Provenance of Training without Training Data: Towards Privacy-Preserving DNN Model Ownership Verification

Yunpeng Liu
liuyp20@mails.tsinghua.edu.cn
Tsinghua University

Kexin Li
cassiekx.li@mail.utoronto.ca
University of Toronto

Zhuotao Liu
zhuotaoliu@tsinghua.edu.cn
Tsinghua University
Zhongguancun Laboratory

Bihan Wen
Bihan.wen@ntu.edu.sg
Nanyang Technological University

Ke Xu
xuke@tsinghua.edu.cn
Tsinghua University
Zhongguancun Laboratory

Weiqiang Wang
weiqiang.wwq@antgroup.com
Ant Group

Wenbiao Zhao
wenbiao.zwb@antgroup.com
Ant Group

Qi Li*
qli01@tsinghua.edu.cn
Tsinghua University
Zhongguancun Laboratory

Abstract

In the era of deep learning, it is critical to protect the intellectual property of high-performance deep neural network (DNN) models. Existing proposals, however, are subject to adversarial ownership forgery (*e.g.*, methods based on watermarks or fingerprints) or require full access to the original training dataset for ownership verification (*e.g.*, methods requiring the replay of the learning process). In this paper, we propose a novel Provenance of Training (PoT) scheme, the first empirical study towards verifying DNN model ownership without accessing any original dataset while being robust against existing attacks. At its core, PoT relies on a coherent model chain built from the intermediate checkpoints saved during model training to serve as the ownership certificate. Through an in-depth analysis of model training, we propose six key properties that a legitimate model chain shall naturally hold. In contrast, it is difficult for the adversary to forge a model chain that satisfies these properties simultaneously without performing actual training. We systematically analyze PoT's robustness against various possible attacks, including the adaptive attacks that are designed given the full knowledge of PoT's design, and further perform extensive empirical experiments to demonstrate our security analysis.

CCS Concepts

• **Computer systems organization** → **Neural networks**; • **Security and privacy** → **Digital rights management**.

Keywords

Deep neural network, ownership verification, IP protection

*Qi Li is the corresponding author.

This work is supported in part by the National Key Research and Development Project of China under Grant 2021ZD0110502, NSFC under Grant 62132011 and 61825204, Beijing Outstanding Young Scientist Program under Grant BJJWZYJH01201910003011, and Ant Group.

WWW '23, April 30–May 04, 2023, Austin, TX, USA

ACM ISBN 978-1-4503-9416-1/23/04.
https://doi.org/10.1145/3543507.3583198

ACM Reference Format:
Yunpeng Liu, Kexin Li, Zhuotao Liu, Bihan Wen, Ke Xu, Weiqiang Wang, Wenbiao Zhao, and Qi Li. 2023. Provenance of Training without Training Data: Towards Privacy-Preserving DNN Model Ownership Verification. In *Proceedings of the ACM Web Conference 2023 (WWW '23), April 30–May 04, 2023, Austin, TX, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3543507.3583198

1 Introduction

Recent advances in deep learning (DL) enable revolutionary breakthroughs for a wide range of Web applications, *e.g.*, recommendation systems, fraud detection and auto speech recognition [6, 26, 41]. Many Web service providers allow users to deploy their DNN models on their platforms so that the models can be accessed by third parties. Also, users can use DNN models sold by these providers without any training costs. However, it is challenging for model owners to ensure that the intellectual property (IP) of the models will not be violated, *e.g.*, by redistributing the models or constructing model stealing attacks [8, 19, 21, 23, 28, 34].

Existing studies [28, 35] protect the IP security of DNN models via verifying the model ownership. DNN ownership verification schemes should achieve *robustness* and *exclusiveness* so that the adversary can neither invalidate the ownership certificate of the model trainer nor forge a fraudulent ownership certificate. Existing methods can be classified into the following three categories. Watermarking schemes [2, 35] require model owners to embed digital signatures into DNN models. For example, feature-based watermarking schemes implant a piece of code into the internal of models [7, 31, 35], and trigger-set-based watermarking schemes embed backdoor functions into the protected models [2, 11, 16, 18, 20, 22, 28, 38, 39]. These designs not only sacrifice model performance but are not robust nor exclusive, since adversaries can remove the existing watermarks or embed their own watermarks into the illegally acquired models [3, 8, 11, 19, 23, 28, 37]. Meanwhile, fingerprint-based methods extract fingerprints from trained models as proofs of ownership while ensuring the model utility [5, 25, 36, 42]. However, adversaries can invalidate the original fingerprints via model modification and construct their own fingerprints to claim ownership [11, 19]. The computation-based method verifies ownership by replaying the training history to validate the training effort of model owners. One critical drawback of the existing computation-based method, Proof of Learning

(PoL) [17], is that *it inevitably requires model owners to expose the full training data to the verifier*. Given that training data is often proprietary and contains sensitive information, the practical application of such kind of method is significantly limited.

To address limitations of existing approaches, we propose PoT, a model ownership verification mechanism that simultaneously achieves robustness, exclusiveness, and training data privacy protection. Unlike watermark-based or fingerprint-based methods that only verify the ownership certificate in the protected model, PoT leverages entire *provenance of the model training process*, which is naturally bound to the specific model. The core is building a *coherent model chain* from the intermediate model checkpoints saved during model training from scratch, such that the model owner can easily verify the model chain coherence without exposing any training data, while it is difficult for adversaries to forge at a reasonable cost.

We systematically study the characteristics of coherent model chains with measurement experiments and then provide the detailed design of PoT based on our observations. We also analyze the security of PoT against both existing attacks and adaptive attacks, and conduct empirical evaluations to validate our analysis. The results show that PoT can defend against all possible model ownership forgery attacks in our consideration. Overall, we make the following contributions:

- **A novel DNN model ownership verification scheme.** We present PoT, a novel model ownership verification scheme that verifies the training provenance *without accessing original training data*. PoT relies only on a chain of intermediate checkpoints saved during training, introducing trivial adoption overhead.
- **Defining robust properties for verification.** Based on a thorough analysis on model training, we discover and formalize six properties that will hold in a coherent model chain naturally, while being difficult for the adversary to forge.
- **Thorough security analysis and experimental evaluations against possible attacks.** We systematically investigate the possible attacks against PoT analytically and experimentally to demonstrate its robustness. We consider adaptive attacks with complete knowledge of PoT along with the existing attacks.

2 Background and Problem Statement

2.1 DNN Model IP Protection

Training a high-performance DNN model requires a substantial amount of computational and data resources. For example, according to Strubell et al. [33], to tune a model performing four NLP tasks in a full research and deployment cycle, 239,942 hours and about $103k to $350k are required. Due to the excessive efforts of training, it is appealing for the adversary to steal the model ownership using various types of attacks, *e.g.*, model extraction attack [21, 34, 43] and model modification attack based on fine-tuning or parameter pruning [24]. Therefore, it is important to protect the intellectual property (IP) of high-performance models.

The common practice for IP protection is to generate an unforgeable *certificate* during model training to verify the training efforts committed by real model owner in case of a model ownership dispute. A model ownership dispute typically involves three types of entities: the original model owner who performed the actual training, the adversary aiming to illegally claim ownership after model stealing, and the verifier responsible for regulating the ownership dispute by checking ownership proofs.

2.2 Threat Model

Adversaries could employ several possible techniques to fraudulently claim ownership. For example, adversaries can conduct model piracy attack by leveraging existing model ownership verification schemes to forge ownership proofs. To this end, we assume that the adversary (i) has white-box access to the illegally acquired model M, and (ii) may even have a labelled auxiliary dataset $\mathcal{D}_{\text{aux}}$ that shares a similar distribution to the original training dataset $\mathcal{D}_{\text{train}}$ of M. *Yet, we assume that the size of $\mathcal{D}_{\text{aux}}$ is much smaller than that of $\mathcal{D}_{\text{train}}$ (*e.g.*, no greater than 10%).* Similarly, we assume that the verifier possesses a validation dataset $\mathcal{D}_{\text{val}}$ with similar distribution to that of the original owner.

We clarify the assumption of limited knowledge on the original training dataset is one key differentiator separating PoT from PoL [17] which requires a *fully trusted* verifier to have *complete access* to $\mathcal{D}_{\text{train}}$. In contrast, PoT enables the verifier to validate ownership with zero-knowledge on $\mathcal{D}_{\text{train}}$, *i.e.*, the verifier learns no additional knowledge about the owner's $\mathcal{D}_{\text{train}}$ (except for the knowledge that the verifier already obtains, such as $\mathcal{D}_{\text{aux}}$).

