Graph-Embedded Structure-Aware Perceptual Hashing for Neural Network Protection and Piracy Detection

Ruiheng Liu[1], Haozhe Chen[2,*], Boyao Zhao[1], Kejiang Chen[1], Weiming Zhang[1],
[1]Anhui Province Key Laboratory of Digital Security, University of Science and Technology of China
[2]Anhui University
{liurh2023@mail., zby26754980@mail., chenkj@, zhangwm@}ustc.edu.cn 23085@ahu.edu.cn

Abstract

The advancement of AI technology has significantly influenced production activities, increasing the focus on copyright protection for AI models. The perceptual hashing of the model offers an efficient solution for retrieve the pirated models. Existing methods, such as handcrafted feature-based and dual-branch network-based perceptual hashing, have proven effective in detecting pirated models. However, these approaches often struggle to differentiate nonpirated models, leading to frequent false positives in model authentication and protection. To address this challenge, this paper proposes a structurally-aware perceptual model hashing technique that achieved reduced false positives while maintaining high true positive rates. Specifically, we introduce a method for converting the diverse neural network structures into graph structures suitable for DNN processing, then utilize a graph neural network to learn their structural features representation. Our approach integrates perceptual parameter-based model hashing, achieving robust performance with higher detection accuracy and fewer false positives. The experimental results show that the proposed method has only 3% false alarm rate when detecting the non-pirated model, and the detection accuracy of the pirated model reaches more than 98%.

1. Introduction

In recent years, a wide variety of neural networks [4, 5, 10, 13, 25, 31] have emerged and been applied to many aspects of daily life. Out of respect for human intelligence and to protect human ethics, the growing importance of neural network certification calls for specialized technologies to assist developers in verifying their models and in identifying potential misuse, such as fine-tuning, pruning, or tampering with replicated models.

*Corresponding author

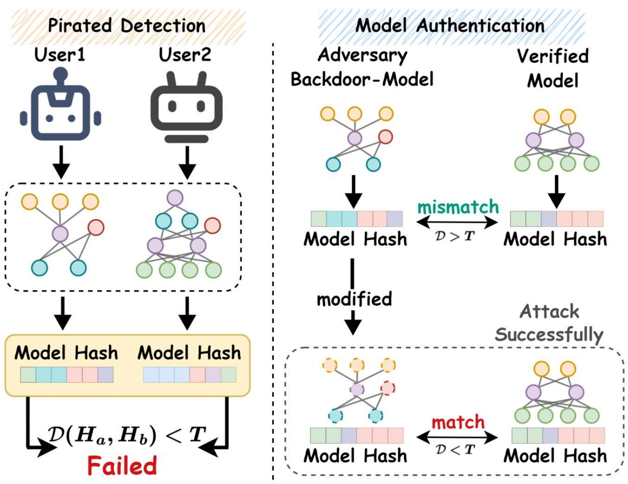

Figure 1. An illustration of model perceptual hash for copyright protection and model verification. The left subfigure illustrates a scenario in which false positives occur in a model pirated detection scenario. User1 and User2 have their own models, which have completely different structures. The model hash vulnerability leads to misidentification of similar models during the discrimination due to the hash distance less than threshold T. The right subfigure depicts the application in a model authentication scenario. The adversary creates a backdoor model and wishes to exploit a vulnerability in the model hash to authenticate it as a legitimate model. The adversary crafts a model modification so that its backdoor model matches the hash of the authenticated legitimate model, fooling the authentication system.

Currently, three main techniques are used for model authentication and protection: model watermarking [6, 33], model fingerprinting [17, 23] and model perceptual hashing [2, 28]. Model watermarking embeds copyright information into a model's parameters, enabling legitimate users to extract the watermark and confirm the model's origin. To avoid the need for retraining, researchers have proposed model fingerprinting, which does not modify model parameters but instead designs adversarial examples to protect intellectual property. However, both methods struggle to efficiently detect infringing models among the vast number of existing models. To address this challenge, Chen et al. [2] and Xiong et al. [28] introduced model perceptual hashing, which is inspired by hash-based image retrieval methods [3, 24, 32]. Model perceptual hash method achieves effective detection of replicated and pirated models by using parameter distribution statistics or deep learning models to learn recognition rules. As shown in Fig. 1, when adversary acquires the owner's original model, the perceptual hashing method—whether modified through pruning or fine-tuning—generates hashes for both the original and pirated models, calculating the hash distance to identify piracy. Specifically, Chen's method employs hand-crafted feature extraction for hashing, which is convenient but relies on a single fixed feature dimension and is susceptible to targeted attacks. In contrast, Xiong et al. [28] utilize deep learning models to capture model characteristics from multiple dimensions, effectively addressing the limitations of manually extracted features.

However, due to the absence of model structure information in the hashing process, the detection accuracy of DNN-based schemes is insufficient, leading to a risk of misidentifying non-pirated models with distinctly different structures as pirated ones. Current methods [28] are subject to higher false positives, *i.e*., a larger number of non-pirated models with different structures are recognized as stolen versions of a particular model at the time of detection. As shown in Fig. 1, the left side of the figure illustrates the model pirated detection scenario where two structurally dissimilar models are recognized as pirated models due to the hash distance being less than the threshold T. The right side of the figure shows a scenario in which an adversary wants to utilize the model-aware hash misreporting vulnerability to authenticate a malicious model as a normal model in the model repository during the model authentication process. In other words, the high false positive rate of model perceptual hashing will not only lead to errors in model piracy detection. Moreover, in the model authentication scenario, it will be utilized by the adversary to authenticate the malicious neural network model as a legitimate model which will lead to more serious consequences.

