Hashed Watermark as a Filter: A Unified Defense Against Forging and Overwriting Attacks in Neural Network Watermarking

Yuan Yao[1], **Jin Song**[2], **Jian Jin**[3*]

[1] Beijing Teleinfo Technology Company Ltd., China Academy of Information and Communications Technology
[2] School of Computer Science, Nanjing University of Posts and Telecommunications
[3] Research Institute of Industrial Internet of Things, China Academy of Information and Communications Technology
yaoyuan.hitsz@gmail.com, jinsongresearch@outlook.com, jin.jian@caict.ac.cn

Abstract

As valuable digital assets, deep neural networks necessitate robust ownership protection, positioning neural network watermarking (NNW) as a promising solution. Among NNW approaches, weight-based methods embed watermarks directly into model parameters; however, they remain generally susceptible to forging and overwriting attacks. To address those challenges, we propose *NeuralMark*, a robust method built around a *hashed watermark filter*. Specifically, we utilize a hash function to generate an irreversible binary watermark from a secret key, which is then used as a filter to select the model parameters for embedding. This design cleverly intertwines the embedding parameters with the hashed watermark, providing a robust defense against both forging and overwriting attacks. Average pooling is also incorporated to resist fine-tuning and pruning attacks. Furthermore, it can be seamlessly integrated into various neural network architectures, ensuring broad applicability. We theoretically analyze its security boundary and highlight the necessity of using a hashed watermark as a filter. Empirically, we demonstrate its effectiveness and robustness across 13 distinct Convolutional and Transformer architectures, covering five image classification tasks and one text generation task.

Introduction

The advancements in artificial intelligence have led to the development of numerous deep neural networks, particularly large language models (Mann et al. 2020; Achiam et al. 2023; Bai et al. 2023; Dubey et al. 2024; Cao et al. 2024). Training such models requires substantial investments in human resources, computational power, and other resources, as exemplified by GPT-4, which costs around $40 million to train (Cottier et al. 2024). Neural networks can thus be regarded as valuable digital assets, making effective ownership protection essential. Motivated by this need, neural network watermarking (NNW) methods (Li, Wang, and Barni 2021; Lukas et al. 2022; Xue et al. 2021) have been proposed. They are generally categorized into three types: (i) White-box methods require access to the model's internal information (*e.g.*, parameters or activations) (Uchida et al. 2017; Liu, Weng, and Zhu 2021; Fan et al. 2021; Li et al. 2024); (ii) Black-box method require querying the model's input–output mapping (Zhang et al. 2021; Huang et al. 2023; An et al. 2025); and (iii) Box-free method require only the model outputs and are particularly suitable for image generative models (Zhang et al. 2021; Huang et al. 2023; An et al. 2025). All three categories have demonstrated significant progress in safeguarding model ownership (Sun et al. 2023; Ngo et al. 2025). Given the distinct challenges associated with each type, this work focuses on white-box NNW, leaving the investigation of other types for future work.

Existing white-box NNW methods can be broadly categorized into three sub-branches: (i) Weight-based methods (Uchida et al. 2017; Feng and Zhang 2020; Li, Tondi, and Barni 2021; Liu, Weng, and Zhu 2021; Li et al. 2024) embed watermarks into model parameters; (ii) Passport-based methods (Fan, Ng, and Chan 2019; Fan et al. 2021; Zhang et al. 2020; Liu et al. 2023) introduce passport layers to replace normalization layers for watermark embedding; and (iii) Activation-based methods (Rouhani, Chen, and Koushanfar 2019; Li et al. 2021; Lim et al. 2022) incorporate watermarks into the activation maps of intermediate layers (see Appendix A for a detailed discussion of related work). Among those methods, weight-based approaches embed watermarks directly into the model's parameters. This allows seamless integration into various network architectures without modifying the original structure (Uchida et al. 2017; Li, Wang, and Barni 2021), providing a direct and easily implementable mechanism for watermark embedding. Although several state-of-the-art weight-based methods (Feng and Zhang 2020; Li, Tondi, and Barni 2021; Liu, Weng, and Zhu 2021; Li et al. 2024) can effectively resist fine-tuning and pruning attacks, *they remain partially vulnerable to forging, overwriting, or both types of attacks*.

On the one hand, forging attacks attempt to fabricate counterfeit watermarks and infer the corresponding secret key through reverse engineering, by freezing the model parameters. In this scenario, the adversary could claim the model's ownership, resulting in ownership ambiguity. On the other hand, overwriting attacks aim to remove the original watermark by embedding a counterfeit one. In particular, adversaries can adaptively increase the embedding strength of their watermarks without being required to match the original watermark's embedding strength. In such cases, the original watermark may be removed while the adversary's watermark is embedded, leading to the invalidation of the

*Corresponding author.

model's ownership. This raises a question: "*How can we design a more robust and effective weighted-based method that defends against both forging and overwriting attacks?*"

To explore this question, we propose *NeuralMark*, a weighted-based method centered on a *hashed watermark filter*. Specifically, we use a hash function to generate an irreversible binary watermark from a secret key, which is then employed as a filter to select the model parameters for embedding. The *avalanche effect* of the hash function (Webster and Tavares 1985) ensures that slight input changes induce significant, unpredictable output variations, impeding gradient calculation and making reverse-engineering-based forging attacks infeasible. Moreover, using distinct hashed watermarks as private filters reduces parameter overlap, especially under repeated filtering, which increases the difficulty for adversaries to locate and manipulate the embedded parameters, thereby hindering overwriting attacks. As a result, the hashed watermark filter cleverly intertwines the embedding parameters with the hashed watermark, providing a robust defense against both forging and overwriting attacks. Furthermore, we also apply an average pooling mechanism to the filtered parameters due to its resilience against fine-tuning and pruning attacks. Upon obtaining the resulting parameters, the hashed watermark is embedded into those parameters using a lightweight embedding loss. During verification, the embedded watermark is extracted to verify model ownership.

The main contributions of this paper are threefold.

