Obfuscation for Deep Neural Networks against Model Extraction: Attack Taxonomy and Defense Optimization

Yulian Sun[1,2], Vedant Bonde[1], Li Duan[1,3], and Yong Li[1,✉]

[1] Data Protection Technology Lab, Huawei Technologies Düsseldorf, Germany
vedant.bonde1@huawei.com, yong.li1@huawei.com
[2] Ruhr University Bochum, Germany
yulian.sun@edu.ruhr-uni-bochum.de
[3] Paderborn University, Germany
liduan@mail.upb.de

Abstract. Well-trained deep neural networks (DNN), including large language models (LLM), are valuable intellectual property assets. To defend against model extraction attacks, one of the major ideas proposed in a large body of previous research is obfuscation: splitting the original DNN and storing the components separately. However, systematically analyzing the methods' security against various attacks and optimizing the efficiency of defenses are still challenging. In this paper, We propose a taxonomy of model-based extraction attacks, which enables us to identify vulnerabilities of several existing obfuscation methods. We also propose an extremely efficient model obfuscation method called $\mathsf{O}^2\mathsf{Splitter}$ using trusted execution environment (TEE). The secrets we store in TEE have $\mathcal{O}(1)$-size, i.e., independent of model size. Although $\mathsf{O}^2\mathsf{Splitter}$ relies on a pseudo-random function to provide a quantifiable guarantee for protection and noise compression, it does not need any complicated training or filtering of the weights. Our comprehensive experiments show that $\mathsf{O}^2\mathsf{Splitter}$ can mitigate norm-clipping and fine-tuning attacks. Even for small noise ($\epsilon = 50$), the accuracy of the obfuscated model is close to random guess, and the tested attacks cannot extract a model with comparable accuracy. In addition, the empirical results also shed light on discovering the relation between DP parameters in obfuscation and the risks of concrete extraction attacks.

Keywords: machine learning model security · model obfuscation · trusted execution environment · intellectual property protection

1 Introduction

Deep neural networks (DNN) and large language models (LLM) are everywhere, from cloud services [1,38] to on-device natural language processing such as Alexa [51]. Meanwhile, privacy and security concerns about these models have attracted growing attention from academia and industry [4, 32, 44].

Model extraction, when successful, allows an adversary to reconstruct a copy of the original model with comparable accuracy. The copy usually reflects the real model architecture and may possess parameters of similar distribution, which make model extraction a stepping stone for more devastating attacks [14,26,32], leading to user privacy leakage [45] or unethical business competition [44]. Model extraction attacks can be categorized into four classes [32]: App-based, device-based, communication-based and model-based attacks. Decompiling application files may directly leak the model [44,53], and being able to dump the memory of a device is essential for a successful device-based attack. Unprotected messages and execution environments also leak information about the model structure and parameters [27].

While App-based, device-based and communication-based attacks are extremely software- and hardware-specific, model-based extraction attack is a general threat and inherent for any public model service. A model-based attacker $\mathcal{A}$ may have datasets similarly distributed as the training dataset [28] or collect input-output pairs by querying the model service [34].

To mitigate model-based extraction attacks, the idea of splitting the original model and storing the components in separate execution environments has constantly inspired previous research [21, 43, 52, 59, 60]. The process of splitting the original model to generate the components is called **obfuscation** and the components exposed to potential attackers are called **obfuscated models**.

Among these proposals, recent solutions using generic Trusted Execution Environment (TEE) [39] are attractive for their efficiency and ubiquity [21, 59].

In principle, a generic TEE-based obfuscation against model-based attacks should have the following properties: (1) *Security* of the model: This may include *effectiveness*: the obfuscated model should have an accuracy close to random guess and cannot be trained efficiently, and *indistinguishability*: if not all the weights are obfuscated, it should be hard for an adversary to distinguish the obfuscated weights from normal ones; (2) *Efficiency* in storage and execution: the storage in the TEE should be sub-linear in the model size, and the obfuscation process should avoid training whenever possible; (3) *Flexibility*: the obfuscation and recovery should be configurable for various models and security requirements.

Unfortunately, when addressing (1), existing work often oversimplifies or underestimates the capabilities of the attackers. For example, the concrete distribution of obfuscated weights is ignored in [49, 59] when discussing about pruning or fine-tuning attacks. In contrast, we show in Section 5.4 that analyzing the obfuscated weight distribution can lead to efficient model extraction.

In this paper, we make the following contributions.

1. We propose a fine-grained taxonomy of extraction attacks against obfuscation according to the attacker's capability and resources, including access to model functionality, knowledge about model architecture, and dataset(s). With the proposed taxonomy, we systematically examine the existing TEE-based obfuscations. One of our key findings is that besides architecture knowledge, the distribution of obfuscated weights greatly impact the effi-

ciency of model-extraction attacks. Based on our taxonomy, we provide a framework of existing model-based extraction attacks, and propose a new fine-tuning attack that is effective against the most recent obfuscation methods, such as [49, 59], confirming our findings.
2. We propose an optimized opfuscation splitting method called O²Splitter. Fig. 1 provides an overview of O²Splitter. Besides being effective, O²Splitter only needs $\mathcal{O}(1)$-size storage for the secrets in the TEE, i.e., independent of model size, and it has quantifiable guarantee for model information leakage. O²Splitter does *not* need any training or filtering to select the weights or masks (noises). In addition, O²Splitter can be configured flexibly for different models and noise levels.
3. We evaluate and compare O²Splitter with current obfuscation methods in terms of its ability to obfuscate DNN/LLM and its resilience against class **C0** and **C1** defined in the taxonomy. Our experiments show that O²Splitter has improved resilience against advanced extraction attacks. In addition, the results shed more light on the relation between DP parameters in obfuscation and the risk level of concrete extraction attacks.

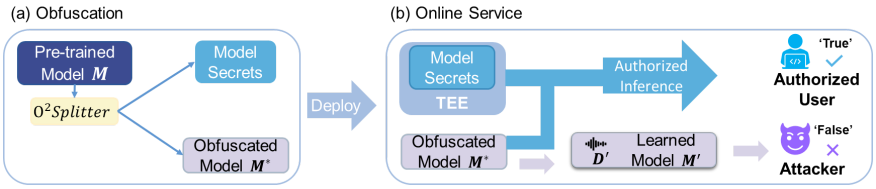


Fig. 1. The pipeline of O²Splitter. (a) Obfuscation: $\mathcal{M}$ is obfuscated by O²Splitter, then split into two parts, i.e., the obfuscated model $\mathcal{M}^*$ and model secrets. (b) Inference: attacker $\mathcal{A}$ can try to learn a model $\mathcal{M}'$ from $\mathcal{M}^*$ with a dataset $\mathcal{D}'$, which has a similar distribution to the original training dataset $\mathcal{D}$.

Outline. The taxonomy of model-based attacks is presented in Section 2, together with an analysis of existing defenses. The construction of O²Splitter is presented in Section 3. The implementation and comprehensive evaluation against various attacks are in Sections 4 and 5. We conclude the paper and discuss possible future work in Section 6.

2 Related Work and Attack Taxonomy

2.1 Related Work, the Defenses

In this paper, we consider only practical obfuscation for *large* models, so solutions with huge communication or computation overheads, such as those using multi-party computation [17] or fully homomorphic encryption [20], are out of scope.

Practical obfuscation methods can rely on designated secure hardwares such as TEE [10, 30, 49]. Optimization algorithms, e.g., search algorithms or complicated training, have also been introduced into obfuscation for efficiency-security trade-offs [21, 43, 52, 60].

