hussain, H., Tamizharasan, P.S., Rahul, C.S.: Design possibilities and challenges of DNN models: a review on the perspective of end devices. Artif. Intell. Rev. **55**(7), 5109–5167 (2022)
10. Xia, C., Zhao, J., Cui, H., Feng, X., Xue, J.: Dnntune: Automatic benchmarking DNN models for mobile-cloud computing. ACM Trans. Archit. Code Optim. **16**(4), 49–14926 (2020)
11. Hinton, G.E., Salakhutdinov, R.R.: Reducing the dimensionality of data with neural networks. Science **313**(5786), 504–507 (2006). https://doi.org/10.1126/science.1127647
12. Yosinski, J., Clune, J., Bengio, Y., Lipson, H.: How transferable are features in deep neural networks? In: Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N.D., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, NIPS 2014, Montreal, pp. 3320–3328 (2014)
13. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network (2015). Preprint at arxiv: 1503.02531
14. Fang, G., Song, J., Shen, C., Wang, X., Chen, D., Song, M.: Data-Free adversarial distillation. Preprint at arxiv **1912**, 11006 (2019)
15. Choi, Y., Choi, J.P., El-Khamy, M., Lee, J.: Data-free network quantization with adversarial knowledge distillation. In: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR Workshops 2020, pp. 3047–3057. Computer Vision Foundation / IEEE, (2020). https://doi.org/10.1109/CVPRW50498.2020.00363
16. Tramèr, F., Zhang, F., Juels, A., Reiter, M.K., Ristenpart, T.: Stealing machine learning models via prediction apis. In: Holz, T., Savage, S. (eds.) 25th USENIX Security Symposium, USENIX Security 16, pp. 601–618. USENIX Association, (2016)
17. Orekondy, T., Schiele, B., Fritz, M.: Knockoff nets: Stealing functionality of black-box models. In: IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, pp. 4954–4963. Computer Vision Foundation / IEEE, (2019). https://doi.org/10.1109/CVPR.2019.00509

18. Yu, H., Yang, K., Zhang, T., Tsai, Y., Ho, T., Jin, Y.: Cloudleak: Large-scale deep learning models stealing through adversarial examples. In: 27th Annual Network and Distributed System Security Symposium, NDSS 2020. The Internet Society, (2020)
19. Kariyappa, S., Prakash, A., Qureshi, M.K.: MAZE: data-free model stealing attack using zeroth-order gradient estimation. In: IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, pp. 13814–13823. Computer Vision Foundation / IEEE, (2021). https://doi.org/10.1109/CVPR46437.2021.01360
20. Sanyal, S., Addepalli, S., Babu, R.V.: Towards data-free model stealing in a hard label setting. In: 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, pp. 13430–13439. Computer Vision Foundation / IEEE, (2022)
21. Rakin, A.S., Chowdhuryy, M.H.I., Yao, F., Fan, D.: Deepsteal: Advanced model extractions leveraging efficient weight stealing in memories. In: 43rd IEEE Symposium on Security and Privacy, SP 2022, pp. 1157–1174. IEEE, (2022). https://doi.org/10.1109/SP46214.2022.9833743
22. Milli, S., Schmidt, L., Dragan, A.D., Hardt, M.: Model reconstruction from model explanations. In: danah boyd, Morgenstern, J.H. (eds.) Proceedings of the Conference on Fairness, Accountability, and Transparency, FAT 2019, pp. 1–9. ACM, (2019). https://doi.org/10.1145/3287560.3287562
23. Batina, L., Bhasin, S., Jap, D., Picek, S.: CSI NN: reverse engineering of neural network architectures through electromagnetic side channel. In: Heninger, N., Traynor, P. (eds.) 28th USENIX Security Symposium, USENIX Security 2019, pp. 515–532. USENIX Association, (2019)
