NNSplitter: An Active Defense Solution for DNN Model via Automated Weight Obfuscation

Tong Zhou [1] **Yukui Luo** [1] **Shaolei Ren** [2] **Xiaolin Xu** [1]

Abstract

As a type of valuable intellectual property (IP), deep neural network (DNN) models have been protected by techniques like watermarking. However, such passive model protection cannot fully prevent model abuse. In this work, we propose an active model IP protection scheme, namely NNSplitter, which actively protects the model by splitting it into two parts: the *obfuscated model* that performs poorly due to weight obfuscation, and the *model secrets* consisting of the indexes and original values of the obfuscated weights, which can only be accessed by authorized users with the support of the trusted execution environment. Experimental results demonstrate the effectiveness of NNSplitter, e.g., by only modifying 275 out of over 11 million (i.e., 0.002%) weights, the accuracy of the obfuscated ResNet-18 model on CIFAR-10 can drop to 10%. Moreover, NNSplitter is stealthy and resilient against norm clipping and fine-tuning attacks, making it an appealing solution for DNN model protection. The code is available at: https://github.com/Tongzhou0101/NNSplitter.

1. Introduction

Despite the success of deep neural networks (DNNs) in various applications (Duong et al., 2019; Wang et al., 2018), building a DNN model with high accuracy is costly, i.e., requiring a large number of labeled samples and massive computational resources (Jiang et al., 2020). As a result, a high-performance DNN model presents valuable intellectual property (IP) of the model owner, which should naturally be adequately protected against potential attacks. However, recent studies have demonstrated that millions of on-device ML models are vulnerable to model IP attacks (Sun et al., 2021), wherein the attacker can extract the model and deploy it on unauthorized devices. Such unauthorized usage leads to significant financial losses for the model owners.

Requirements	Definitions
Effectiveness	The obfuscated model exhibits poor performance (e.g., random-guess accuracy).
Efficiency	The number of model secrets stored in the secure space should be minimized.
Integrity	The functionality of the model is preserved for the legitimate users.
Resilience	The obfuscated model should be resilient against potential attack surfaces.
Stealthiness	The obfuscated weights should be indistinguishable from normal weights.

Table 1. The design requirements for an efficient model protection scheme, and the guidance for our proposed NNSplitter.

Several studies have addressed the issue of DNN model protection, which can be broadly classified into two categories: passive protection (after IP infringement) and active protection (before IP infringement). Although passive protection techniques, e.g., watermarking, help model owners declare the ownership and guard their rights (Yang et al., 2021; Zhang et al., 2018), they cannot effectively prevent unauthorized usage as the model can perform very well in most cases. Thus, attackers are still motivated to steal the well-performed model and use it without the knowledge of the model owner.

In contrast, active protection only allows legitimate users to use the well-performed model, while intentionally degrading the model functionality for attackers, thus protecting the interests of the model owner (Chakraborty et al., 2020; Fan et al., 2019; Zhou et al., 2022). Nonetheless, such an advantage of the active protection methods is not free, which either requires hardware support, e.g., a hardware root-of-trust (Chakraborty et al., 2020), or introduces extra model parameters (Fan et al., 2019). Moreover, the existing active protection approaches are not generic, i.e., they require special training strategies for model protection, rendering

[1]Northeastern University, Boston, MA [2]UC Riverside, Riverside, CA. Correspondence to: Xiaolin Xu <x.xu@northeastern.edu>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

them inapplicable to pre-trained models. It is also worth noting that some fault injection methods can also cause accuracy deterioration (Liu et al., 2017), using software-oriented (Rakin et al., 2019) or hardware-oriented (Luo et al., 2021; Rakin et al., 2021) attacking schemes. However, the design of those works is from the perspective of an attacker, which can not satisfy requirements (shown in Tab. 1) for active protection, with detailed discussion in Sec. 2.4.

Considering these limitations of existing defense strategies, we are motivated to develop a generic active model IP protection scheme. Specifically, we propose to split the victim model into an obfuscated model and model secrets, which should fulfill the requirements detailed in Tab. 1. The design of such a scheme presents the following substantial challenges (**C**). **C1:** Given the limited size of secure memory we can leverage, e.g., the trusted execution environment (TEE) (Costan & Devadas, 2016), the stored model secrets need to be kept small, while there are millions of, if not more, weights in modern DNN models. **C2:** The model functionality should be preserved for legitimate users. **C3:** The obfuscated weights should be imperceptible and not easily identified by attackers. **C4:** Attackers can not significantly improve the degraded accuracy with reasonable efforts.

To address **C1**, our proposed scheme, namely *NNSplitter*, generates a mask that selectively obfuscates weights within a small range. This range is chosen to be small enough, so that the original values of the obfuscated weights can be replaced by a single value, thereby reducing the storage requirements for model secrets. To achieve this goal, we utilize a reinforcement learning (RL) algorithm to design a controller that efficiently identifies important filters with significant influences on model predictions. By focusing on these filters, we can minimize the number of obfuscated weights while still achieving a significant accuracy drop. For **C2**, we profile the model weights and adjust the aforementioned small range to ensure that the original model accuracy can be preserved, after applying the obfuscated weights restoration rule (details are given in Sec. 3.1). Besides, we set a limit to ensure the obfuscated weights remain within the original weight range to avoid being identified by attackers (addressing **C3**). Last, we force the weight changes to spread across various layers to increase the resilience against potential attack surfaces to improve accuracy (addressing **C4**).

Overall, NNSplitter achieves model IP protection by splitting a victim model into two parts: the *obfuscated model* and the *model secrets*. Specifically, the obfuscated model is vulnerable to model extraction, but its degraded accuracy resulting from weight obfuscation renders it practically useless, effectively mitigating the vulnerability. Meanwhile, the model secrets are secured by TEE to provide authorized inference, which can only be accessed by authorized users.

The contributions of this work are as follows:

- We systematically define the requirements for active model protection and propose NNSplitter that can automatically split the victim model into the obfuscated model and model secrets with all of these design requirements fulfilled.
- The accuracy of the obfuscated model can drop to random guess by modifying only 0.001% weights ($\sim$ 300) of the victim model, which is hardware-friendly due to low secure memory requirement.
- We demonstrate that the proposed NNSplitter is resilient against potential attacks, including norm clipping and fine-tuning attacks.