An overview of existing practical obfuscation proposals is in Table 1. When examining each method, we focus on (i) the type of original models, (ii) the major idea in the obfuscation method, (iii) how the obfuscated weights are distributed, and (iv) whether the security analysis in the paper considers an attacker with datasets related to the training dataset(s). Spintronic-Based [30] is only valid for Binary Deep Neural Network (BDNN), HardwareObfuscation [10], Progressive Neural Architecture Search (ProgressiveNAS) [21] and DeepObfuscation [52] are considering Deep Convolutional Neural Network (DCNN), CoreLocker [49], NNSplitter [59], AdvTraining [43] and Obfunas [60] are more generally looking into Deep Neural Network (DNN). Key-based and training/learning are common tools in obfuscation, and the obfuscated weight distributions are usually not well addressed in these works.

Table 1. Overview of the defenses: original model, obfuscation method, and weight distribution of the obfuscated model. **Data:** whether this research assumes that the attacker has knowledge of the training dataset.

Defense Solutions	Original Model	Obfuscation	Obf. Weight Dist.	[a] Data
Spintronic-Based [30]	BDNN	Key-based	Weight values are 0 or 1	●
HardwareObfuscation [10]	DCNN	Key-based	Not mentioned	●
ProgressiveNAS [21]	DCNN	RL-based	Not mentioned	○
DeepObfuscation [52]	DCNN	Structural Design	Not mentioned	●
CoreLocker [49]	DNN	Access Key Extraction	Sub-Gaussian distributions	●
NNSplitter [59]	DNN	RL-based	Weight distribution is similar to original model	●
AdvTraining [43]	DNN	Adversarial Training	Not mentioned	●
Obfunas [60]	DNN	NAS-based	Not mentioned	●

[a] Obfuscation Weight Distribution
● : "Yes"; ○ : "No";

2.2 Taxonomy of Extraction Attacks against Obfuscation

We define extraction attacks as a post-exposure attack, i.e., the attack starts when the obfuscated model has been leaked to the attacker. To classify the attacks, we consider three types of resources that an attacker can have: (1) black-

box queries to the complete model, (2) knowledge about the training dataset, and (3) knowledge about the model architecture.

Considering whether (1) is available allows us to first classify the attacks into two groups: passive and active. Then, inside each group, we consider the mildest (with no knowledge) and strongest versions of attacks (with all knowledge). Therefore, we define four classes of extraction attacks against obfuscation ordered by the attacker's resources (low to high). The more resources an attacker has, the easier the extraction is and the more accurate the result can be.

C0. (Obfuscated Model Only) $\mathcal{A}$ only has access to an obfuscated model. $\mathcal{A}$ has neither knowledge about the model architecture nor the training dataset.

C1. (Passive with Dataset) $\mathcal{A}$ has access to an obfuscated model and has knowledge about the architecture and training dataset. The adversarial knowledge ranges from basic information about the data format to a usable dataset with a distribution close to the original dataset.

C2. (Active Type I) In addition to the obfuscated model, $\mathcal{A}$ can query the complete model in a black-box way, but $\mathcal{A}$ does **not** know the exact model architecture. Note that the query already leaks knowledge about the original dataset.

C3. (Active Type II) $\mathcal{A}$ has all the resources available, as in C2, and **complete** knowledge about the model architecture.

We consider the major inference attack paradigms and their variants: Adversarial Query (QR), Norm Clipping (NC) [54], Model Pruning (MP) [24], and Fine-tuning (FT) attacks [2]. Table 2 summarizes our classification. Each representative work may use more than one major attack.

Table 2. The Model-based Attack Taxonomy.

Attack Class	Attacks	Represent. Work
C0 (Obfuscated Model Only)	**NC, MP**	Fine Pruning [24]
C1 (Passive with Dataset)	**NC, MP, FT**	FT in [2]
C2 (Active Type I)	**QR, FT**	Reverse Blackbox [33] Hyperparameters [48] KnockoffNets [34] ML-Stealer [23] MAZE [15] CloudLeak [57]
C3 (Active Type II)	**QR, FT**	SimulatorAttack [28] Activethief [35]

C3-class attackers are the most resourceful. For example, if highly accurate pre-trained models and different categories of training data are at hand, SimulatorAttack [28] can extract the victim model weights with remarkable accuracy. A

common mitigation is to limit the number of queries that $\mathcal{A}_3$ can make, as there can be frequency-based or distribution-based monitoring for abnormal queries. Activethief [35] used unannotated public data and made their attack follow a natural distribution that a query distribution-based monitoring approach cannot detect.

A **C2** attacker $\mathcal{A}_2$ can first extract as much information about the model architecture as possible and then turn to a C3 strategy. For instance, $\mathcal{A}_2$ can employ Reverse Blackbox [33] and Hyperparameters [48] to collect the activation types, optimization algorithms, and model hyperparameters via queries, and then $\mathcal{A}_2$ can use SimulatorAttack [28] (classified as **C3**) for the weights. However, the accumulated error can be much larger than a direct C3 attack. In contrast, if $\mathcal{A}_2$ needs only the functionality of the original model, it can use KnockoffNets [34] to obtain the functionality without much knowledge of the model family or training data.

Without querying the victim model, extracting the weights or functionality becomes much more challenging for **C1** attackers. Attackers of the **C1** class can still use FT or re-train the obfuscated model to extract the functionality. For example, DNNWatermarking [2] provides detailed descriptions of different ways to fine-tune or re-train a model on a small representative dataset. As pointed out in [26], the complexity of the dataset used to train the victim model and the dataset used for the model-extraction attack can greatly affect the attack's accuracy.

A **C0** attacker $\mathcal{A}_0$ only has an obfuscated model. $\mathcal{A}_0$ may first filter out the noisy weights to keep as much usable part of the obfuscated model as possible. However, the filtering cannot be guaranteed. For example, if the obfuscated model is a ciphertext generated with an IND-CCA secure encryption scheme [16], then it only tells $\mathcal{A}_1$ about the size of the victim model. Methods like AdvParams [54] use techniques like ℓ_1 weight pruning to partly recover model performance. Other methods like Fine-Pruning [24] try to clip the weights of extreme values potentially generated by obfuscation. Unfortunately, these are some of the most rudimentary attacks that suffer from poor model recovery performance against many state-of-the-art defense techniques.

Defending against C3 and C2 attacks is a continuous arms race, and authentication, rate-limiting, and query/sample filtering are the mainstream guards [35, 46, 55]. On the other hand, immunity to C1 and C0 attacks should be the security bottom line of effective model protection.

With this taxonomy, we can systematically check the security of existing TEE-based obfuscation against these attacks, give a more precise evaluation and propose optimization.

Table 3 shows the resilience of the existing solutions against different classes of attackers. Although training-based parameter search benefits obfuscation efficiency, the result is not guaranteed to be resilient against **C1** attacks (See our attack experiment in Section 5). Moreover, we show in later sections that obfuscating the model without training is also possible (See Table 4).

Table 3. Defenses against different attack classes.

Defense Solutions	**C0**	**C1**	**C2**	**C3** [a]
[10, 30, 43, 49, 52, 59, 60]	◓	◓	○	○
[21]	○	○	○	○
This Work	●	●	○	○

[a] Symbols refer to demonstrated defense against ● : all attacks; ◓ : some attacks; ○ : no attacks;

Table 4. Efficiency comparisons with related work.

Method	**W.o. Learn.**[a]	**Const. Storage**[b]	**Obf. Compl.**[c]
[10, 30, 49, 59]	●	○	$\mathcal{O}(N)$
[21, 43, 52, 60]	○	●	$\mathcal{O}(N)$
This work	●	●	$\mathcal{O}(N)$

[a] Without Learning
[b] Constant Storage
[c] Obfuscation Complexity
● : "Yes"; ○ : "No"; N: the number of model parameters.