24. Merrer, E.L., Pérez, P., Trédan, G.: Adversarial frontier stitching for remote neural network watermarking. Neural Comput. Appl. **32**(13), 9233–9244 (2020)
25. Yang, P., Lao, Y., Li, P.: Robust watermarking for deep neural networks via bi-level optimization. In: 2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, pp. 14821–14830. IEEE, (2021). https://doi.org/10.1109/ICCV48922.2021.01457
26. Adi, Y., Baum, C., Cissé, M., Pinkas, B., Keshet, J.: Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In: Enck, W., Felt, A.P. (eds.) 27th USENIX Security Symposium, USENIX Security 2018, pp. 1615–1631. USENIX Association, (2018)
27. Zhang, J., Gu, Z., Jang, J., Wu, H., Stoecklin, M.P., Huang, H., Molloy, I.M.: Protecting intellectual property of deep neural networks with watermarking. In: Kim, J., Ahn, G., Kim, S., Kim, Y., López, J., Kim, T. (eds.) Proceedings of the 2018 on Asia Conference on Computer and Communications Security, AsiaCCS 2018, pp. 159–172. ACM, (2018). https://doi.org/10.1145/3196494.3196550
28. Guo, J., Potkonjak, M.: Watermarking deep neural networks for embedded systems. In: Bahar, I. (ed.) Proceedings of the International Conference on Computer-Aided Design, ICCAD 2018, p. 133. ACM, (2018). https://doi.org/10.1145/3240765.3240862
29. Li, Z., Hu, C., Zhang, Y., Guo, S.: How to prove your model belongs to you: a blind-watermark based framework to protect intellectual property of DNN. In: Balenson, D. (ed.) Proceedings of the 35th Annual Computer Security Applications Conference, ACSAC 2019, pp. 126–137. ACM, (2019). https://doi.org/10.1145/3359789.3359801
30. Lukas, N., Jiang, E., Li, X., Kerschbaum, F.: Sok: How robust is image classification deep neural network watermarking? In: 43rd IEEE Symposium on Security and Privacy, SP 2022, pp. 787–804. IEEE, (2022). https://doi.org/10.1109/SP46214.2022.9833693
31. Shafieinejad, M., Lukas, N., Wang, J., Li, X., Kerschbaum, F.: On the robustness of backdoor-based watermarking in deep neural networks. In: Borghys, D., Bas, P., Verdoliva, L., Pevný, T., Li, B., Newman, J. (eds.) Proceedings of the 2021 ACM Workshop on Information Hiding and Multimedia Security, pp. 177–188. ACM, (2021). https://doi.org/10.1145/3437880.3460401
32. Guo, S., Zhang, T., Qiu, H., Zeng, Y., Xiang, T., Liu, Y.: Fine-tuning is not enough: A simple yet effective watermark removal attack for DNN models. In: Zhou, Z. (ed.) Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, pp. 3635–3641. ijcai.org, (2021). https://doi.org/10.24963/ijcai.2021/500
33. Hitaj, D., Mancini, L.V.: Have You Stolen My Model? Evasion attacks against deep neural network watermarking techniques. Preprint at arxiv: 1809.00615 (2018)
34. Wang, T., Kerschbaum, F.: Attacks on digital watermarks for deep neural networks. In: IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2019, pp. 2622–2626. IEEE, (2019). https://doi.org/10.1109/ICASSP.2019.8682202
35. Chen, J., Wang, J., Peng, T., Sun, Y., Cheng, P., Ji, S., Ma, X., Li, B., Song, D.: Copy, right? a testing framework for copyright protection of deep learning models. In: 43rd IEEE Symposium on Security and Privacy, SP 2022, pp. 824–841. IEEE, (2022). https://doi.org/10.1109/SP46214.2022.9833747
36. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, (2016)
37. Devlin, J., Chang, M., Lee, K., Toutanova, K.: BERT: pre-training of deep bidirectional transformers for language understanding. In: Burstein, J., Doran, C., Solorio, T. (eds.) Proceedings of the 2019

Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, pp. 4171–4186. Association for Computational Linguistics, (2019). https://doi.org/10.18653/v1/n19-1423
