Hashed Watermark as a Filter: Boosting Robustness of Weight-based Neural-Network Watermarking

Anonymous Author(s)

Abstract

As valuable digital assets, deep neural networks necessitate robust ownership protection, positioning neural-network watermarking (NNW) as a promising solution. Among various NNW approaches, weight-based methods are favored for their simplicity and practicality; however, they remain vulnerable to forging and overwriting attacks. To address those challenges, we propose *NeuralMark*, a robust approach 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 employed as a filter to select the model parameters for embedding. This design cleverly intertwines the embedding parameters with the hash function, providing robust defense against both forging and overwriting attacks. An average pooling is also incorporated to resist fine-tuning and pruning attacks. As a result, NeuralMark offers robust resilience against a wide range of attacks without compromising model performance. Also, it can be seamlessly integrated into various neural network architectures, ensuring broad applicability. Theoretically, we analyze its security boundary. Empirically, we verify its effectiveness and robustness across 13 distinct Convolutional and Transformer architectures, covering five image classification tasks and one text generation task. The source codes are available at https://anonymous.4open.science/r/NeuralMark.

CCS Concepts

• **Security and privacy → Digital rights management**; *Domain-specific security and privacy architectures.*

Keywords

Neural-network Watermarking, Weight-based Approach, Hashed Watermark Filter

ACM Reference Format:

Anonymous Author(s). 2025. Hashed Watermark as a Filter: Boosting Robustness of Weight-based Neural-Network Watermarking. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (CCS '25)*. ACM, New York, NY, USA, 14 pages. https://doi.org/XXXXXXX.XXXXXXX

CCS '25, Taipei, Taiwan

ACM ISBN 978-1-4503-XXXX-X/2025/06
https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

The advancements in artificial intelligence have led to the development of numerous deep neural networks, particularly large language models [1, 3, 7, 32, 36]. 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 [5]. Thus, they can be regarded as valuable digital assets, necessitating urgent measures for ownership protection. To this end, neural-network watermarking (NNW) approaches [35, 42, 46] have been proposed to protect model ownership by embedding watermarks within the neural network. Methods that require access to model weights for watermark embedding and verification fall under white-box neural network watermarking (NNW) [25, 30, 31, 44], whereas those that do not require access to the model weights belong to black-box NNW [2, 15, 19, 23, 26]. Both approaches have demonstrated significant progress in safeguarding model ownership [42] and hold promise for integration in practical applications [9, 10]. Given the distinct challenges inherent in each approach, this paper concentrates on white-box NNW, with black-box NNW reserved for future exploration.

Existing white-box NNW methods can be broadly categorized into three sub-branches: (i) Weight-based methods [12, 25, 28, 30, 44] embed watermarks into model weights; (ii) Passport-based methods [9, 10, 31, 49] introduce passport layers to replace normalization layers for watermark embedding; and (iii) Activation-based methods [27, 29, 40] incorporate watermarks into the activation maps of intermediate layers. Among these approaches, weight-based methods are particularly appealing due to their inherent simplicity and practicality. By embedding watermarks directly into the model's weights, those methods offer a straightforward process that can be seamlessly integrated into various network architectures without altering the original structure. This feature renders them especially valuable for various practical applications. Although several state-of-the-art weight-based methods [12, 25, 28, 30] 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 model's ownership. This raises a question: "*How can we design a more robust and effective weight-based NNW method that defends against all of the aforementioned attacks?*"

To answer this question, we propose *NeuralMark*, a robust approach built around 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 hash function [45] ensures that even slight changes in the input lead to significant, unpredictable variations in the output, effectively impeding gradient calculation and making reverse-engineering infeasible. Moreover, because the hashed watermarks generated by the model owner and the adversary are distinct, using them as private filters reduces the overlap in selected parameters, especially when the filtering process is performed repeatedly. This mechanism significantly increases the difficulty for adversaries to identify and manipulate the filtered parameters, thereby protecting the original watermark. Therefore, the hashed watermark filter cleverly intertwines the embedding parameters with the hash function, providing robust defense against both forging and overwriting attacks. We also apply an average pooling mechanism to the filtered parameters due to its resilience against parameter perturbations. Upon obtaining the resulting parameters, the hashed watermark is embedded into those parameters using a lightweight watermarking embedding loss. When a potentially unauthorized model is identified, the corresponding watermark can be extracted from those parameters to verify ownership. As a result, NeuralMark provides strong resilience against those attacks while preserving model performance.

The main contributions of this paper are highlighted as follows.

- We propose a NeuralMark, which, to the best of our knowledge, is the first method to utilize the hashed watermark filter to boost the robustness of weight-based NNW. Also, we provide a theoretical analysis of its security boundary.
- In NeuralMark, an elegant hashed watermark filter is developed to cleverly intertwine the embedding parameters with the hash function, offering robust defense 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.

2 Related Work

In this section, we review weight-based, passport-based, and activation-based methods, respectively.

Weight-based Method. This kind of methods [12, 25, 28, 30, 44] embeds watermarks into the model weights of neural networks. For instance, [44] propose the first weight-based method, which embeds the watermark into the model weights of an intermediate layer in the neural network. Another example is that [28] propose a method based on spread transform dither modulation that enhances the secrecy of the watermark. However, those two methods cannot effectively resist forging and overwriting attacks. Moreover, [12] utilize the secret keys to pseudo-randomly select weights for watermark embedding and apply spread-spectrum modulation to disperse the modulated watermark across different layers. This method effectively defends overwriting attacks while neglecting forging attacks. Additionally, [30] propose to greedily choose important model parameters for watermark embedding without an additional secret key. Although this method is effective against forging attacks, it fails to provide strong resistance to overwriting attacks of varying strength levels. Recently, [25] introduce random noises into the watermarked parameters and then employ a majority voting scheme to aggregate the verification results across multiple rounds. While this method enhances the watermark's robustness to some extent, it remains ineffective against forging and overwriting attacks.

Passport-based Method. This group of methods [9, 10, 31, 49] integrates the watermark into the normalization layers in neural networks. Specifically, [9, 10] propose the first passport-based method, which utilizes additional passport samples (*e.g.*, images) to generate affine transformation parameters for the normalization layers, tightly binding them to the model performance. Subsequently, [49] integrate a private passport-aware branch into the normalization layers, which is trained jointly with the target model and is used solely for watermark verification. Recently, [31] argue that binding the model performance is insufficient to defend against forging attacks, and thus propose establishing a hash mapping between passport samples and watermarks.

Activation-based Method. This category of methods [27, 29, 40] incorporates watermarks into the activation maps of intermediate layers in neural networks. For instance, [40] incorporate the watermark into the mean vector of activation maps generated by predetermined trigger samples. Similarly, [27] directly integrate the watermark into the activation maps associated with the trigger samples. Additionally, [29] embed the watermark into the hidden memory state of a recurrent neural network.

3 Preliminary

In this section, we elaborate on the principle and vulnerability of the first weight-based method [44], which we refer to as VanillaMark, serving as the foundation for subsequent weight-based watermarking methods [12, 25, 28, 30]. Specifically, it begins by selecting, averaging, and flattening a subset of model parameters θ into a parameter vector $\widetilde{\mathbf{w}} \in \mathbb{R}^k$. A secret key matrix $\mathbf{K} \in \mathbb{R}^{k \times n}$ is then used to derive the extracted watermark via $\widetilde{\mathbf{b}} = \delta(\widetilde{\mathbf{w}}\mathbf{K})$, where $\delta(\cdot)$ denotes the sigmoid function. To embed a target binary watermark $\mathbf{b}$ into $\widetilde{\mathbf{w}}$, VanillaMark optimizes the following objective:

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

where $\mathcal{L}_m$ denotes the main task loss (e.g., classification loss), $\mathcal{L}_e(\cdot,\cdot)$ represents the binary cross-entropy loss, and λ is a positive trade-off hyper-parameter. Although VanillaMark is simple and pioneering, and can resist fine-tuning and pruning attacks [44], it remains vulnerable to the following two critical threats:

- **Forging Attack**: An adversary can learn the secret key for any arbitrary watermark. Specifically, given a counterfeit watermark $\mathbf{b}_a$, the attacker can learn a corresponding key $\mathbf{K}_a$ by minimizing $\mathcal{L}_e(\mathbf{b}_a, \widetilde{\mathbf{b}}_a)$, *i.e.*, $\mathbf{K}_a = \arg\min_{\mathbf{K}_a} \mathcal{L}_e(\mathbf{b}_a, \widetilde{\mathbf{b}}_a)$.
- **Overwriting Attack**: VanillaMark neither protects the confidentiality of watermarked parameters nor ensures non-overlapping usage between the model owner's and the adversary's parameters. Once the watermarked parameters are identified, an adversary can forge a counterfeit watermark tuple $\{\mathbf{K}_a, \mathbf{b}_a\}$ and embed $\mathbf{b}_a$ into the model parameters by optimizing $\min_\theta \mathcal{L}_m + \lambda \mathcal{L}_e(\mathbf{b}_a, \widetilde{\mathbf{b}})$. Since different watermarks often induce conflicting

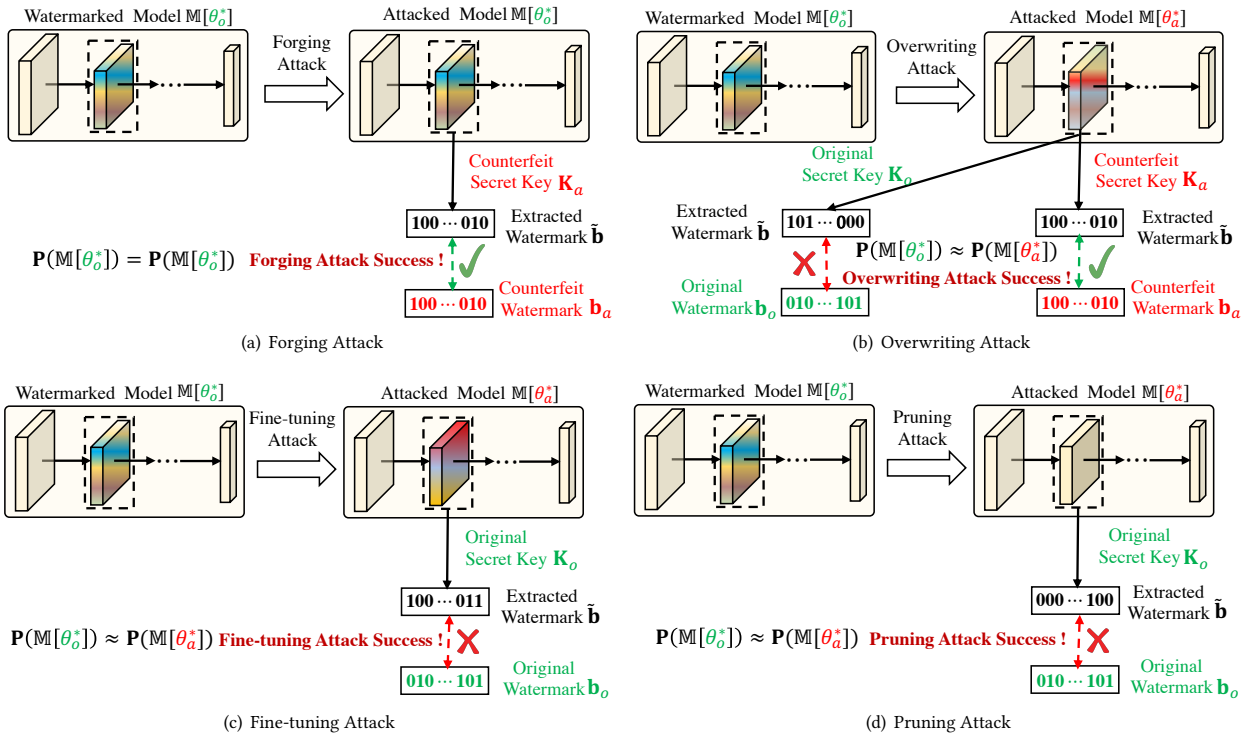


Figure 1: Illustrations of different types of attacks. (a) Forging attack: the adversary aims to generate a counterfeit secret key–watermark pair without modifying the model parameters. (b) Overwriting attack: the adversary embeds a counterfeit watermark to overwrite the original one. (c) Fine-tuning attack: the adversary fine-tunes the model in an attempt to remove the original watermark. (d) Pruning attack: the adversary prunes the model parameters to remove the original watermark.

gradients on the same parameters, the newly embedded watermark can easily overwrite the original one.

4 Problem Formulations

In this section, we introduce several key formulations used throughout this paper.

4.1 Weight-based NNW

In the weighted-based NNW problem, we are provided with a training dataset $\mathcal{D}$ and a watermark tuple $\mathcal{W} = \{\mathbf{K}, \mathbf{b}\}$, where $\mathbf{K}$ is a secret key and $\mathbf{b}$ is a binary watermark consisting of ones and zeros. The goal is to train a watermarked model $\mathbb{M}(\theta^*)$ using $\mathcal{D}$ such that the model parameters θ^* effectively embed $\mathbf{b}$ while satisfying the following criteria: (i) The watermark should minimally affect the model performance and be difficult for adversaries to detect; and (ii) The watermark must be resilient against a wide range of adversarial attacks.

4.2 Threat Model

We assume that an adversary can illegally obtain a watermarked model and identify the layers containing the watermark. Additionally, the adversary has access to the training datasets but is constrained by limited computational resources. As discussed above, this paper primarily focuses on forging and overwriting attacks, while also considering fine-tuning and pruning attacks. Those attack scenarios are described below.

- **Forging Attack**: As illustrated in Figure 1(a), in a 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 [9, 10].
- **Overwriting Attack**: As presented in Figure 1(b), in an overwriting attack, the adversary embeds a counterfeit watermark to overwrite the original watermark [30].
- **Fine-tuning Attack**: As depicted in Figure 1(c), in a fine-tuning attack, the adversary fine-tunes the model in an attempt to remove the original watermark.
- **Pruning Attack**: As shown in Figure 1(d), in a pruning attack, the adversary attempts to remove the original watermark by pruning the model parameters.

4.3 Success Criteria for Threat Model

Building on insights from [9, 10, 24, 50], 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

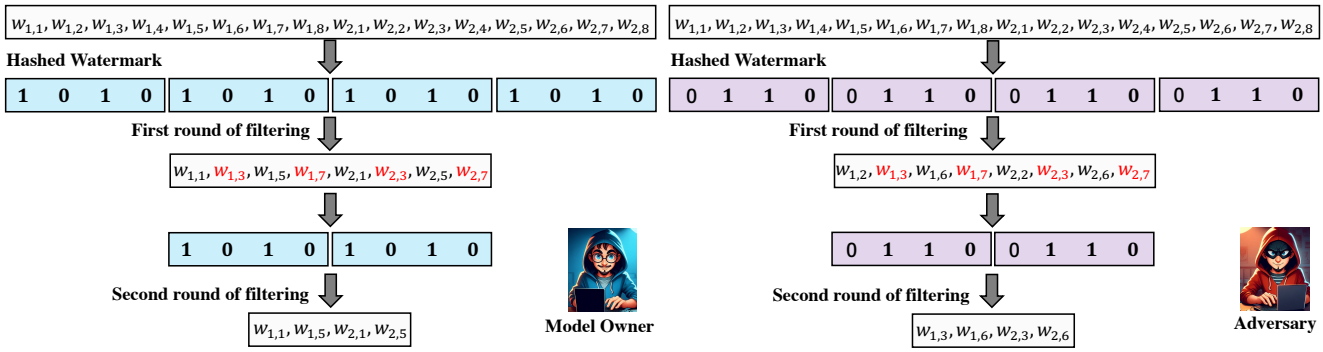


Figure 2: Illustration of hashed watermark filter. Here, the model owner's hashed watermark is $[1, 0, 1, 0]$, while the adversary's is $[0, 1, 1, 0]$. Without filtering, all 16 parameters overlap. After one round of filtering, each retains eight parameters, with four overlapping. A second round leaves four parameters each, with no overlap.

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:

- **Success Criteria for Forging Attack**: Forge a counterfeit watermark that passes verification without modifying the model parameters.
- **Success Criteria for Overwriting Attack**: Remove the original watermark and embed a counterfeit one by modifying the model parameters, while maintaining model performance.
- **Success Criteria for Fine-tuning Attack**: Remove the original watermark through fine-tuning, while maintaining model performance.
- **Success Criteria for Pruning Attack**: Remove the original watermark through parameter pruning, while maintaining model performance.

5 Methodology

In this section, we present the proposed NeuralMark.