3 Construction of O²Splitter

3.1 Threat Model

We consider a **C1** attacker $\mathcal{A}$ that is defined in Section 2.2 and runs in probabilistic polynomial time (PPT) with the following resources:

R0. $\mathcal{A}$ has obtained the obfuscated model $\mathcal{M}^*$;
R1. $\mathcal{A}$ has knowledge about the original training dataset $\mathcal{D}$. For simplicity, we assume that $\mathcal{A}$ knows some $\mathcal{D}'$ that relates to $\mathcal{D}$, but the number of samples and classes can be quite different.

This threat model is practical since most industrial models, as mentioned in [49, 59], are variants of public research and adapted for or fine-tuned with a private dataset $\mathcal{D}$. In this case, $\mathcal{A}$ can use the public datasets as $\mathcal{D}'$.

3.2 Tools for Efficient Obfuscation.

DP-like Obfuscation. Since the adversary has access to the obfuscated model, we consider the obfuscated weights as *published data*. Thus, we can have quantifiable indistinguishability by following the differential privacy paradigm (DP). An effective DP mechanism guarantees the hardness to distinguish one value from another after processing, and this hardness is quantifiable.

Definition 1 (Differential Privacy, DP [6, 29]) *We say that a mechanism $M()$ on data space $\mathcal{W}$ has (ϵ, δ) differential privacy if*

$$\forall W, W' \in \mathcal{W} : \Pr[M(W) = y] \leq e^{\epsilon} \Pr[M(W') = y] + \delta \tag{1}$$

Definition 2 (ℓ_2 Local Sensitivity) *For $f : \mathcal{W} \to \mathcal{R} \subset \mathbb{R}$, the ℓ_2 local sensitivity $\Delta_{f,2}$ of $f()$ is defined as*

$$\Delta_{f,2} = \max_{W,W'}(||f(W) - f(W')||_2). \tag{2}$$

The term ϵ is called the *privacy budget*, and δ the *failing probability*, i.e., the probability that the difference caused by $x \neq x'$ goes beyond the factor e^{ϵ}.

If w has dimension one, the ℓ_1-sensitivity has the same value as ℓ_2-sensitivity, i.e., $\Delta_{f,1} = \Delta_{f,2}$ in this case.

Definition 3 (Gaussian Mechanism for DP [6]) *If a function $f(W)$ of an input $W \in \mathcal{W}$ is to be released, the Gaussian release mechanism is defined as*

$$G(W) := f(W) + \mathcal{N}(0, \sigma^2 I) = f(W) + \sigma\mathcal{N}(0, I). \tag{3}$$

Fixing ϵ and δ, we can derive the standard deviation σ for the Gaussian mechanism from $\Delta_{f,2}$ as $\sigma = \frac{\Delta_{f,2}}{\epsilon}\sqrt{2\log\frac{1.25}{\delta}}$ [6]. Here, $f()$ is the identity function $f(W) = W$. Each weight $w_i \in W$ will then be obfuscated by a random noise $m_i \xleftarrow{\$} \sigma\mathcal{N}(0, I)$, where $\xleftarrow{\$}$ means sampling from.

3.3 Overview and Noise Compression

In O^{2}Splitter, critical obfuscation secrets are derived from a trained model $\mathcal{M}$. The derived secrets are used to generate and scale the noises. An obfuscated model $\mathcal{M}^*$ is computed from the noises and $\mathcal{M}$. After authentication, an authorized user can instruct the TEE to recover $\mathcal{M}$ from $\mathcal{M}^*$ and the secrets. This procedure is illustrated in Figure 1. Details follow.

Noise Compression. Although the Gaussian mechanism can provide excellent protection, storing each individual noise for obfuscation is infeasible regarding the scale of contemporary ML models. Thus, we need to securely "compress" the presentation of noises. Given a uniform random seed s_i, we can re-write the sampling as a deterministic procedure, i.e.,

$$m_i = \sigma\mathcal{N}(0, I; s_i). \tag{4}$$

Now, the challenge is reduced to computing and reproducing each s_i, and the tool is a pseudo-random function.

Definition 4 (Pseudo-random Function, PRF [16]) *A pseudo-random function (PRF) is a pair of two algorithms* (FKGen, PRF)*, where* FKGen *and* PRF *are described below.*

- $\mathsf{FKGen}(1^\lambda) \xrightarrow{\$} \mathsf{K}$. *The non-deterministic key generation algorithm* $\mathsf{FKGen}()$ *takes the security parameter* 1^λ *as the input and outputs the secret key* K.
- $\mathsf{PRF}(\mathsf{K}, x) = y$. *The deterministic evaluation algorithm* $\mathsf{PRF}()$ *takes the secret key* K *and a value* x *in the domain as the input and outputs an image* y.

3.4 $\mathsf{O^2Splitter}$ and its Theoretical Security.

By using a cryptographic PRF defined above, we can define an invariant between weight w_i and the obfuscated w_i^* as follows:

$$w_i^* = w_i + \sigma\mathcal{N}(0, I; \mathsf{PRF}(\mathsf{K}, \mathsf{ID}_i)), \tag{5}$$

where K is the PRF key chosen uniformly at random, and ID_i is the index of W_i. For security and privacy, we require that K never be used for different models and ID_i be unique for each w_i.

As long as K is unknown to adversary $\mathcal{A}$, it remains intractable for $\mathcal{A}$ to distinguish PRF outputs from real random values or predict the next output [58], i.e., each s_i output by the PRF is computationally close to a real random s_i. Moreover, PRF always outputs the same s_i given the same K and ID_i. Thus, we only have to store K and σ securely in the trusted environment for weight recovery, i.e., $\mathcal{O}(1)$ secure storage.

Details of the assembled $\mathsf{O^2Splitter}$ can be found in Algorithm 1, where a complete index is the concatenation of the model index, layer index, and weight index. The weight recovery is straightforward: use K and σ to compute each m_i, and compute $w_i = w_i^* - m_i$. $\mathsf{O^2Splitter}$ security is summarized in Theorem 1.

Theorem 1 *The obfuscation algorithm defined as Algorithm 1 has* $(\epsilon, \delta + 2\cdot\delta_{\mathsf{PRF}})$ *differential privacy against any efficient C1-class adversary, where C1 is defined in Section 2, and* δ_{PRF} *is the advantage of any efficient adversary against* PRF.

The proof can be found in Appendix A

Comparison with Encryption. Using noise for obfuscation is a more generic approach and covers the case of encryption. For example, suppose that we instantiate PRF with AES and use XOR instead of scaling and addition in Algorithm 1. In that case, $\mathsf{O^2Splitter}$ essentially performs encryption with semantic security (AES-CTR [22], assuming unique ID_i), leading to $\epsilon = 0$ and negligible δ for PPT attackers.

$\mathsf{O^2Splitter}$ is more efficient and flexible than simple encryption. Existing blockciphers, such as ASCON [5] and AES [7], have an output block length of 64 or 128 bits, whereas a typical weight in DNN has only 8 to 16 bits. If each individual weight is encrypted with a single block to generate the obfuscated model $\mathcal{M}^*$, a space about 4-16 times as big as the original model size is needed by $\mathcal{M}^*$ on disk or in RAM. If multiple weights are encrypted within one block, the bookkeeping introduces extra overhead, and the configuration can be very model-specific.