2. Related Works and Background

2.1. Threat Model

To ensure highly effective model protection, we consider a strong attacker who has the capability to extract the exact victim DNN model, including its architecture and model parameters, using techniques like in-memory extraction mentioned in (Sun et al., 2021). For example, attackers can download a mobile application built with a DNN model, decompile it, extract the model file, and deploy it on their own devices. Besides, we assume the attackers only have limited training data; otherwise, they can train a competitive model on their own, without strong incentives to steal the victim model. By considering these scenarios, we aim to design a model protection scheme that can effectively safeguard the victim model IP against such strong attackers.

2.2. Trusted Execution Environment

While passive model IP protection fails to protect models from being stolen or used, we envision the TEE (e.g., ARM TrustZone on mobile devices (Ngabonziza et al., 2016)) as a promising solution to achieve active model protection. TEE provides a physical isolation scheme in the hardware devices that separates memory into the normal (untrusted) world and the secure (trusted) world, where the normal world can communicate with the secure world by invoking a secure monitor call (Ye et al., 2018). This setup ensures that only legitimate users can access the secure world, while attackers are blocked. Given the effectiveness of TEE in model protection as demonstrated in previous works (Chen et al., 2019; Sun et al., 2023), we adopt the TEE implementation scheme following (Sun et al., 2023) without delving into the technical details or considering the vulnerability of TEE (e.g., side-channel attacks), as it is not the primary focus in this work.

It is important to note that the secure memory of TEE is limited, e.g., $\sim$ 10 MB for trusted applications (Sun et al.,

2023). On the other hand, the size of state-of-the-art (SOTA) DNN models continues increasing, e.g., large models like ResNet-101 exceed 155M parameters (He et al., 2016). To accommodate this limitation, our approach NNSplitter aims to obfuscate as few weights as possible, to minimize the overhead on secure memory usage.

2.3. Model IP Protection

The existing literature has actively addressed model security issues on edge devices (Sun et al., 2021; Xu et al., 2019; Shukla et al., 2021), and demonstrated that attackers can easily extract the model even without sophisticated skills (Sun et al., 2021). As discussed above, the existing passive model protection methods like watermarking (Yang et al., 2021) have limitations in fully preventing model piracy. On the other hand, active protection methods, such as model encryption (Al-Garadi et al., 2020), have been proposed where the model files are encrypted and stored in memory. However, the encrypted model needs to be decoded at runtime for inference, which can still be vulnerable to attacks.

To enhance the model IP security, Chakraborty *et al.*leverage secure hardware support and propose a key-dependent back-propagation algorithm to train a DNN architecture with the weight space obfuscated (Chakraborty et al., 2020). After obfuscation, only authorized users are allowed to use the model on trusted hardware with the key embedded on-chip, while the model accuracy will drop significantly if attackers extract the model and deploy it on other devices. However, this method requires hardware modification and cannot be generally used to protect pre-trained models. Similarly, Fan *et al.* propose a method to protect the model IP by embedding a passport layer within the DNN model, so that the DNN inference performance of an original task will be significantly deteriorated due to forged passports (Fan et al., 2019). However, this work aims to defend against ambiguity attacks, and can only be applied to the models already embedded with watermarks. These existing approaches provide valuable insights into model protection, but they either require hardware modifications or have specific limitations in their applicability.

2.4. Difference from Fault Injection

A key point of active model protection is to introduce performance degradation (e.g., accuracy drop) into the protected model. Although the objective is similar to fault injection attacks that manipulate the DNN model parameters to cause abnormal inference (Liu et al., 2017), the fundamental design requirements are largely different: (i) **Stealthiness:** fault injection attacks do not consider stealthiness in model manipulations, which introduce extremely large magnitudes changes and can be easily distinguished and removed by applying weights range restriction (Chen et al., 2021; Liu et al., 2017). (ii) **Resilience:** most fault injection attacks only target the most direct parameters of outputs, e.g., those in the last layer. However, such an attack is not resilient against fine-tuning. Also, although existing attacks like bit-flip (Rakin et al., 2019) modify the weight bits in different layers to degrade model accuracy, such gradient-ranking-based attacks can be mitigated by weights reconstruction (Li et al., 2020). Moreover, bit-flip targets the quantized DNN models, where the weight magnitude is constrained based on the quantization method, while how to ensure the stealthiness and resilience of attacks on the floating-point precision DNN models is significantly under-explored.

In sharp contrast to these studies on attacks, we rethink and address all the aforementioned design limitations from a defense perspective. Specifically, we mainly explore an active defense scheme leveraging hardware support from the TEE, to actively prevent attackers from obtaining functional DNN models and make such model extraction attacks less motivated. Our work is orthogonal to the existing literature and can be generally applied to any pre-trained models.

3. Our Proposed Method: NNSplitter

This section presents our proposed active DNN model protection method, NNSplitter, which meets the requirements of effectiveness, efficiency, integrity, resilience, and stealthiness, as described in Tab. 1.

The overview of NNSplitter is illustrated in Fig. 1, including the offline model obfuscation and the online secured inference. In the offline phase, taking the pre-trained DNN model as input (①), the mask generator profiles the weight distribution to determine the parameters of the mask following certain rules (Sec. 3.1). The mask parameters and the DNN model will be fed into the optimization loop (②) along with the dataset. In the loop, we build a RL-based controller to help form a filter-wise mask, which is used to guide the weight obfuscation optimized by the model optimizer. Then the negative accuracy evaluated on the test dataset will serve as a reward to optimize the controller. When the reward converges, i.e., the accuracy stops decreasing, the optimization loop will generate two parts — the obfuscated model (④) deployed in the normal world (untrusted memory), and the model secrets (③) that include the indexes and the original values of the obfuscated weights stored in the secure world (trusted memory).