5.1 Motivation

As discussed in Section 3, most weight-based methods struggle to simultaneously defend against both forging and overwriting attacks. On the 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. Therefore, to resist such attacks, it is essential to keep the watermarked parameters confidential and ensure that those used by the model owner and the adversary are non-overlapping.

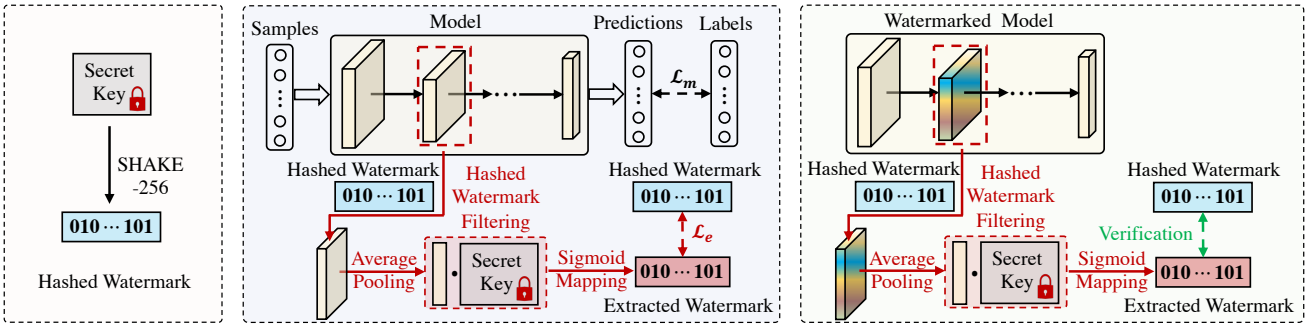
To address both threats, we propose a *hashed watermark filter* that leverages an irreversible binary watermark as a private filter to restrict embedding to a secret subset of model parameters. Specifically, we utilize a hash function to generate an irreversible binary watermark from a secret key, which is then applied as a filter to select the model parameters for embedding. This design cleverly intertwines the embedding parameters with the hash function, providing two key properties:

- **Gradient Obfuscation**: The avalanche effect of hash function ensures that even minor changes in the input lead to large, unpredictable changes in the output, effectively impeding gradient computation and rendering reverse-engineering infeasible.
- **Embedding Isolation**: Since the hashed watermarks derived by the model owner and the adversary are 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 2, 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. Consequently, 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, those properties enable the hashed watermark filter to offer strong resistance against both forging and overwriting attacks, forming the core of NeuralMark. Next, we elaborate on the NeuralMark.

5.2 NeuralMark

As depicted in Figure 3, NeuralMark consists of three primary steps: (i) watermark generation; (ii) watermark embedding; and (iii) watermark verification. Next, we detail how each step works.



(a) Watermark Generation (b) Watermark Embedding (c) Watermark Verification

Figure 3: Illustrations of the processes for watermark generation (a), embedding (b), and verification (c).

5.2.1 Hashed Watermark Generation. As aforementioned, we construct a hash-based mapping from a secret key to a binary watermark, as shown in Figure 3(a). 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.*, normalized 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 [8], an extendable-output function from the SHA-3 family that allows dynamic adjustment of output length. Furthermore, auxiliary content C (*e.g.*, textual descriptors or unique identifiers) can also be incorporated into the hash function, yielding $\mathbf{b} = \mathcal{H}(\mathbf{K}||C)$, 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 subsequent experiments.

5.2.2 Watermark Embedding. As illustrated in Figure 3(b), to embed the hashed watermark $\mathbf{b}$ into the model $\mathbb{M}(\theta)$, we first select and flatten a subset of parameters (*e.g.*, one-layer parameters) from θ 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 [13] 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

$$\min_{\theta} \mathcal{L}_m + \lambda \mathcal{L}_e(\widetilde{\mathbf{b}}, \mathbf{b}), \tag{2}$$

where $\mathcal{L}_m$ denotes the main task loss (*e.g.*, classification loss), $\mathcal{L}_e(\cdot,\cdot)$ represents the binary cross-binary 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. (2), the watermark can be embedded into model parameters during the main task training. The watermark embedding process is summarized in Algorithm 1.

Algorithm 1 Watermark Embedding in NeuralMark

Input: Training dataset $\mathcal{D}$, secret key $\mathbf{K}$, index of embedding layer $\mathbf{I}$, hyper-parameters λ, T, and filter rounds R.
Output: Watermarked model $\mathbb{M}(\theta^*)$.

```
Randomly initialize the model parameter θ.
Generate the watermark b = H(K).
for t = 0 to T − 1 do
    Use I to select a subset from θ and flatten it to create w.
    for r = 1 to R do
        Perform watermark filtering on w to obtain w^(r).
    end for
    Apply average pooling on w^(R) to yield w̃.
    Execute sigmoid mapping on w̃K to produce b̃.
    Update θ based on Eq. (2).
end for
```

5.2.3 Watermark Verification. The watermark verification process is similar to the embedding process, as depicted in Figure 3(c). 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

$$\rho = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}\left[b_i = \mathcal{T}(\widetilde{b}_i)\right], \tag{3}$$

where $\mathcal{T}(x)$ is a threshold function that outputs 1 if $x > 0.5$ and 0 otherwise, and $\mathbb{1}(\psi)$ 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:

- The watermark detection rate ρ exceeds a theoretical security boundary ρ^*, which will be theoretically analyzed later.
- 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.

Algorithm 2 Watermark Verification in NeuralMark

Input: Watermarked model $\mathbb{M}(\theta^*)$, secret key $\mathbf{K}$, watermark $\mathbf{b}$, index of embedding layer $\mathbf{I}$, filter rounds R, and security boundary ρ^*.
Output: True (Verification Success) or False (Verification Failure).

```
Use I to select a subset from θ* and flatten it to create w.
for r = 1 to R do
    Perform watermark filtering on w to obtain w^(r).
end for
Apply average pooling on w^(R) to yield w̃.
Execute sigmoid mapping on w̃K to produce b̃.
Calculate watermark detection rate ρ based on Eq. (3).
if ρ ≥ ρ* and H(K) = b then
    return True
else
    return False
end if
```

5.3 Theoretical Analysis

We present a theoretical analysis to determine the security boundary of NeuralMark in Proposition 5.1.

Proposition 5.1. *Under the assumption that the hash function produces uniformly distributed outputs [4], 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 5.1 is provided in Appendix A. Proposition 5.1 provides a theoretical benchmark for establishing the security boundary of the watermark detection rate. Specifically, with $n = 256$, if the watermark detection rate $\rho \geq 88.28\%$, 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 88.28% as the security bound for the watermark detection rate in the experiments.

6 Experiments

In this section, we evaluate NeuralMark across a variety of datasets, architectures, and tasks.

6.1 Experimental Setup

Datasets and Architectures. We use five image classification datasets: CIFAR-10, CIFAR-100 [20], Caltech-101 [11], Caltech-256 [14], and TinyImageNet [22], as well as one text generation dataset, E2E [37]. Additionally, we utilize 11 image classification architectures, including eight Convolutional architectures: AlexNet [21], VGG-13, VGG-16 [41], GoogLeNet [43], ResNet-18, ResNet-34 [16], WideResNet-50 [48], and MobileNet-V3-L [17], as well as three Transformer architectures: ViT-B/16 [6], Swin-V2-B, and Swin-V2-S [33]. Furthermore, we adopt two text generation architectures: GPT-2-S and GPT-2-M [39].

Baselines and Metrics. We compare NeuralMark with VanillaMark [44], and two state-of-the-art weight-based methods proposed in [30] and [25] (see Section 2 for details). For clarity, we refer to those two methods as GreedyMark and VoteMark, respectively. Additionally, we include a comparison with a method that does not involve watermark embedding, referred to as Clean. For the image classification task, we evaluate model performance using classification accuracy, while the watermark embedding task is assessed based on the watermark detection rate. As for the text generation task, we follow [18] 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.

Implementation Details. We implement NeuralMark using the PyTorch framework [38] and conduct all experiments on three NVIDIA V100 series GPUs. The specific hyper-parameters are summarized below.