3 Observations

In this section, we introduce the key observations behind the design of PoT with measurement studies on the basic properties of the model checkpoints generated during training. These properties are based on common assumptions in DNN training and validation:

- The training and validation data are drawn from *similar* underlying data distribution. Therefore, the validation accuracy of the model improves continuously as the training accuracy does.
- The DNN model is usually trained with a sufficiently *small learning rate* (*e.g.*, 0.1 for the SGD optimizer), and the parameters are initialized *randomly* and *independently*.
- The convergence of the model training algorithm is *well-behaved*. Thus, the parameters of the intermediate checkpoints gradually approach those of a nearby local optimum.

Based on these assumptions, we focus on the following quantitative measures of intermediate model checkpoints: *validation accuracy*, *parameter distribution distance*, *independence among initial parameters* and *weight distance*. We empirically study these quantitative measures to show that their unique behaviors can be used to verify whether intermediate checkpoints are generated from training.

3.1 Validation Accuracy

Validation accuracy is an important metric to evaluate the performance of a model. By training the DNN model properly (*e.g.*, without gradient explosion or vanishing), the training loss is likely to decrease monotonically until convergence. Therefore, when the validation data share a distribution similar to the training data without overfitting, the validation accuracy of the model is expected to be monotonically non-decreasing as the training proceeds.

To demonstrate such property in practice, we conduct measurement experiments by training DNN models over popular computer vision datasets. Figure 1 plots each validation accuracy as a function of the epoch number. The result aligns with our analysis. Note that although intermediate checkpoints with non-monotonic accuracy are typical in practice due to the local minima in the loss function and unstable gradients, the overall trend still holds.

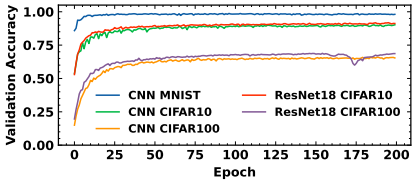


Figure 1: Trend of validation accuracy. The validation accuracy is continuously improving over training epochs with small fluctuations.

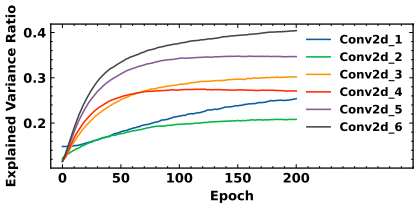


Figure 2: Trend of the explained variance ratio of the first principle component in a CNN model trained on CIFAR100 dataset over training epochs.

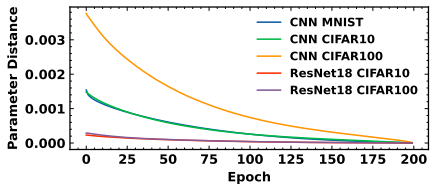
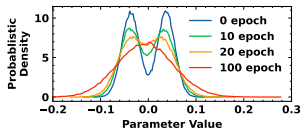
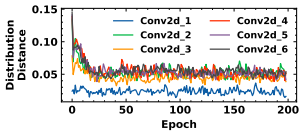


Figure 3: Trend of parameter distance between the checkpoint and the final converged model over training epochs.



(a) The change of the parameter distribution in the fourth convolutional layer.



(b) The parameter distribution distance between consecutive checkpoints.

Figure 4: The trend of parameter distribution of an 8-layer CNN model trained on CIFAR100 dataset.

3.2 Parameter Distribution Distance

To ensure convergence, deep learning models are usually trained with a sufficiently small learning rate (*e.g.*, 0.1 for SGD algorithm). Meanwhile, the magnitude of gradients decreases as the model converges. Hence, the parameters are updated by a small step in each iteration. This leads to the observation that the parameter distribution should *evolve continuously without sharp changes*. To measure the change of parameter distribution, we formally define the *parameter distribution distance* as the Earth Mover's Distance [32] between parameter distributions. In practice, the parameter distribution can be approximated by the histogram.

To demonstrate the above observation, we train an 8-layer CNN model on CIFAR100 dataset and analyze the parameter distributions. We initialize the model parameters with a special Gaussian Mixture Model (GMM) distribution to make the distribution change obviously. Figure 4a shows the distribution for the parameters in the 4th convolutional layer at different training stages. It is clear that the parameter distribution gradually evolves from the original GMM distribution to a Gaussian distribution. Figure 4b shows the trend of parameter distribution distances between *consecutive* checkpoints. Throughout the training process, the parameter distribution distance remains relatively small, demonstrating the coherence of parameter distribution in a normal training process.

3.3 Independence among Initial Parameters

It is common to randomly initialize the parameters of DNN models before training, making parameters *mutually independent*. During model training, the parameters will become correlated with each other to empower the functionality of corresponding neurons.

To demonstrate this, we use the Principal Component Analysis (PCA) algorithm to decompose the weights of the convolutional layers of a trained CNN model. The basic idea here is that the *explained variance ratio* of the *first* principle component (*i.e.*, the component with the largest explained variance) will be *small* if different features of analyzed data samples are independent. By grouping parameters into instances of high-dimension parameter vectors according to their relationship, we can apply PCA algorithm

Table 1: Comparison of model distances. The random reinitialization is repeated 1000 times for each model to report the mean and standard deviation of $d(M_{rand}, M_{final})$.

Model	Dataset	$d(M_{init}, M_{final})$	$d(M_{rand}, M_{final})$	
			mean	std
CNN	MNIST	1.54e-3	5.11e-3	1.12e-5
	CIFAR10	1.48e-3	3.76e-3	7.29e-6
	CIFAR100	3.76e-3	6.17e-3	9.80e-6
ResNet18	CIFAR10	2.32e-4	1.22e-3	8.85e-7
	CIFAR100	2.91e-4	1.30e-3	9.36e-7
TextCNN	DBpedia	5.51e-4	4.36e-3	9.70e-6
	SOGOU	3.84e-3	5.04e-3	1.18e-5
TextRNN	DBpedia	2.85e-3	1.36e-2	1.78e-4

to reveal the intra-group parameter correlation. The explained variance ratio of the first principle component reaches the smallest possible value if parameters are randomly initialized. As long as the parameters become correlated, *e.g.*, the training process introduces latent parameter correlation, the ratio will increase. Figure 2 shows the trend of the explained variance ratio of the first principle component with respect to the number of training epochs. The explained variance ratios in all layers increase as the training proceeds, which validates our analysis.

3.4 Weight Distance

The parameter distance between two models with the same architecture can be measured using the L2 norm. Formally, for an L-layer model M, its parameters are represented as $\mathbf{W} = \{\mathbf{w}_1, \mathbf{w}_2, \cdots, \mathbf{w}_L\}$. The *weight distance* between two models M_i and M_j sharing the same architecture is defined as:

$$d(M_i, M_j) = \frac{1}{N_{\text{param}}} \sqrt{\sum_{l=1}^{L} \left\| \mathbf{w}_l^i - \mathbf{w}_l^j \right\|_F^2}, \tag{1}$$

where N_{param} is the total number of model parameters and $\|\cdot\|_F$ denotes the Frobenius norm.

In model training, the model parameters will eventually converge to a local optimum from the initial position, and the distance between the intermediate model and converged model will gradually change. Given the gradient descent algorithm's nature of convergence, the parameters of the model under training shall *gradually* approach to an optimum that is *close to* the initial model. Thus the weight distance between the intermediate model and converged model is *monotonically non-increasing*. We illustrate this observation in Figure 3 where we plot, for multiple model training processes, the parameter distances between the intermediate checkpoints and the final converged models over the epochs. The

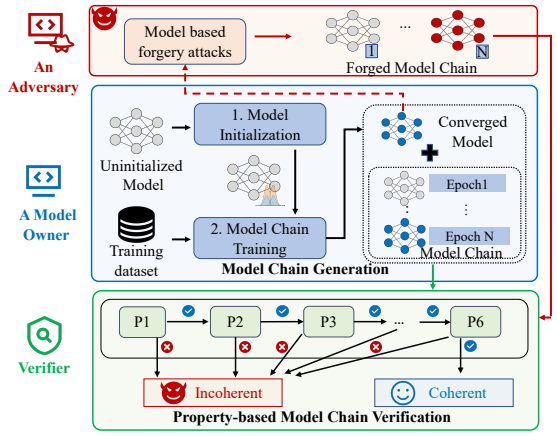


Figure 5: The overview of PoT.

distances are continuously decreasing, which demonstrates our observation.