- We propose NeuralMark, a weight-based method designed to safeguard model ownership. Also, we provide a theoretical analysis of its security boundary.
- In NeuralMark, an elegant hashed watermark filter is developed to defend against both forging and overwriting attacks.
- Extensive experimental results across 13 distinct Convolutional and Transformer architectures, covering five image classification tasks and one text generation task, verify the effectiveness and robustness of NeuralMark.

Threat Model

In this section, we present the threat model considered in this work, detailing the adversary's capabilities and the corresponding success criteria.

Adversary Capabilities

We assume a *fully trusted* third-party verifier responsible for watermark verification. An adversary can illegally access a watermarked model, identify the watermark-containing layers, and obtain the original training datasets, *but is limited in computational resources*. This constraint is reasonable, as an attacker with sufficient computational resources could train a new model from scratch, making model theft unnecessary. As discussed above, this work focuses on forging and overwriting attacks, while also considering fine-tuning and pruning attacks. Those threat scenarios are detailed as follows. (1) Forging Attack: The adversary aims to generate a counterfeit secret key–watermark pair without modifying the model parameters. Specifically, the adversary first randomly forges a counterfeit watermark and then derives a corresponding secret key by optimizing it while keeping the model parameters frozen (Fan, Ng, and Chan 2019; Fan et al. 2021). (2) Overwriting Attack: The adversary attempts to embed a counterfeit watermark to overwrite the original one (Liu, Weng, and Zhu 2021). (3) Fine-tuning Attack: The adversary aims to fine-tune the model to remove the original watermark. (4) Pruning Attack: The adversary attempts to remove the original watermark by parameter pruning.

Attack Success Criteria

Building on insights from (Fan, Ng, and Chan 2019; Fan et al. 2021; Zhu et al. 2020; Li et al. 2022), a successful attack on a watermarked model typically requires the adversary to either (i) *forge a counterfeit watermark without altering the model parameters*, or (ii) *remove the original watermark through parameter modifications, all while preserving model performance*. If the adversary only embeds a counterfeit watermark without removing the original one, the resulting model contains both. In this case, the model owner can submit a version containing only the original watermark to an authoritative third-party for verification. In contrast, the adversary cannot provide a model with only the counterfeit watermark, as the original watermark remains intact. As a result, the adversary cannot convincingly claim ownership unless they train a new model embedded solely with their own watermark. This not only makes stealing the original model unnecessary but also incurs significant training costs. Accordingly, we define the success criteria for each type of attack as follows. (1) Success Criteria for Forging Attack: Forge a counterfeit watermark that passes verification without modifying the model parameters. (2) Success Criteria for Overwriting Attack: Remove the original watermark and embed a counterfeit one by modifying the model parameters, while maintaining model performance. (3) Success Criteria for Fine-tuning Attack: Remove the original watermark through fine-tuning, while maintaining model performance. (4) Success Criteria for Pruning Attack: Remove the original watermark through parameter pruning, while maintaining model performance.

Methodology

In this section, we present NeuralMark, a weight-based method designed to protect model ownership. The objective is to train a watermarked model $\mathbb{M}(\boldsymbol{\theta}^*)$ on a given training dataset $\mathcal{D}$ such that the model parameters $\boldsymbol{\theta}^*$ embed a binary watermark $\mathbf{b}$[1] while satisfying the following criteria: (i) the watermark imposes negligible impact on the model performance and remains difficult for adversaries to detect; and (ii) the embedded watermark exhibits robustness against the adversarial attacks defined in the Threat Model section.

Motivation

As aforementioned, most weight-based methods struggle to defend against both forging and overwriting attacks. On the

[1]Watermarks in this paper are binary vectors of 0s and 1s.

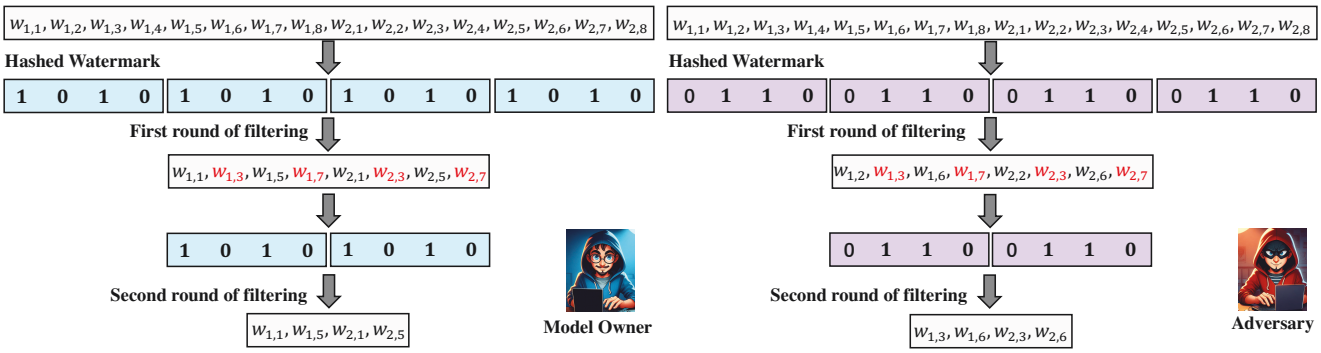


Figure 1: Illustration of the hashed watermark filter. The model owner's hashed watermark is $[1, 0, 1, 0]$, while the adversary's is $[0, 1, 1, 0]$. The watermark is repeated to match the parameter length before each round of filtering. Without filtering, all 16 parameters overlap. After the first round, each watermark retains eight parameters with four overlapping; after the second round, only four parameters remain for each, with no overlap.

one hand, forging attacks aim to generate a counterfeit watermark and derive the corresponding secret key via gradient backpropagation, while keeping the model parameters fixed. *Defending against such attacks requires disrupting gradient computation to hinder reverse-engineering*. On the other hand, overwriting attacks attempt to remove the original watermark by embedding a counterfeit one. Once watermarked parameters are identified, the adversary can overwrite the original watermark. Since each watermark updates the model parameters in a distinct and often conflicting direction, embedding a new watermark can easily disrupt the original one. *Defending against such attacks is essential to preserving the confidentiality of watermarked parameters and ensuring distinct parameter usage between the model owner and the adversary*.