- For all the image classification architectures, we train for 200 epochs with a multi-step learning rate schedule from scratch, with learning rates set to 0.01, 0.001, and 0.0001 for epochs 1 to 100, 101 to 150, and 151 to 200, respectively. We apply a weight decay of 5×10^{-4} and set the momentum to 0.9. The batch sizes for the training and test datasets are set to 64 and 128, respectively. In addition, we set hyper-parameter λ to 1 and the number of filter rounds R to 4.
- For the GPT-2-S and GPT-2-M architectures, we utilize the Low-Rank Adaptation (LoRA) technique [18]. Each architecture is trained for 5 epochs with a linear learning rate scheduler, starting at 2×10^{-4}. We set the warm-up steps to 500, apply a weight decay with a coefficient of 0.01, and enable bias correction in the AdamW optimizer [34]. The dimension and the scaling factor for LoRA are set to 4 and 32, respectively, with a dropout probability of 0.1 for the LoRA layers. The batch sizes for the training and test sets are 8 and 4, respectively. Moreover, we set hyper-parameter λ to 1 and the number of filter rounds R to 10.

6.2 Fidelity Evaluation

Question 1. Is NeuralMark capable of reliably embedding watermarks while preserving model performance across a variety of datasets, architectures, and tasks?

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. Table 2 lists the results of NeuralMark on the CIFAR-100 dataset using VGG-13, VGG-16, GoogLeNet, ResNet-34, WideResNet-50, MobileNet-V3-L, ViT-B/16, Swin-V2-B, and Swin-V2-S. We find that NeuralMark maintains a 100% watermark detection rate across a wide range of architectures while exerting minimal impact on model performance.

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

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 2: Comparison of classification accuracy (%) on CIFAR-100 using various architectures. 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 3: Comparison on E2E using GPT-2-S and GPT-2-M. 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

Those observations suggest that NeuralMark is highly generalizable across various architectures.

Text Generation Tasks. Finally, we evaluate the effect of NeuralMark on the text generation tasks. Table 3 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 validate the potential of NeuralMark in safeguarding the ownership of generative models.

Overall, NeuralMark demonstrates consistent fidelity across various datasets, architectures, and tasks.

6.3 Robustness Evaluation

Question 2. Is NeuralMark capable of withstanding forging attacks?

We adopt the setting detailed in Section 4.2 to assess the robustness of NeuralMark against forging attacks. Concretely, for VanillaMark and VoteMark, we first randomly generate a counterfeit watermark and then learn the corresponding secret key by freezing the model parameters. Since GreedyMark does not require a secret key, we utilize 10 sets of randomly forged watermarks to directly verify them using the watermarked model. For NeuralMark, due to the avalanche effect of hash functions, a method similar to GreedyMark is employed, where 10 sets of randomly forged watermarks are directly verified using the watermarked model. Table 4 presents the watermark detection rates of forging attacks, and we present the following significant 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. In summary, those results suggest that NeuralMark effectively withstands forging attacks.

Table 4: Comparison of watermark detection rate (%) against forging attacks using ResNet-18.

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

Question 3. Is NeuralMark robust to overwriting attacks, especially with varying attack strength levels?

We follow the setting outlined in Section 4.2 to assess the robustness of NeuralMark against overwriting attacks. Specifically, we analyze two key factors: the hyper-parameter λ in Eq. (2) 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 λ. First, 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 and CIFAR-10 to CIFAR-100 tasks 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 Section 4.3, the original watermark must be effectively removed for overwriting attacks to be deemed successful. Thus, the

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 watermark detection rate and classification accuracy, respectively.

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)
CIFAR-10 to CIFAR-100	1	71.78 (100)	72.68 (98.82)	71.34 (55.07)	72.97 (98.43)	0.001	71.78 (100)	72.68 (98.82)	71.34 (55.07)	72.97 (98.43)
	10	72.6 (100)	72.03 (98.04)	72.30 (49.21)	72.08 (98.04)	0.005	71.04 (99.60)	70.02 (69.53)	70.25 (48.04)	71.11 (71.09)
	50	72.73 (100)	72.45 (95.70)	70.92 (46.87)	72.38 (97.26)	0.01	69.14 (96.48)	69.02 (59.76)	69.25 (46.09)	68.88 (62.11)
	100	71.49 (100)	71.92 (92.18)	72.05 (48.04)	72.72 (93.75)	0.1	51.88 (60.54)	51.76 (53.90)	51.71 (51.56)	51.74 (56.25)
	1000	71.81 (100)	71.35 (57.42)	71.74 (51.95)	70.73 (56.64)	1	1.00 (44.53)	1.00 (53.15)	1.00 (50.00)	1.00 (53.51)

overwriting attack experiments focus solely on whether the original watermark can be successfully removed. We can summarize several insightful observations.

- 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 tasks are 100%, 53.90%, 49.60%, and 59.37%, respectively. Those results indicate that NeuralMark exhibits strong robustness against overwriting attacks.
- 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.

Distinct Values of η. Second, 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 and CIFAR-10 to CIFAR-100 tasks using ResNet-18. We have the following important observations.

- As η increases, model performance declines due to its substantial impact on performance. Thus, the adversary cannot arbitrarily increase η to strengthen the attack.
- At $\eta = 0.005$, the original watermark detection rates for VanillaMark, GreedyMark, and VoteMark drop dramatically, whereas NeuralMark maintains a detection rate close to 100%. 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 88.28% defined in Section 5.3, while those for the other methods fall significantly. 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.

On the whole, all results confirm NeuralMark's robustness against overwriting attacks.

Question 4. Is NeuralMark robust to fine-tuning attacks?

We adhere to the setting stated in Section 4.2 to evaluate the robustness of NeuralMark against fine-tuning attacks.

Fine-tuning All Model Parameters. Following [30], we adopt the same hyper-parameters for fine-tuning attacks as during training, except for setting the learning rate to 0.001. Also, we replace the task-specific classifier with randomly initialized parameters and optimize all parameters by minimizing the main task loss $\mathcal{L}_m$ for 100 epochs. Table 6 reports the results of fine-tuning attacks, 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.

Fine-tuning Watermark-Specific Layer Parameters. Furthermore, Table 7 reports the experimental results of fine-tuning the watermark-specific layer and classifier. As can be seen, the watermark detection rate remains at 100%, but the model performance of all methods exhibits a substantial decline. Specifically, for the CIFAR-10 to CIFAR-100 task using ResNet-18, the accuracy achieved by NeuralMark through fine-tuning the watermark embedding layer and classifier is 49.77%, which is markedly lower than the 71.67% accuracy obtained when all parameters are fine-tuned. Similar trends are observed across other methods. Those results indicate that fine-tuning only the watermark embedding layer and classifier makes it challenging to maintain effective model performance, resulting in the failure of fine-tuning attacks.

Overall, NeuralMark can effectively defend against fine-tuning attacks.

Question 5. Is NeuralMark robust to pruning attacks?

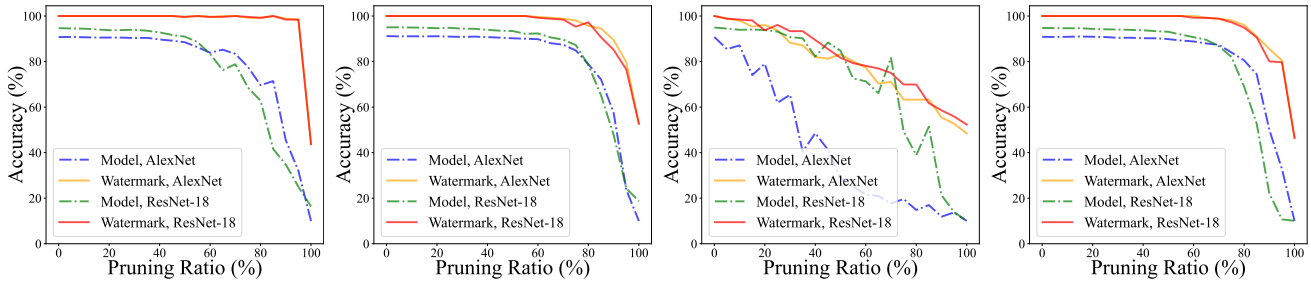
We now verify the robustness of NeuralMark in resisting pruning attacks. Specifically, we randomly reset a specified proportion of model parameters in the watermark embedding layer to zero. Figure 4 shows the results of pruning attacks on the CIFAR-10 dataset using AlexNet and ResNet, respectively. As can be seen, as the

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

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	89.44	93.21	89.11(100)	93.74(100)	89.00(100)	93.29(100)	89.34(99.21)	93.21(100)	89.03(100)	93.59(100)
CIFAR-10 to CIFAR-100	65.46	72.17	64.60(100)	71.67(100)	65.03(92.18)	72.49(97.26)	64.57(98.82)	72.06(100)	64.83(96.09)	72.27(98.04)
Caltech-256 to Caltech-101	72.69	76.93	73.55(100)	76.60(100)	72.90(100)	78.48(100)	73.12(100)	77.19(100)	72.90(100)	77.41(100)
Caltech-101 to Caltech-256	43.39	46.48	43.15(100)	44.42(100)	43.21(98.43)	45.69(99.60)	43.47(99.60)	45.25(100)	43.78(98.43)	45.29(100)

Table 7: Comparison of resistance to fine-tuning attacks against watermark embedding layer using ResNet-18. Values (%) inside and outside the bracket are the watermark detection rate and classification accuracy, respectively.