During online secured inference, the model is executed layer by layer. At each layer, the obfuscated weights are used to compute an output feature map, which may contain errors in certain output channels. These errors are intentionally propagated to subsequent layers, resulting in a substantial drop in accuracy. This mechanism effectively prevents unauthorized use by attackers, as the model they extract from

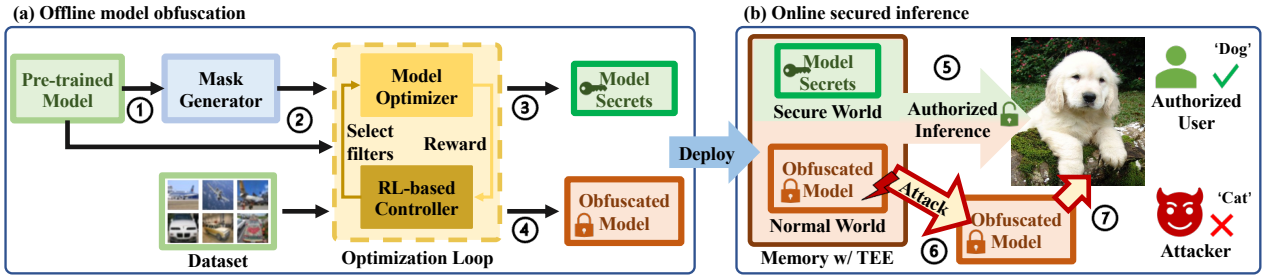

Figure 1. An overview of NNSplitter. (a) Offline model obfuscation: NNSplitter splits the pre-trained model into two parts once the reward is converged, i.e., the obfuscated model and the model secrets (including the indexes and the original weight values). (b) Online secured inference: an attacker can only extract the obfuscated model stored in the normal world, which exhibits poor performance. However, the original model accuracy can be preserved by integrating the model secrets stored in the secure world of the victim's device.

the normal world will perform poorly due to the presence of obfuscated weights (⑥ and ⑦). However, in the secure world, the model secrets are utilized to correct these errors in the specific output channels, ensuring that the model functions as intended for legitimate users who have access to the secure world (⑤). In this way, the majority of the DNN inference computation is performed in the main memory of the normal world, reducing the computation overhead within the secure world.

3.1. Problem Formulation

Given a pre-trained DNN model $\mathcal{M}$ containing L convolutional/fully connected layers with weights $\mathbf{W} := \{\mathbf{W}^{(l)}\}_{l=1}^{L}$, we aim to find the optimal *weight changes* $\Delta\mathbf{W}$ (the same size as $\mathbf{W}$) that maximize the classification loss function $\mathcal{L}_{\mathcal{M}}$. For simplicity, we denote each element in $\mathbf{W}$ and $\Delta\mathbf{W}$ as w_i and Δw_i, respectively, where $i \in [1, N]$ and N is the total number of model weights. Upon achieving the optimal weight obfuscation, we store the indexes of non-zero Δw_i and original w_i to preserve the performance of the victim model for legitimate users.

Mask Generator. To reduce the secure storage requirement, we design a mask $\mathbf{M}$ for $\Delta\mathbf{W}$ to determine the weights to be obfuscated, which is defined by:

$$\mathbf{M}(w_i) = \begin{cases} 1 & if\ |w_i - c| \leq \epsilon, \\ 0 & otherwise, \end{cases} \tag{1}$$

where c and ϵ are both controllable hyper-parameters. Using this mask, we can refine the weight changes $\Delta\mathbf{W}' := \Delta\mathbf{W} \odot \mathbf{M}$, where $\odot$ denotes element-wise multiplication. The benefits of the mask design are two-fold: **(i)** $\mathbf{M}$ only allows weights in the range $[c - \epsilon, c + \epsilon]$ to be obfuscated. By selecting a small ϵ, we ensure that the obfuscated weights are close to a constant value c. This allows us to store a single value for these obfuscated weights instead of multiple different values, thus saving the secure space while preserving the model functionality; **(ii)** by carefully selecting c, we can distribute the weight obfuscation across various layers, significantly improving the resilience against the potential attack surfaces, such as fine-tuning (see results in Sec. 5.2). Besides, we apply ℓ_0-norm regularization to $\Delta\mathbf{W}'$ to further save the secure storage space.

Model Optimizer. To improve the stealthiness of weight obfuscation, we restrict the values of *obfuscated weights*, i.e., $\mathbf{W} + \Delta\mathbf{W}'$, within the original value range of $\mathbf{W}$, which is achieved by the hyperparameters α and β in Eq. (2). Thus, the optimal $\Delta\mathbf{W}'$ can be found by minimizing the loss function $\mathcal{L}(\Delta\mathbf{W}')$:

$$\begin{aligned} \min_{\Delta\mathbf{W}'} \mathcal{L}(\Delta\mathbf{W}') &= -\mathcal{L}_{\mathcal{M}}\left(f\left(\boldsymbol{x}; \mathbf{W} + \Delta\mathbf{W}'\right), \boldsymbol{y}\right) + \lambda \left\|\Delta\mathbf{W}'\right\|_0 \\ \text{s.t.} \quad & \alpha * \min\{w_i\} \leq w_i + \Delta w_i' \leq \beta * \max\{w_i\} \ \ \forall i, \end{aligned} \tag{2}$$

where f denotes the functionality of the DNN model $\mathcal{M}$, $\boldsymbol{x}$ is the training samples with $\boldsymbol{y}$ being the corresponding labels, and λ controls the sparsity of weight changes.

However, considering the SOTA DNN models consisting of millions of parameters, only using ℓ_0-norm to minimize the number of weight changes is not sufficient. Inspired by the fact that the importance of different filters varies (You et al., 2019), e.g., the filters learning the background features contribute less compared to these learning the object edge, we propose to embed the filter-wise weights selection strategy into the mask design. This strategy involves adding weight changes only to selected important filters while still satisfying the constraints in Eq. (1). By doing so, we can further reduce the storage space required for weight obfuscation, while still achieving the desired level of accuracy degradation.

Nonetheless, manually selecting filters to design an optimal filter-wise mask is impractical due to the large number of filters in SOTA DNNs. Thus, we propose a RL-based controller to automatically select the optimal filters.