To address both attacks, we propose a *hashed watermark filter*, which uses an irreversible watermark generated from a secret key via a hash function as a private filter, restricting watermark embedding to a secret parameter subset. This design provides two key properties:

- **Gradient Obfuscation**: The avalanche effect of the hash function ensures that even minor input changes lead to large, unpredictable output variants, effectively impeding gradient computation and rendering reverse-engineering-based forging attacks infeasible.
- **Embedding Isolation**: Since the hashed watermarks of the model owner and the adversary are inherently distinct, using them as private filters can effectively reduce the overlap in selected parameters, especially when the filtering process is performed repeatedly. As exemplified in Figure 1, the model owner's hashed watermark is $[1, 0, 1, 0]$, while the adversary's is $[0, 1, 1, 0]$. Without filtering, all 16 model parameters are shared, yielding a 100% overlap ratio. After the first round of filtering, each party retains eight parameters, with four overlapping, reducing the overlap to 50%. A second filtering round results in four parameters per party, with zero overlap, achieving a 0% overlap ratio. This progressive isolation ensures that as filtering continues, the overlap between the model owner's and the adversary's selected parameters is significantly reduced. Thus, it becomes increasingly difficult for the adversary to identify and manipulate the owner's watermarked parameters, even when increasing the embedding strength of their watermarks, thereby preserving the integrity of the original watermark against overwriting attacks.

In summary, these properties allow the hashed watermark filter to tightly entangle the embedding parameters with the hashed watermark, which is essential for resisting both forging and overwriting attacks (see the Security Analysis section for details). This mechanism forms the core of NeuralMark, which we will elaborate on next.

NeuralMark

NeuralMark consists of three primary steps: (i) hashed watermark generation; (ii) watermark embedding; and (iii) watermark verification. Figure 5 in Appendix C illustrates the workflow of each step. We now elaborate on each step.

Hashed Watermark Generation As aforementioned, we construct a hash-based mapping from a secret key to a binary watermark. Formally, the watermark $\mathbf{b} \in \{0, 1\}^n$ is generated by $\mathbf{b} = \mathcal{H}(\mathbf{K})$, where $\mathbf{K} \in \mathbb{R}^{k \times n}$ is a secret key matrix with elements drawn from a random distribution (*e.g.*, standard Gaussian distribution), $\mathcal{H}(\cdot)$ denotes a hash function, and n indicates the length of the watermark. To accommodate various application requirements, we adopt SHAKE-256 (Dworkin 2015), an extendable-output function from the SHA-3 family that allows dynamic adjustment of output length. Furthermore, auxiliary content $\mathcal{C}$ (*e.g.*, textual descriptors or unique identifiers) can also be incorporated into the hash function, yielding $\mathbf{b} = \mathcal{H}(\mathbf{K}||\mathcal{C})$, where $||$ denotes the concatenation operation. This mechanism enables context-aware watermark generation without compromising the avalanche effect of the hash function. For simplicity, we omit auxiliary content in the experiments.

Watermark Embedding To embed the hashed watermark $\mathbf{b}$ into the model $\mathbb{M}(\boldsymbol{\theta})$, we first select and flatten a subset of parameters (*e.g.*, one-layer parameters) from $\boldsymbol{\theta}$ into a parameter vector $\mathbf{w} \in \mathbb{R}^m$. Then, we utilize the hashed

watermark filter to select the model parameters for embedding. Specifically, let $\mathbf{w}^{(0)} = \mathbf{w}$ be the initial parameter vector. In the r-th ($r \in \{1, \cdots, R\}$) filtering round, the watermark $\mathbf{b}$ is repeated to match the length of $\mathbf{w}^{(r-1)}$, forming $\mathbf{b}^{(r)}$, with any excess parameters in $\mathbf{w}^{(r-1)}$ discarded. The parameter vector $\mathbf{w}^{(r)}$ is constructed by selecting the elements from $\mathbf{w}^{(r-1)}$ at positions where $\mathbf{b}^{(r)}$ equals one, *i.e.*, $\mathbf{w}^{(r)} = \left[w_i^{(r-1)} \mid i \in \{j \mid b_j^{(r)} = 1\}\right]$, where $w_i^{(r-1)}$ is the i-th element of $\mathbf{w}^{(r-1)}$, and $b_j^{(r)}$ is the j-th element of $\mathbf{b}^{(r)}$. After completing the whole watermark filtering process, the filtered parameter vector $\mathbf{w}^{(R)}$ is obtained. Next, we adopt the average pooling AVG$(\cdot)$ operation (Gholamalinezhad and Khosravi 2020) to calculate the final parameters as $\widetilde{\mathbf{w}} = \text{AVG}(\mathbf{w}^{(R)}) \in \mathbb{R}^k$. This operation aggregates parameters across broader regions, thereby enhancing robustness against parameter perturbations caused by fine-tuning and pruning attacks. Finally, we formulate the overall optimal objective as

$$\min_{\theta} \mathcal{L}_m + \lambda \mathcal{L}_e(\widetilde{\mathbf{b}}, \mathbf{b}), \tag{1}$$

where $\mathcal{L}_m$ denotes the main task loss (*e.g.*, classification loss), $\mathcal{L}_e(\cdot, \cdot)$ represents the binary cross-entropy loss, $\widetilde{\mathbf{b}} = \delta(\widetilde{\mathbf{w}}\mathbf{K})$ denotes the extracted watermark, with $\delta(\cdot)$ being the sigmoid function, and λ is a positive trade-off hyper-parameter. By minimizing Eq. (1), the watermark can be embedded into model parameters during the main task training. The watermark embedding process is summarized in Algorithm 1 in Appendix D.