38. Cheng, Y., Wang, D., Zhou, P., Zhang, T.: A survey of model compression and acceleration for deep neural networks. Preprint at arxiv: 1710.09282 (2017)
39. Choudhary, T., Mishra, V.K., Goswami, A., Sarangapani, J.: A comprehensive survey on model compression and acceleration. Artif. Intell. Rev. **53**(7), 5113–5155 (2020)
40. LeCun, Y., Denker, J.S., Solla, S.A.: Optimal brain damage. In: Touretzky, D.S. (ed.) Advances in Neural Information Processing Systems 2, NIPS 1989, pp. 598–605. Morgan Kaufmann, (1989)
41. Han, S., Pool, J., Tran, J., Dally, W.J.: Learning both weights and connections for efficient neural network. In: Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R. (eds.) Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, NIPS 2015, Montreal, pp. 1135–1143 (2015)
42. Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P.: Pruning filters for efficient convnets. In: 5th International Conference on Learning Representations, ICLR 2017. OpenReview.net, (2017)
43. Polino, A., Pascanu, R., Alistarh, D.: Model compression via distillation and quantization. In: 6th International Conference on Learning Representations, ICLR 2018. OpenReview.net, (2018)
44. Rigamonti, R., Sironi, A., Lepetit, V., Fua, P.: Learning separable filters. In: 2013 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2013, pp. 2754–2761. IEEE Computer Society, (2013). https://doi.org/10.1109/CVPR.2013.355
45. Jagielski, M., Carlini, N., Berthelot, D., Kurakin, A., Papernot, N.: High accuracy and high fidelity extraction of neural networks. In: Capkun, S., Roesner, F. (eds.) 29th USENIX Security Symposium, USENIX Security 2020, pp. 1345–1362. USENIX Association, (2020)
46. Gong, X., Wang, Q., Chen, Y., Yang, W., Jiang, X.: Model extraction attacks and defenses on cloud-based machine learning models. IEEE Commun. Mag. **58**(12), 83–89 (2020)
47. Li, Y., Wang, H., Barni, M.: A survey of deep neural network watermarking techniques. Neurocomputing **461**, 171–193 (2021)
48. Uchida, Y., Nagai, Y., Sakazawa, S., Satoh, S.: Embedding watermarks into deep neural networks. In: Ionescu, B., Sebe, N., Feng, J., Larson, M.A., Lienhart, R., Snoek, C. (eds.) Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval, ICMR 2017, pp. 269–277. ACM, (2017). https://doi.org/10.1145/3078971.3078974
49. Chen, H., Rouhani, B.D., Fu, C., Zhao, J., Koushanfar, F.: Deepmarks: A secure fingerprinting framework for digital rights management of deep learning models. In: El-Saddik, A., Bimbo, A.D., Zhang, Z., Hauptmann, A.G., Candan, K.S., Bertini, M., Xie, L., Wei, X. (eds.) Proceedings of the 2019 on International Conference on Multimedia Retrieval, ICMR 2019, pp. 105–113. ACM, (2019). https://doi.org/10.1145/3323873.3325042
50. Wang, T., Kerschbaum, F.: RIGA: covert and robust white-box watermarking of deep neural networks. In: Leskovec, J., Grobelnik, M., Najork, M., Tang, J., Zia, L. (eds.) Proceedings of the Web Conference 2021, WWW 2021, pp. 993–1004. ACM / IW3C2, (2021). https://doi.org/10.1145/3442381.3450000
51. Liu, H., Weng, Z., Zhu, Y.: Watermarking deep neural networks with greedy residuals. In: Meila, M., Zhang, T. (eds.) Proceedings of the 38th International Conference on Machine Learning, ICML 2021. Proceedings of Machine Learning Research, vol. 139, pp. 6978–6988. PMLR, (2021)
52. Rouhani, B.D., Chen, H., Koushanfar, F.: Deepsigns: An end-to-end watermarking framework for ownership protection of deep neural networks. In: Bahar, I., Herlihy, M., Witchel, E., Lebeck, A.R. (eds.) Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2019, pp. 485–497. ACM, (2019). https://doi.org/10.1145/3297858.3304051