Algorithm 1 $\mathsf{O^2Splitter}$ weight obfuscation with noise compression

Input: $\mathcal{M}$, Params = $(\epsilon, \delta, \lambda)$
Output: $\mathcal{M}^*$, K, σ
Algorithm Starts:
$\mathsf{K} \xleftarrow{\$} \mathsf{FKGen}(1^\lambda)$
Compute $\Delta_{f,2}$ for all the weights
Compute $\sigma = \frac{\Delta_{f,2}}{\epsilon}\sqrt{2\log\frac{1.25}{\delta}}$
for each layer j **do**
    for each w_i on layer j **do**
        Let ID_i be the complete index of w_i
        $s_i = \mathsf{PRF}(\mathsf{K}, \mathsf{ID}_i)$
        Generate noise $m_i = \sigma\mathcal{N}(0, I; s_i)$
        $w_i^* = w_i + m_i$
    end for
end for
return $\mathcal{M}^* = \{w_i^*\}$, K, σ

Moreover, extra expertise is required to balance security and efficiency in simple encryption-based solutions in practice. The security level of block ciphers are configured by the key length, which should be at least 128-bit [3]. One may attempt to sacrifice key lengths for efficiency; for example, use a 64-bit key instead of 128-bit one. However, a black-box call of block cipher may require the short key to be padded to 128-bit so that each round function inside a block cipher is computed as before. Thus, this attempt gives no efficiency improvement, and reducing the number of rounds can have unforeseeable consequences [11] [4].

In contrast, $\mathsf{O^2Splitter}$ can be easily configured with different $(\epsilon, \delta, \lambda)$ values, which always guarantee quantifiable privacy and security.

Quantifying Resilience against Attacks. Unlike solutions using only DP, TEE-based obfuscation does not inherently suffer from DP-specific attacks such as [12], because the obfuscated model does not work alone and does *not* to have any utility. Therefore, the ϵ parameter in $\mathsf{O^2Splitter}$ can be configured to 0 for the highest privacy guarantee in terms of DP.

On the other hand, we consider mapping $(\epsilon, \delta, \lambda)$ to risks of specific extraction attacks in a theoretically sound way as an open problem. Although previous work such as noise calibration [19] and [31] has attempted to map noise scale to risks of membership inference attacks targeting the training dataset, to the best of our knowledge, there is no generic and theoretically sound way of mapping DP parameters or noise scale to the success probability of any model extraction attack. Thus, we empirically address this mapping in Sections 5.3 and 5.4, shedding more light on the discovery of the underlying relation.

[4] The distinguisher in [11] can tell a ciphertext generated by 4-round 128-bit-key AES from random bit-strings within $\mathcal{O}(2^{33.3})$-time.

4 Experiments

This section provides details of the O²Splitter implementation, the datasets and the baseline models for benchmarking and comparing with current obfuscation methods.

4.1 Implementation Setup

Datasets. We evaluate and compare the effectiveness of O²Splitter on the CIFAR-10 [18], which contains 50k training images and 10k test images of dimension 32×32 and includes 10 classes, and CIFAR-100 [18], which includes around 60k images equally distributed across 100 classes. We also use Google's Speech Commands [50] and PolyAI's MINDS-14 [9] datasets.

The Speech Commands consists of 65000 one-second audio clips of 32 different short words for the task of Limited-Vocabulary Speech Recognition. The MINDS-14 spans 14 languages and covers 14 different intents obtained from the e-banking domain with the task of Intent Classification.

Furthermore, to evaluate the obfuscation of LLMs, we choose the models for the reading comprehension tasks, and use the SQuAD dataset (Stanford Question Answering Dataset) [37], which consists of over 100,000 question-answer pairs based on more than 500 Wikipedia articles.

Baseline Models. As O²Splitter is a model-independent technique, we consider DNNs such as VGG-11 [42], MobileNet-v2 [40], and ResNet18 [13] trained on the CIFAR-10 dataset. We also consider Transformer-based models like XLSR-Wav2Vec2 [5] and Audio Spectrogram Transformer (AST) [6], which are trained on the MINDS-14 and Speech Commands datasets, respectively. We use and modify the pre-trained weights but do not alter the model's overall structure to show the model-independent nature of the O²Splitter.

We also use O²Splitter to obfuscate Large Language Models. More specifically, we choose two language models: RobERTa [25] (encoder-only) and GPT-2 [36] (decoder-only) fine-tuned on SQuAD (Stanford Question Answering Dataset) for question-answering tasks.

Metrics. For the evaluation of obfuscated models on classification tasks, the percentage of correct classes predicted by the models, also denoted as top-1 accuracy, is considered. For the generative tasks on language models like question-answering, we use exact match, which calculates the percentage of predictions that match the ground-truth answers.

[5] https://huggingface.co/anton-l/xtreme_s_xlsr_300m_minds14
[6] https://huggingface.co/MIT/ast-finetuned-speech-commands-v2

4.2 Implementation Details

LLM Fine-tuning. Since models like RoBERTa and GPT-2 have been pre-trained for generalized masked-language modeling tasks, we fine-tune both models on domain-specific task like question-answering. We utilize the transformer library of Huggingface to obtain the pre-trained models of RoBERTa [7] and GPT-2 [8] to fine-tune these models on the SQuAD dataset over three epochs. We adopt a linear learning rate scheduler with an initial value of $2e-5$. We collect the optimal models during fine-tuning. Then, we execute obfuscation and attacks on the same models.

Compared Work. Within existing methods for obfuscation, we select two representative solutions. (1) NNSplitter: We choose NNSplitter because it is currently one of the most effective methods utilizing reinforcement learning to obfuscate neural networks with minimal weight changes. It also demonstrated some resilience against certain adversarial attacks. However, the current NNSplitter methodology only works on the classification tasks, so we only compare the results of NNSplitter on DNNs designed for classification. (2) CoreLocker: We choose CoreLocker as it is the most recent paper on obfuscation (2024) with simple yet effective method for weight obfuscation that is compatible with different model architectures. Unlike NNSplitter, CoreLocker has also demonstrated resilience against language models like BERT. Therefore, we compare the results of model $\mathsf{O^2Splitter}$ and CoreLocker on classification models, and LLMs for question-answering task.

5 Results

This section provides a quantitative analysis to evaluate the practical results of applying $\mathsf{O^2Splitter}$ to commonly used DNNs. Besides optimized storage, the performance evaluation is focused on the remaining properties: effectiveness and flexibility. Moreover, we demonstrate that $\mathsf{O^2Splitter}$ has resilience against an adversarial attack, to which NNSplitter [59] and CoreLocker [49], the state-of-the-art, are vulnerable.

5.1 Basic Effectiveness

Table 5 demonstrates that the accuracy obtained from an $\mathsf{O^2Splitter}$-obfuscated model is close to that of random guesses for all datasets. More specifically, for the CIFAR-10 dataset, which has 10 classes, the top-1 accuracy for all of the obfuscated models is close to 10%. For CIFAR-100, which contains 100 classes, the top-1 accuracy of the obfuscated models is close to 1%. For audio datasets like the Speech Commands dataset, which contains 32 classes, the top-1 accuracy

[7] https://huggingface.co/FacebookAI/roberta-base
[8] https://huggingface.co/openai-community/gpt2-medium

Table 5. O^2Splitter was applied to multiple datasets with different DNNs for diverse classification and question-answering tasks.

Dataset	#Classes	Model	Baseline Acc. (%)	Std. Dev. (σ)[9]	Obfu. Acc. (%)	Restored Acc. (%)
CIFAR-10	10	VGG-11	92.39	0.03	10.14	92.39
		ResNet-18	93.07	0.05	9.54	93.07
		MobileNet-v2	93.91	0.04	10.00	93.91
CIFAR-100	100	VGG-11	68.77	0.05	1.01	68.77
		ResNet-18	71.91	0.08	0.94	71.91
		MobileNet-v2	75.45	0.06	0.91	75.45
Speech Commands	32	AST	98.11	0.22	2.63	98.11
MINDS-14	14	XLSR-Wav2Vec2	93.30	1.44	6.29	93.29
SQuAD	-	GPT-2	67.63	3.77	0.39	67.63
	-	RoBERTa	85.04	0.23	0.23	85.04

Table 6. Obfuscation Accuracy (%) for different ϵ values for $\delta = 2^{-32}$ in VGG-11, AST, and XLSR-Wav2Vec2.