3.2. RL-based Controller

As an important component of NNSplitter, the RL-based controller aims to form a filter-wise mask. While a straightforward approach would be to use the controller to generate all the hyperparameters required by the design of $\mathbf{M}$, including c and ϵ in Eq. (1), this design principle would increase the complexity and optimization difficulty of developing the controller. To overcome the challenges while maintaining the effectiveness of the controller, we leverage the domain knowledge about the distribution of the model weights to determine the values of these two hyper-parameters (see details in Sec. 4.2), and leave the difficult part, i.e., selecting important filters, to the controller.

The developed controller consists of three parts: an encoder for encoding the initialized state, a policy network for decision-making, and decoders for different layers to decode the output of policy networks into filter indexes. In this controller, an agent selects a filter with index k for each layer (i.e., actions), where $k \in [1, K^{(l)}]$ and $K^{(l)}$ denotes the number of filters (i.e., output channels) of the l-th layer. Since the state $K^{(l)}$ is determined by the architecture of the victim model $\mathcal{M}$, the environment is static for the agents. To select n filters for each layer (n could be 1), we will have n agents making $n * L$ actions in total, denoted as $a_{1:n*L}$. All agents will share the same controller with weights θ, which will be optimized by maximizing the expected reward $J(\theta)$:

$$J(\theta) = E_{\pi(a_{1:n*L};\theta_c)}[R], \tag{3}$$

where $\pi(\cdot)$ denotes the probabilities of taken actions given θ, and reward R is constructed by the negative inference accuracy of the obfuscated model, defined by Eq. (4):

$$R = -ACC\left(f\left(\boldsymbol{x^t}; \mathbf{W} + \Delta\mathbf{W}'\right), \boldsymbol{y^t}\right), \tag{4}$$

where ACC is accuracy, $\boldsymbol{x^t}$ is the validation dataset and $\boldsymbol{y^t}$ denotes the corresponding labels. Considering R is non-differentiable with respect to the controller output, we use a policy gradient method: REINFORCE algorithm (Williams, 1992) to maximize $J(\theta)$, which is the same as minimizing the loss function of the controller:

$$\mathcal{L}_c(\theta) = -\frac{1}{m}\sum_{j=1}^{m}\sum_{t=1}^{n*L}\log\pi\left(a_t;\theta_c\right)\left(R_j - b\right), \tag{5}$$

where m represents the number of trails in each episode of the controller, and b denotes an exponential moving average of the rewards used to reduce the variance for updating θ.

The obfuscated model generation is described in Alg. 1. With the mask parameters c and ϵ obtained from the mask generator (line 1), the initialized controller will first design a filter-wise mask to optimize the victim model by minimizing the Eq. (2) (line 4-8), then the controller use rewards obtained from the victim model to optimize itself (line 9-14). When the reward converges, NNSplitter will output two parts, which are the obfuscated model and model secrets that will be stored in the secure world.

Algorithm 1 Offline obfuscated model generation

Input: pre-trained model $\mathcal{M}$ with weights $\mathbf{W}$; initialized controller with θ; training data $(\boldsymbol{x}, \boldsymbol{y})$; test data $(\boldsymbol{x^t}, \boldsymbol{y^t})$; $K^{(l)}$, α, β, λ.
Parameters: learning rate η_1, η_2.
Output: model secrets (the indexes of $\Delta w_i'$ and c), obfuscated model $\mathcal{M}'$.

1: Feed $\mathcal{M}$ into model generator and obtain c and ϵ
2: **repeat**
// Optimization loop
3: **for** m batches **do**
4: Use controller to generate filter indexes
5: Form filter-wise mask $\mathbf{M}$ and feed into model optimizer
// Optimize $\Delta\mathbf{W}'$
6: **for** training epochs **do**
7: Minimize $\mathcal{L}(\Delta\mathbf{W}')$ ▷ Eq. (2)
8: Update $\Delta\mathbf{W}' \leftarrow \Delta\mathbf{W}' - \eta_1\nabla\mathcal{L}(\Delta\mathbf{W}')$
9: Measure accuracy on $(\boldsymbol{x^t}, \boldsymbol{y^t})$
10: Collect the reward R ▷ Eq. (4)
11: **end for**
// Optimize the controller θ
12: Calculate the average reward b
13: Minimize $\mathcal{L}_c(\theta)$ ▷ Eq. (5)
14: Update controller: $\theta \leftarrow \theta - \eta_2\nabla\mathcal{L}_c(\theta)$
15: **end for**
16: **until** Reward R is converged

4. Experimental Validation

4.1. Experimental Setup

Datasets. We evaluate the effectiveness of NNSplitter on models trained with three datasets: Fashion-MNIST (Xiao et al., 2017), CIFAR-10, and CIFAR-100 (Krizhevsky et al., 2009). For Fashion-MNIST, there are 60k 28×28 grayscale images from 10 classes in the training dataset and 10k images in the test dataset. Besides, CIFAR-10/100 both have 50k training images and 10k test images of 32×32, except that CIFAR-10 includes 10 classes while CIFAR-100 has 100 classes.

Baseline DNN Models. While NNSplitter applies to any pre-trained models, here we consider several commonly-used DNNs as the proof-of-concept, including VGG-11 (Simonyan & Zisserman, 2015), MobileNet-v2 (Sandler et al., 2018), and ResNet-18/20 (He et al., 2016) trained on the aforementioned datasets. To demonstrate that NNSplitter is

a generic defense solution regardless of the victim model's training strategies, i.e., training-free, we use pre-trained models with weights public online, where the parameter settings (e.g., layer dimensions) could be different for the same DNN class for different datasets. We use the structures and pre-trained weights as they are released online, *despite that they may not reach the best-known accuracy on these datasets*.

Comparison Methods. Since there are no existing works that follow the same settings and objectives as NNSplitter, we propose the following methods for comparison in order to demonstrate its effectiveness. (i) **Random:** instead of using domain knowledge and the RL-based controller to design a filter-wise mask, we assume a model protection method that randomly generates a binary mask to select weights and obfuscate them by optimizing Eq. (2). For a fair comparison, the binary mask will select the same number of obfuscated weights as NNSplitter. (ii) **Base-NNSplitter**: this method randomly selects filters in each layer instead of using the RL-based controller to optimize the selection.