53. Namba, R., Sakuma, J.: Robust watermarking of neural network with exponential weighting. In: Galbraith, S.D., Russello, G., Susilo, W., Gollmann, D., Kirda, E., Liang, Z. (eds.) Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security, AsiaCCS 2019, pp. 228–240. ACM, (2019). https://doi.org/10.1145/3321705.3329808
54. Jia, H., Choquette-Choo, C.A., Chandrasekaran, V., Papernot, N.: Entangled watermarks as a defense against model extraction. In: Bailey, M., Greenstadt, R. (eds.) 30th USENIX Security Symposium, USENIX Security 2021, pp. 1937–1954. USENIX Association, (2021)
55. Szyller, S., Atli, B.G., Marchal, S., Asokan, N.: DAWN: dynamic adversarial watermarking of neural networks. In: Shen, H.T., Zhuang, Y., Smith, J.R., Yang, Y., Cesar, P., Metze, F., Prabhakaran, B. (eds.) Proceedings of the 29th ACM International Conference on Multimedia, MM 2021, pp. 4417–4425. ACM, (2021). https://doi.org/10.1145/3474085.3475591

56. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego (2015)
57. Kurakin, A., Goodfellow, I.J., Bengio, S.: Adversarial examples in the physical world. In: 5th International Conference on Learning Representations, ICLR 2017. OpenReview.net, (2017)
58. Ilyas, A., Santurkar, S., Tsipras, D., Engstrom, L., Tran, B., Madry, A.: Adversarial examples are not bugs, they are features. In: Wallach, H.M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E.B., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, Vancouver, pp. 125–136 (2019)
59. Gu, T., Dolan-Gavitt, B., Garg, S.: BadNets: Identifying vulnerabilities in the machine learning model supply chain. Preprint at https://arxiv.org/abs/1708.06733 (2017)
60. Liu, Y., Ma, S., Aafer, Y., Lee, W., Zhai, J., Wang, W., Zhang, X.: Trojaning attack on neural networks. In: 25th Annual Network and Distributed System Security Symposium, NDSS 2018. The Internet Society, (2018)
61. Jia, H., Yaghini, M., Choquette-Choo, C.A., Dullerud, N., Thudi, A., Chandrasekaran, V., Papernot, N.: Proof-of-learning: Definitions and practice. In: 42nd IEEE Symposium on Security and Privacy, SP 2021, pp. 1039–1056. IEEE, (2021). https://doi.org/10.1109/SP40001.2021.00106
62. Cao, X., Jia, J., Gong, N.Z.: Ipguard: Protecting intellectual property of deep neural networks via fingerprinting the classification boundary. In: Cao, J., Au, M.H., Lin, Z., Yung, M. (eds.) Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security, AsiaCCS 2021, pp. 14–25. ACM, (2021). https://doi.org/10.1145/3433210.3437526
63. Lukas, N., Zhang, Y., Kerschbaum, F.: Deep neural network fingerprinting by conferrable adversarial examples. In: 9th International Conference on Learning Representations, ICLR 2021. OpenReview.net, (2021)
64. Peng, Z., Li, S., Chen, G., Zhang, C., Zhu, H., Xue, M.: Fingerprinting deep neural networks globally via universal adversarial perturbations. In: 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, pp. 13430–13439. Computer Vision Foundation / IEEE, (2022)
65. Li, Y., Zhang, Z., Liu, B., Yang, Z., Liu, Y.: Modeldiff: Testing-based DNN similarity comparison for model reuse detection. In: Cadar, C., Zhang, X. (eds.) Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2021, pp. 139–151. ACM, (2021). https://doi.org/10.1145/3460319.3464816
66. Maini, P., Yaghini, M., Papernot, N.: Dataset inference: Ownership resolution in machine learning. In: 9th International Conference on Learning Representations, ICLR 2021. OpenReview.net, (2021)