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)



(a) NeuralMark (b) VanillaMark (c) GreedyMark (d) VoteMark

Figure 4: Comparison of resistance to pruning attacks at various pruning ratios on CIFAR-10 using AlexNet and ResNet-18.

pruning ratio increases, the performance of NeuralMark degrades while the detection rate remains nearly 100%. This indicates NeuralMark's robustness against pruning attacks. Moreover, we observe that both VanillaMark and VoteMark exhibit strong resistance to pruning attacks, while GreedyMark demonstrates relatively weak resistance. One possible reason is that GreedyMark depends on several important parameters, and their removal may affect its robustness. More results of pruning attacks across distinct datasets are provided in Appendix B. All the results suggest that NeuralMark effectively resists pruning attacks.

6.4 Hashed Watermark Filter Analysis

Question 6. How does the number of filtering rounds impact the fidelity of NeuralMark?

To evaluate the impact of filtering rounds on NeuralMark's fidelity, we conduct experiments with 6 and 8 filters, compared to the default 4 filters. Table 8 presents the impact of watermark embedding on the model performance across distinct filtering rounds. The results demonstrate that NeuralMark, even with varying filtering rounds, has a minimal effect on the model performance while successfully embedding watermarks.

Table 8: Comparison of classification accuracy (%) with various distinct filter rounds on CIFAR-10 and CIFAR-100 using ResNet-18. Watermark detection rates are omitted as they all reach 100%.

Dataset	4 Filters	6 Filters	8 Filters
CIFAR-10	94.79	94.74	94.88
CIFAR-100	76.74	75.59	76.16

Question 7. How does the number of filtering rounds influence the robustness of NeuralMark?

To assess the influence of the number of filtering rounds on NeuralMark's robustness against attacks, we conduct experiments with 6 and 8 filters, compared to the default 4 filters. We omit forging attacks, as the hashed watermark filter is resilient to them.

Table 9: Comparison of resistance to overwriting attacks at various trade-off hyper-parameters (λ) and learning rates (η) with distinct filtering rounds using ResNet-18. Values (%) inside and outside the bracket are watermark detection rate and classification accuracy, respectively.

Overwriting	λ	4 Filters	6 Filters	8 Filters	η	4 Filters	6 Filters	8 Filters
CIFAR-100 to CIFAR-10	1	93.65 (100)	93.13(100)	93.40(100)	0.001	93.65 (100)	93.13(100)	93.40(100)
	10	93.44 (100)	93.06(100)	93.41(100)	0.005	91.76 (99.60)	92.10(100)	91.62(100)
	50	93.46 (100)	93.06(100)	93.54(100)	0.01	91.58 (92.18)	91.64(94.92)	90.48(89.84)
	100	93.53 (100)	92.88(100)	92.99(100)	0.1	75.2 (50.78)	75.84(58.2)	74.54(51.56)
	1000	93.09 (100)	93.03(100)	93.39(100)	1	10.00 (44.53)	10.00(47.26)	10.00(50.39)
CIFAR-10 to CIFAR-100	1	71.78 (100)	71.69(100)	72.63(100)	0.001	71.78 (100)	71.69(100)	72.63(100)
	10	72.6 (100)	72.06(100)	72.81(100)	0.005	71.04 (99.60)	70.65(100)	71.46(100)
	50	72.73 (100)	71.85(100)	72.85(100)	0.01	69.14 (96.48)	69.47(97.26)	67.88(95.70)
	100	71.49 (100)	71.88(100)	72.00(100)	0.1	51.88 (60.54)	55.18(62.10)	50.36(55.07)
	1000	71.81 (100)	72.22(100)	72.39(100)	1	1.00 (44.53)	1.00(47.26)	1.00(50.39)

Overwriting Attacks. Table 9 lists the results of overwriting attacks across distinct filtering rounds. From the results, we find that when the number of filtering rounds is set to 6, NeuralMark exhibits superior robustness compared to 4 and 8 filter rounds. Specifically, at $\eta = 0.01$, the original watermark detection rates for 4, 6, and 8 filter rounds on the CIFAR-100 to CIFAR-10 task are 92.18%, 94.92%, and 89.84%, respectively. Those results indicate that increasing the number of filtering rounds can enhance robustness against overwriting attacks to a certain extent. However, when the number of filtering rounds exceeds a certain threshold, the robustness may be slightly compromised due to the reduction in the number of parameters.

Fine-tuning Attacks. Table 10 reports the results of fine-tuning attacks across distinct filtering rounds. We can observe that NeuralMark maintains a watermark detection rate of 100% across all filtering rounds, with negligible impact on the model performance.

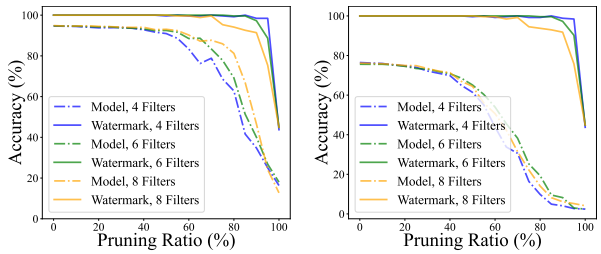
Table 10: Comparison of resistance to fine-tuning attacks with distinct filter rounds using ResNet-18. Watermark detection rates are omitted as they all reach 100%.

Fine-tuning	Clean	4 Filters	6 Filters	8 Filters
CIFAR-100 to CIFAR-10	93.21	93.74	93.01	93.55
CIFAR-10 to CIFAR-100	72.17	71.67	72.68	72.27

Pruning Attacks. Figure 5 shows the results of pruning attacks on the CIFAR-10 and CIFAR-100 datasets using ResNet-18 across different filtering rounds. As can be seen, as the number of filtering rounds increases, the robustness of NeuralMark in resisting pruning attacks exhibits a slight decline. One reason is that increasing the number of filter rounds reduces the number of filtered parameters, leading to a smaller average pooling window size, which affects the robustness against pruning attacks.

Question 8. How does the hashed watermark filter affect the overlap rate between the model owner and the adversary?

To analyze the effect of the hashed watermark filter on the overlap rate between the model owner and the adversary, we generate five counterfeit watermarks and calculate the overlap ratio between the parameters filtered by those and the original watermark. As shown in Figure 6, the overlap rate decreases towards zero with more filtering rounds, indicating that watermark filtering enhances the confidentiality of the watermarked parameters.



(a) CIFAR-10 (b) CIFAR-100

Figure 5: Comparison of resistance to pruning attacks with distinct filter rounds on CIFAR-10 and CIFAR-100 using ResNet-18 at various pruning ratios.