To address the challenge of achieving robust performance in both copy detection and distinguishing models with different structures, we introduce, for the first time, the inclusion of model structure information into perceptual hashing. Specifically, to encode diverse and complex model structures into unified statistical features, we propose a novel scheme that converts neural network structures into graph representations. In order to avoid multiple edges as well as circles that may result from the existing methods, we change the node connection strategy of the current scheme and design a model structure directed acyclic graph extraction strategy. Specifically, the original nodes representing layer inputs and layer outputs are directly connected to one or several intermediary nodes instead. These intermediary nodes can represent either fully connected layers or convolutional layers. Moreover, we train a model structure feature extractor on a large set of neural network graph data to generate structural features. Finally, we merge the trained network structure feature extractor with the parameter feature extractor via a merge layer to form a new three-branch network. Experimental results show that our proposed method has only 3% false positive rate in distinguishing different structural models and has more than 98% pirated model detection accuracy. Thus, our approach not only improves detection accuracy in copy detection scenarios but also enhances security in model authentication tasks. In summary, the main contributions of this work are as follows:

- For the first time, we encode model structure information into model perceptual hashing, significantly improving detection accuracy in model copy detection and enhancing security in model authentication tasks.
- We design a conversion strategy that translates diverse and complex model structures into graph representations, making it more suitable for our graph embedding model to extract structural information. This approach effectively reduces the number of nodes and edges in the generated graphs, reducing memory complexity from $\mathcal{O}(mn)$ to $\mathcal{O}(m+n)$ compared to existing methods.
- Experiments demonstrate that our method significantly reduces false positive rates and improves the accuracy of distinguishing models with different structures. And it also improves the accuracy of piracy model detection.

2. Related Work

2.1. Graph Neural Networks (GNNs)

Graph Neural Networks (GNNs) are generally divided into two categories: spectral graph convolutions and spatial graph convolutions. Spectral graph convolutions, based on the solid mathematical foundation of Graph Signal Processing (GSP) [8, 19, 21], aim to capture patterns in the frequency domain by leveraging the spectral properties of graphs. In contrast, spatial graph convolutions adopt the message-passing framework from Recurrent Graph Neural Networks (RecGNNs) [26], where node features are computed by aggregating the features of their neighboring nodes. Since message-passing networks like GCN [11] and GraphSAGE [7] are indistinguishable when confronted with certain graph structures. In order to solve this problem and enhance the representation capability of graph neural networks, Xu et al. proposed the graph isomorphism network (GIN) [29], which enhances the ability to distinguish different graph structures by assigning adjustable weights to the center node features. In this work, we adopt graph representation learning to generate features for structural information of DNNs.

2.2. Model Hashing

Inspired by perceptual hashing in image processing [1, 15, 20], Chen et al. [2] were the first to propose a perceptual hashing method for models, leveraging Normal Test Statistics (NTS) of convolutional neural network parameters to detect pirated models. Their work claimed effective piracy detection for convolutional networks with promising results. However, their method failed to account for the diverse differences that may exist between non-pirated and original models, resulting in suboptimal perceptual hashing performance. Subsequently, Xiong et al. [28]introduced the use of a dual-branch network to replace statistical features and manual quantisation methods, partially addressing the issue of hash distance independence from model structure. However, the high false positive rate of their method posed challenges for practical applications.

Based on these observations, we propose a structured hashing method that integrates model structure as part of the perceptual hash. Our method reduces the false alarm rate for different model detection to below 5% under the same threshold while simultaneously improving the accuracy of pirated model detection at that threshold.

3. Problem Formulation

Given that the hash sequences H_a and H_b are generated for two neural network models, M_a and M_b, using a hash generator, the hash distance D between H_a and H_b can be computed to assess the similarity or dissimilarity between models a and b. During the final hash calculation, we use mean squared error(MSE):

$$\mathcal{D}(H_a, H_b) = \frac{1}{L} \times \sum_{t=1}^{L} (h_{a_t} - h_{b_t})^2, \tag{1}$$

where $h_{a_t} \in \{0,1\}^t$ and $h_{b_t} \in \{0,1\}^t$ represent the t-th bit of hash sequences H_a and H_b, respectively, and L denotes the length of the hash sequence.

3.1. Robustness

If the model to be tested is a common modified version of the query model (common modified version means a model that has undergone model fine-tuning and model pruning), then the hash distance between the test model M_a and the query model M_b should be less than threshold T. formally, $\forall M_a, M_b$, we have follow:

$$\text{if } M_a = f(M_b) \text{ then, } \mathcal{D}(H_a, H_b) < T, \tag{2}$$

where H_a, H_b and f denote model hash of M_a and M_b and the common modifications, and T is threshold.

3.2. Discrimination

The distance of model hash should be larger than threshold T, if the model is a non-pirated model(*i.e.*, a test model is neither the query model nor its common modified version). Formally, we have follow:

$$\text{if } M_a \neq M_b \text{ then, } \mathcal{D}(H_a, H_b) > T. \tag{3}$$

4. Methodology

To reduce security risks from high false positives in non-pirated model hashing and improve platform retrieval and model use efficiency, we propose a structured perceptual hashing method. As shown in Fig. 2, this method consists of two components: the parameter hashing module and the structure hashing module. When generating a hash for a model, we first separate its parameters and structure. The parameters are preprocessed and fed into the parameter hashing generator. And the structural information is first converted into a Directed Acyclic Graph (DAG) to facilitate feature learning. Then the DAGs are fed into the Model structure hash generator to generate the structure hash. In the end, structure hash and parameter hash are merged to generate the finally model hash.