67. Chen, H., Fu, C., Rouhani, B.D., Zhao, J., Koushanfar, F.: Deepattest: An end-to-end attestation framework for deep neural networks. In: Manne, S.B., Hunter, H.C., Altman, E.R. (eds.) Proceedings of the 46th International Symposium on Computer Architecture, ISCA 2019, pp. 487–498. ACM, (2019). https://doi.org/10.1145/3307650.3322251
68. Chakraborty, A., Mondal, A., Srivastava, A.: Hardware-assisted intellectual property protection of deep learning models. In: 57th ACM/IEEE Design Automation Conference, DAC 2020, pp. 1–6. IEEE, (2020). https://doi.org/10.1109/DAC18072.2020.9218651
69. Fan, L., Ng, K.W., Chan, C.S.: Rethinking deep neural network ownership verification: embedding passports to defeat ambiguity attacks. In: Wallach, H.M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E.B., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, Vancouver, pp. 4716–4725 (2019)
70. Zhang, J., Chen, D., Liao, J., Zhang, W., Hua, G., Yu, N.: Passport-aware normalization for deep model protection. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H. (eds.) Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020 (2020)
71. Lin, N., Chen, X., Lu, H., Li, X.: Chaotic weights: A novel approach to protect intellectual property of deep neural networks. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **40**(7), 1327–1339 (2021)
72. Xue, M., Sun, S., He, C., Zhang, Y., Wang, J., Liu, W.: ActiveGuard: An active DNN IP protection technique via adversarial examples. Preprint at arxiv: 2103.01527 (2021)
73. Lee, T., Edwards, B., Molloy, I.M., Su, D.: Defending against neural network model stealing attacks using deceptive perturbations. In: 2019 IEEE Security and Privacy Workshops, SP Workshops 2019, pp. 43–49. IEEE, (2019). https://doi.org/10.1109/SPW.2019.00020
74. Orekondy, T., Schiele, B., Fritz, M.: Prediction poisoning: Towards defenses against DNN model stealing attacks. In: 8th International Conference on Learning Representations, ICLR 2020. OpenReview.net, (2020)

75. Juuti, M., Szyller, S., Marchal, S., Asokan, N.: PRADA: protecting against DNN model stealing attacks. In: IEEE European Symposium on Security and Privacy, EuroS &P 2019, pp. 512–527. IEEE, (2019). https://doi.org/10.1109/EuroSP.2019.00044
76. Kariyappa, S., Qureshi, M.K.: Defending against model stealing attacks with adaptive misinformation. In: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, pp. 767–775. Computer Vision Foundation / IEEE, (2020). https://doi.org/10.1109/CVPR42600.2020.00085
77. Regazzoni, F., Palmieri, P., Smailbegovic, F., Cammarota, R., Polian, I.: Protecting artificial intelligence ips: a survey of watermarking and fingerprinting for machine learning. CAAI Transactions on Intelligence Technology **6**(2), 180–191 (2021)
78. Barni, M., Pérez-González, F., Tondi, B.: DNN watermarking: Four challenges and a funeral. In: Borghys, D., Bas, P., Verdoliva, L., Pevný, T., Li, B., Newman, J. (eds.) Proceedings of the 2021 ACM Workshop on Information Hiding and Multimedia Security, pp. 189–196. ACM, (2021). https://doi.org/10.1145/3437880.3460399
79. Xue, M., Wang, J., Liu, W.: Dnn intellectual property protection: Taxonomy, attacks and evaluations. In: Chen, Y., Zhirnov, V.V., Sasan, A., Savidis, I. (eds.) Proceedings of the 2021 on Great Lakes Symposium on VLSI, GLSVLSI 2021, pp. 455–460. ACM, (2021). https://doi.org/10.1145/3453688.3461752
80. Nagai, Y., Uchida, Y., Sakazawa, S., Satoh, S.: Digital watermarking for deep neural networks. Int. J. Multim. Inf. Retr. **7**(1), 3–16 (2018)