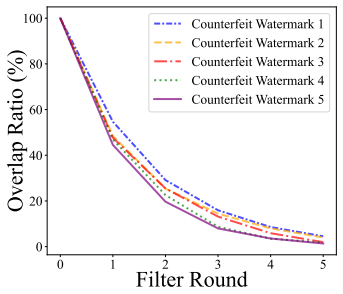
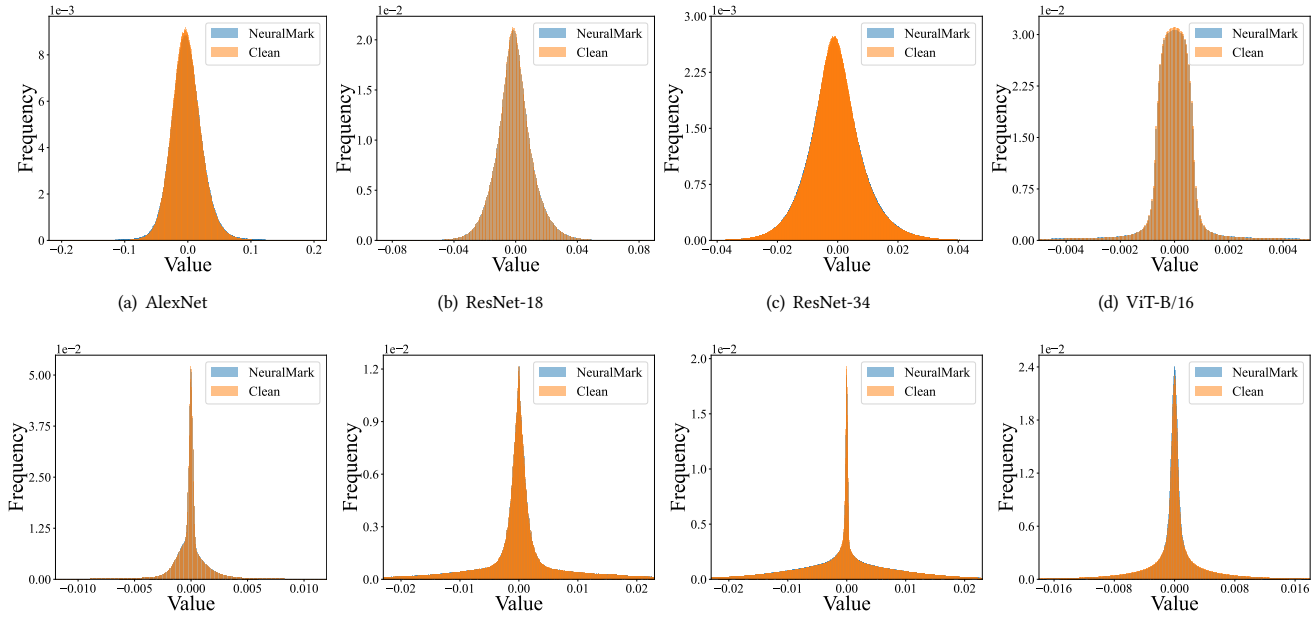


Figure 6: Comparison of parameter overlap ratio with different filter rounds on CIFAR-100 using ResNet-18.

6.5 Additional Analysis

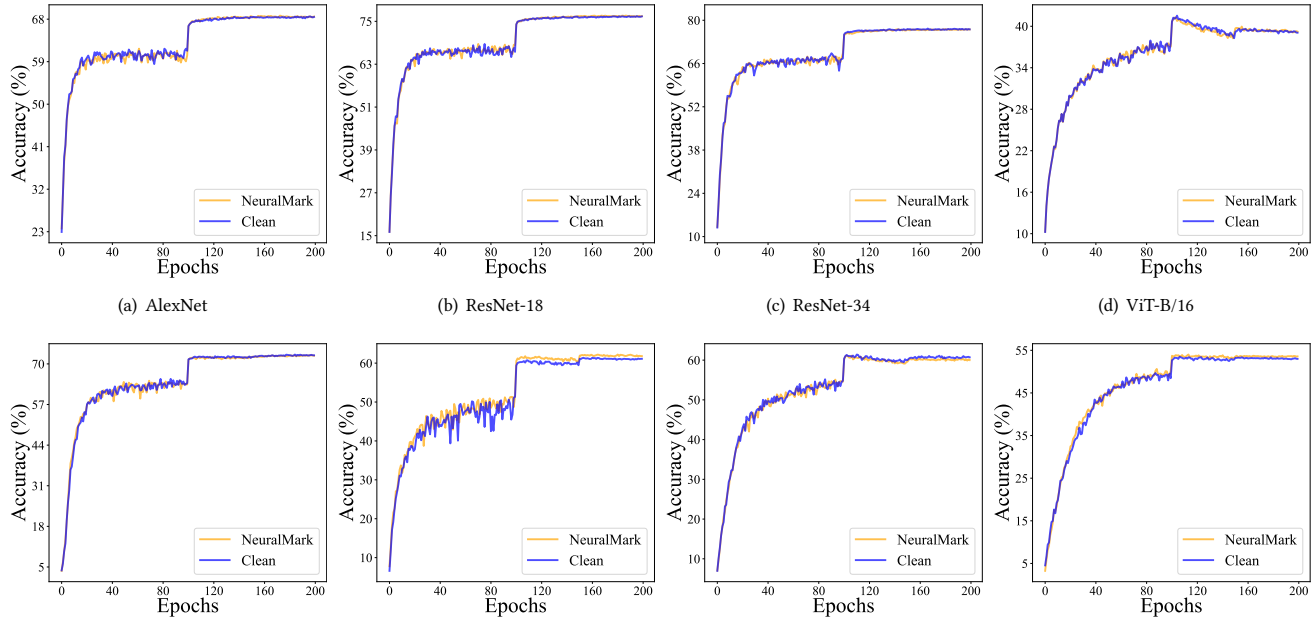
Question 9. How does NeuralMark affect the parameter distribution?



(a) AlexNet (b) ResNet-18 (c) ResNet-34 (d) ViT-B/16

(e) VGG-16 (f) MobileNet-V3-L (g) GoogLeNet (h) Swin-V2-B

Figure 7: Comparison of parameter distributions on CIFAR-100 with distinct architectures.



(a) AlexNet (b) ResNet-18 (c) ResNet-34 (d) ViT-B/16

(e) VGG-16 (f) MobileNet-V3-L (g) GoogLeNet (h) Swin-V2-B

Figure 8: Comparison of model performance convergence across distinct architectures on CIFAR-100.

To assess the influence of NeuralMark on the parameter distribution, Figure 7 present the parameter distributions on the CIFAR-100 dataset with various architectures. As can be seen, the parameter distributions of Clean and NeuralMark are nearly indistinguishable. Thus, it is challenging for adversaries to detect the embedded watermarks within the model.

Question 10. How does NeuralMark influence the performance convergence?

To examine the impact of NeuralMark on model performance convergence, Figure 8 show the results on the CIFAR-100 dataset with various architectures. We find that the performance curves of

Table 11: Comparison of the effects of average pooling on resistance to fine-tuning and pruning attacks using ResNet-18. Values (%) inside and outside the bracket are watermark detection rate and classification accuracy, respectively.

Method	CIFAR-100 to CIFAR-10 Fine-tuning (Learning Rate)			CIFAR-100 Pruning (Pruning Ratio)		
	0.001	0.005	0.01	40%	60%	80%
NeuralMark (w/o AP)	93.26 (100)	92.20 (100)	90.68 (81.64)	71.82 (90.62)	57.50 (78.51)	16.14 (69.92)
NeuralMark	93.74 (100)	92.25 (100)	91.25 (96.87)	69.86 (100)	43.88 (99.21)	9.85 (99.21)

Table 12: Comparison of average time cost (in seconds) on CIFAR-100 using ResNet-18. Here, R is the number of filtering rounds.

Method	Clean	NeuralMark ($R = 1$)	NeuralMark ($R = 2$)	NeuralMark ($R = 3$)	NeuralMark ($R = 4$)	VanillaMark	GreedyMark	VoteMark
Time (s)	23.60	24.49	24.94	25.01	25.19	24.34	47.43	35.17

Clean and NeuralMark exhibit a similar trend of change and are closely aligned, indicating that NeuralMark does not negatively affect the convergence of model performance.

Question 11. How does average pooling impact NeuralMark?

To verify the efficacy of average pooling, we compare NeuralMark with its variant without average pooling, *i.e*, NeuralMark w/o AP. As shown in Table 11, both versions resist fine-tuning attacks at lower learning rates. However, at a learning rate of 0.01, the detection rate for NeuralMark (w/o AP) drops to 81.64%, below the security boundary, while NeuralMark maintains at 96.87%. In addition, the detection rate of NeuralMark (w/o AP) rapidly declines with increasing pruning rates, reaching 69.92% at an 80% pruning rate, while NeuralMark achieves 99.21%. Those results confirm that average pooling enhances resistance to both fine-tuning and pruning attacks.

Question 12. Does NeuralMark impose a significant additional computational burden during training?

Table 12 list the average time cost (in seconds) per training epoch over five epochs on the CIFAR-100 dataset using ResNet-18. NeuralMark's running time is comparable to that of Clean and VanillaMark, highlighting the efficiency of NeuralMark. Also, NeuralMark outperforms GreedyMark in terms of speed due to GreedyMark's reliance on costly sorting operations for parameter selection. Moreover, NeuralMark demonstrates significantly faster running times compared to VoteMark, as it avoids the multiple rounds of watermark embedding loss calculations required by VoteMark. Those results highlight the superior efficiency of NeuralMark.