Besides, existing studies show that the loss surface of a sufficiently complex DNN has a large number of equivalent local minima that have similar performances [9, 29]. At the beginning of training, the direction of the gradient will point to a region of interest that is *close* to the starting point. Therefore, the model will converge to an optimum that is *close* to the initial model compared with other local minima. Moreover, $d(M_{rand}, M_{final})$ is the sum of the distances between all corresponding parameters in the two models, and all parameters in the same layer are equivalent and thus follow the same distribution. Thus, according to the central limit theorem, $d(M_{rand}, M_{final})$ will follow a normal distribution whose standard deviation is inversely proportional to the square root of the number of parameters. In DNNs with a large number of neurons, the standard deviation of $d(M_{rand}, M_{final})$ will be small. Combining the analysis, the probability is low for any random model M_{rand} to have $d(M_{rand}, M_{final})$ close to $d(M_{init}, M_{final})$, and it is extremely hard to find one randomly initialized model sufficiently close to M_{final} by repetitive re-initialization. We conduct an empirical study to demonstrate this observation. The experiment results in Table 1 clearly show that $d(M_{init}, M_{final})$ is much smaller than $d(M_{rand}, M_{final})$ for all models, and the standard deviation of $d(M_{rand}, M_{final})$ is small.

4 PoT Design

In this section, we elaborate on the design of PoT. Based on the observations in §3, we provide property definitions for model chain evaluation and the model ownership verification procedure.

The key design goal of PoT is to build the provenance of model training by carefully analyzing the end-to-end training process of the DNN model, which is tightly linked to the training effort of the model owner. Towards this end, we propose to use the *chain of model checkpoints* generated during the training process to serve as the ownership certificate of a DNN model. In DNN training, it is a common practice to save intermediate model weights as checkpoints after each epoch to deal with overfitting. It is also widely supported by deep learning frameworks including PyTorch [30] and TensorFlow [1]. We call the checkpoint sequence generated during model training a *model chain*. In the following sections, we use M to denote the model protected by PoT, C_i to denote the checkpoint, and $C = \{C_0, C_1, \cdots, C_{N-1}, C_N\}$ to denote the model chain. Note that the last checkpoint C_N is exactly the protected model M.

Generally, our framework works as shown in Figure 5. The requirement for model owners to use PoT is light: they only need to initialize the model according to a specific Gaussian Mixture Model (GMM) distribution and conduct the normal training procedure with L2 regularization (weight decay) (see details in § 4.2). In a model ownership dispute, the verifier regulates the process by favoring the party providing a coherent model chain that satisfies key properties simultaneously. These properties are *naturally* held by real model chains, but are not in the forged ones. We present the property definitions used by PoT in §4.1, and describe the details of training and verification with PoT in §4.2 and 4.3.

4.1 Model Chain Properties

Our observations and measurements in §3 lead to six key properties that a coherent model chain should hold.

(1) Validation Accuracy. In §3.1, we analyze the property of checkpoints' validation accuracies, which is formally defined as follows:

Property 1 (Monotonicity of Validation Accuracy). Given a validation dataset $\mathcal{D}_{\text{val}}$ with a data distribution similar to $\mathcal{D}_{\text{train}}$, the validation accuracies of the checkpoints $\{C_0, C_1, \cdots, C_N\}$ over $\mathcal{D}_{\text{val}}$ are likely to be monotonically non-decreasing. Formally, for most of the timestamp i,

$$\text{acc}(C_i, \mathcal{D}_{\text{val}}) \geq \text{acc}(C_j, \mathcal{D}_{\text{val}}) \quad \forall j \leq i.$$

We define a metric to quantitatively measure this property by calculating the Spearman's rank correlation coefficient [10] between the accuracy and the number of training epochs:

$$\rho_{acc} \triangleq 1 - \frac{6 \times \sum_i^N \left(\text{rank}(\text{acc}(C_i, \mathcal{D}_{\text{val}})) - i\right)^2}{N(N^2-1)}, \tag{2}$$

where $\text{rank}(\text{acc}(C_i, \mathcal{D}_{\text{val}}))$ denotes the ranking index of the accuracy of C_i in the model chain. A model chain with higher ρ_{acc} indicates a higher confidence in its authenticity. Note that there are corner cases where the validation accuracy is not sufficiently monotonic, *e.g.*, the model overfits to the dataset or the task is overly hard. We use early stop during verification to alleviate the overfitting issue and constrain the metric to deal with the non-monotonicity.

(2) Parameter Distribution Distance. In §3.2, we empirically show that model parameter distribution keeps evolving without sharp changes during training due to incremental parameter updates. Hereafter, We denote the weight distribution of $\mathbf{w}$ as $P(\mathbf{w})$, which can be approximated with the histogram in practice. We use $\text{EMD}(\cdot, \cdot)$ to denote the Earth Mover's Distance between two distributions. Formally, we can obtain the following property:

Property 2 (Parameter Distribution Continuity). Given that the model is trained with a sufficiently small learning rate, for any two consecutive checkpoints C_i and C_{i+1} and a small threshold δ, the weights in all the L layers should satisfy:

$$\forall l \in [1, L], \quad \text{EMD}\left(P(\mathbf{w}_l^i), P(\mathbf{w}_l^{i+1})\right) < \delta.$$

We quantify the property as follows:

$$c_{weight} \triangleq \max_{i \in [1, N-1]} \left\{ \max_{l \in [1, L]} \left\{ \text{EMD}\left(P(\mathbf{w}_l^i), P(\mathbf{w}_l^{i+1})\right) \right\} \right\}, \tag{3}$$

c_{weight} calculates the maximum of layer-wise parameter distribution distances between consecutive checkpoints. This property also prevents the adversary from directly concatenating model chains.

(3) Initial Model Parameters. In §3.3, we empirically show that the mutual independence among the initial model parameters can reveal whether the model is randomly initialized. In PoT, the parameters of a protected DNN model are randomly initialized with a specific GMM distribution, where the required GMM distribution for a weight $\mathbf{w}$ is represented as $P_{\text{GMM}}(\mathbf{w})$. This constraint is important because an adversary may forge a model chain by directly manipulating the stolen model to generate a fake "initial model" that is not randomly initialized. Therefore, in a model ownership dispute, the parameters of the initial model should be checked to verify whether the parameters are initialized according to the requirement. Based on the analysis above, we can obtain the following two properties of the initial model parameters.

Property 3 (Distribution of the Initial Parameters). The parameters of the initial model should conform to the required *GMM distribution*. Given an initial model C_0, its weights in all layers should satisfy:

$$\forall l \in [1, L], \quad \text{EMD}\left(P(\mathbf{w}_l^0), P_{\text{GMM}}(\mathbf{w}_l^0)\right) \approx 0.$$

Property 3 can be quantified by calculating the distance between the distribution of initial parameters and the required GMM distribution. The property is quantified as:

$$d_{init} \triangleq \max_{l \in [1,L]} \left\{\text{EMD}\left(P\left(\mathbf{w}_l^0\right), P_{\text{GMM}}\left(\mathbf{w}_l^0\right)\right)\right\}. \tag{4}$$

In a nutshell, Eq. (4) calculates the *maximum* distance between the required distribution and the actual distribution across all layers.

Property 4 (Independence among Initialized Parameters). The parameters of the initial model should be *independent*. Given the weight $\mathbf{w}$ in an initialized layer, any two different parameters $\mathbf{w}[i]$ and $\mathbf{w}[j]$ in $\mathbf{w}$ should satisfy the following equation if we view them as random variables:

$$\text{cov}(\mathbf{w}[i], \mathbf{w}[j]) = 0,$$

where $\text{cov}(\cdot, \cdot)$ is the covariance of two random variables.

To verify Property 4, we need to evaluate whether the parameters of the initial model are independently and randomly initialized. In §3.3, we show that the verification of parameter independence can be achieved with the PCA algorithm. Here we further elaborate the detailed procedures. First, group the parameters in the same layer according to their relationship (*e.g.*, the parameters belonging to the same neuron in a linear layer or belonging to the same channel in a kernel filter of a convolutional layer are viewed as a group). Second, run the PCA algorithm on the grouped parameters, and get the explained variance ratio of the *first* principle components:

$$p_{pca} \triangleq \max_{l \in [1,L]} \left\{MaxExplainedRatio\left(Group\left(\mathbf{w}_0^l\right)\right)\right\}, \tag{5}$$

where $MaxExplainedRatio(\cdot)$ returns the largest explained variance ratio in the PCA result, and $Group(\cdot)$ is the parameter grouping function. In verification, a large p_{pca} suggests that the initial model of the model chain is not likely to be randomly initialized.