81. Chen, X., Wang, W., Bender, C., Ding, Y., Jia, R., Li, B., Song, D.: REFIT: A unified watermark removal framework for deep learning systems with limited data. In: Cao, J., Au, M.H., Lin, Z., Yung, M. (eds.) Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security, ASIA CCS 2021, pp. 321–335. ACM, (2021). https://doi.org/10.1145/3433210.3453079
82. Kornblith, S., Norouzi, M., Lee, H., Hinton, G.E.: Similarity of neural network representations revisited. In: Chaudhuri, K., Salakhutdinov, R. (eds.) Proceedings of the 36th International Conference on Machine Learning, ICML 2019. Proceedings of Machine Learning Research, vol. 97, pp. 3519–3529. PMLR, (2019)
83. Salakhutdinov, R., Hinton, G.E.: Learning a nonlinear embedding by preserving class neighbourhood structure. In: Meila, M., Shen, X. (eds.) Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics, AISTATS 2007. JMLR Proceedings, vol. 2, pp. 412–419. JMLR.org, (2007)
84. Breier, J., Hou, X., Jap, D., Ma, L., Bhasin, S., Liu, Y.: Practical fault attack on deep neural networks. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, pp. 2204–2206. ACM, (2018). https://doi.org/10.1145/3243734.3278519
85. Hong, S., Frigo, P., Kaya, Y., Giuffrida, C., Dumitras, T.: Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks. In: Heninger, N., Traynor, P. (eds.) 28th USENIX Security Symposium, USENIX Security 2019, pp. 497–514. USENIX Association, (2019)
86. Zhang, R., Liu, J., Ding, Y., Wang, Z., Wu, Q., Ren, K.: Adversarial examples for proof-of-learning. In: 43rd IEEE Symposium on Security and Privacy, SP 2022, pp. 1408–1422. IEEE, (2022). https://doi.org/10.1109/SP46214.2022.9833596
87. Moosavi-Dezfooli, S., Fawzi, A., Fawzi, O., Frossard, P.: Universal adversarial perturbations. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, pp. 86–94. IEEE Computer Society, (2017). https://doi.org/10.1109/CVPR.2017.17
88. Cai, Y., Chen, X., Tian, L., Wang, Y., Yang, H.: Enabling secure in-memory neural network computing by sparse fast gradient encryption. In: Pan, D.Z. (ed.) Proceedings of the International Conference on Computer-Aided Design, ICCAD 2019, pp. 1–8. ACM, (2019). https://doi.org/10.1109/ICCAD45719.2019.8942041
89. Peterson, G.: Arnold's cat map. Math linear algebra **45**, 1–7 (1997)
90. Papernot, N., McDaniel, P.D., Goodfellow, I.J., Jha, S., Celik, Z.B., Swami, A.: Practical black-box attacks against machine learning. In: Karri, R., Sinanoglu, O., Sadeghi, A., Yi, X. (eds.) Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, AsiaCCS 2017, pp. 506–519. ACM, (2017). https://doi.org/10.1145/3052973.3053009
91. Quan, Y., Teng, H., Chen, Y., Ji, H.: Watermarking deep neural networks in image processing. IEEE Trans. Neural Networks Learn. Syst. **32**(5), 1852–1865 (2021)
92. Ong, D.S., Chan, C.S., Ng, K.W., Fan, L., Yang, Q.: Protecting intellectual property of generative adversarial networks from ambiguity attacks. In: IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, pp. 3630–3639. Computer Vision Foundation / IEEE, (2021). https://doi.org/10.1109/CVPR46437.2021.00363

93. Zhao, X., Wu, H., Zhang, X.: Watermarking graph neural networks by random graphs. In: Varol, A., Karabatak, M., Varol, I. (eds.) 9th International Symposium on Digital Forensics and Security, ISDFS 2021, pp. 1–6. IEEE, (2021). https://doi.org/10.1109/ISDFS52919.2021.9486352
94. Xu, J., Picek, S.: Watermarking Graph Neural Networks based on Backdoor Attacks. Preprint at arxiv: 2110.11024 (2021)