4.2. Hyper-parameters Setting

Weight Constraints. To enhance the stealthiness of weight changes, we use two hyper-parameters α and β in Eq. (2) to ensure that the values of *obfuscated weights* are indistinguishable from the normal weights, thereby avoiding outlier detection. Considering $\min\{w_i\} < 0$ and $\max\{w_i\} > 0$ in general, the values of α and β are in the range (0,1]. Specifically, they are set to 0.95 for the following experiments.

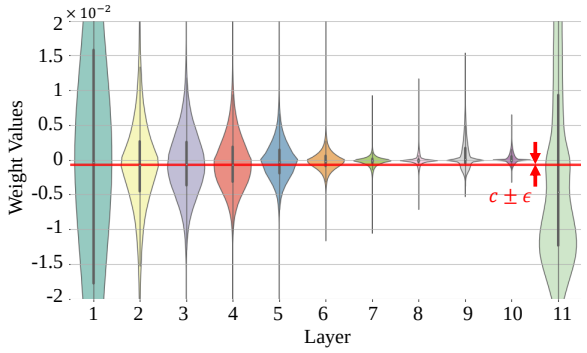

Figure 2. The obfuscated weights spread across all layers, illustrated in a VGG-11 model trained on CIFAR-10 as an example.

Mask Design. The mask design depends on the domain knowledge of the weight distribution. Specifically, to determine the mask hyper-parameters c and ϵ, we profile the weight distribution of each layer and take the average of the median values as the c, which will encourage the weight changes to spread across various layers, as shown in Fig. 2. As for determining ϵ, the principle is to ensure that the accuracy can be preserved when replacing weights in the range of $[c-\epsilon, c+\epsilon]$ with c. Therefore, the closer c is to the median of the total weights, the smaller ϵ should be. Otherwise, the baseline accuracy cannot be restored due to a great precision loss. The details are shown in Tab. 2.

Dataset	Model	Hyper-parameters	
		c	ϵ
Fashion MNIST	VGG-11	-1.7e-3	1e-4
	ResNet-18	-2.3e-4	3e-5
	MobileNet-v2	-3.8e-4	4e-3
CIFAR-10	VGG-11	-7.0e-4	1e-4
	ResNet-18	-1.2e-3	8e-5
	MobileNet-v2	-2.5e-4	1e-4
CIFAR-100	VGG-11	-1.9e-3	5e-4
	ResNet-20	-6.7e-3	6e-4
	MobileNet-v2	-9.4e-4	3e-4

Table 2. The settings of mask hyper-parameters.

Controller Design. The RL-based controller in our approach follows a similar design to the neural architecture search in (Zoph & Le, 2017; Zoph et al., 2018; Pham et al., 2018), i.e., using a recurrent neural network (RNN) to build the policy network, where the embedding dimension and the hidden dimension of the RNN policy network are set to 256 and 512, respectively. Besides, we use one-hot encoding to encode the initialized state as the input of the policy network. For decoding the output of the policy network into filter indexes, we build a decoder for each layer in the DNN victim model with a linear layer, where its output dimension is equal to the number of output channels in the corresponding DNN layer.

4.3. Performance Evaluation

To find the optimal changes added to the pre-trained models, we leverage the designed RL-based controller to select filters in both convolutional layers and fully connected layers. Here, we also refer to each output channel of the fully connected layer as a filter for simplicity. The results of NNSplitter, baseline, and *random* methods are shown in Tab. 3. Following our defined requirements for DNN model protection schemes in Tab. 1, we evaluate the performance of NNSplitter from three perspectives: effectiveness, efficiency, and integrity.

Effectiveness. As shown in column 5 and column 6 in Tab. 3, NNSplitter successfully degrades the victim model inference accuracy to random guessing, rendering the attacker's effort useless. Specifically, for 10-class datasets like Fashion-MNIST and CIFAR-10, the obfuscated top-1 accuracy of all victim models is lower than 11%, while for CIFAR-100 including 100 classes, the top-1 accuracy of victim models after obfuscation is lower than 2%. In contrast, randomly selecting weights to achieve model obfuscation

Dataset	Model	Baseline		Obfu. Weights	Obfu. Acc. (%)		Restored Acc. (%)
		Acc. [1] (%)	Para. (M)	Num. / Ratio [2](%)	**NNSplitter**	Random	
Fashion MNIST	VGG-11	93.73	28.14	**313/ 0.001**	**10.00**	92.90±0.40	93.73
	ResNet-18	93.71	11.17	**231/ 0.002**	**10.00**	92.03±0.93	93.71
	MobileNet-v2	93.97	2.24	**340/ 0.015**	**10.00**	86.41±1.58	93.97
CIFAR-10	VGG-11	92.39	28.15	**876/ 0.003**	**10.78**	91.42±0.25	92.39
	ResNet-18	93.07	11.17	**275/ 0.002**	**10.00**	91.35±0.27	93.07
	MobileNet-v2	93.91	2.24	**835/ 0.037**	**10.48**	78.18±1.38	93.91
CIFAR-100	VGG-11	70.50	9.80	**782/ 0.008**	**1.34**	64.34±0.75	70.50
	ResNet-20	68.28	0.28	**96/ 0.034**	**1.31**	56.35±1.38	68.27
	MobileNet-v2	74.29	2.25	**447/ 0.019**	**1.00**	50.92±1.33	74.28

Table 3. NNSplitter applied to multiple DNN models on three datasets. The number of obfuscated weights is the median value when the obfuscated (Obfu.) accuracy degraded to random guess (<11% for Fashion-MNIST/CIFAR-10, and <2% for CIFAR-100). The obfuscated accuracy of random is reported as mean±std with the same number of obfuscated weights.

only causes a limited accuracy drop (column 7 in Tab. 3), e.g., ∼1% (92.90±0.40 % vs. 93.73%) accuracy drop for VGG-11 model trained on Fashion-MNIST. Besides, the number of obfuscated weights is below 1k for all cases, which is small enough to store in TEE (Costan & Devadas, 2016). The smaller storage requirement can support more models deployed on the same device.