Given the lack of similar datasets, we adopt a pre-training and fine-tuning strategy to learn effective representations from the structural DAG. Specifically, we begin by pre-training a graph classification neural network on the MUTAG dataset to obtain a generalizable pre-trained model. We then use 30% and 10% of the generated model DAGs for fine-tuning and validation, respectively, to create the final structure hash generator. Additionally, we modify the pre-trained graph classification neural network to meet the specific requirements of our fine-tuning tasks, ensuring the generator performs as expected.

4.1. Structure Hashing Module

DNNs exhibit diverse structures with varying numbers and types of layers. To facilitate the use of deep learning models in processing DNN structure information, we convert the diverse model structures into features of uniform size. In this study, we first transform the model structure into a graph structure suitable for DNN processing and then use a DNN-based graph embedding algorithm to convert the graph structure into features of uniform size. As shown in Fig. 2, our structural information extraction module consists of two parts: the DAG extraction module and the structural feature generation module.

DAG Extraction Module. In graph learning, neural networks often struggle to efficiently generate graphs for mainstream models, typically requiring space complexity of $\mathcal{O}(mn)$ or more (e.g., $\mathcal{O}(mnp)$ for convolutional layers, where p is the number of kernels, and m, n represent the input and output dimensions of each layer), which is impractical for our task. Therefore, we optimized the graph generation process based on the properties of neural network models. Specifically, we adjusted the generation methods for fully connected and convolutional layers, changing the original two-layer design to a three-layer structure and avoiding designs that could lead to cyclic graphs.

For fully connected layers, as shown in Fig. 2 (b), three groups of nodes are used: the top group corresponds to the input dimensions, the bottom group corresponds to the output dimensions, and the middle group corresponds the activation process between the input and output nodes. For convolutional layers, we also use three groups of nodes. However, unlike existing methods that represent the number of kernels with multiple edges, our intermediate layer contains nodes corresponding to each dimension of every kernel. For example, if there are two kernels, each of size 2×2, the middle group would contain $2 \times 2 \times 2 = 8$ graph nodes. This approach not only avoids multiple edges and simplifies the graph into a directed acyclic graph (DAG) but also reduces space complexity to $\mathcal{O}(p \times (m+n))$. We generated corresponding graphs for 497 models from the TIMM.

Structural Feature Generation Module. After obtaining the DAG from the model structure, we input it into a graph neural network to generate the final structural features. We use graph isomorphism network [29] as generator in our method. We first pre-trained a graph classification network on the MUTAG dataset. However, since the graphs in MUTAG datasets are relatively small, directly adapting the trained network to our task would result in poor performance. Therefore, we performed finetuning using a subset of DAG data from our task to align the model with the structure of the generated DAGs. During finetuning, we applied noise to the original graph G_o to generate a similar graph G_s and a dissimilar graph G_d. Specifically, we employ contrastive learning, defining two graphs as similar if the added nodes do not exceed 5% of the original nodes within the same layer, and their feature vectors are as close as possible. Conversely, graphs with more than 5% added nodes or new layers were considered dissimilar, and their feature vectors were expected to be as distant as possible. Ultimately, we obtained a 128-bit structure hash. Finally, we constructed a fusion layer composed of fully connected layers to merge the structure and parameter hashes into a 100-bit perceptual hash.

4.2. Parameter Hashing Module

In designing the parameter hashing module, we adopted the approach from mainstream practices. First, the parameters are preprocessed and divided into convolutional layer parameters and fully connected layer parameters, both of which are pruned to reduce redundancy. The pruned parameters are then input into the parameter hashing generator. Since we must handle parameters of various sizes and shapes, the first layer of the hashing generator employs a dynamic convolutional layer to map inputs of different dimensions into a unified space, ensuring effective processing by subsequent modules. Finally, the hashes from the convolutional and fully connected layers are merged through a fusion module to generate a 50-bit parameter hash. This approach reduces redundant computations through pruning, and the dynamic adjustment capabilities of the hashing generator ensure adaptability, providing a consistent hash representation for models of varying scales.

4.3. Loss Functions

The loss functions used in the experiment fall into three categories for various training stages: the parameter hash loss $\mathcal{L}_w$, the structure hash loss $\mathcal{L}_s$ and the merge hash loss $\mathcal{L}_m$:

$$\mathcal{L} = \begin{cases} \mathcal{L}_w, & \text{if train parameter hash generator;} \\ \mathcal{L}_s, & \text{if train structure hash generator;} \\ \mathcal{L}_m, & \text{if train merge hash.} \end{cases} . \tag{4}$$

In Eq. (4), $\mathcal{L}_w$ is aimed at ensuring that the hash codes generated from similar models' parameters are closely clustered, while those from dissimilar ones are more dispersed. $\mathcal{L}_w$ consists of two parts: $\mathcal{D}(H_p, H_o)$ and $\mathcal{D}(H_{np}, H_o)$:

$$\mathcal{L}_w = \mathcal{D}(H_p, H_o) - \mathcal{D}(H_{np}, H_o), \tag{5}$$

where H_o, H_p and H_{np} represent the hash codes of the original, pirated and non-pirated models' parameters, respectively.