(4) Weight Distance. In §3.4, we empirically study the natural properties of the weight distance during model training. Our observations can be further formalized as the following two properties that a legitimate model chain shall hold.

Property 5 (Monotonicity of Weight Distance). Given that the training of the model converges normally, the distance between the *intermediate* checkpoint and the *converged* model decreases monotonically to zero, *i.e.*, the following inequality on the model distance is expected to hold:

$$d(C_1, C_N) \geq d(C_2, C_N) \geq \cdots \geq d(C_{N-1}, C_N) \gtrsim 0.$$

Property 5 can also be measured using the Spearman's rank correlation coefficient of the distance sequence:

$$\rho_{dis} \triangleq \left|1 - \frac{6 \times \sum_i^N \left(\text{rank}\left(d(C_i, C_N)\right) - i\right)^2}{N(N^2 - 1)}\right|, \tag{6}$$

Property 6 (Small Distance between the Initial Model and Converged Model). Given that the DNN model is sufficiently complex, it is highly possible that the distance of the *converged* model to the *initial* model in the same model chain is much smaller than that to other randomly initialized models. Formally, given the *initial* model C_0 and *converged* model C_N in a model chain C and any other random model M_{rand} initialized with the same method as C_0, the following inequality holds with a high probability:

$$d(C_0, C_N) \ll d(M_{rand}, C_N).$$

Property 6 can be quantified as follows:

$$d_{chain} \triangleq d(C_0, C_N). \tag{7}$$

This property prevents the adversary from constructing a fake model chain by concatenating several short model chains, because the distance pattern is discontinuous and non-monotonic.

4.2 DNN Model Training with PoT

In order to achieve effective ownership verification, PoT has the following requirements on the model training.

(1) Initialize the model parameters with a special two-component GMM distribution. The mean values of the two components in the GMM distribution are $\pm 2\sqrt{2/5n_{in}}$ and the standard deviations are both equal to $\sqrt{2/5n_{in}}$, where n_{in} is the fan-in of the layer [4]. Under this setting, the standard deviation of the GMM distribution equals $\sqrt{2/n_{in}}$, which helps stabilize the magnitudes of the model activation values [13].

(2) Use L2 regularization term during training. It is a common practice to avoid overfitting by applying a Gaussian prior to the parameters [27].

PoT only has minimal requirements on the parameter initialization and regularization, and it does not conflict with advanced training techniques like knowledge distillation [15]. Following the requirements of PoT, the model owner needs to save the initial model and the model checkpoints after each training epoch. After the model converges to its optimum, the model owner can secretly store the model chain in the form of a list of training checkpoints, and use it as the ownership certificate.

Table 2: The deficiencies of possible adaptive attacks. P1-P6 are the verification properties of PoT.

Attack	P1	P2	P3	P4	P5	P6
Param. Interpolation	N/A	✓	◑	◑	✓	◐[a]
Regulated Distillation	✓	✓	◑	◑	✓	◐
Backward Construction	N/A[b]	N/A	✗	✗	N/A	N/A

[a] ◐ and ◑ mean that the two types of properties cannot be faked simultaneously.
[b] Technically difficult to be faked by the adversary (see details in §5.2).

4.3 Model Ownership Verification using PoT

Ownership verification in PoT is conceptually simple: only the entity possessing a *coherent model chain* that satisfies all the aforementioned properties can claim ownership successfully. Adopting PoT is lightweight, since the construction of a coherent model chain is straightforward for the party that performs actual training.

However, designing the verification protocol is non-trivial. In particular, values of certain properties (*e.g.*, Properties 4 and 6) can be largely influenced by the complexity of models and datasets. Therefore, we leverage two strategies to perform the property verification. For properties whose values are less affected (such as Properties 1, 2, 3, and 5), we consider a model chain to satisfy these properties as long as their values are within the predefined thresholds. Meanwhile, for Property 4, since the verifier knows the architecture and the initialization method of the model, we need the verifier to randomly re-initialize the model several times and compute the property value to estimate its valid range. Property 4 of a coherent model chain should be within this range because its initial model is randomly initialized, and the incurred overhead for range estimation is low. Property 6 is verified in a similar way by randomly sampling the initial model to estimate a range of rejection. The basic idea is that d_{chain} of a coherent model chain is much smaller than that of a model chain with a forged initial checkpoint, so the valid d_{chain} should be smaller than the lower bound of the estimated range.

The verification proceeds as follows. It firstly uses a threshold-based approach to verify ρ_{acc}, ρ_{dis}, c_{weight}, and d_{init}. Then it estimates the valid range of p_{pca} and d_{chain} by randomly sampling initial models and verifies the model chains with the estimated ranges. A model chain is coherent only if it passes the verification of all the properties. We further provide the verification algorithm in Appendix A and discuss the threshold selection in Appendix B. We also discuss potential issues in the application of PoT in Appendix F.

5 Security Analysis

In this section, we analyze the security of PoT against possible attacks to demonstrate its effectiveness. We discuss both existing attacks against IP protection and possible adaptive attacks.

5.1 Security Against Existing Attacks

We follow the taxonomy of existing attacks on model ownership verification methods in [24], which divides them into three categories:

Input Preprocessing Attack. Input preprocessing attacks modify the input data samples of DNN models before classification. For instance, adding noises or smoothing the input to break the possible backdoor triggers in the input. Since the attacks do not forge model chains, they are not applicable to PoT.

Model Modification Attack. Model modification attacks manipulate illegally acquired models with techniques such as fine-tuning and parameter pruning to erase the embedded watermarks or decouple the original fingerprints from the modified model. Since such attacks are irrelevant to model training from scratch, they cannot forge a valid model chain and thus cannot pass PoT's verification.

Model Extraction Attack. Model extraction attacks derive a surrogate model by distilling M with an auxiliary dataset [15]. An adversary may distill the stolen model following the training requirements of PoT to generate a surrogate model and the corresponding model chain. We call this attack *direct distillation attack*. However, as a prerequisite, the adversary is required to own a relatively large $\mathcal{D}_{\text{aux}}$ with similar distribution to $\mathcal{D}_{\text{train}}$ of M, which essentially imposes an attack barrier. We empirically study this attack barrier in Appendix C. The experimental results show that, in order to achieve comparable performance with the original model, the size of $\mathcal{D}_{\text{aux}}$ is required to be (at least) 30% of the size of $\mathcal{D}_{\text{train}}$. As a result, the effectiveness of the direct distillation attack greatly depends on a non-trivial $\mathcal{D}_{\text{aux}}$ that is comparable with the original $\mathcal{D}_{\text{train}}$ in both size and distribution. Such a barrier of launching this attack nullifies its effectiveness under our threat model defined in §2.2.

In summary, existing ownership forgery attacks fail to invalidate PoT because they cannot forge ownership certificates under our threat model. Therefore, we primarily focus on adaptive attacks.

5.2 Security Against Adaptive Attacks

Adaptive attacks are performed by adversaries with capabilities defined in our threat model (§2.2). We consider three such attacks: parameter interpolation, regulated distillation and backward construction. Our following discussion shows that none of them can defeat PoT under our threat model. The findings are in Table 2.

Parameter Interpolation Attack. In this attack, the adversary forges the intermediate checkpoints for the illegally acquired model by linearly interpolating the parameters of a randomly initialized model and the stolen model with different ratios:

$$\mathbf{w}_{checkpoint} = (1-\alpha)\mathbf{w}_{rand} + \alpha\mathbf{w}_{final},$$

where $\mathbf{w}_{rand}$ is the parameters of a randomly initialized model, $\mathbf{w}_{final}$ is the parameters of the stolen model, and $\alpha \in [0,1]$ is the interpolation ratio. By varying α, the adversary can forge a series of models with different validation performances and distances to the final model and fake a model chain without a dataset.

However, the model chain forged by this attack cannot simultaneously satisfy all the properties PoT requires, especially Property 6. This is because the adversary has to choose a randomly initialized model M_{rand} and interpolate it with the stolen model C_N to meet the requirements of Property 3 and 4. However, according to the measurement study in §3.4, $d(M_{rand}, C_N)$ is much larger than $d(C_0, C_N)$. Meanwhile, if the adversary chooses to directly forge an initial model that is close to C_N by manipulating the parameters of C_N, the initial model will be not randomly initialized, which violates Properties 3 and 4. Thus, the parameter interpolation attack cannot satisfy all the properties simultaneously.