Question 13. How do the watermark embedding layers impact the model's performance?

To investigate the impact of watermark embedding layers on the model performance, we randomly choose four individual layers and all layers from ResNet-18 for watermark embedding. Table 13 presents the results on the CIFAR-100 dataset, showing that embedding different layers or all layers does not significantly affect the model performance.

Table 13: Comparison of classification accuracy (%) on different watermarking layers on CIFAR-100 using ResNet-18. Here, Layers 1-4 denote randomly chosen layers, while All Layer refers to all layers. Watermark detection rates are omitted as they all reach 100%.

Watermarking Layer	Layer 1	Layer 2	Layer 3	Layer 4	All Layer
Accuracy	76.51	76.68	76.30	76.73	75.86

Question 14. What is the effect of varying the watermark length on model performance?

To evaluate the influence of watermark length on the model performance, we set watermark lengths to 64, 128, 256, 512, 1024, and 2048, respectively. Table 14 lists the results on the CIFAR-100 dataset using ResNet-18, indicating that NeuralMark can achieve a 100% detection rate with various watermark lengths while preserving nearly lossless model performance.

Table 14: Comparison of classification accuracy (%) for distinct watermark lengths on CIFAR-100 using ResNet-18. Watermark detection rates are omitted as they all reach 100%.

Watermark Length	64	128	256	512	1024	2048
Accuracy	75.84	75.90	76.46	76.18	76.51	76.27

7 Conclusion

In this paper, we present NeuralMark, a method designed to enhance the robustness of weight-based NNW. 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. This design cleverly intertwines the embedding parameters with the hash function, providing robust protection against both forging and overwriting attacks. Moreover, the incorporation of average pooling provides resilience against fine-tuning and pruning attacks, ensuring comprehensive defense without compromising model performance. We provide a theoretical analysis of NeuralMark's security boundary. Extensive experiments on various datasets, architectures, and tasks confirm NeuralMark's effectiveness and robustness. In the future, we plan to extend NeuralMark to more complex scenarios, for instance, federated learning [47].

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. arXiv preprint arXiv:2303.08774 (2023).

[2] Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. 2018. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In USENIX Security. 1615–1631.

[3] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. 2023. Qwen technical report. arXiv preprint arXiv:2309.16609 (2023).

[4] Mihir Bellare and Phillip Rogaway. 1993. Random oracles are practical: A paradigm for designing efficient protocols. In CCS. 62–73.

[5] Ben Cottier, Robi Rahman, Loredana Fattorini, Nestor Maslej, and David Owen. 2024. The rising costs of training frontier AI models. arXiv preprint arXiv:2405.21015 (2024).

[6] Alexey Dosovitskiy. 2021. An image is worth 16x16 words: Transformers for image recognition at scale. In ICLR.

[7] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. arXiv preprint arXiv:2407.21783 (2024).

[8] Morris J Dworkin. 2015. SHA-3 standard: Permutation-based hash and extendable-output functions. (2015).

[9] Lixin Fan, Kam Woh Ng, and Chee Seng Chan. 2019. Rethinking deep neural network ownership verification: Embedding passports to defeat ambiguity attacks. In NeurIPS, Vol. 32.

[10] Lixin Fan, Kam Woh Ng, Chee Seng Chan, and Qiang Yang. 2021. Deepipr: Deep neural network ownership verification with passports. IEEE Transactions on Pattern Analysis and Machine Intelligence 44, 10 (2021), 6122–6139.

[11] Li Fei-Fei, Rob Fergus, and Pietro Perona. 2004. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. In CVPRW. 178–178.

[12] Le Feng and Xinpeng Zhang. 2020. Watermarking neural network with compensation mechanism. In KSEM. 363–375.

[13] Hossein Gholamalinezhad and Hossein Khosravi. 2020. Pooling methods in deep neural networks, a review. arXiv preprint arXiv:2009.07485 (2020).

[14] Gregory Griffin, Alex Holub, Pietro Perona, et al. 2007. Caltech-256 object category dataset. Technical Report. Technical Report 7694, California Institute of Technology Pasadena.

[15] Chaoxiang He, Xiaofan Bai, Xiaojing Ma, Bin Benjamin Zhu, Pingyi Hu, Jiayun Fu, Hai Jin, and Dongmei Zhang. 2024. Towards Stricter Black-box Integrity Verification of Deep Neural Network Models. In ACM MM.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In CVPR. 770–778.

[17] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. 2019. Searching for mobilenetv3. In ICCV. 1314–1324.

[18] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In ICLR.

[19] Ju Jia, Yueming Wu, Anran Li, Siqi Ma, and Yang Liu. 2022. Subnetwork-lossless robust watermarking for hostile theft attacks in deep transfer learning models. IEEE Transactions on Dependable and Secure Computing (2022).

[20] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. Technical Report. Technical report, University of Toronto.

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In NeurIPS, Vol. 25.

[22] Ya Le and Xuan Yang. 2015. Tiny imagenet visual recognition challenge. CS 231N 7, 7 (2015), 3.

[23] Erwan Le Merrer, Patrick Perez, and Gilles Trédan. 2020. Adversarial frontier stitching for remote neural network watermarking. Neural Computing and Applications 32, 13 (2020), 9233–9244.

[24] Fangqi Li, Lei Yang, Shilin Wang, and Alan Wee-Chung Liew. 2022. Leveraging Multi-task Learning for Umambiguous and Flexible Deep Neural Network Watermarking.. In SafeAI@ AAAI.

[25] Fangqi Li, Haodong Zhao, Wei Du, and Shilin Wang. 2024. Revisiting the Information Capacity of Neural Network Watermarks: Upper Bound Estimation and Beyond. In AAAI. 21331–21339.

[26] Peixuan Li, Pengzhou Cheng, Fangqi Li, Wei Du, Haodong Zhao, and Gongshen Liu. 2023. Plmmark: a secure and robust black-box watermarking framework for pre-trained language models. In AAAI, Vol. 37. 14991–14999.

[27] Yue Li, Lydia Abady, Hongxia Wang, and Mauro Barni. 2021. A feature-map-based large-payload DNN watermarking algorithm. In IWDW. 135–148.

[28] Yue Li, Benedetta Tondi, and Mauro Barni. 2021. Spread-transform dither modulation watermarking of deep neural network. Journal of Information Security and Applications 63 (2021), 103004.

[29] Jian Han Lim, Chee Seng Chan, Kam Woh Ng, Lixin Fan, and Qiang Yang. 2022. Protect, show, attend and tell: Empowering image captioning models with ownership protection. Pattern Recognition 122 (2022), 108285.

[30] Hanwen Liu, Zhenyu Weng, and Yuesheng Zhu. 2021. Watermarking Deep Neural Networks with Greedy Residuals.. In ICML. 6978–6988.

[31] Hanwen Liu, Zhenyu Weng, Yuesheng Zhu, and Yadong Mu. 2023. Trapdoor normalization with irreversible ownership verification. In ICML. PMLR, 22177–22187.

[32] Yiheng Liu, Tianle Han, Siyuan Ma, Jiayue Zhang, Yuanyuan Yang, Jiaming Tian, Hao He, Antong Li, Mengshen He, Zhengliang Liu, et al. 2023. Summary of chatgpt-related research and perspective towards the future of large language models. Meta-Radiology (2023), 100017.

[33] Ze Liu, Han Hu, Yutong Lin, Zhuliang Yao, Zhenda Xie, Yixuan Wei, Jia Ning, Yue Cao, Zheng Zhang, Li Dong, et al. 2022. Swin transformer v2: Scaling up capacity and resolution. In CVPR. 12009–12019.

[34] Ilya Loshchilov, Frank Hutter, et al. 2017. Fixing weight decay regularization in adam. arXiv preprint arXiv:1711.05101 5 (2017).

[35] Nils Lukas, Edward Jiang, Xinda Li, and Florian Kerschbaum. 2022. Sok: How robust is image classification deep neural network watermarking?. In S&P. IEEE, 787–804.

[36] Ben Mann, N Ryder, M Subbiah, J Kaplan, P Dhariwal, A Neelakantan, P Shyam, G Sastry, A Askell, S Agarwal, et al. 2020. Language models are few-shot learners. arXiv preprint arXiv:2005.14165 1 (2020).