$\mathcal{L}_s$ in Eq. (4) is used to train structure hash generator. We employed different loss functions for various training phases:

$$\mathcal{L}_s = \begin{cases} \mathcal{L}_p, & \text{when pre-training phase;} \\ \mathcal{L}_f, & \text{when fine-tuning phase.} \end{cases} , \tag{6}$$

where $\mathcal{L}_p$ represents mean squared error loss function:

$$\mathcal{L}_p = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2. \tag{7}$$

To make the graph neural network better suited to the structural characteristics of our data, the pre-trained model is fine-tuned using the loss function $\mathcal{L}_f$

$$\mathcal{L}_f = \mathcal{D}_e(H_s, H_o) - \mathcal{D}_e(H_d, H_o), \tag{8}$$

where H_o, H_s and H_d represent the hash of original graph, the similar graph, and the dissimilar graph, respectively, and $\mathcal{D}_e(\cdot,\cdot)$is used to calculate the distance between two hashes. The specific formula is as follows:

$$\mathcal{D}_e(H_a, H_b) = \frac{1}{1 + e^{-\mathcal{D}_o(H_a, H_b)}}, \tag{9}$$

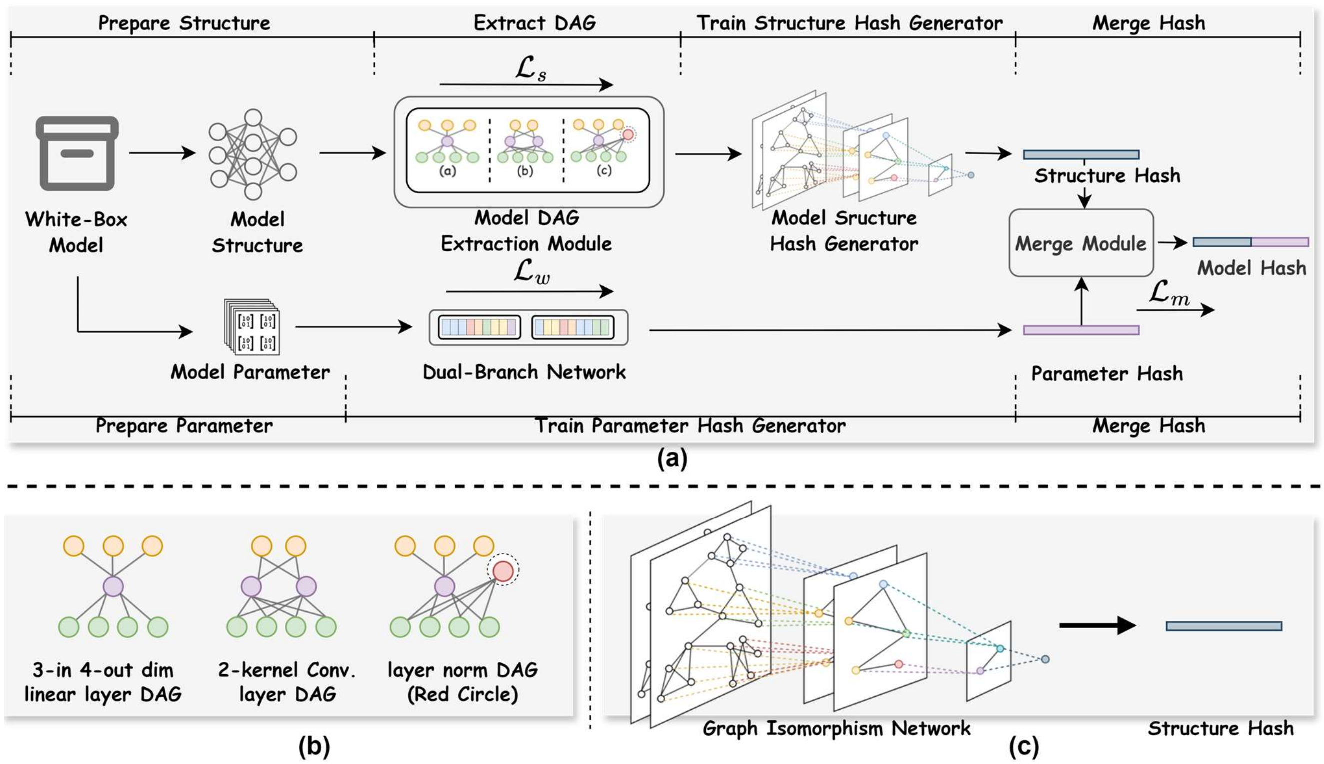

Figure 2. Network framework of our proposed method. In part (a), model structure and model parameter matrices are firstly extracted. Model parameter metrices are fed into dual-branch network to generate parameter hash. While the model structure is extracted to get the DAG. The DAG is then fed into well-trained GIN to generate structure. Finally, structure hash and parameter hash are merged to generate the model hash. In part (b), we illustrates our DAG extraction method. The left diagram represents a linear layer with three input dimensions and four output dimensions; the middle diagram depicts a convolutional layer with two 1×1 convolutional kernels, having two inputs dimensions and four outputs dimensions; and the right diagram shows the layer norm, indicated by the dashed circle. And in part (c), the left diagram shows the model structure hash generator which is graph isomorphism network with 3 graph convolutional layers.

where $\mathcal{D}(H_a, H_b)$ denotes the Euclidean distance between the graph embeddings H_a and H_b. Under this setup, the embedding distance for similar graphs should be as small as possible, while the distance for dissimilar graphs should be as large as possible. As the loss function $\mathcal{L}_f$ converges, $\mathcal{D}_e(H_s, H_o)$ approaches 0.5, indicating that the original and similar graphs are very close in embedding space, while $\mathcal{D}_e(H_d, H_o)$ approaches 1, showing that the original and dissimilar graphs are far apart. These outcomes ensure that the final loss $\mathcal{L}_f$ lies within the open interval $(-0.5, 0.5)$.