Regulated Distillation Attack. To overcome the limitation of dataset availability in the direct distillation attack, the adversary can add an additional regularization term to the distillation loss to penalize the distance between the student and stolen models. Formally, let $\mathcal{L}$ denote the original loss function of the direct distillation attack, the loss function of the regulated distillation attack

is:

$$\mathcal{L}' = \mathcal{L} + \gamma \cdot d(M_{student}, M_{teacher}), \tag{8}$$

By minimizing $\mathcal{L}'$, the distance between $M_{student}$ and $M_{teacher}$ reduces, and thus $M_{student}$ gradually converges to $M_{teacher}$.

Despite the ease of resource constraints, it is still hard for the adversary to forge the properties. Particularly, to satisfy Properties 3 and 4, the adversary should randomly initialize a model M_{rand} with the required GMM distribution. However, as our observation in §3.4 implies, $d(M_{rand}, M_{teacher})$ is highly probable to be larger than d_{chain} of the real model chain. Since $d(M_{rand}, M_{distill}) \approx d(M_{rand}, M_{teacher}) > d_{chain}$, the forged model chain fails to satisfy Property 6 and thus cannot pass verification. To forge Property 6, the adversary has to find an initial model with the GMM distribution that is close to $M_{teacher}$. It is difficult to formulate this into an optimization problem since Property 3 is not differentiable. Instead, the adversary could transform the parameters of $M_{teacher}$ back into the GMM distribution via histogram matching [12]. In this way, the adversary may forge an initial model that satisfies Properties 3 and 6 and meanwhile has a small distance to $M_{teacher}$. However, since the parameters are transformed from a trained model, p_{pca} of the forged model chain will be large and thus it cannot pass the verification of Property 4. In summary, neither initial model forgery strategy can fake all properties simultaneously. Therefore, PoT is secure against the regulated distillation attack.

Backward Construction Attack. In this attack, the adversary fine-tunes the stolen model with a wrongly-labeled dataset. The adversary adjusts the proportion of wrong labels to degrade the model performance gradually, building a model chain from the tail. To satisfy Properties 3 and 4, the parameters should revert to the original GMM distribution, which can be achieved by choosing a random model M_{rand} and minimizing its distance to the model:

$$\mathcal{L}' = \mathcal{L} + \beta \cdot d(M, M_{rand}). \tag{9}$$

It is difficult for this attack to forge all the properties simultaneously. First, it is technically hard to precisely control the hyperparameters (*e.g.*, the learning rate and ratio of wrong labels) to forge properties in coherence. For example, because of overparameterization, the model may "remember" the wrong labels and thus model accuracy may not decrease monotonically. Besides, since the dataset is noisy, the fine-tuning process may be unstable and violate Properties 2, 5, and 6. Meanwhile, the distribution distance between parameters of the converged model and the required GMM distribution is large, making it hard to restore the distribution through fine-tuning, which violates Property 3. Moreover, the inherent parameter correlation in a trained model cannot be removed via simple fine-tuning. Therefore, the parameters of the forged initial model still exhibit such inherent correlation, violating Property 4.

6 Experiment Evaluation

In this section, we implement the attacks discussed in §5 to demonstrate the robustness of PoT.

6.1 Experiment Setup

Datasets and Models. To demonstrate that PoT is scalable, we evaluate PoT on various common CV and NLP models and datasets. The data allocations among the model owner, the adversary and the verifier are as follows: the model owner owns $(1 - X)\%$ of the original training samples, the adversary owns the remaining $X\%$ training samples, and the verifier owns the original testing samples. Per our threat model in §2.2, we set $X = 10$ to stress test PoT. We evaluate PoT on five commonly used datasets, including three CV datasets (MNIST, CIFAR10 and CIFAR100) and two NLP datasets (DBpedia and SOGOU). For the CV task, we implement an 8-layer CNN model and a ResNet18 model [14]. For the NLP task, we implement a TextCNN model with three 1-D convolutional layers and a TextRNN model with two LSTM layers. All the models are trained following the requirements of PoT.

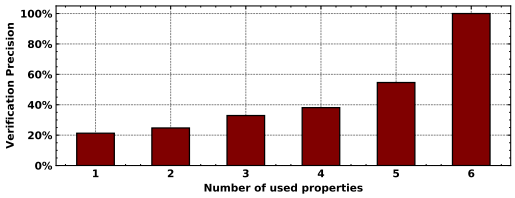


Figure 6: Verification precision of PoT when using the first $k \in [1, 6]$ properties in the verification algorithm.

Implementation of Attacks. For the parameter interpolation attack, we interpolate a randomly initialized model with the converged model and change the interpolation ratio α from 0 to 1 with a uniform step of 0.01. For the regulated interpolation attack, we distill the target model for 100 epochs with a distillation temperature of 2 and a regularization coefficient γ of 0.005. For the backward construction attack, we fine-tune the converged model with $\beta = 0.005$. We set the initial poisoned label rate in $\mathcal{D}_{\text{aux}}$ to 40% and increase it by 10% every 10 epochs until reaching 80%.

6.2 Overall Performance

We use precision as the evaluation metric, which is the proportion of real model chains in the model chains predicted to be coherent by PoT. We evaluate the effectiveness of PoT by repeating the attacks on each trained model 10 times. To understand the overall performance under different scenarios, we mix all the four attacking strategies and eight model architectures. We report the overall precision of PoT on verifying forged model chains and corresponding real model chains, *i.e.*, 320 forged model chains and 32 real model chains in total. Besides, to study the impact of the six properties, we evaluate PoT using different numbers of properties in verification. Specifically, since PoT verifies the six required properties sequentially, we report the precision achieved at each step (*e.g.*, the precision at the fourth step is achieved using the first four properties). The evaluation results of PoT on differentiating the real and forged model chains generated by different attacks are shown in Figure 6 (we present the detailed verification precision on different attacks in Appendix E). The results show that PoT is able to achieve a precision of 100% on all the evaluated attacks when using all six properties. Besides, the precisions using different numbers of properties also imply that all the properties contribute to the verification. Thus, combining them together can increase the difficulty for the attack and ensure the robustness of PoT.

6.3 Empirical Studies on Attacks

Now, we analyze in-depth why the evaluated attacks fail to evade PoT, in order to validate our security analysis in §5.

Against Regulated Distillation Attacks. In §5.2, we analyze that both of the two strategies of the regulated distillation attack fail to satisfy all the properties simultaneously. For the attack with the random initialization, we conclude that it faces the problem of a large d_{chain} (Property 6). To demonstrate this, we report the validation accuracy and d_{chain} of the models forged by this attack

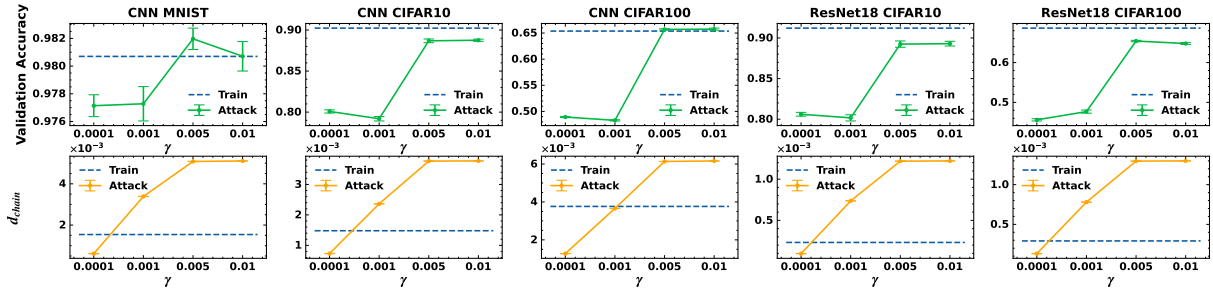


Figure 7: Impact of regularization coefficient γ on the performance of the regulated distillation Attack.

Table 3: d_{chain} and p_{pca} of model chains generated by the original training and the regulated distillation attack with histogram matching initialization.