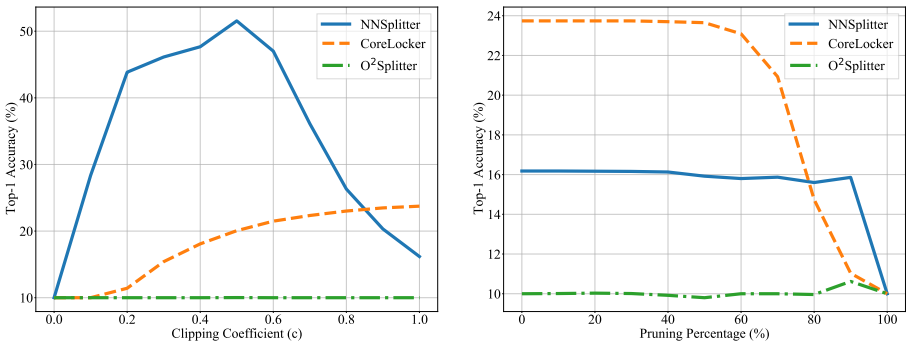
Method	Obfu. Acc. (%)							
	$\epsilon = 0.1$	$\epsilon = 1$	$\epsilon = 10$	$\epsilon = 50$	$\epsilon = 100$	$\epsilon = 200$	$\epsilon = 500$	$\epsilon = 1000$
VGG-11	9.99	10.00	10.55	10.14	30.95	80.45	91.45	92.03
AST	3.69	3.53	3.61	2.63	1.71	3.08	96.79	97.18
XLSR	6.80	6.24	6.71	6.29	6.78	7.27	7.63	64.73

of the obfuscated AST model is just 2.63%, which is close to the random guessing accuracy. Other similar results can also be observed for the MINDS-14 dataset, highlighting the basic effectiveness of the O^2Splitter.

Unlike classification tasks, in question-answering, there is a near-zero probability to randomly guess a correct answer, especially for a large answer space, so the obfuscated model's accuracy should drop to random guess probability, and our experiments confirm this for O^2Splitter. For the question-answering task for language models, the exact match accuracy for the obfuscated GPT-2 model drops to 0.39%, and for the obfuscated RoBERTa model, 0.23%. The extremely low accuracy highlights the basic effectiveness of obfuscating the LLMs with O^2Splitter.

5.2 Flexibility

Table 6 illustrates the obfuscated top-1 accuracy of VGG-11, AST, and XLSR-Wav2Vec2 models on CIFAR-10, Speech Commands, and MINDS-14 datasets for different ϵ values. The strength of the noise drops when the ϵ value increases. O^2Splitter reduces the model accuracy significantly after obfuscation. Adjusting of the amount of noise, i.e., adjusted ϵ, the model accuracy changes correspondingly. The large range of ϵ selections all degrade model accuracy to random



(a) Clipping Attack (b) Pruning Attack

Fig. 2. Comparison of applying 2(a) clipping attacks and 2(b) pruning attacks on different methods of obfuscated VGG-11 models evaluated on the CIFAR-10 dataset.

guesses for all three models. The number of parameters and the magnitude of deviation in each layer of the neural network, which depend inversely on ϵ, are the primary causes of the variance in performance of various models as ϵ increases. While VGG-11 recovers its accuracy for smaller values of ϵ, AST and XLSR are large parameter models with different weight distributions and hence, addition of noise with small deviations can significantly impact the model performance.

5.3 Resilience against C0-class Attacks

The access to obfuscated model weights provides C0-class attackers with the possibility to investigate and filter out outliers, which can be an effective way to extract a functionally comparable model. Model Pruning (MP) and Norm Clipping (NC) are the most famous types of attacks on obfuscated weights, but many recent methods [49, 54] only focus on a limited subset of these attacks. Hence, we aim to analyze the strongest C0-class attacks on existing obfuscation methods and compare the results of these attacks with $\mathsf{O^2Splitter}$.

Norm Clipping (NC) Attack. The NC attack was proposed in [56, 59]. As the obfuscation method relies on the abrupt magnitude change of some weights to cause a drop in model performance, NC can clip the (possibly) obfuscated weights to particular intervals. More specifically, for the set of weights W^* of an obfuscated model $\mathcal{M}^*$, the clipping interval I can be defined as:

$$\mathsf{I} = [c * \min\{W^*\},\ c * \max\{W^*\}], \tag{6}$$

where c is a coefficient in the range $[0, 1]$, used to determine the range of values to be clipped.

We compare the recovered accuracy using different clipping intervals to assess the **C0**-attack resilience of the $\mathsf{O^2Splitter}$ with that of the existing methods. The

experiments are conducted on obfuscated VGG-11 models, and top-1 accuracy is evaluated on the CIFAR-10 dataset. Figure 2(a) illustrates the result of NC attacks with different incremental values of c in the range $[0, 1]$. The results demonstrate that NNSplitter is exceptionally prone to clipping attacks, reaching a maximum top-1 accuracy of more than 50%. This can be explained by the obfuscation strategy of NNSplitter, which pushes the weights of the VGG-11 model to extreme values (See Figure 3(b)). Thus, clipping the extreme values back to smaller intervals helps increase the accuracy of the obfuscated model. CoreLocker and $\mathsf{O}^2\mathsf{Splitter}$ exhibit resilience against clipping attacks, showing no noticeable increase in top-1 accuracy.

Model Pruning (MP) Attack. The MP attack is another effective method to recover model accuracy by pruning the obfuscated parameters. It has been noted that MP attack works especially well against noise-based obfuscation algorithms [49, 54]. Recent methods like CoreLocker have demonstrated resilience against pruning ratios up to 40% of the total weights.

The experiment setup is similar to NC attack, and obfuscated VGG-11 models are evaluated on CIFAR-10 dataset. Figure 2(b) shows the result of applying the pruning attack with different ratios to the obfuscated models using different techniques. It can observed that all the methods are completely robust to the pruning attacks irrespective of the pruning percentage. This provides evidence of the $\mathsf{O}^2\mathsf{Splitter}$ being robust against clipping attacks as it systematically obfuscates all the weights of the neural networks.

5.4 Resilience against C1-class Attacks.

Current models are trained on large-scale datasets from a variety of sources, which might be partially collected by attackers to launch **C1**-class fine-tuning attacks against the obfuscated models to approximate the original functionality. This is one of the most effective ways to use the knowledge still left in an obfuscated model. The type of fine-tuning attack used is determined by the computational power that an attacker has: it may have considerable resources to fine-tune the whole neural network, or it can only work with limited resources to fine-tune only the last few layers of the models. This difference is critical in the context of LLMs, which require extensive GPU resources to be trained. This section demonstrates four existing fine-tuning methods and presents a detailed analysis of usage of specific fine-tuning attacks against the discussed obfuscation methods. Furthermore, we also propose a Selective Fine-tuning (SFT) Attack, to provide an effective way to attack recent obfuscation methods like [59] to extract a model with comparable accuracy.

Existing Fine-tuning Attacks. Most of the current literature on model obfuscation like [49, 59] does not provide the exact description of the strategy that a fine-tuning attack uses. To address these inconsistencies, inspired by [2], we propose four different types of fine-tuning attacks:

- **Fine-Tune All Layers (FTAL)**: Unfreeze the weights of all layers of the model and back-propagate through the entire network while training on the attacker's dataset.
- **Fine-Tune Last Layer (FTLL)**: Unfreeze the weights of the last layer only, allowing backpropagation to update only the weights of the last layer.
- **Re-train Last Layer (RTLL)**: Initialize the parameters of the last layer with weights using Kaiming initialization, and allow backpropagation over the last layer only while fine-tuning. Many attackers might prefer this method in case of limited computing budget.
- **Re-train + Fine-Tune (RTFL)**: Initialize the parameters of the last layer with Kaiming initialized weights, but unfreeze the parameters of all the layers to allow backpropagation over the entire network. This strategy exploits the fact that some obfuscation methods only store the weights of the last layer of the model in the TEE to lower costs and increase execution efficiency. Although the attacker does not have the weights of the last layer, they can randomly initialize the last layer, use the rest of the obfuscated model as it is, and perform fine-tuning on their own datasets.