In the merge phase, the loss function integrates the parameter and structure hashes into the final perceptual hash. Thus, we expect the pirated model's perceptual hash to closely align with that of the original model, while that of the non-pirated model remains distant. The specific form of $\mathcal{L}_m$ is as follows:

$$\mathcal{L}_m = \mathcal{D}(H_p^T, H_o^T) - \mathcal{D}(H_{np}^T, H_o^T), \tag{10}$$

where $\mathcal{D}(\cdot,\cdot)$ is same as Eq. (1) and H_o^T, H_p^T and H_{np}^T denote the original, the pirated and the non-pirated model.

4.4. Training details

During the training stage, we first train the dual-branch network with $\mathcal{L}_w$ by $5,000$ epochs to get a well-trained parameter hash generator. Concurrently, the structure hash generator, based on graph neural network, undergoes initial training with the $\mathcal{L}_p$ to enhance its feature recognition capability. Subsequently, 30% of our dataset is allocated for fine-tuning, and the generator is refined using $\mathcal{L}_f$ over 200 epochs, yielding the final structure hash generator. The outputs of the dual-branch model are ultimately integrated with that of the structural branch, followed by joint fine-tuning of all three branches using $\mathcal{L}_m$.

5. Experiments

5.1. Experimental Setup

Model Dataset. We used ResNet (ResNet18, 34, 50, 101, 152) [9], VGG (VGG11, 13, 16, 19) [22], ShuffleNet [30], and MobileNetv2 [18] as the training set for the parameter identification module, and these mod-

Non-pirated model identification	Hash Distance				False Alarm Rate		Accuracy Rate	
	Max.	Mean.	Std.	Min.	T=0.1	T=0.5	T=0.1	T=0.5
Proposed	374.7232	48.8895	77.4429	0.08	2.69%	10.45%	97.31%	88.55%
Baseline	45.8825	11.3158	13.377	0	9.50%	17.46%	91.51%	82.54%

Table 1. Statistics of hash distances, false alarm rates and identification accuracy rates for non-pirated models.

			Hash Distance				Accuracy Rate			
			Max.	Mean.	Std.	Min.	T=0.1	T=0.5	T=1.0	T=2.0
Pirated Model Identification	Fine-tuning	Proposed	0.02	0.0015	0.0019	0	100.00%	100.00%	100.00%	100.00%
		Baseline	0.2109	0.0241	0.0283	0	96.88%	100.00%	100.00%	100.00%
	L1 pruning	Proposed	0.0131	0.0014	0.0019	0	100.00%	100.00%	100.00%	100.00%
		Baseline	0.2362	0.0244	0.0295	0	96.35%	100.00%	100.00%	100.00%
	Random pruning	Proposed	0.013	0.0015	0.0018	0	100.00%	100.00%	100.00%	100.00%
		Baseline	0.2031	0.0256	0.0289	0	95.50%	100.00%	100.00%	100.00%

Table 2. Statistics of hash distances and identification accuracy rates for pirated models.

els were trained on classical vision task datasets, including CIFAR-10 [12], CIFAR-100 [12], MNIST [14], and FashionMNIST [27].During fine-tuning, five models were saved per epoch (10 epochs total) to capture varying refinement stages. We evaluated generalization using 325 TIMM visual models as test data, resulting in a total of 2,569 models. Pruned/tampered models were dynamically generated during training. For structural hashing, MUTAG molecular graphs served as pretraining data (188 samples, avg. 17.9 nodes/19.8 edges), with performance enhancement achieved through fine-tuning on 30% (98) of our TIMM-derived graph structures - the remaining 70% (227) comprised the test set for large graph recognition validation.
Training Strategy. The model hashing generator was trained on 1,683 models (75% of 2,569 original and corresponding fine-tuned models) with real-time generation of pruned/tampered variants from originals. Using AdaMax optimizer (learning rate=0.0001) for 5,000 iterations, the generator processed randomly selected pirated models including fine-tuned, pruned, and tampered versions. For structural hashing, we employed MUTAG molecular graphs (188 samples, avg. 17.9 nodes/19.8 edges) and enhanced generalization by fine-tuning on 30% (98) of TIMM's 325 graph structures to enable scalable model hashing.
Environment. All experiments were run on Linux platform with four RTX 4090 GPUs, using the PyTorch [16].
Comparison with Baseline Methods. We compared our approach with existing DNN-based perceptual hashing methods (e.g., Xiong [28]) from two perspectives: pirated model detection and non-pirated model distinction. And the result of the experiment shows in Tabs. 1 and 2.