Model	Dataset	d_{chain}		p_{pca}	
		Original	Attack	Original	Attack
CNN	MNIST	1.58e-3	6.76e-4 ± 3.0e-8	0.148	0.221 ± 0.000
	CIFAR10	1.48e-3	2.56e-4 ± 9.9e-7	0.158	0.308 ± 0.000
	CIFAR100	3.76e-3	6.17e-4 ± 5.0e-7	0.148	0.392 ± 0.000
ResNet18	CIFAR10	2.32e-4	1.28e-4 ± 9.8e-8	0.148	0.297 ± 0.000
	CIFAR100	2.92e-4	1.23e-4 ± 4.6e-8	0.151	0.349 ± 0.000
TextCNN	DBpedia	5.51e-4	1.14e-4 ± 2.2e-8	0.110	0.154 ± 0.000
	SOGOU	3.84e-3	2.29e-4 ± 3.1e-8	0.110	0.286 ± 0.000
TextRNN	DBpedia	2.85e-3	6.28e-4 ± 5.1e-7	0.098	0.126 ± 0.000

with different values of γ in Figure 7. It shows that values of d_{chain} of forged model chains are larger than that of the real model chains as long as $\gamma > 0.001$, leading to the failure of the attack. Although d_{chain} can be reduced by decreasing γ (*e.g.*, d_{chain} of forged model chains can be smaller than real model chains when $\gamma = 0.0001$), this will lead to poor performance of the surrogate models and the adversary is less motivated to perform the attack.

For the attack with histogram matching initialization, we analyzed that although it can satisfy Property 6, it has the problem of breaking Property 4 because of a large p_{pca}. To demonstrate this, we report d_{chain} and p_{pca} of the forged model chains and the real coherent model chains in Table 3. Results show that although the d_{chain} of the attack can be smaller or similar to real model chains generated by training from scratch, p_{pca} of the attack is larger than that of real model chains, which confirms our analysis.

Against Backward Construction Attacks. In §5.2, we analyze that it is difficult for model chains forged by the backward construction attack to satisfy the required properties. We demonstrate this by utilizing Properties 3 and 4. Results (see Table 4) show that d_{init} and p_{pca} of the forged model chains are always larger than those of the real one. This means that the parameter distributions of the forged model chain are not restored to the GMM distribution, and the parameters of the forged initial model are still correlated. Hence, the forged model chain cannot pass the verification of PoT.

7 Related Work

Watermark-based methods. Watermarking DNNs to protect model ownership has been extensively studied. It can be classified into two categories, *i.e.*, feature-based and trigger set-based. Feature-based methods [7, 31, 35] embed information into model parameters. For example, DeepMarks [7] implants binary code into target layers of the pre-trained model. Trigger set-based methods embed triggers into the models. Adi et al. [2] implant backdoors as watermarks into DNNs by over-parameterizing the model. Jia et al. [16] try to improve watermark robustness by entangling normal and watermark data distributions. Unfortunately, watermarking schemes are not robust and exclusive [3, 8, 19, 23, 28, 37]. Adversaries can conduct watermark removal or model piracy attacks.

Table 4: d_{init} and p_{pca} of model chains generated by the original training and the backward construction attack.

Model	Dataset	d_{init}		p_{pca}	
		Original	Attack	Original	Attack
CNN	MNIST	0.133	1.541 ± 0.000	0.148	0.233 ± 0.000
	CIFAR10	0.110	1.210 ± 0.000	0.158	0.319 ± 0.000
	CIFAR100	0.091	1.939 ± 0.000	0.148	0.404 ± 0.000
ResNet18	CIFAR10	0.133	3.523 ± 0.000	0.148	0.325 ± 0.000
	CIFAR100	0.133	2.582 ± 0.000	0.151	0.384 ± 0.000
TextCNN	DBpedia	0.086	0.507 ± 0.000	0.110	0.166 ± 0.000
	SOGOU	0.086	1.780 ± 0.000	0.110	0.294 ± 0.000
TextRNN	DBpedia	0.109	0.909 ± 0.000	0.098	0.133 ± 0.000

Fingerprint-based methods. Fingerprint-based methods *extract* ownership proofs from models instead of embedding, which preserves the model utility [5, 25, 36, 42]. For example, Cao et al. [5] and Wang et al. [36] show that data near the unique classification boundaries of DNN classifiers can be used as fingerprints. Although these methods do not impact performance, they still cannot ensure robustness or exclusiveness. Particularly, since fingerprints are not timestamped, adversaries can extract their own fingerprints and use as their proof of ownership [11].

Computation-based method. Proof-of-Learning (PoL) [17] verifies model ownership by checking the reproducibility of the training process. It retrains checkpoints from the previous ones according to training logs, which only uses the local property of the model chain. Due to the weak security guarantee, the adversary can develop adaptive attacks given PoL's design details [40]. In contrast, our method exploits both the local nature (Properties 2, 3 and 4) and the global nature (Properties 1, 5 and 6) of the model chain. More critically, PoL has privacy leakage issues because it requires the verifier to have full access to the proprietary training data.

8 Conclusion

In this paper, we present PoT, the first exclusive, robust, and privacy-preserving design for DNN model ownership verification. PoT addresses the limitations of the state of the arts that are either vulnerable to various attacks or require full access to the private training dataset. In contrast, PoT proposes to verify the provenance of the training effort using the naturally coherent model chain saved during training. We perform a comprehensive security analysis and conduct extensive empirical experiments to validate the robustness of PoT against various attacks.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*. 265–283.

[2] Yossi Adi, Carsten Baum, Moustapha Cissé, Benny Pinkas, and Joseph Keshet. 2018. Turning Your Weakness Into a Strength: Watermarking Deep Neural Networks by Backdooring. In *27th USENIX Security Symposium, USENIX Security 2018*. 1615–1631.

[3] William Aiken, Hyoungshick Kim, Simon S. Woo, and Jungwoo Ryoo. 2021. Neural network laundering: Removing black-box backdoor watermarks from deep neural networks. *Comput. Secur.* 106 (2021), 102277.

[4] Yoshua Bengio. 2012. Practical recommendations for gradient-based training of deep architectures. arXiv:1206.5533 [cs.LG]

[5] Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. 2021. IPGuard: Protecting Intellectual Property of Deep Neural Networks via Fingerprinting the Classification Boundary. In *ASIA CCS '21: ACM Asia Conference on Computer and Communications Security*. ACM, 14–25.

[6] William Chan, Navdeep Jaitly, Quoc V. Le, and Oriol Vinyals. 2016. Listen, Attend and Spell: A Neural Network for Large Vocabulary Conversational Speech Recognition. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2016* (2016). IEEE, 4960–4964.

[7] Huili Chen, Bita Darvish Rouhani, Cheng Fu, Jishen Zhao, and Farinaz Koushanfar. 2019. DeepMarks: A Secure Fingerprinting Framework for Digital Rights Management of Deep Learning Models. In *Proceedings of the 2019 on International Conference on Multimedia Retrieval, ICMR 2019*. ACM, 105–113.

[8] Xinyun Chen, Wenxiao Wang, Chris Bender, Yiming Ding, Ruoxi Jia, Bo Li, and Dawn Song. 2021. REFIT: A Unified Watermark Removal Framework For Deep Learning Systems With Limited Data. In *ASIA CCS '21: ACM Asia Conference on Computer and Communications Security*. ACM, 321–335.

[9] Anna Choromanska, Mikael Henaff, Michaël Mathieu, Gérard Ben Arous, and Yann LeCun. 2015. The Loss Surfaces of Multilayer Networks. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2015* (2015), Vol. 38.

[10] Yadolah Dodge. 2010. *The Concise Encyclopedia of Statistics*. New York : Springer. 502–503 pages.

[11] Lixin Fan, KamWoh Ng, and Chee Seng Chan. 2019. Rethinking Deep Neural Network Ownership Verification: Embedding Passports to Defeat Ambiguity Attacks. In *NeurIPS 2019*. 4716–4725.

[12] Rafael C. Gonzalez and Richard E. Woods. 2006. *Digital Image Processing (3rd Edition)*. Prentice-Hall, Inc., USA. 150–160 pages.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *2015 IEEE International Conference on Computer Vision (ICCV)* (2015-12). 1026–1034.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016* (2016). 770–778.

[15] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the Knowledge in a Neural Network. *CoRR* abs/1503.02531 (2015).

[16] Hengrui Jia, Christopher A. Choquette-Choo, Varun Chandrasekaran, and Nicolas Papernot. 2021. Entangled Watermarks as a Defense against Model Extraction. In *30th USENIX Security Symposium (USENIX Security 21)*. 1937–1954.

[17] Hengrui Jia, Mohammad Yaghini, Christopher A. Choquette-Choo, Natalie Dullerud, Anvith Thudi, Varun Chandrasekaran, and Nicolas Papernot. 2021. Proof-of-Learning: Definitions and Practice. In *Proceedings of the 42nd IEEE Symposium on Security and Privacy* (2021).

[18] Erwan Le Merrer, Patrick Pérez, and Gilles Trédan. 2020. Adversarial Frontier Stitching for Remote Neural Network Watermarking. *Neural Computing and Applications* 32 (2020), 9233–9244.