Watermark Verification The watermark verification process is similar to the embedding process. Concretely, upon identifying a potentially unauthorized model, the relevant subset of model parameters is extracted and subjected to hashed watermark filtering and average pooling to derive an extracted watermark $\widetilde{\mathbf{b}}$. This extracted watermark is then compared to the model owner's watermark $\mathbf{b}$ using the *watermark detection rate*, which is defined by

$$\rho = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}\left[b_i = \mathcal{T}(\widetilde{b}_i)\right], \tag{2}$$

where $\mathcal{T}(x)$ is a threshold function that outputs 1 if $x > 0.5$ and 0 otherwise, and $\mathbf{1}(\psi)$ is an indicator function that returns 1 if ψ is true and 0 otherwise. The unauthorized model is confirmed to belong to the model owner if both of the following conditions are satisfied: **(1)** The watermark detection rate ρ exceeds a theoretical security boundary ρ^*, which will be theoretically analyzed later. **(2)** The watermark must correspond to the output of the hash function applied to the secret key, ensuring cryptographic consistency with the predefined hash function. The watermark verification process is outlined in Algorithm 2 in Appendix D.

Security Analysis

Security Boundary Analysis We present a theoretical analysis to determine the security boundary of NeuralMark in Proposition 1.

Propositions 1 *Under the assumption that the hash function produces uniformly distributed outputs (Bellare and Rogaway 1993), for a model watermarked by NeuralMark with a watermark tuple* $\{\mathbf{K}, \mathbf{b}\}$*, where* $\mathbf{b} = \mathcal{H}(\mathbf{K})$*, if an adversary attempts to forge a counterfeit watermark tuple* $\{\mathbf{K}', \mathbf{b}'\}$ *such that* $\mathbf{b}' = \mathcal{H}(\mathbf{K}')$ *and* $\mathbf{K}' \neq \mathbf{K}$*, then the probability of achieving a watermark detection rate of at least* ρ *(i.e.,* $\geq \rho$*) is upper-bounded by* $\frac{1}{2^n} \sum_{i=0}^{n-\lceil \rho n \rceil} \binom{n}{i}$.

The proof of Proposition 1 is provided in Appendix B. Proposition 1 provides a theoretical benchmark for establishing the security boundary of the watermark detection rate. Specifically, with $n = 256$, if the watermark detection rate $\rho \geq 88.29\%$, the probability of this occurring by forgery is less than $1/2^{128}$. This negligible probability allows us to confirm ownership with high confidence. *Thus, we set* $n = 256$ *and use* 88.29% *as the security bound for the watermark detection rate in the experiments*.

Necessity of Hashed Watermark Filter We analyze the necessity of the hashed watermark filter by comparing it to a baseline mechanism that employs a private filter rather than a hashed watermark. While this mechanism offers resistance to overwriting attacks, it remains vulnerable to forging attacks. For example, an adversary can use a 256×256 *identity matrix* as a secret key $\mathbf{K}$ to generate a hashed watermark $\mathbf{b}$. By selecting embedding parameters $\widehat{\mathbf{w}}$ whose signs correspond to $\mathbf{b}$ (with 0 representing a negative value and 1 a positive value), the adversary can derive a private filter that selects those parameters accordingly. This allows bypassing watermark verification, *i.e.*, $\mathcal{T}(\delta(\widehat{\mathbf{w}}\mathbf{K})) = \mathbf{b}$ and $\mathcal{H}(\mathbf{K}) = \mathbf{b}$. In contrast, the hashed watermark filter cleverly intertwines the embedding parameters with the hashed watermark, rendering it essential for defending against both forging and overwriting attacks.

Experiments

In this section, we evaluate the proposed NeuralMark.

Experimental Setup

Datasets and Architectures We use five image classification datasets: CIFAR-10 (Krizhevsky, Hinton et al. 2009), CIFAR-100 (Krizhevsky, Hinton et al. 2009), Caltech-101 (Fei-Fei, Fergus, and Perona 2004), Caltech-256 (Griffin et al. 2007), and TinyImageNet (Le and Yang 2015), as well as one text generation dataset, E2E (Novikova, Dušek, and Rieser 2017). Additionally, we utilize 11 image classification architectures, including eight Convolutional architectures: AlexNet (Krizhevsky, Sutskever, and Hinton 2012), VGG-13, VGG-16 (Simonyan and Zisserman 2015), GoogLeNet (Szegedy et al. 2015), ResNet-18, ResNet-34 (He et al. 2016), WideResNet-50 (Zagoruyko 2016), and MobileNet-V3-L (Howard et al. 2019), as well as three Transformer architectures: ViT-B/16 (Dosovitskiy 2021), Swin-V2-B, and Swin-V2-S (Liu et al. 2022). Furthermore, we adopt two text generation architectures: GPT-2-S and GPT-2-M (Radford et al. 2019).

Baselines and Metrics We compare NeuralMark with three popular weight-based methods presented in (Uchida

Dataset	Clean		NeuralMark		VanillaMark		GreedyMark		VoteMark	
	AlexNet	ResNet-18	AlexNet	ResNet-18	AlexNet	ResNet-18	AlexNet	ResNet-18	AlexNet	ResNet-18
CIFAR-10	91.05	94.76	90.93	94.50	91.01	94.87	90.88	94.69	90.86	94.79
CIFAR-100	68.24	76.23	68.57	76.34	68.43	76.22	68.31	76.14	68.53	76.74
Caltech-101	68.07	68.83	68.38	68.47	68.54	68.99	68.59	69.08	68.88	67.91
Caltech-256	44.27	54.09	44.55	53.71	44.73	53.47	44.64	53.28	44.43	54.71
TinyImageNet	42.42	53.48	42.31	53.22	42.50	53.36	42.94	53.31	42.50	53.47

Table 1: Comparison of classification accuracy (%) across distinct datasets using AlexNet and ResNet-18. Watermark detection rates are omitted as they all reach 100%.