5.2. Non-pirated Model Distinction

As a key focus of this study, we evaluated the ability to distinguish non-pirated models. We used the same validation dataset $\mathcal{D}_t^2$ and randomly selected two models to generate hash values in each experiment. We performed 1,000 tests in total, and the results are summarized in Tab. 1. Our method can achieve a non-pirated detection false positive rate of only 2.69% at a T that guarantees a higher piracy detection accuracy, compared to 9.5% for the baseline method. According to the T in Tab. 2, if the piracy detection accuracy is to be taken into account, the baseline false alarm rate will be 17.46%. It is important to note that the identification accuracy for non-pirated models decreases as the threshold increases, which is different from the trend observed in pirated model detection. Thus, ideally, the distance between non-pirated models should be larger, while the distance between pirated models should be smaller. Therefore, as the threshold increases, both the maximum hash value recognized as a pirated model and the minimum hash value recognized as a non-pirated model increase, leading to an inverse trend in accuracy growth between the two tasks.

5.3. Robustness Test

5.3.1 Model Fine-tuning

Following Sec. 3, we conducted experiments on pirated model detection under fine-tuning, pruning, and tampering scenarios. For fine-tuning tests, we used a validation dataset $\mathcal{D}_t^1$ consisting of 25% of the original models and their fine-tuned versions, totaling $(2569 - 325) \times 0.25 = 561$ models. We generated two hash sequences for each original model and its fine-tuned version and calculated their hash distances H_p. The results for fine-tuning detection are summarized in Tab. 2 under the "Fine-tuning" row, where our method achieves 100% detection accuracy at the baseline's minimum threshold $T = 0.1$, with a maximum hash distance of only 0.02. As shown in Fig. 3, our method make better performance than baseline with different T.

		$\alpha = 1\%$					$\alpha = 20\%$					$\alpha = 50\%$				
	σ	0.01	0.05	0.1	0.15	0.2	0.01	0.05	0.1	0.15	0.2	0.01	0.05	0.1	0.15	0.2
D	Max.	0.04	0.00	0.02	0.04	0.04	0.13	0.46	0.73	0.47	0.93	0.20	0.68	0.54	0.75	0.58
	Mean.	0.00	0.00	0.00	0.00	0.00	0.01	0.06	0.09	0.10	0.10	0.02	0.07	0.09	0.10	0.11
	Std.	0.00	0.00	0.00	0.01	0.01	0.02	0.08	0.09	0.09	0.10	0.03	0.08	0.09	0.10	0.10
	Min.	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
R_p	T=0.1	100%	100%	100%	100%	100%	98.96%	81.46%	74.48%	69.04%	67.66%	95.41%	77.89%	71.30%	64.87%	60.06%
	T=0.5	100%	100%	100%	100%	100%	100%	100%	99.70%	96.20%	94.70%	100%	99.85%	93.78%	95.60%	92.94%
	T=1.0	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
	T=2.0	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%

Table 3. Statistics of hash distances and identification accuracy rates for pirated models under tampering operation. Here, α, σ, D, and R_p denote the tampering rate, Gaussian noise deviation, hash distance, and detection accuracy at threshold T, respectively.

5.3.2 Model Pruning

For pruning attack detection, we tested our hashing generator with both L1-norm pruning and random pruning. The validation dataset $\mathcal{D}_t^2$ consisted of $(11 \times 4) \times 0.25 + 325 = 336$ models. During the experiment, we randomly selected an original model and generated its pruned version in real-time, with the pruning rate pr randomly chosen from 0.1 to 0.9 with the step of 0.1. We performed 1,000 experiments in total and recorded the corresponding hash distances. As shown in Tab. 2, the maximum hash distances for L1-norm and random pruning were 0.0131 and 0.013, respectively, while the baseline method yielded maximum distances of 0.2362 and 0.2301. Additionally, our method achieved the same 100% detection accuracy as in the fine-tuning scenario, outperforming the baseline method, which reported only 96.3% and 96.8% accuracy for L1-norm and random pruning, respectively. Also shown in Fig. 3, performance of ours method is better.

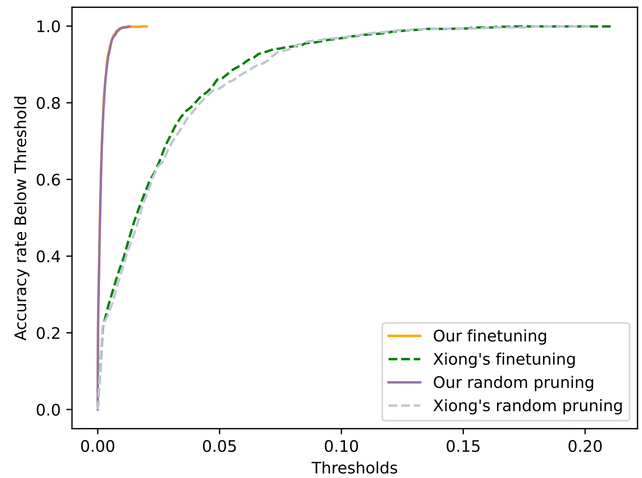

Figure 3. Recognition accuracy comparison under different threshold T. The figure respectively describes the accuracies of our scheme and the scheme of Xiong et al. under different thresholds in the scenarios of pruning and fine-tuning.