Efficiency. Given the ever-increasing size of DNN models, we aim to achieve active model protection by modifying only a very small fraction of the model weights. Specifically, by obfuscating 0.001% weights of the VGG-11 model on Fashion-MNIST, the model becomes completely malfunctional, i.e., with inference accuracy equal to random guess. Besides, for even more complicated datasets like CIFAR-100, the ratio of weight obfuscation is still small, e.g., 0.008% for VGG-11. Note that our proposed design could further reduce this ratio by tuning the mask hyper-parameters c and ϵ. However, for a *fair comparison*, we follow the generic strategy for all victim models to determine these parameters as described in Sec. 4.2.

Furthermore, Fig. 3 demonstrates that fewer weight changes are required when the desired accuracy degradation is smaller. For example, with 300 obfuscated weights and 301 model secrets (including 300 indexes and the value of c), NNSplitter achieves an accuracy drop to 10.23% for the VGG-11 model on Fashion-MNIST. Besides, a noticeable accuracy drop can still be observed as the number of obfuscated weights decreases to 150. In contrast, by randomly obfuscating 300 weights, the accuracy only drops to 92.49%, while the number of secrets is almost doubled, i.e., 600, since the original values of obfuscated weights are not close and thus cannot be replaced by a single value.

Integrity. Ensuring normal model inference for legitimate users is essential for an active model protection method. Thus our method should securely eliminate the adverse effects of obfuscated weights for authorized use. Specifically,

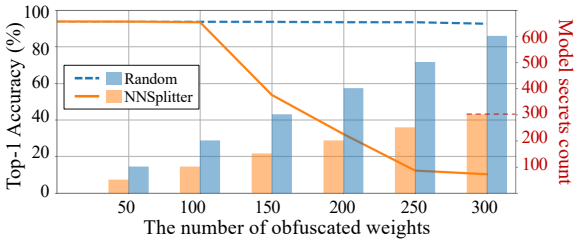

Figure 3. Number of obfuscated weights vs. accuracy for VGG-11 trained on CIFAR-10 (line plot), with the number of corresponding model secrets shown in the bar plot (associated with the y-axis on the right).

with access to the model secrets stored in TEE, the obfuscated weights can be located according to stored indexes. Since our proposed method carefully selects weights within $[c-\epsilon, c+\epsilon]$ with a very small ϵ (reported in Tab. 2), we can replace the constant c with the obfuscated weights during computation, thus preserving the baseline accuracy, as shown in column 8 in Tab. 3.

4.4. Ablation Study

We conduct an ablation study to verify the effectiveness of the RL-based controller. By applying the Base-NNSplitter defined in Sec. 4.1 to the same victim models, we can measure the number of obfuscated weights required to cause the same accuracy drop, the increment ratio of the Base-NNSplitter compared to NNSplitter is reported in Tab. 4. The increment can reach up to 125% in the worst case, demonstrating the effectiveness of the controller in optimizing filter selection. In conclusion, our developed RL-based controller achieves a drastic accuracy drop with fewer obfuscated weights.

[1]Acc. denotes the top-1 accuracy for all cases.

[2]Ratio is calculated by the number of obfuscated weights divided by the total number of model parameters.

Dataset	Model	Ratio
Fashion MNIST	VGG-11	+53.03%
	ResNet-18	+81.82%
	MobileNet-v2	+26.18%
CIFAR-10	VGG-11	+41.32%
	ResNet-18	+81.45%
	MobileNet-v2	+36.05%
CIFAR-100	VGG-11	+51.28%
	ResNet-20	+125.00%
	MobileNet-v2	+89.71%

Table 4. The increment ratio of the obfuscated weights for Base-NNSplitter compared to NNSplitter when both cause the random-guessing accuracy.

5. Discussion

In addition to the effectiveness, NNSplitter also considers the potential attack surfaces, i.e., whether an adversary can identify the obfuscated weights and mitigate their adverse effects, or improve the accuracy of the obfuscated model through further attacks, such as fine-tuning the model using limited training data. Thus, we evaluate the stealthiness and resilience of NNSplitter, following our defined requirements in Tab. 1. Furthermore, we conduct a comparison between a straightforward obfuscation strategy and our method to highlight the superiority of NNSplitter in terms of mitigating potential strong attacks as in Sec. 5.2.

5.1. Stealthiness

As discussed in Sec. 2.4, previous works achieving accuracy drop by manipulating weights fall into two categories: magnitude-based and gradient-ranking-based (Liu et al., 2017; Rakin et al., 2019). However, compared to the former category (Liu et al., 2017), NNSplitter constrains the obfuscated weights within the original range of weight values, thus avoiding being easily identified. As for the latter category, attackers can potentially locate the obfuscated weights by examining the weight gradients, allowing them to improve the degraded accuracy through weight reconstructions (Chen et al., 2021). However, NNSplitter mitigates this threat by employing an optimization method instead of a greedy method. This makes it more difficult for attackers to reverse engineer the obfuscated weights and improve the accuracy based on existing knowledge, thus ensuring a high level of stealthiness.

5.2. Resilience against Potential Attack Surfaces

Following our threat model in Sec. 2, we assume a strong attacker, who strives to improve the accuracy of the obfuscated models using SOTA techniques, like norm clipping (Yu et al., 2021) and fine-tuning (Adi et al., 2018).

Against Norm Clipping. The norm clipping proposed in (Yu et al., 2021) aims to defend against universal adversarial patches by restricting the norm of feature vectors. In our case, since the accuracy drop is caused by the magnitude change of some weights (from small to large), attackers may adopt norm clipping to weights and try to clip the obfuscated weights and eliminate their adverse effect. Specifically, the weight values outside an interval will be clipped to the interval edges, where the interval is defined by

$$Interval = t * [\min\{\mathbf{W} + \Delta\mathbf{W}'\}, \max\{\mathbf{W} + \Delta\mathbf{W}'\}] \quad (6)$$

and t is a coefficient in the range [0, 1].

We conduct experiments to evaluate the effectiveness of norm clipping as an attack against NNSplitter. The results, shown in Fig. 4, demonstrate that as the clipping threshold decreases, the accuracy of the obfuscated models initially increases due to more obfuscated weights being clipped. However, after reaching a certain point, the accuracy starts to decrease because normal weights are also being clipped. It is important to note that the highest accuracy achieved by the attacker is still below 50%, indicating the resilience of NNSplitter against norm clipping attacks.