[19] Suyoung Lee, Wonho Song, Suman Jana, Meeyoung Cha, and Sooel Son. 2021. Evaluating the Robustness of Trigger Set-Based Watermarks Embedded in Deep Neural Networks. *CoRR* abs/2106.10147 (2021). arXiv:2106.10147

[20] Meng Li, Qi Zhong, Leo Yu Zhang, Yajuan Du, Jun Zhang, and Yong Xiangt. 2020. Protecting the Intellectual Property of Deep Neural Networks with Watermarking: The Frequency Domain Approach. In *TrustCom*. IEEE, 402–409.

[21] Tommy Li and Cory E. Merkel. 2021. Model Extraction and Adversarial Attacks on Neural Networks using Switching Power Information. *CoRR* abs/2106.08299 (2021). arXiv:2106.08299

[22] Zheng Li, Chengyu Hu, Yang Zhang, and Shanqing Guo. 2019. How to prove your model belongs to you: a blind-watermark based framework to protect intellectual property of DNN. In *ACSAC*. ACM, 126–137.

[23] Xuankai Liu, Fengting Li, Bihan Wen, and Qi Li. 2021. Removing Backdoor-Based Watermarks in Neural Networks with Limited Data. In *2020 25th International Conference on Pattern Recognition (ICPR)*. 10149–10156.

[24] N. Lukas, E. Jiang, X. Li, and F. Kerschbaum. 2022. SoK: How Robust is Image Classification Deep Neural Network Watermarking?. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 52–69.

[25] Nils Lukas, Yuxuan Zhang, and Florian Kerschbaum. 2021. Deep Neural Network Fingerprinting by Conferrable Adversarial Examples. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event*. OpenReview.net.

[26] Xiaoxiao Ma, Jia Wu, Shan Xue, Jian Yang, Chuan Zhou, Quan Z. Sheng, Hui Xiong, and Leman Akoglu. 2021. A Comprehensive Survey on Graph Anomaly Detection with Deep Learning. *IEEE Transactions on Knowledge and Data Engineering* (2021), 1–1. https://doi.org/10.1109/TKDE.2021.3118815

[27] Kevin P. Murphy. 2012. *Machine learning - a probabilistic perspective*. MIT Press.

[28] Ryota Namba and Jun Sakuma. 2019. Robust Watermarking of Neural Network with Exponential Weighting. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security, AsiaCCS 2019*. ACM, 228–240.

[29] Quynh Nguyen and Matthias Hein. 2017. The Loss Surface of Deep and Wide Neural Networks. In *Proceedings of the 34th International Conference on Machine Learning* (2017-08-06/2017-08-11), Vol. 70. PMLR, 2603–2612.

[30] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019), 8026–8037.

[31] Bita Darvish Rouhani, Huili Chen, and Farinaz Koushanfar. 2019. DeepSigns: An End-to-End Watermarking Framework for Ownership Protection of Deep Neural Networks. In *ASPLOS 2019*. 485–497.

[32] Sameer Shirdhonkar and David W. Jacobs. 2008. Approximate earth mover's distance in linear time. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*. 1–8.

[33] Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2020. Energy and Policy Considerations for Modern Deep Learning Research. In *AAAI 2020*. AAAI Press, 13693–13696.

[34] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. 2016. Stealing Machine Learning Models via Prediction APIs. In *25th USENIX Security Symposium, USENIX Security 16*. USENIX Association, 601–618.

[35] Yusuke Uchida, Yuki Nagai, Shigeyuki Sakazawa, and Shin'ichi Satoh. 2017. Embedding Watermarks into Deep Neural Networks. In *ICMR 2017*. 269–277.

[36] Si Wang and Chip-Hong Chang. 2021. Fingerprinting Deep Neural Networks - a DeepFool Approach. In *IEEE International Symposium on Circuits and Systems, ISCAS 2021*. IEEE, 1–5.

[37] Tianhao Wang and Florian Kerschbaum. 2019. Attacks on Digital Watermarks for Deep Neural Networks. In *ICASSP*. IEEE, 2622–2626.

[38] Jie Zhang, Dongdong Chen, Jing Liao, Han Fang, Weiming Zhang, Wenbo Zhou, Hao Cui, and Nenghai Yu. 2020. Model Watermarking for Image Processing Networks. In *AAAI*. AAAI Press, 12805–12812.

[39] Jialong Zhang, Zhongshu Gu, Jiyong Jang, Hui Wu, Marc Ph. Stoecklin, Heqing Huang, and Ian Molloy. 2018. Protecting Intellectual Property of Deep Neural Networks with Watermarking. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security - ASIACCS '18*. ACM Press, 159–172.

[40] Rui Zhang, Jian Liu, Yuan Ding, Qingbiao Wu, and Kui Ren. 2021. "Adversarial Examples" for Proof-of-Learning. *CoRR* abs/2108.09454 (2021).

[41] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. 2019. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR)* 52, 1 (2019), 1–38.

[42] Jingjing Zhao, Qingyue Hu, Gaoyang Liu, Xiaoqiang Ma, Fei Chen, and Mohammad Mehedi Hassan. 2020. AFA: Adversarial Fingerprinting Authentication for Deep Neural Networks. *Computer Communications* 150 (2020), 488–497.

[43] Yuankun Zhu, Yueqiang Cheng, Husheng Zhou, and Yantao Lu. 2021. Hermes Attack: Steal DNN Models with Lossless Inference Accuracy. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 1973–1988.

A Verification Algorithm

Algorithm 1 shows the pseudocode of the verification algorithm of PoT.

Algorithm 1: PoT Verification

Input: The model chain C to be verified by PoT
Output: A Boolean value indicating whether the model chain C is coherent

Calculate $\rho_{acc}, \rho_{dis}, d_{chain}, c_{weight}, d_{init}, p_{pca}$ for the model chain C based on the definitions in §4.1;
if $C \rightarrow \rho_{acc} < \delta_1$ **then**
  return *False*; // *Report C to be incoherent.*
else if $C \rightarrow \rho_{dis} < \delta_2$ **then**
  return *False*;
else if $C \rightarrow c_{weight} > \delta_3$ **then**
  return *False*;
else if $C \rightarrow d_{init} > \delta_4$ **then**
  return *False*;
estPCAmean, estPCAstd = estimatePCAratio(C) ; // *Estimate the range of valid p_{pca} by randomly reinitializing the protected model for several times and calculate p_{pca}.*
if $C \rightarrow p_{pca} >$ estPCAmean + 10 × estPCAstd **then**
  return *False*;
estDisMean, estDisSTD = estimateChainDistance(C) ; // *Estimate the range of d_{chain} of forged model chains based on randomly re-initializing.*
if $d_{chain} <$ estDisMean − 30 × estDisSTD **then**
  return *True*;
else
  return *False*;

B Threshold Selection for Verification

The four properties that require predetermined thresholds (Properties 1, 2, 3, and 5) can naturally hold as long as the model is randomly initialized and stably trained (which is the case in the training of high-performance models). Therefore, these properties are less sensitive to the model architecture and dataset and their thresholds can be chosen a priori. Empirically, we set the thresholds δ_1, δ_2 , δ_3, and δ_4 in Algorithm 1 to be 0.5, 0.8, 0.8, and 0.3 respectively. In Table 5, we provide the four properties of model chains generated under normal training settings, where early stopping is used. The table shows that all the model chains satisfy the thresholds. Therefore, the algorithm does not produce false negatives, *i.e.*, wrongly labeling a model chain generated by real training as an incoherent one. Moreover, most property values are not sensitive to thresholds under most training settings. Note that the value of Property 1 is small under the setting of the CNN model trained with the MNIST dataset due to the simple task. Therefore, PoT is not sensitive to threshold selection.

C Experiments on the Direct Distillation Attack

In §5.2, we analyze that the direct distillation attack requires a relatively large auxiliary dataset to derive a surrogate model with comparable performance to the original model. To demonstrate this, we train teacher models with 50% of each dataset, and then perform the direct distillation attack on the trained models with different proportions of the remaining 50% of the dataset. We set the number of training epochs to 200, weight decay to 0.0001 and batch size to 128. We repeat the attack 10 times and report the accuracy of the distilled model on $\mathcal{D}_{\text{aux}}$ with respect to the relative size of the distillation dataset in Figure 8. Note that the accuracies of the teacher models are lower than usual (*e.g.*, the accuracies in Figure 7) because the models are trained with small subsets of the original training datasets.