Method	ViT-B/16	Swin-V2-B	Swin-V2-S	VGG-16	VGG-13	ResNet-34	WideResNet-50	GoogLeNet	MobileNet-V3-L
Clean	39.07	52.99	55.88	72.75	72.71	77.06	59.67	60.71	61.11
NeuralMark	39.22	53.57	55.87	72.61	71.49	77.03	58.41	60.02	61.8

Table 2: Comparison of classification accuracy (%) on CIFAR-100 using various architectures. Watermark detection rates are omitted as they all reach 100%.

GPT-2-S	BLEU	NIST	MET	ROUGE-L	CIDEr	GPT-2-M	BLEU	NIST	MET	ROUGE-L	CIDEr
Clean	69.36	8.76	46.06	70.85	2.48	Clean	68.7	8.69	46.38	71.19	2.5
NeuralMark	69.59	8.79	46.01	70.85	2.48	NeuralMark	67.73	8.57	46.07	70.66	2.47

Table 3: Comparison on E2E using GPT-2-S and GPT-2-M. Watermark detection rates are omitted as they all reach 100%.

et al. 2017), (Liu, Weng, and Zhu 2021), and (Li et al. 2024), referred to as **VanillaMark**, **GreedyMark**, and **VoteMark**, respectively (see the Related Work section in Appendix A for details). Additionally, we include a comparison with a method that does not involve watermark embedding, referred to as **Clean**. For the image classification task, we assess model performance using classification accuracy, while the watermark embedding task is evaluated based on the watermark detection rate. As for the text generation task, we follow (Hu et al. 2022) and evaluate model performance using BLEU, NIST, MET, ROUGE-L, and CIDEr metrics, with the watermark embedding task assessed based on the watermark detection rate. More experimental details are provided in Appendix E.

Fidelity Evaluation

Diverse Datasets First, we evaluate the influence of watermark embedding on the model performance across diverse datasets. Table 1 reports the results across five image datasets using AlexNet and ResNet-18. We observe that all methods have minimal impact on model performance while successfully embedding watermarks, indicating that NeuralMark and other methods maintain model performance across diverse datasets during watermark embedding.

Various Architectures Next, we assess the impact of NeuralMark on model performance across various architectures. **??** lists the results of NeuralMark on the CIFAR-100 dataset using ViT-B/16, Swin-V2-B, Swin-V2-S, VGG-16, VGG-13, ResNet-34, WideResNet-50, GoogLeNet, and MobileNet-V3-L. We find that NeuralMark maintains a 100% watermark detection rate across a wide range of architectures while exerting minimal impact on model performance. Those observations indicate that NeuralMark exhibits a good level of generalizability across architectures.

Dataset	NeuralMark	VanillaMark	GreedyMark	VoteMark
CIFAR-10	48.56	100.00	50.70	100.00
CIFAR-100	49.41	100.00	52.85	100.00

Table 4: Comparison of detection rate (%) of counterfeit watermarks using ResNet-18.

Text Generation Tasks Finally, we evaluate the effect of NeuralMark on the text generation tasks. **??** presents the results of NeuralMark applied to the GPT-2-S and GPT-2-M architectures on the E2E dataset. We can observe that NeuralMark achieves a 100% watermark detection rate while maintaining nearly lossless model performance. Those results demonstrate NeuralMark's potential and generality in ownership protection of text generative models.

Robustness Evaluation

Forging Attacks We follow the setting described in the Threat Model section to evaluate the robustness of NeuralMark against forging attacks. Specifically, for VanillaMark and VoteMark, we randomly generate a counterfeit watermark and then attempt to learn the corresponding secret key while keeping the model parameters fixed. As for GreedyMark and NeuralMark, we directly verify 10 randomly forged watermarks using the watermarked model because GreedyMark does not require a secret key, and NeuralMark benefits from the avalanche effect of the hash function and the tight coupling between the embedding parameters and the hashed watermark, making reverse-engineering in-

Overwriting	λ	NeuralMark	VanillaMark	GreedyMark	VoteMark	η	NeuralMark	VanillaMark	GreedyMark	VoteMark
CIFAR-100 to CIFAR-10	1	93.65 (**100**)	93.30 (**100**)	93.45 (**48.82**)	93.63 (**100**)	0.001	93.65 (**100**)	93.30 (**100**)	93.45 (**48.82**)	93.63 (**100**)
	10	93.44 (**100**)	93.58 (**100**)	93.29 (**51.17**)	93.13 (**100**)	0.005	91.76 (**99.60**)	92.17 (**73.04**)	92.13 (**50.00**)	92.45 (**78.90**)
	50	93.46 (**100**)	93.50 (**100**)	93.07 (**55.07**)	93.39 (**100**)	0.01	91.58 (**92.18**)	91.79 (**62.10**)	91.53 (**49.60**)	91.76 (**60.15**)
	100	93.53 (**100**)	92.95 (**94.53**)	93.18 (**54.29**)	93.53 (**96.48**)	0.1	75.2 (**50.78**)	79.68 (**47.26**)	72.42 (**53.12**)	70.92 (**54.29**)
	1000	93.09 (**100**)	92.89 (**53.90**)	92.85 (**49.60**)	92.77 (**59.37**)	1	10.00 (**44.53**)	10.00 (**53.51**)	10.00 (**48.04**)	10.00 (**53.51**)

Table 5: Comparison of resistance to overwriting attacks at various trade-off hyper-parameters (λ) and learning rates (η) using ResNet-18. Values (%) inside and outside the bracket are the watermark detection rate and classification accuracy, respectively. Adversary watermarks, which are consistently detected at 100%, are omitted.