5.3.3 Model Tampering

According to the definition, tampered versions of the original models should be regarded as pirated models and identified as such. To verify the robustness of our method under different tampering configurations, we conducted tests with varying tampering rates α, set at 1%, 20%, and 50%. Additionally, the mean of the Gaussian noise ε was set to 0, while the standard deviation σ was assigned values of 0.01, 0.05, 0.10, 0.15, and 0.20. In each experiment, a model was randomly selected from Ω_2, and a tampered version was generated in real time by altering its parameters. Hash sequences were then generated for both the original and tampered models, and their hash distance S_p was calculated. A total of 700 tampering operations were performed for each combination of σ and α, resulting in $700 \times 3 \times 5 = 10,500$ hash distances and $3 \times 5 \times 4 = 60$ recognition accuracies.

As shown in Tab. 3, the maximum hash distance was only 0.93, with both the mean and standard deviation below 0.10, indicating that most hash distances were small, and nearly all tampered pirated models were successfully identified. When α and the threshold T were fixed, the recognition accuracy decreased as σ increased. This is because larger σ values amplify the differences between the original and tampered models, resulting in greater hash distances. When T was set to 0.5, 1.0, and 2.0, the recognition accuracy of pirated models consistently exceeded 90%.

5.4. Impact of Parameters on Hashing Performance

5.4.1 Performance with Different Component

Our ablation study compares: (1) direct concatenation vs. neural fusion (merge module) for hash integration, and (2) hard-coded computational graphs vs. graph learning networks for structural hash generation. Hard-coded structural hashes produce sparse vectors that amplify parametric hash dominance (e.g., 53.19% vs. 89.55% Non-pirated Model Identification at T=0.5), while graph learning resolves this imbalance. As shown in Tab. 5, our method achieves 100% Fine-tuning and L1 Random accuracy at T=0.1, outperforming baselines by 4.91–9.38% across key metrics.

			Hash Distance				Accuracy Rate			
			Max.	Mean.	Std.	Min.	T=0.1	T=0.5	T=1.0	T=2.0
Pirated Model Identification	Fine-tuning	50-bit	0.0189	0.0015	0.0019	0	100.00%	100.00%	100.00%	100.00%
		75-bit	0.0147	0.0011	0.0013	0	100.00%	100.00%	100.00%	100.00%
		100-bit	0.02	0.0015	0.0019	0	100.00%	100.00%	100.00%	100.00%
	Random pruning	50-bit	0.0578	0.0048	0.0047	0.0001	100.00%	100.00%	100.00%	100.00%
		75-bit	0.0455	0.0038	0.0037	0.0002	100.00%	100.00%	100.00%	100.00%
		100-bit	0.012	0.0015	0.0018	0	100.00%	100.00%	100.00%	100.00%
Non-pirated model identification		50-bit	417.5456	43.8611	70.4111	0.0119	95.16%	89.78%	82.80%	76.88%
		75-bit	284.7927	26.135	49.3706	0.0038	92.47%	82.26%	76.81%	63.44%
		100-bit	366.8084	37.505	63.7281	0.0181	95.48%	89.50%	81.01%	78.93%

Table 4. Performance under different length of model hash.

Accuracy Rate	Threshold	Fine tuning	L1 Random	Non-pirated Model Identification
Concatenation	T=0.1	95.09%	62.44%	89.13%
	T=0.5	**100.00%**	94.12%	66.85%
Hard Code	T=0.1	90.62%	46.41%	95.74%
	T=0.5	**100.00%**	98.64%	53.19%
Ours	T=0.1	**100.00%**	**100.00%**	**97.31%**
	T=0.5	**100.00%**	**100.00%**	**89.55%**

Table 5. Ablation study with concatenating and hard coding method.

5.4.2 Performance with Different Threshold

To demonstrate the efficiency and low false alarm rate of the proposed scheme, we conducted experiments on the recognition accuracy under the same threshold and plotted the accuracy curves for identifying different types of pirated models across various thresholds, as shown in Fig. 3. Our scheme achieves 100% recognition accuracy at a threshold of less than 0.05, whereas existing methods require a threshold of approximately 0.15 to achieve the same accuracy. Similarly, to further validate the low false positive rate of our proposed scheme, we performed experiments on the false positive rate under different thresholds.

5.4.3 Performance with Different Length of Merge Hash

To evaluate the impact of different fusion lengths H_w and H_s on hashing performance, we randomly selected 200 model pairs at lengths of 50, 75, and 100 bits across fine-tuning, random pruning, and dissimilar model scenarios. We assessed detection accuracy and false alarm rates using various thresholds, where smaller hash values are preferred in the fine-tuning and pruning contexts, and larger hash values are favored in dissimilar scenarios. A higher detection accuracy is desirable across all conditions. Tab. 4 summarizes the results for different hash lengths. Notably, a length of 100 bits demonstrated strong performance in both similar and dissimilar scenarios, leading us to adopt it as our final fusion hash length in the experiments.

	Length		
	32	64	128
Accuracy Rate	92.11%	89.47%	97.37%
$\mathcal{L}_{nll}$	0.3019	0.3556	0.1826

Table 6. Performance with different length of structure hash.

5.4.4 Different Length of Structure Hash

During the structure hash generator implementation, we tested hash lengths for optimal discrimination. Structural DAG detection tests with 32, 64, and 128 -bit outputs were conducted, results in Tab. 6, where $\mathcal{L}_{nll}$ denotes negative log - likelihood loss. Under the same training, 128 -bit hash showed better performance and accuracy. Thus, 128 bits was chosen for our experiments.