[37] Jekaterina Novikova, Ondřej Dušek, and Verena Rieser. 2017. The E2E dataset: New challenges for end-to-end generation. arXiv preprint arXiv:1706.09254 (2017).

[38] 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. In NeurIPS, Vol. 32.

[39] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. OpenAI blog 1, 8 (2019), 9.

[40] Bita Darvish Rouhani, Huili Chen, and Farinaz Koushanfar. 2019. Deepsigns: an end-to-end watermarking framework for protecting the ownership of deep neural networks. In ASPLOS, Vol. 3.

[41] Karen Simonyan and Andrew Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. In ICLR.

[42] Yuchen Sun, Tianpeng Liu, Panhe Hu, Qing Liao, Shaojing Fu, Nenghai Yu, Deke Guo, Yongxiang Liu, and Li Liu. 2023. Deep intellectual property protection: A survey. arXiv preprint arXiv:2304.14613 (2023).

[43] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In CVPR. 1–9.

[44] Yusuke Uchida, Yuki Nagai, Shigeyuki Sakazawa, and Shin'ichi Satoh. 2017. Embedding watermarks into deep neural networks. In ACM ICMR. 269–277.

[45] Arthur F Webster and Stafford E Tavares. 1985. On the design of S-boxes. In Eurocrypt. Springer, 523–534.

[46] Mingfu Xue, Yushu Zhang, Jian Wang, and Weiqiang Liu. 2021. Intellectual property protection for deep learning models: Taxonomy, methods, attacks, and evaluations. IEEE Transactions on Artificial Intelligence 3, 6 (2021), 908–923.

[47] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. 2019. Federated machine learning: Concept and applications. ACM Transactions on Intelligent Systems and Technology 10, 2 (2019), 1–19.

[48] Sergey Zagoruyko. 2016. Wide residual networks. In BMVC.

[49] Jie Zhang, Dongdong Chen, Jing Liao, Weiming Zhang, Gang Hua, and Nenghai Yu. 2020. Passport-aware normalization for deep model protection. In NeurIPS, Vol. 33. 22619–22628.

[50] Renjie Zhu, Xinpeng Zhang, Mengte Shi, and Zhenjun Tang. 2020. Secure neural network watermarking protocol against forging attack. EURASIP Journal on Image and Video Processing 2020 (2020), 1–12.

A Proof for Proposition 5.1

Proposition 5.1. *Under the assumption that the hash function produces uniformly distributed outputs [4], 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* $\frac{1}{2^n}\sum_{i=0}^{n-\lceil \rho n \rceil}\binom{n}{i}$.

Proof. Since the hash function produces uniformly distributed outputs, each bit of the counterfeit watermark matches the corresponding bit of the extracted watermark from model parameters with a probability of $\frac{1}{2}$. The number of matching bits follows a binomial distribution with parameters n and $p = \frac{1}{2}$. To achieve a detection rate of at least ρ, the adversary needs at least $\lceil \rho n \rceil$ bits to match out of n bits. Thus, the probability of having at least $\lceil \rho n \rceil$ matching bits is given by

$$\begin{aligned} \Pr\left[X \geq \lceil \rho n \rceil\right] &= \sum_{i=\lceil \rho n \rceil}^{n} \binom{n}{i}\left(\frac{1}{2}\right)^{i}\left(\frac{1}{2}\right)^{n-i} \\ &= \frac{1}{2^n}\sum_{i=\lceil \rho n \rceil}^{n}\binom{n}{i} = \frac{1}{2^n}\sum_{i=0}^{n-\lceil \rho n \rceil}\binom{n}{i}. \end{aligned} \tag{4}$$

Accordingly, the probability of an adversary forging a counterfeit watermark that achieves a watermark detection rate of at least ρ (*i.e.*, $\geq \rho$) is upper-bounded by $\frac{1}{2^n}\sum_{i=0}^{n-\lceil \rho n \rceil}\binom{n}{i}$.

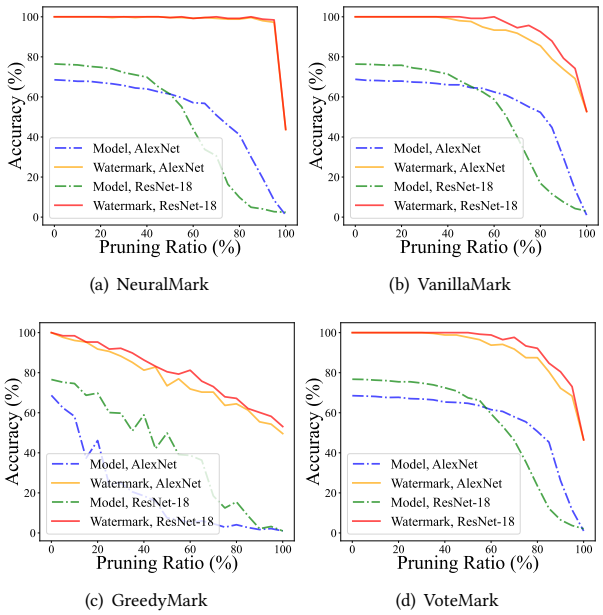


Figure 9: Comparison of resistance to pruning attacks at various pruning ratios on CIFAR-100 using AlexNet and ResNet-18.

B Additional Results of Pruning Attacks

Figure 9-11 provide the results from pruning attacks conducted on the CIFAR-100, Caltech-101, and Caltech-256 datasets, respectively. As can be seen, as the pruning ratio increases, the performance of NeuralMark degrades while the detection rate remains nearly 100%. This indicates NeuralMark's robustness against pruning attacks. Those results collectively suggest NeuralMark exhibits superior robustness in resisting pruning attacks compared to other methods.

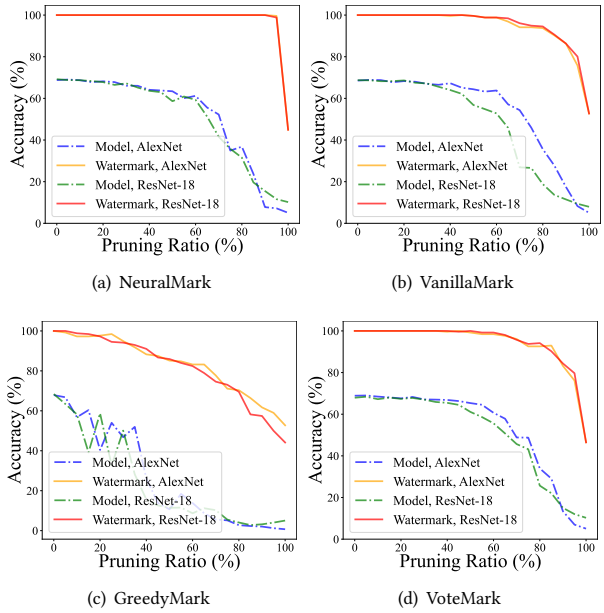


Figure 10: Comparison of resistance to pruning attacks at various pruning ratios on Caltech-101 using AlexNet and ResNet-18.

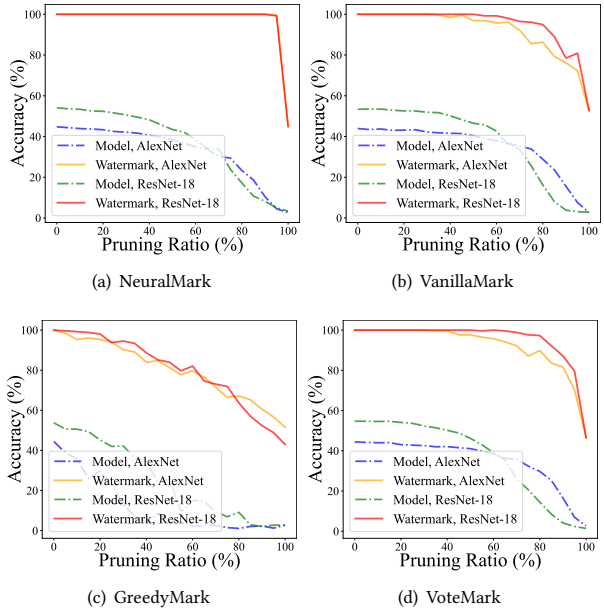


Figure 11: Comparison of resistance to pruning attacks at various pruning ratios on Caltech-256 using AlexNet and ResNet-18.