We evaluate the resilience of $\mathsf{O}^2\mathsf{Splitter}$ and compare the results of these attacks on the victim-obfuscated models obtained via NNSplitter and CoreLocker on the CIFAR-10 dataset. More specifically, for the case of CoreLocker, as there are no current public implementations, we follow the steps provided in the paper to implement CoreLocker with ℓ_1-norm as the extraction indicator and use an extraction ratio of 0.1 in our experiments (indicating the ratio of weights extracted from the neural network).

The results of the four fine-tuning procedures for classification are illustrated in Table 7. Overall, $\mathsf{O}^2\mathsf{Splitter}$ demonstrates greater resilience to all fine-tuning attacks than CoreLocker and NNSplitter. Specifically, FTAL shows poor top-1 accuracy with limited fine-tuning data (1%) but achieves the highest accuracy with larger datasets (10%) due to more comprehensive learning. However, obtaining large datasets may be challenging for attackers. FTLL often outperforms FTAL with smaller fine-tuning datasets due to the effectiveness of fine-tuning only the last layer with limited data. Both CoreLocker and NNSplitter are vulnerable to FTLL with less data. RTLL performs well on NNSplitter but poorly on CoreLocker, with no significant top-1 accuracy increase on $\mathsf{O}^2\mathsf{Splitter}$.

RTFL assumes that the final layer's weights are stored securely. This attack achieves good top-1 accuracy with limited data on CoreLocker but fails to recover accuracy on NNSplitter and $\mathsf{O}^2\mathsf{Splitter}$. With larger datasets, FTAL and RTFL significantly increase the top-1 accuracy of the obfuscated $\mathsf{O}^2\mathsf{Splitter}$ model. CoreLocker is particularly susceptible to whole-network fine-tuning attacks, while NNSplitter is more vulnerable to last-layer fine-tuning attacks. Our study demonstrates that $\mathsf{O}^2\mathsf{Splitter}$ exhibits robust defense against these fine-tuning procedures.

The results of obfuscation and fine-tuning attacks for LLMs on question-answering tasks are compared in Table 8. An important advantage of $\mathsf{O}^2\mathsf{Splitter}$ over CoreLocker is its lower exact match accuracy on the SQuAD evaluation

Table 7. Comparison of different fine-tuning attack methods on obfuscated VGG-11 models evaluated on CIFAR-10 ($\epsilon = 50, \delta = 2^{-32}$).

Method	Data	Obf. Acc. (%)	Fine-tune Acc. (%)				
			FTAL	FTLL	RTLL	RTFL	SFT
CoreLocker	1%	23.74	45.21	79.82	44.64	78.51	84.53
NNSplitter	1%	16.81	20.87	91.09	91.12	23.37	91.41
O^{2}Splitter	1%	10.14	11.02	15.42	11.00	11.51	13.27
CoreLocker	5%	23.74	68.50	80.25	78.96	82.95	88.23
NNSplitter	5%	16.81	80.25	90.86	91.03	83.95	91.80
O^{2}Splitter	5%	10.14	28.35	19.30	15.59	16.52	12.93
CoreLocker	10%	23.74	84.74	80.35	80.19	82.88	88.79
NNSplitter	10%	16.81	86.51	90.96	90.83	86.69	90.62
O^{2}Splitter	10%	10.14	48.53	19.91	18.52	44.22	12.59

Table 8. Comparison of different fine-tuning attack methods on obfuscated RoBERTa models evaluated on the question-answering dataset SQuAD ($\epsilon = 50, \delta = 2^{-32}$).

Method	Data (%)	Obf. Acc. (%)	Fine-tune Acc. (%)				
			FTAL	FTLL	RTLL	RTFL	SFT
CoreLocker	1%	2.75	0.82	5.57	3.21	0.55	0.78
O^{2}Splitter	1%	0.23	0.46	0.33	0.27	0.49	0.45
CoreLocker	5%	2.75	0.72	5.98	5.62	0.47	0.77
O^{2}Splitter	5%	0.23	0.50	0.39	0.33	0.47	0.52
CoreLocker	10%	2.75	0.84	6.72	6.42	0.96	0.85
O^{2}Splitter	10%	0.23	0.52	0.39	0.27	0.49	0.53

data. Both CoreLocker and O^{2}Splitter demonstrate resilience against fine-tuning attacks. FTLL and RTLL are the most effective attacks against CoreLocker, achieving an accuracy of 6.72% on the 10% data split. In contrast, whole-model fine-tuning works better on O^{2}Splitter, with FTAL achieving only 0.52% accuracy on the 10% data split. Overall, while both methods are robust against fine-tuning attacks, O^{2}Splitter exhibits greater resilience compared to CoreLocker.

Selective Fine-tuning attack (SFT). Besides these fine-tuning attacks, we also propose a modified fine-tuning attack to compare the resilience of NNSplitter and O^{2}Splitter. Attackers may attempt to recover an equivalent model by fine-tuning the obfuscated models on a similar but much smaller dataset $\mathcal{D}'$. Identification of obfuscated weights is easy in NNSplitter, as it clips the obfuscated weights between constant maximum and minimum values while performing gradient updates, as observed in Figure 3. As most of the *obfuscated* weights ($> 90\%$) lie on these common maximum and minimum values, it becomes extremely easy to identify the obfuscated weights and perform fine-tuning on these particular weights. The SFT Attack utilizes the approximate common maxima and minima (c_{min}, c_{max}) obtained from the obfuscated neural network from NNSplitter to fine-tune a small subset of the weights on the smaller dataset $\mathcal{D}'$.

The adversary marks the set of "obfuscated" weights W^* as the ones lying either in the upper selection region $\mathsf{U} = [c_{max} - \mu_1, c_{max} + \mu_2]$ or the lower selection region $\mathsf{L} = [c_{min} - \ell_1, c_{min} + \ell_2]$. All the obfuscated weights are unfreezed, while the rest of the weights are frozen to prevent any gradient updates. Subsequently, the adversary fine-tunes the obfuscated model $\mathcal{M}^*$ on $\mathcal{D}'$ by optimizing the obfuscated weights to minimize the loss function $\mathcal{L}$. Specifically, the weight update is defined as:

$$w' = w - \eta \frac{\partial \mathcal{L}}{\partial w} \odot M_w, \tag{7}$$

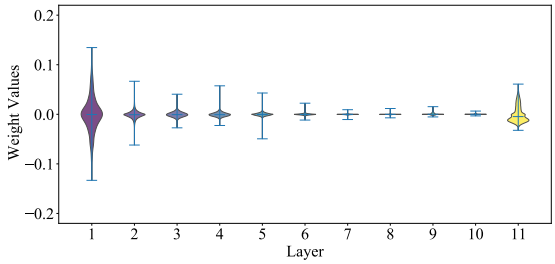
where M_w describes the mask obtained from W^*. The intuition behind the selective gradient update is to preserve the pre-trained model weights, ensuring faster convergence and lesser computation.