6. Conclusion and Limitation

We propose structure hashing, a model-perceptual method employing graph neural networks to extract structural fingerprints through directed acyclic graph (DAG) mapping. Our results demonstrate that structural integration with parameter hashing significantly improves piracy detection robustness (38% fewer false positives). This novel graph-based approach progresses through three stages: DAG conversion, graph feature learning, and multi-hash fusion. Future work will extend this framework to LLM by analyzing their massive parameter sequences as distributional patterns in parameter space, given their architectural complexity.

7. Acknowledgement

This work was supported in part by the National Natural Science Foundation of China under Grant U2336206, 62472398, 62121002, and U2436601.

References

[1] Sani M Abdullahi, Hongxia Wang, and Tao Li. Fractal coding-based robust and alignment-free fingerprint image hashing. *IEEE Transactions on Information Forensics and Security*, 15:2587–2601, 2020. 3

[2] Haozhe Chen, Hang Zhou, Jie Zhang, Dongdong Chen, Weiming Zhang, Kejiang Chen, Gang Hua, and Nenghai Yu. Perceptual hashing of deep convolutional neural networks for model copy detection. *ACM Transactions on Multimedia Computing, Communications and Applications*, 19(3):1–20, 2023. 1, 2, 3

[3] Thanh-Toan Do, Tuan Hoang, Dang-Khoa Le Tan, Huu Le, Tam V Nguyen, and Ngai-Man Cheung. From selective deep convolutional features to compact binary representations for image retrieval. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 15(2):1–22, 2019. 2

[4] Kaipeng Fang, Jingkuan Song, Lianli Gao, Pengpeng Zeng, Zhi-Qi Cheng, Xiyao Li, and Heng Tao Shen. Pros: Prompting-to-simulate generalized knowledge for universal cross-domain retrieval. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17292–17301, 2024. 1

[5] Bahare Fatemi, Jonathan Halcrow, and Bryan Perozzi. Talk like a graph: Encoding graphs for large language models. *arXiv preprint arXiv:2310.04560*, 2023. 1

[6] Guanhao Gan, Yiming Li, Dongxian Wu, and Shu-Tao Xia. Towards robust model watermark via reducing parametric vulnerability. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4751–4761, 2023. 1

[7] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017. 2

[8] Jiaqi Han, Yu Rong, Tingyang Xu, and Wenbing Huang. Geometrically equivariant graph neural networks: A survey. *arXiv preprint arXiv:2202.07230*, 2022. 2

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 5

[10] Jitesh Jain, Jianwei Yang, and Humphrey Shi. VCoder: Versatile Vision Encoders for Multimodal Large Language Models. In *CVPR*, 2024. 1

[11] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. 2

[12] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009. 6

[13] Rudolf Larenz. On overcoming the alleged alienation between christianity and physics. 2023. 1

[14] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010. 6

[15] Xiaoping Liang, Zhenjun Tang, Ziqing Huang, Xianquan Zhang, and Shichao Zhang. Efficient hashing method using 2d-2d pca for image copy detection. *IEEE Transactions on Knowledge and Data Engineering*, 35(4):3765–3778, 2021. 3

[16] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017. 6

[17] Yuhui Quan, Huan Teng, Ruotao Xu, Jun Huang, and Hui Ji. Fingerprinting deep image restoration models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 13285–13295, 2023. 1

[18] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018. 5

[19] Aliaksei Sandryhaila and José M. F. Moura. Discrete signal processing on graphs. *IEEE Transactions on Signal Processing*, 61(7):1644–1656, 2013. 2

[20] Yuming Shen, Li Liu, Fumin Shen, and Ling Shao. Zero-shot sketch-image hashing. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3598–3607, 2018. 3

[21] David I Shuman, Sunil K. Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 30(3):83–98, 2013. 2

[22] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 5

[23] Hae Jin Song, Mahyar Khayatkhoei, and Wael AbdAlmageed. Manifpt: Defining and analyzing fingerprints of generative models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10791–10801, 2024. 1

[24] Cheng-Hao Tu, Huei-Fang Yang, Shih-Min Yang, Mei-Chen Yeh, and Chu-Song Chen. Semantichash: Hash coding via semantics-guided label prototype learning. *IEEE Transactions on Artificial Intelligence*, 2(1):42–57, 2021. 2

[25] Rongyuan Wu, Tao Yang, Lingchen Sun, Zhengqiang Zhang, Shuai Li, and Lei Zhang. Seesr: Towards semantics-aware real-world image super-resolution. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 25456–25467, 2024. 1

[26] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020. 2

[27] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017. 6

[28] Cheng Xiong, Guorui Feng, Xinran Li, Xinpeng Zhang, and Chuan Qin. Neural network model protection with piracy identification and tampering localization capability. In *Proceedings of the 30th ACM International Conference on Multimedia*, pages 2881–2889, 2022. 1, 2, 3, 6

[29] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018. 2, 4

[30] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6848–6856, 2018. 5

[31] Chengfeng Zhao, Juze Zhang, Jiashen Du, Ziwei Shan, Junye Wang, Jingyi Yu, Jingya Wang, and Lan Xu. I'm hoi: Inertia-aware monocular capture of 3d human-object interactions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 729–741, 2024. 1

[32] Han Zhu, Mingsheng Long, Jianmin Wang, and Yue Cao. Deep hashing network for efficient similarity retrieval. In *Proceedings of the AAAI conference on Artificial Intelligence*, 2016. 2

[33] Peifei Zhu, Tsubasa Takahashi, and Hirokatsu Kataoka. Watermark-embedded adversarial examples for copyright protection against diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 24420–24430, 2024. 1