Fine-tuning	Clean		NeuralMark		VanillaMark		GreedyMark		VoteMark	
	AlexNet	ResNet-18	AlexNet	ResNet-18	AlexNet	ResNet-18	AlexNet	ResNet-18	AlexNet	ResNet-18
CIFAR-100 to CIFAR-10	85.55	89.15	85.35(**100**)	88.83(**100**)	85.48(**91.01**)	89.35(**85.93**)	80.41(**96.48**)	76.15(**94.14**)	84.97(**89.06**)	89.66(**85.54**)
CIFAR-10 to CIFAR-100	58.96	49.74	58.50(**100**)	49.77(**100**)	58.75(**74.21**)	49.97(**70.31**)	51.75(**97.65**)	19.94(**82.42**)	58.81(**80.07**)	49.08(**71.87**)
Caltech-256 to Caltech-101	47.65	74.09	71.29(**100**)	73.12(**100**)	71.56(**100**)	74.03(**100**)	72.04(**100**)	68.45(**100**)	71.62(**100**)	72.47(**99.60**)
Caltech-101 to Caltech-256	40.61	40.00	40.34(**100**)	40.34(**100**)	40.71(**96.09**)	39.04(**93.36**)	40.68(**100**)	36.45(**98.82**)	39.52(**95.31**)	39.73(**93.75**)

Table 6: Comparison of resistance to fine-tuning attacks using ResNet-18. Values (%) inside and outside the bracket are the watermark detection rate and classification accuracy, respectively.

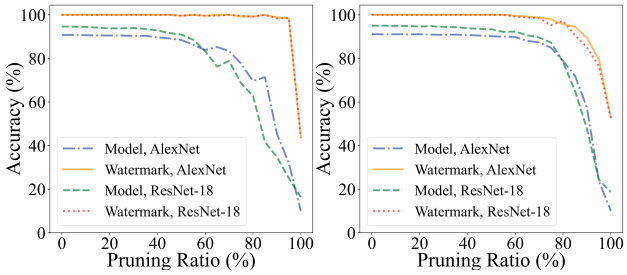
feasible. Table 4 presents the detection rates of counterfeit watermarks, from which we draw the following observations. (1) For VanillaMark and VoteMark, a pair of counterfeited secret key and watermark can be successfully learned through reverse-engineering, indicating their vulnerability to forging attacks. (2) NeuralMark and GreedyMark demonstrate robust resistance against forging attacks, which aligns with our expectations.

Overwriting Attacks We conduct overwriting attacks targeting the watermark embedding layers, with the number of training epochs fixed at 100 to reflect limited computational resources. The optimization is guided by the loss function $\mathcal{L}_m + \lambda\mathcal{L}_e(\widetilde{\mathbf{b}}, \mathbf{b}_a)$, where $\mathbf{b}_a$ denotes the adversary's watermark. Also, we analyze the effects of two key factors: the hyperparameter λ and the learning rate η. Here, λ controls the strength of the watermark embedding, with larger values leading to stronger embedding, while η primarily affects model performance.

Distinct Values of $\boldsymbol{\lambda}$. We investigate the influence of λ in overwriting attacks. Specifically, we set λ to 1, 10, 50, 100, and 1000, respectively. Table 5 presents the results on the CIFAR-100 to CIFAR-10 task using ResNet-18. We report only the original watermark detection rate, as the adversary's watermark detection rate reaches 100%. As defined in the success criterion in the Threat Model section, the original watermark must be effectively removed for overwriting attacks to be deemed successful. Thus, the overwriting attack experiments focus solely on whether the original watermark can be successfully removed. We can summarize several insightful observations. **(1)** As λ increases, the original watermark detection rate of NeuralMark remains at 100%, while those of VanillaMark, GreedyMark, and VoteMark significantly decline. In particular, when $\lambda = 1000$, the embedding strength of the adversary's watermark is 1000 times greater than that of the original watermark. At this point, the original watermark detection rates for NeuralMark, VanillaMark, GreedyMark, and VoteMark on the CIFAR-100 to CIFAR-10 task are 100%, 53.90%, 49.60%, and 59.37%, respectively. Those results indicate that NeuralMark exhibits strong robustness against overwriting attacks. **(2)** As λ increases, model performance remains relatively stable. This is because overwriting attacks jointly train both the main task and the watermark embedding task, enabling the model parameters to effectively adapt to both. More results are offered in Appendix F.1.

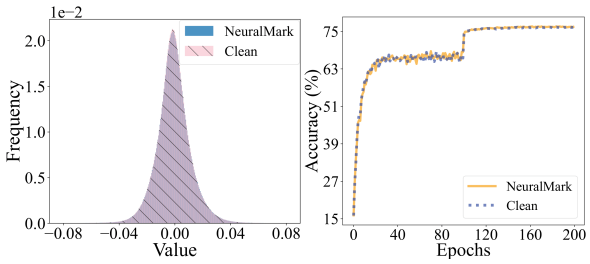
Distinct Values of $\boldsymbol{\eta}$. We examine the impact of η in overwriting attacks. Concretely, we set η to 0.001, 0.005, 0.01, 0.1, and 1, respectively. Table 5 lists the results on the CIFAR-100 to CIFAR-10 task using ResNet-18. We have several important observations. **(1)** Larger η values hurt model performance, implying that the adversary cannot arbitrarily increase the attack strength. **(2)** At $\eta = 0.005$, the original watermark detection rates for VanillaMark, GreedyMark, and VoteMark drop sharply, whereas NeuralMark maintains a detection rate close to 100%. **(3)** When $\eta = 0.01$, the model performance of NeuralMark on the CIFAR-100 to CIFAR-10 task decreases by 2.07%, but its original watermark detection rate remains above the security boundary of 88.29% defined in the Security Boundary Analysis section, while those for the other methods fall significantly. **(4)** For $\eta >= 0.1$, although the original watermark detection rate of NeuralMark drops below the security boundary, the model performance is completely compromised, indicating that the attack is ineffective. More results are provided in Appendix F.1.

Fine-tuning Attacks We perform fine-tuning attacks on the watermark embedding layers. During the attack, the task-specific classifier is first replaced with randomly initialized parameters, after which only the parameters of the watermark embedding layers and the classifier are updated, while all other parameters remain frozen. The optimization is guided solely by the main task loss $\mathcal{L}_m$. Following (Liu, Weng, and Zhu 2021), we adopt the same hyperparameters for fine-tuning attacks as during training, except



(a) NeuralMark (b) VanillaMark

Figure 2: Comparison of resistance to pruning attacks under various pruning ratios on CIFAR-10 using AlexNet and ResNet-18.