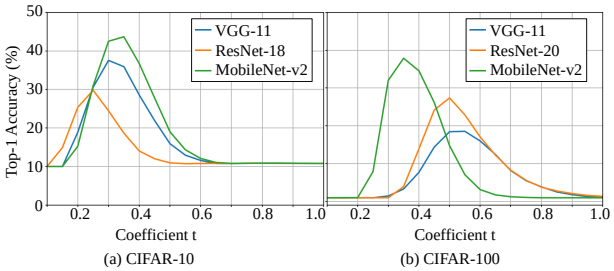

Figure 4. Apply norm clipping to improve the accuracy of obfuscated models on CIFAR-10/100.

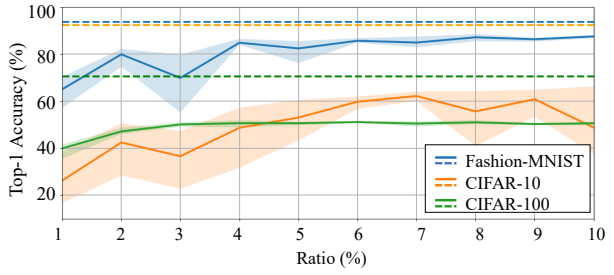

Figure 5. Apply fine-tuning to improve the accuracy of obfuscated VGG-11 models on different datasets. For each ratio, the average (solid line) and the error band (shadow region) are taken from 5 trials, with baseline accuracy as comparisons (dotted line).

Against Fine-tuning. Assuming stronger attackers who are aware of weight obfuscation in various layers (as illustrated in Fig. 2), they may attempt to reconstruct the weights through fine-tuning the obfuscated models using limited data (Adi et al., 2018). To evaluate the resilience of NNSplitter

against fine-tuning attacks, we consider different sizes of datasets available to the attackers, ranging from 1% to 10% of the training data used by the victim models. As shown in Fig. 5, in general, the accuracy will improve with the increased ratio of datasets used for fine-tuning. However, since the dataset is randomly sampled for each trial, some data may contribute more to the model fine-tuning than others, which explains the fluctuation in Fig. 5.

Moreover, our study demonstrates that distributing weight changes across multiple layers is more effective in protecting against the fine-tuning attack compared to concentrating them in a single layer. This finding highlights the benefit **(ii)** of our mask design as discussed in Section 3.1. Specifically, with the number of model secrets fixed, we add weight changes only to the first or the last layer of VGG-11 models on three datasets, respectively, and fine-tune the obfuscated models with 10% of training data. As shown in Fig. 6, obfuscating only the last layer results in a slight accuracy drop ($< 2\%$), which could be recovered close to the baseline accuracy through the fine-tuning attack. Although obfuscating the first layer achieves a drastic accuracy drop as NNSplitter from the defense perspective, its defense effects are not resilient against the fine-tuning attack at all. In summary, our proposed NNSplitter outperforms these strategies in the expected design requirements in Tab. 1.

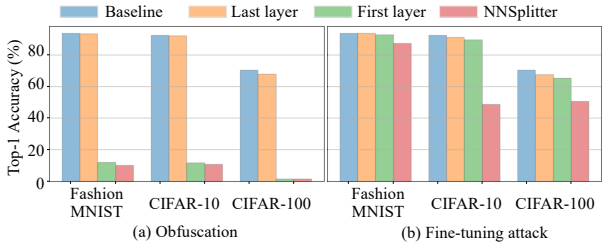

Figure 6. The performance comparison between the obfuscated model generated by NNSplitter and obfuscating a single layer (either the first or the last layer). Figure (a) displays the accuracy of the obfuscated model, while (b) shows the improved accuracy achieved through fine-tuning.

5.3. Obfuscation Strategy

We conduct experiments using VGG-11 on CIFAR-10 to compare the straightforward obfuscation method that modifies the scale and bias parameters of the normalization layer with NNSplitter. The results are presented in Tab. 5. By obfuscating the scale parameter to 1 and the bias parameter to 0 in the normalization layer, resulting in 5504 altered parameters, the obfuscated accuracy of the model decrease significantly to 13.77%. This demonstrates the effectiveness of the straightforward obfuscation technique in degrading the model's performance.

However, we observe that this obfuscated model is less effective in providing long-term protection against fine-tuning attacks. In particular, when attackers have access to only 10% of the training dataset and perform fine-tuning, they are able to restore the accuracy to 59.15%. In contrast, our proposed NNSplitter achieves a greater accuracy drop, i.e., 10.4% lower than obfuscating the normalization statistics, while obfuscating fewer weights (876 vs. 5504). This finding demonstrates the effectiveness of our proposed defense approach.

	Num. / Ratio (%)	Obfu. Acc.	Fine-tuned Acc.
Scale/bias	5504/0.019	13.77%	59.15%
Weights **(our)**	**876/0.003**	**10.78%**	**48.75%**

Table 5. Comparison of different obfuscation strategies.

Furthermore, this experiment comparison verifies our intuition that reconstructing the convolutional weights is more challenging for attackers compared to reconstructing the normalization statistics, which serves as a motivation for us to design a sophisticated weight obfuscation strategy as part of our model protection approach.

6. Conclusion

We propose a novel model IP protection scheme NNSplitter to actively protect the DNN model by preserving the model functionality exclusively for legitimate users. By leveraging the support of TEE, NNSplitter automatically splits a victim model into two components: the obfuscated model, stored in the normal world, and the model secrets, securely stored in the secure world. Through extensive experiments, we demonstrate the effectiveness of NNSplitter in achieving efficient model protection, e.g., by modifying around 0.001% weights (313 out of 28.14M), the victim model only outputs random prediction, rendering it useless for model attackers. Conversely, legitimate users can successfully execute authorized inferences by utilizing the safeguarded model secrets. Furthermore, we address the important aspects of stealthiness and resilience against potential attacks in the design of NNSplitter. This ensures that attackers are unable to identify our obfuscation technique or improve the degraded accuracy with reasonable efforts. By fulfilling these critical design requirements, NNSplitter emerges as a promising solution for protecting DNN models in real-world scenarios. Its ability to maintain the integrity and functionality of the models while preventing attackers from unauthorized use makes it an attractive option for model owners looking to safeguard their valuable intellectual property.