We evaluate and compare the results of this attack on the victim obfuscated models obtained via NNSplitter, CoreLocker, and $\mathsf{O}^2\mathsf{Splitter}$ on the CIFAR-10 dataset, and only compare CoreLocker and $\mathsf{O}^2\mathsf{Splitter}$ on the SQuAD dataset for LLMs as previously mentioned. For CoreLocker, as mentioned before, we follow the steps provided in the paper to implement CoreLocker with ℓ_1-norm as the extraction indicator and use an extraction ratio of 0.1 in our experiments. Furthermore, we perform an SFT attack on CoreLocker with $\mathsf{U} = \mathsf{L} = [0.0,\ 0.0]$, basically indicating that only weights of 0 values are extracted.

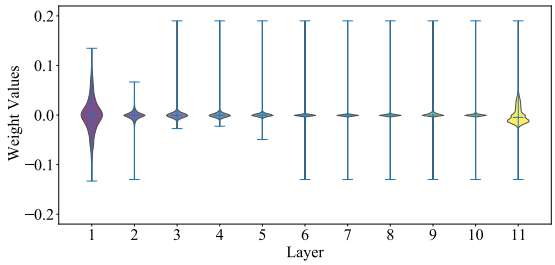
The results of the SFT attack are shown in Table 7. More specifically, the VGG-11 model fine-tuned on the NNSplitter obfuscated models using just 1% of the CIFAR-10 dataset is able to attain a top-1 accuracy close to the baseline accuracy (91.41%). In contrast, the recovered model from the $\mathsf{O}^2\mathsf{Splitter}$-obfuscated model (VGG-11) only reaches an accuracy of 12.59% when fine-tuned on 10% of the size of CIFAR-10. The reason for the success of this attack on NNSplitter is that most of its obfuscated weights are concentrated around two common values, i.e., c_{max} and c_{min}, making the identification easy. Similarly, the SFT attack also reaches a top-1 accuracy of 84.53% on just 1% when fine-tuned on the model obfuscated via CoreLocker. SFT shows a much better increase in top-1 accuracy compared to existing fine-tuning methods like FTAL and FTLL on lesser data availability, with the exception of $\mathsf{O}^2\mathsf{Splitter}$

The obfuscation accuracy and the results of the SFT attack on the LLMs are shown in Table 8. In general, it can be observed that both CoreLocker and $\mathsf{O}^2\mathsf{Splitter}$ are resilient against SFT attacks, with $\mathsf{O}^2\mathsf{Splitter}$ being slightly more resilient. For CoreLocker, the fine-tuning accuracy using SFT reaches 0.85% on the 10% data split, while for $\mathsf{O}^2\mathsf{Splitter}$, it only reaches 0.53% on the same split. This provides empirical evidence for the $\mathsf{O}^2\mathsf{Splitter}$ being flexible and robust against all the mentioned **C1**-class fine-tuning attacks on models for classification and generative tasks.

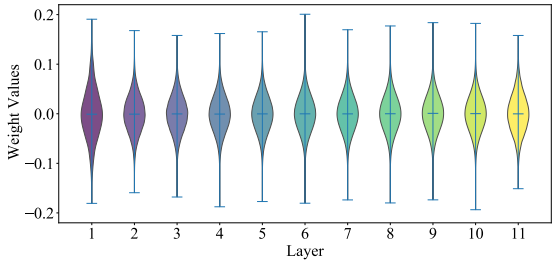
Limited Label Fine-tuning Attack. To assess the case where attackers work with fewer or different labels, we perform a fine-tuning attack (RTLL) on the last layer of obfuscated models, using 5% of CIFAR-10 training data with 1) 2 classes and 2) 5 classes, and evaluate on testing data limited to the same classes.



(a) VGG-11



(b) Obfuscated VGG-11 (NNSplitter)



(c) Obfuscated VGG-11 ($\mathsf{O}^2\mathsf{Splitter}$)

Fig. 3. Comparison of weight distributions of original VGG-11 (3(a)), Obfuscated VGG-11 model via NNSplitter (3(b)) and Obfuscated VGG-11 model via $\mathsf{O}^2\mathsf{Splitter}$ (3(c)). All models are trained on the CIFAR-10 dataset.

Table 9. Comparison of fine-tuning attack (RTLL) on limited-label availability setting with 2 and 5 class datasets. ($\epsilon = 50, \delta = 2^{-32}$).

Method	#Classes	Obf. Acc. (%)	Fine-tune Acc. (%) ↓
CoreLocker	2	45.23	50.95
NNSplitter	2	51.88	80.10
$\mathsf{O^2Splitter}$	2	46.62	47.85
CoreLocker	5	20.22	81.28
NNSplitter	5	23.08	93.54
$\mathsf{O^2Splitter}$	5	20.00	21.80

Table 9 summarizes the results of the attack in the limited-label setting. Both CoreLocker and NNSplitter show vulnerability to fine-tuning attacks, regaining performance on a subset of original labels. Notably, NNSplitter achieves 80.10% accuracy when fine-tuned on a 2-class dataset, while CoreLocker recovers 81.28% accuracy, and NNSplitter achieves 93.54% on the 5-class testing data. In contrast, the $\mathsf{O^2Splitter}$ remains robust, limiting the attacker to 21.80% accuracy on the 5-class test dataset. This robustness arises from the obfuscation of "all" weights in $\mathsf{O^2Splitter}$, whereas CoreLocker and NNSplitter partially obfuscate weights, preserving the feature representation in the obfuscated models.

5.5 Potential Vulnerabilities and Mitigation

Risk of Misconfiguration. A common pitfall is to use the same $\{\mathsf{ID}_i\}$ in $\mathsf{O^2Splitter}$ for different versions of one original model. Since the PRF key is a long-term secret that does not change, the same $\{\mathsf{ID}_i\}$ leads to the same noises. In the worst case of this misconfiguration, where one version of the original model is also leaked, attacker $\mathcal{A}$ can compute the noises and recover every other version. Effective mitigation is to use a unique model identifier for each model type and version and include them in each ID_i, which makes it intractable to find repetition in PRF outputs and noises. Rotation of the PRF key can also help.

Vulnerabilities against Other Attacks. Since we assume that the model is recovered completely in the memory for services, $\mathcal{A}$ may corrupt the working memory and find out the original model. Effective mitigation includes memory isolation [8, 41] and sandboxing [47], which are orthogonal to obfuscation methods.

6 Conclusion and Future Work

In comparison with state-of-the-art methods, $\mathsf{O^2Splitter}$ shows high effectiveness and strong resilience against **C1**-attacks in the experiments. In the future, we seek for a combination of the $\mathsf{O^2Splitter}$ with model architectural obfuscation. We would also like to investigate the possible ways to upgrade $\mathsf{O^2Splitter}$ so that it can have resilience against **C2** and **C3** attackers. In addition, formalizing the relation between $\mathsf{O^2Splitter}$-parameters (or noise scale) and the risk of concrete extraction attacks, in theory, is also interesting.