(a) Distribution (b) Convergence

Figure 3: Parameter distribution and performance convergence on the CIFAR-100 dataset using ResNet-18.

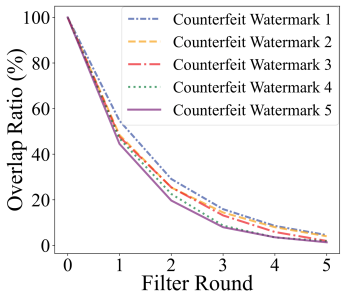


Figure 4: Comparison of parameter overlap ratio with different filter rounds on CIFAR-100 using ResNet-18.

for setting the learning rate to 0.001. As shown in Table 6, we find that watermarks embedded with NeuralMark maintain a 100% watermark detection rate across all fine-tuning tasks. In contrast, watermarks embedded with VanillaMark, GreedyMark, and VoteMark experience a slight reduction in detection rates across several tasks. Those results indicate that fine-tuning attacks cannot effectively remove watermarks embedded with NeuralMark. Furthermore, we conduct a fine-tuning attack by updating all model parameters, as detailed in Appendix F.2.

Pruning Attacks We evaluate the robustness of NeuralMark against pruning attacks by randomly resetting a specified proportion of parameters in the watermark embedding layer to zero. Figure 2 presents the results of NeuralMark and VanillaMark on the CIFAR-10 dataset using AlexNet and ResNet-18, respectively. As the pruning ratio increases, NeuralMark's performance degrades slightly, while the detection rate remains nearly 100%, indicating a good level of robustness. Additional results for all baselines across different datasets are provided in Appendix F.3.

Analysis

Parameter Distribution Figure 3a shows the parameter distributions learned by Clean and NeuralMark on the CIFAR-100 dataset using ResNet-18. As observed, their distributions are nearly indistinguishable, making it difficult for adversaries to detect the embedded watermarks. Additional results across various architectures are provided in Appendix F.4.

Performance Convergence Figure 3b shows the performance convergence of Clean and NeuralMark on the CIFAR-100 dataset using ResNet-18. The two curves follow a similar trajectory and remain closely aligned, indicating that NeuralMark does not hinder model convergence. Additional results across various architectures are provided in Appendix F.5.

Filtering Rounds To analyze watermark filtering efficacy, we generate five counterfeit watermarks and calculate the overlap ratio between parameters filtered with those and the original watermark. As shown in Figure 4, the overlap rate decreases towards zero with more filtering rounds, indicating that watermark filtering enhances the secrecy of the watermarked parameters. Furthermore, Appendix G presents additional experiments with 6 and 8 filtering rounds to evaluate their impact on NeuralMark's effectiveness robustness against various attacks, compared to the default setting of 4. The results show that the number of filtering rounds has a negligible effect on robustness.

Additional Analyses The impact of the watermark embedding layers and watermark length on model performance, as well as the training efficiency, is analyzed in Appendices F.6–F.8, respectively. Those results demonstrate the flexibility, effectiveness, and efficiency of NeuralMark.

Conclusion

In this paper, we present NeuralMark, a white-box method designed to protect model ownership. At the core of NeuralMark is a hashed watermark filter, which utilizes a hash function to generate an irreversible binary watermark from a secret key, subsequently employing this watermark as a filter to select model parameters for embedding. We provide a theoretical analysis of its security boundary and highlight the necessity of employing a hashed watermark as a filter. Extensive experiments on various datasets, architectures, and tasks confirm NeuralMark's effectiveness and robustness. In future work, we plan to investigate how the proposed hashed watermark filter can be incorporated with existing watermarking approaches to offer complementary protection against broader attack scenarios.

References

Achiam, J.; Adler, S.; Agarwal, S.; Ahmad, L.; Akkaya, I.; Aleman, F. L.; Almeida, D.; Altenschmidt, J.; Altman, S.; Anadkat, S.; et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

An, H.; Hua, G.; Fang, Z.; Xu, G.; Rahardja, S.; and Fang, Y. 2025. Decoder Gradient Shield: Provable and High-Fidelity Prevention of Gradient-Based Box-Free Watermark Removal. In *CVPR*, 13424–13433.

Bai, J.; Bai, S.; Chu, Y.; Cui, Z.; Dang, K.; Deng, X.; Fan, Y.; Ge, W.; Han, Y.; Huang, F.; et al. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Bellare, M.; and Rogaway, P. 1993. Random oracles are practical: A paradigm for designing efficient protocols. In *CCS*, 62–73.

Cao, Y.; Zhao, H.; Cheng, Y.; Shu, T.; Chen, Y.; Liu, G.; Liang, G.; Zhao, J.; Yan, J.; and Li, Y. 2024. Survey on large language model-enhanced reinforcement learning: Concept, taxonomy, and methods. *IEEE Transactions on Neural Networks and Learning Systems*.

Cottier, B.; Rahman, R.; Fattorini, L.; Maslej, N.; and Owen, D. 2024. The rising costs of training frontier AI models. *arXiv preprint arXiv:2405.21015*.

Dosovitskiy, A. 2021. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*.

Dubey, A.; Jauhri, A.; Pandey, A.; Kadian, A.; Al-Dahle, A.; Letman, A.; Mathur, A.; Schelten, A.; Yang, A.; Fan, A.; et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Dworkin, M. J. 2015. SHA-3 standard: Permutation-based hash and extendable-output functions.

Fan, L.; Ng, K. W.; and Chan, C. S. 2019. Rethinking deep neural network ownership verification: Embedding passports to defeat ambiguity attacks. In *NeurIPS*, volume 32.

Fan, L.; Ng, K. W.; Chan, C. S.; and Yang, Q. 2021. Deepipr: Deep neural network ownership verification with passports. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(10): 6122–6139.

Fei-Fei, L.; Fergus, R.; and Perona, P. 2004. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. In *CVPRW*, 178–178.