7. Acknowledgments

This work was supported in part by the U.S. National Science Foundation under grants CNS-2247892, CNS-2153690, and CNS-2238672.

References

Adi, Y., Baum, C., Cisse, M., Pinkas, B., and Keshet, J. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In *27th USENIX Security Symposium (USENIX Security 18)*, pp. 1615–1631, 2018.

Al-Garadi, M. A., Mohamed, A., Al-Ali, A. K., Du, X., Ali, I., and Guizani, M. A survey of machine and deep learning methods for internet of things (iot) security. *IEEE Communications Surveys & Tutorials*, 22(3):1646–1685, 2020.

Chakraborty, A., Mondai, A., and Srivastava, A. Hardware-assisted intellectual property protection of deep learning models. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6. IEEE, 2020.

Chen, H., Fu, C., Rouhani, B. D., Zhao, J., and Koushanfar, F. Deepattest: an end-to-end attestation framework for deep neural networks. In *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, pp. 487–498. IEEE, 2019.

Chen, Z., Li, G., and Pattabiraman, K. A low-cost fault corrector for deep neural networks through range restriction. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 1–13. IEEE, 2021.

Costan, V. and Devadas, S. Intel sgx explained. *Cryptology ePrint Archive*, 2016.

Duong, C. N., Quach, K. G., Jalata, I., Le, N., and Luu, K. Mobiface: A lightweight deep learning face recognition on mobile devices. In *2019 IEEE 10th international conference on biometrics theory, applications and systems (BTAS)*, pp. 1–6. IEEE, 2019.

Fan, L., Ng, K. W., and Chan, C. S. Rethinking deep neural network ownership verification: Embedding passports to defeat ambiguity attacks. *Advances in neural information processing systems*, 32, 2019.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

Jiang, Y., Zhu, Y., Lan, C., Yi, B., Cui, Y., and Guo, C. A unified architecture for accelerating distributed {DNN} training in heterogeneous {GPU/CPU} clusters. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pp. 463–479, 2020.

Krizhevsky, A., Nair, V., and Hinton, G. Cifar-10 (canadian institute for advanced research). 2009. URL http://www.cs.toronto.edu/~kriz/cifar.html.

Li, J., Rakin, A. S., Xiong, Y., Chang, L., He, Z., Fan, D., and Chakrabarti, C. Defending bit-flip attack through dnn weight reconstruction. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6. IEEE, 2020.

Liu, Y., Wei, L., Luo, B., and Xu, Q. Fault injection attack on deep neural network. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 131–138. IEEE, 2017.

Luo, Y., Gongye, C., Fei, Y., and Xu, X. Deepstrike: Remotely-guided fault injection attacks on dnn accelerator in cloud-fpga. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pp. 295–300. IEEE, 2021.

Ngabonziza, B., Martin, D., Bailey, A., Cho, H., and Martin, S. Trustzone explained: Architectural features and use cases. In *2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)*, pp. 445–451. IEEE, 2016.

Pham, H., Guan, M., Zoph, B., Le, Q., and Dean, J. Efficient neural architecture search via parameters sharing. In *International conference on machine learning*, pp. 4095–4104. PMLR, 2018.

Rakin, A. S., He, Z., and Fan, D. Bit-flip attack: Crushing neural network with progressive bit search. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1211–1220, 2019.

Rakin, A. S., Luo, Y., Xu, X., and Fan, D. Deep-dup: An adversarial weight duplication attack framework to crush deep neural network in multi-tenant fpga. In *30th USENIX Security Symposium*, 2021.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

Shukla, S., Manoj, P. S., Kolhe, G., and Rafatirad, S. On-device malware detection using performance-aware and robust collaborative learning. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pp. 967–972. IEEE, 2021.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *3rd International Conference on Learning Representations (ICLR)*, 2015.

Sun, Z., Sun, R., Lu, L., and Mislove, A. Mind your weight (s): A large-scale study on insufficient machine learning model protection in mobile apps. In *30th USENIX Security Symposium (USENIX Security 21)*, pp. 1955–1972, 2021.

Sun, Z., Sun, R., Liu, C., Chowdhury, A., Lu, L., and Jha, S. Shadownet: A secure and efficient on-device model inference system for convolutional neural networks. In *2023 2023 IEEE Symposium on Security and Privacy (SP) (SP)*, pp. 1489–1505, Los Alamitos, CA, USA, may 2023. IEEE Computer Society. doi: 10.1109/SP46215.2023.00085. URL https://doi.ieeecomputersociety.org/10.1109/SP46215.2023.00085.

Wang, R. J., Li, X., and Ling, C. X. Pelee: A real-time object detection system on mobile devices. *Advances in neural information processing systems*, 31, 2018.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Xu, M., Liu, J., Liu, Y., Lin, F. X., Liu, Y., and Liu, X. A first look at deep learning apps on smartphones. In *The World Wide Web Conference*, pp. 2125–2136, 2019.

Yang, P., Lao, Y., and Li, P. Robust watermarking for deep neural networks via bi-level optimization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 14841–14850, 2021.

Ye, M., Sherman, J., Srisa-An, W., and Wei, S. Tzslicer: Security-aware dynamic program slicing for hardware isolation. In *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 17–24. IEEE, 2018.

You, Z., Yan, K., Ye, J., Ma, M., and Wang, P. Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. *Advances in neural information processing systems*, 32, 2019.

Yu, C., Chen, J., Xue, Y., Liu, Y., Wan, W., Bao, J., and Ma, H. Defending against universal adversarial patches by clipping feature norms. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 16434–16442, 2021.

Zhang, J., Gu, Z., Jang, J., Wu, H., Stoecklin, M. P., Huang, H., and Molloy, I. Protecting intellectual property of deep neural networks with watermarking. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pp. 159–172, 2018.

Zhou, T., Ren, S., and Xu, X. Obfunas: A neural architecture search-based dnn obfuscation approach. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, pp. 1–9, 2022.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. *5th International Conference on Learning Representations (ICLR)*, 2017.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8697–8710, 2018.