The results show that the adversary needs a sufficiently large auxiliary dataset (at least 30% of the original dataset size) to obtain a surrogate model with comparable performance with the original model. In some cases, the surrogate model's performance is never comparable with the original model, regardless of the size of the auxiliary dataset. Thus, due to the requirement of a non-trivial auxiliary dataset that is comparable with the original dataset in both size and distribution, we consider this attack to be out of the scope of our threat model defined in §2.2.

D Experiments on Regulated Distillation Attack with Small-scale Auxiliary Dataset

We empirically evaluate the impact of the size of auxiliary dataset $\mathcal{D}_{\text{aux}}$ on the regulated distillation attack with random initialization. We perform the attack on the models with a small $\mathcal{D}_{\text{aux}}$ whose size is only 1% of $\mathcal{D}_{\text{train}}$, and all the other settings are the same as §6. The validation accuracy and d_{chain} of the models derived by the attack are reported in Table 6. Experiment results show that although the attack is able to derive a model with similar performance to the stolen model with limited data under most of the settings, d_{chain} of the forged model chain is significantly larger than that of the real model chain, which violates Property 6. Therefore, the attack can be

Table 5: Property values of real model chains. The models are training for 200 epochs with early stopping and the patience of early stopping is 20 epochs.

Model	Dataset	Stopping Epoch	ρ_{acc}	ρ_{dis}	c_{weight}	d_{init}
CNN	MNIST	68	0.672	1	0.431	0.133
	CIFAR10	120	0.941	1	0.273	0.11
	CIFAR100	77	0.978	1	0.344	0.091
ResNet18	CIFAR10	98	0.963	1	0.14	0.133
	CIFAR100	154	0.982	1	0.116	0.133
TextCNN	DBpedia	187	0.998	1	0.095	0.086
	SOGOU	29	0.919	1	0.66	0.086
TextRNN	DBpedia	91	0.98	1	0.118	0.109

Table 6: Validation accuracy of the final model and d_{chain} of training from scratch and the regulated distillation attack with a small dataset whose size is 1% of $\mathcal{D}_{\text{train}}$.

Model	Dataset	Validation Accuracy		d_{chain}	
		Training from Scratch	Attack	Training from Scratch	Attack
CNN	MNIST	0.981	0.982 ± 0.001	1.54e-3	5.08e-03 ± 4.02e-06
	CIFAR10	0.902	0.894 ± 0.001	1.48e-3	3.75e-03 ± 1.73e-06
	CIFAR100	0.654	0.659 ± 0.001	3.76e-3	6.10e-03 ± 3.71e-06
ResNet18	CIFAR10	0.912	0.904 ± 0.001	2.32e-4	1.21e-03 ± 2.94e-07
	CIFAR100	0.686	0.662 ± 0.002	2.92e-4	1.29e-03 ± 3.24e-07
TextCNN	DBpedia	0.985	0.893 ± 0.001	5.51e-4	4.37e-03 ± 3.93e-06
	SOGOU	0.947	0.899 ± 0.000	3.84e-3	5.04e-03 ± 3.98e-06
TextRNN	DBpedia	0.978	0.963 ± 0.001	3.40e-3	1.37e-02 ± 5.01e-05

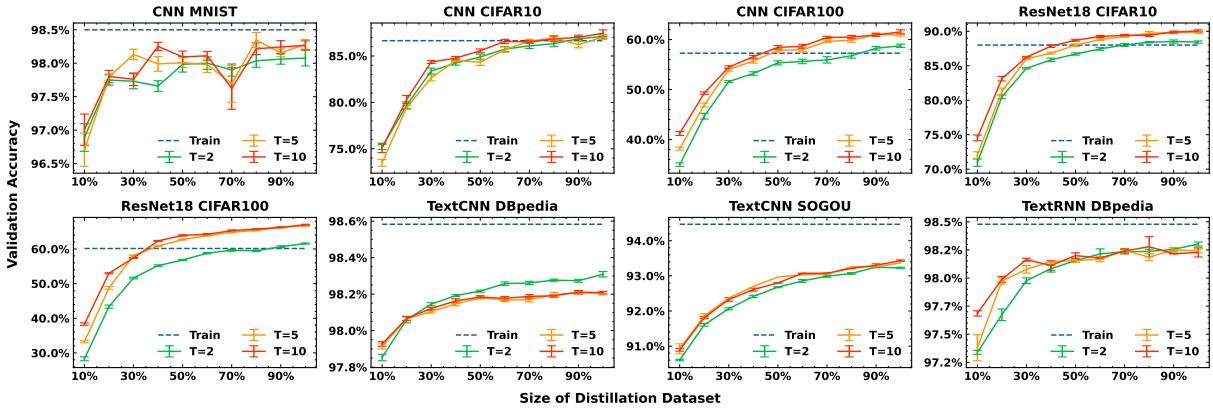
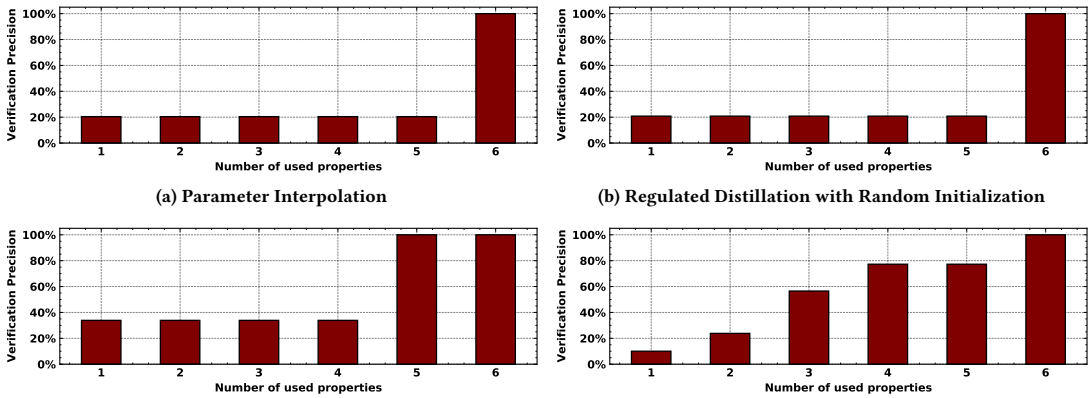


Figure 8: Validation accuracy of direct distillation attack w.r.t the relative size of distillation dataset. T is the temperature of distillation.



(a) Parameter Interpolation

(b) Regulated Distillation with Random Initialization

(c) Regulated Distillation with Histogram Matching Initialization

(d) Backward Construction

Figure 9: Verification precision of PoT on different attacks with different numbers of properties used in the verification algorithm.

detected by PoT. We further provide the detection precisions of PoT on differentiating the real and forged model chains in Appendix D, and the results show that PoT is able to perform the detection with 100% precision.

E Verification Precision on Different Attacks

Figure 9 shows the detailed verification precision of PoT on different attacks when the size of $\mathcal{D}_{\text{aux}}$ is 10% of $\mathcal{D}_{\text{train}}$. The results show that PoT is able to successfully defend against all these attacks.

F Discussion

Validity of validation dataset. $\mathcal{D}_{\text{val}}$ plays an important role in resolving an ownership dispute, since PoT asks for verifying whether the model chain has the property of the validation accuracy monotonicity. To enforce a valid and disinterested verification, the construction of $\mathcal{D}_{\text{val}}$ should be *fair and unbiased* for all parties involved in the dispute. To achieve this, the validation dataset can be built by crowdsourcing with the help from the community. Such datasets can be collected in advance by building an open platform, so that PoT can be applied to models on different tasks and fields.

Transfer learning in model training. To prove model ownership, PoT requires model trainers to initialize their model parameters with a specific GMM distribution. If model trainers want to conduct transfer learning on a pre-trained model, this prerequisite cannot be satisfied, and thus PoT cannot be directly applied to the transfer learning scenario. However, pre-trained models are not available in some scenarios, such as learning-based traffic analysis and anomaly detection. Besides, applying pre-trained models may need to deal with licensing issues. Therefore, PoT focuses on protecting models trained from scratch, and we consider ownership verification in the transfer learning scenario as future work.

Dishonest verifier. One security concern of PoT is that a dishonest verifier may illegally copy the model chain under verification and later falsely claim the ownership of the model. This concern can be addressed as follows: the model owner calculates the hash values of the model chain upon finishing training, and commits them on a public blockchain. This enables the model owner to claim exclusive ownership of the model chain, even if later handing the model chain over to the verifier.