Feng, L.; and Zhang, X. 2020. Watermarking neural network with compensation mechanism. In *KSEM*, 363–375.

Gholamalinezhad, H.; and Khosravi, H. 2020. Pooling methods in deep neural networks, a review. *arXiv preprint arXiv:2009.07485*.

Griffin, G.; Holub, A.; Perona, P.; et al. 2007. Caltech-256 object category dataset. Technical report, Technical Report 7694, California Institute of Technology Pasadena.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *CVPR*, 770–778.

Howard, A.; Sandler, M.; Chu, G.; Chen, L.-C.; Chen, B.; Tan, M.; Wang, W.; Zhu, Y.; Pang, R.; Vasudevan, V.; et al. 2019. Searching for mobilenetv3. In *ICCV*, 1314–1324.

Hu, E. J.; Shen, Y.; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, S.; Wang, L.; and Chen, W. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *ICLR*.

Huang, Z.; Li, B.; Cai, Y.; Wang, R.; Guo, S.; Fang, L.; Chen, J.; and Wang, L. 2023. What can discriminator do? towards box-free ownership verification of generative adversarial networks. In *CVPR*, 5009–5019.

Krizhevsky, A.; Hinton, G.; et al. 2009. Learning multiple layers of features from tiny images. Technical report, Technical report, University of Toronto.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, volume 25.

Le, Y.; and Yang, X. 2015. Tiny imagenet visual recognition challenge. *CS 231N*, 7(7): 3.

Li, F.; Yang, L.; Wang, S.; and Liew, A. W.-C. 2022. Leveraging Multi-task Learning for Umambiguous and Flexible Deep Neural Network Watermarking. In *SafeAI@ AAAI*.

Li, F.; Zhao, H.; Du, W.; and Wang, S. 2024. Revisiting the Information Capacity of Neural Network Watermarks: Upper Bound Estimation and Beyond. In *AAAI*, 21331–21339.

Li, Y.; Abady, L.; Wang, H.; and Barni, M. 2021. A feature-map-based large-payload DNN watermarking algorithm. In *IWDW*, 135–148.

Li, Y.; Tondi, B.; and Barni, M. 2021. Spread-transform dither modulation watermarking of deep neural network. *Journal of Information Security and Applications*, 63: 103004.

Li, Y.; Wang, H.; and Barni, M. 2021. A survey of deep neural network watermarking techniques. *Neurocomputing*, 461: 171–193.

Lim, J. H.; Chan, C. S.; Ng, K. W.; Fan, L.; and Yang, Q. 2022. Protect, show, attend and tell: Empowering image captioning models with ownership protection. *Pattern Recognition*, 122: 108285.

Liu, H.; Weng, Z.; and Zhu, Y. 2021. Watermarking Deep Neural Networks with Greedy Residuals. In *ICML*, 6978–6988.

Liu, H.; Weng, Z.; Zhu, Y.; and Mu, Y. 2023. Trapdoor normalization with irreversible ownership verification. In *ICML*, 22177–22187. PMLR.

Liu, Z.; Hu, H.; Lin, Y.; Yao, Z.; Xie, Z.; Wei, Y.; Ning, J.; Cao, Y.; Zhang, Z.; Dong, L.; et al. 2022. Swin transformer v2: Scaling up capacity and resolution. In *CVPR*, 12009–12019.

Lukas, N.; Jiang, E.; Li, X.; and Kerschbaum, F. 2022. Sok: How robust is image classification deep neural network watermarking? In *S&P*, 787–804. IEEE.

Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; Agarwal, S.; et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 1.

Ngo, A. T.; Heng, C. S.; Chattopadhyay, N.; and Chattopadhyay, A. 2025. Persistence of Backdoor-based Watermarks for Neural Networks: A Comprehensive Evaluation. *IEEE Transactions on Neural Networks and Learning Systems*.

Novikova, J.; Dušek, O.; and Rieser, V. 2017. The E2E dataset: New challenges for end-to-end generation. *arXiv preprint arXiv:1706.09254*.

Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; Sutskever, I.; et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8): 9.

Rouhani, B. D.; Chen, H.; and Koushanfar, F. 2019. Deepsigns: an end-to-end watermarking framework for protecting the ownership of deep neural networks. In *ASPLOS*, volume 3.

Simonyan, K.; and Zisserman, A. 2015. Very deep convolutional networks for large-scale image recognition. In *ICLR*.

Sun, Y.; Liu, T.; Hu, P.; Liao, Q.; Fu, S.; Yu, N.; Guo, D.; Liu, Y.; and Liu, L. 2023. Deep intellectual property protection: A survey. *arXiv preprint arXiv:2304.14613*.

Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; and Rabinovich, A. 2015. Going deeper with convolutions. In *CVPR*, 1–9.

Uchida, Y.; Nagai, Y.; Sakazawa, S.; and Satoh, S. 2017. Embedding watermarks into deep neural networks. In *ACM ICMR*, 269–277.

Webster, A. F.; and Tavares, S. E. 1985. On the design of S-boxes. In *Eurocrypt*, 523–534. Springer.

Xue, M.; Zhang, Y.; Wang, J.; and Liu, W. 2021. Intellectual property protection for deep learning models: Taxonomy, methods, attacks, and evaluations. *IEEE Transactions on Artificial Intelligence*, 3(6): 908–923.

Zagoruyko, S. 2016. Wide residual networks. In *BMVC*.

Zhang, J.; Chen, D.; Liao, J.; and et al. 2021. Deep model intellectual property protection via deep watermarking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(8): 4005–4020.

Zhang, J.; Chen, D.; Liao, J.; Zhang, W.; Hua, G.; and Yu, N. 2020. Passport-aware normalization for deep model protection. In *NeurIPS*, volume 33, 22619–22628.

Zhu, R.; Zhang, X.; Shi, M.; and Tang, Z. 2020. Secure neural network watermarking protocol against forging attack. *EURASIP Journal on Image and Video Processing*, 2020: 1–12.