References

1. Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F.L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al.: Gpt-4 technical report. arXiv preprint arXiv:2303.08774 (2023)
2. Adi, Y., Baum, C., Cisse, M., Pinkas, B., 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)
3. Barker, E., Roginsky, A.: Transitioning the use of cryptographic algorithms and key lengths. Tech. rep., National Institute of Standards and Technology (2018)
4. Carlini, N., Tramer, F., Wallace, E., Jagielski, M., Herbert-Voss, A., Lee, K., Roberts, A., Brown, T., Song, D., Erlingsson, U., et al.: Extracting training data from large language models. In: 30th USENIX Security Symposium (USENIX Security 21). pp. 2633–2650 (2021)
5. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon, submission to the caesar competition. Institute for Applied Information Processing and Communications, Graz University of Technology, Graz (2014)
6. Dwork, C., Roth, A., et al.: The algorithmic foundations of differential privacy. Foundations and Trends® in Theoretical Computer Science **9**(3–4), 211–407 (2014)
7. Dworkin, M.J., Barker, E., Nechvatal, J.R., Foti, J., Bassham, L.E., Roback, E., Dray Jr, J.F., et al.: Advanced encryption standard (aes) (2001)
8. Frassetto, T., Jauernig, P., Liebchen, C., Sadeghi, A.R.: {IMIX}:{In-Process} memory isolation {EXtension}. In: 27th USENIX Security Symposium (USENIX Security 18). pp. 83–97 (2018)
9. Gerz, D., Su, P.H., Kusztos, R., Mondal, A., Lis, M., Singhal, E., Mrkšić, N., Wen, T.H., Vulić, I.: Multilingual and cross-lingual intent detection from spoken data. arXiv preprint arXiv:2104.08524 (2021)
10. Goldstein, B.F., Patil, V.C., Ferreira, V.C., Nery, A.S., França, F.M., Kundu, S.: Preventing dnn model ip theft via hardware obfuscation. IEEE Journal on Emerging and Selected Topics in Circuits and Systems **11**(2), 267–277 (2021)
11. Grassi, L.: Mixture differential cryptanalysis: a new approach to distinguishers and attacks on round-reduced aes. IACR Transactions on Symmetric Cryptology pp. 133–160 (2018)
12. Hayes, J., Balle, B., Mahloujifar, S.: Bounding training data reconstruction in dp-sgd. Advances in neural information processing systems **36**, 78696–78722 (2023)
13. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
14. Hong, S., Davinroy, M., Kaya, Y., Dachman-Soled, D., Dumitraş, T.: How to 0wn nas in your spare time. arXiv preprint arXiv:2002.06776 (2020)
15. Kariyappa, S., Prakash, A., Qureshi, M.K.: Maze: Data-free model stealing attack using zeroth-order gradient estimation. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 13814–13823 (2021)
16. Katz, J., Lindell, Y.: Introduction to modern cryptography (2020)
17. Knott, B., Venkataraman, S., Hannun, A., Sengupta, S., Ibrahim, M., van der Maaten, L.: Crypten: Secure multi-party computation meets machine learning. Advances in Neural Information Processing Systems **34**, 4961–4973 (2021)
18. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images (2009)

19. Kulynych, B., Gomez, J.F., Kaissis, G., du Pin Calmon, F., Troncoso, C.: Attack-aware noise calibration for differential privacy. Advances in Neural Information Processing Systems **37**, 134868–134901 (2024)
20. Lee, J.W., Kang, H., Lee, Y., Choi, W., Eom, J., Deryabin, M., Lee, E., Lee, J., Yoo, D., Kim, Y.S., et al.: Privacy-preserving machine learning with fully homomorphic encryption for deep neural network. iEEE Access **10**, 30039–30054 (2022)
21. Li, J., He, Z., Rakin, A.S., Fan, D., Chakrabarti, C.: Neurobfuscator: A full-stack obfuscation tool to mitigate neural architecture stealing. In: 2021 IEEE International Symposium on Hardware Oriented Security and Trust (HOST). pp. 248–258. IEEE (2021)
22. Lipmaa, H., Rogaway, P., Wagner, D.: Ctr-mode encryption. In: First NIST Workshop on Modes of Operation. vol. 39. Citeseer. MD (2000)
23. Liu, G., Wang, S., Wan, B., Wang, Z., Wang, C.: Ml-stealer: Stealing prediction functionality of machine learning models with mere black-box access. In: 2021 IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom). pp. 532–539. IEEE (2021)
24. Liu, K., Dolan-Gavitt, B., Garg, S.: Fine-pruning: Defending against backdooring attacks on deep neural networks. In: International symposium on research in attacks, intrusions, and defenses. pp. 273–294. Springer (2018)
25. Liu, Y.: Roberta: A robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692 **364** (2019)
26. Liu, Y., Wen, R., He, X., Salem, A., Zhang, Z., Backes, M., De Cristofaro, E., Fritz, M., Zhang, Y.: {ML-Doctor}: Holistic risk assessment of inference attacks against machine learning models. In: 31st USENIX Security Symposium (USENIX Security 22). pp. 4525–4542 (2022)
27. Liu, Y., Srivastava, A.: Ganred: Gan-based reverse engineering of dnns via cache side-channel. In: Proceedings of the 2020 ACM SIGSAC Conference on Cloud Computing Security Workshop. pp. 41–52 (2020)
28. Ma, C., Chen, L., Yong, J.H.: Simulating unknown target models for query-efficient black-box attacks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 11835–11844 (2021)
29. Mironov, I., Pandey, O., Reingold, O., Vadhan, S.: Computational differential privacy. In: Annual International Cryptology Conference. pp. 126–142. Springer (2009)
30. Mohseni, A., Moaiyeri, M.H., Amirany, A., Rezayati, M.H.: Protecting the intellectual property of binary deep neural networks with efficient spintronic-based hardware obfuscation. IEEE Transactions on Circuits and Systems I: Regular Papers (2024)
31. Nanayakkara, P., Smart, M.A., Cummings, R., Kaptchuk, G., Redmiles, E.M.: What are the chances? explaining the epsilon parameter in differential privacy. In: 32nd USENIX Security Symposium (USENIX Security 23). pp. 1613–1630 (2023)
32. Nayan, T., Guo, Q., Al Duniawi, M., Botacin, M., Uluagac, S., Sun, R.: {SoK}: All you need to know about {On-Device}{ML} model extraction-the gap between research and practice. In: 33rd USENIX Security Symposium (USENIX Security 24). pp. 5233–5250 (2024)
33. Oliynyk, D., Mayer, R., Rauber, A.: I know what you trained last summer: A survey on stealing machine learning models and defences. ACM Computing Surveys **55**(14s), 1–41 (2023)
34. Orekondy, T., Schiele, B., Fritz, M.: Knockoff nets: Stealing functionality of black-box models. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 4954–4963 (2019)

35. Pal, S., Gupta, Y., Shukla, A., Kanade, A., Shevade, S., Ganapathy, V.: Activethief: Model extraction using active learning and unannotated public data. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 34, pp. 865–872 (2020)
36. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al.: Language models are unsupervised multitask learners. OpenAI blog **1**(8), 9 (2019)
37. Rajpurkar, P., Zhang, J., Lopyrev, K., Liang, P.: SQuAD: 100,000+ questions for machine comprehension of text. In: Su, J., Duh, K., Carreras, X. (eds.) Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing. pp. 2383–2392. Association for Computational Linguistics, Austin, Texas (Nov 2016). https://doi.org/10.18653/v1/D16-1264, https://aclanthology.org/D16-1264/
38. Ren, X., Zhou, P., Meng, X., Huang, X., Wang, Y., Wang, W., Li, P., Zhang, X., Podolskiy, A., Arshinov, G., et al.: Pangu-{\Sigma}: Towards trillion parameter language model with sparse heterogeneous computing. arXiv preprint arXiv:2303.10845 (2023)
39. Sabt, M., Achemlal, M., Bouabdallah, A.: Trusted execution environment: What it is, and what it is not. In: 2015 IEEE Trustcom/BigDataSE/Ispa. vol. 1, pp. 57–64. IEEE (2015)
40. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., 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)
41. Schrammel, D., Weiser, S., Steinegger, S., Schwarzl, M., Schwarz, M., Mangard, S., Gruss, D.: Donky: Domain keys–efficient {In-Process} isolation for {RISC-V} and x86. In: 29th USENIX Security Symposium (USENIX Security 20). pp. 1677–1694 (2020)
42. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
43. Sternby, J., Johansson, B., Liljenstam, M.: Neural network model obfuscation through adversarial training. In: 2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid). pp. 782–789. IEEE (2022)
44. Sun, Z., Sun, R., Lu, L., 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)
45. Suya, F., Mahloujifar, S., Suri, A., Evans, D., Tian, Y.: Model-targeted poisoning attacks with provable convergence. In: International Conference on Machine Learning. pp. 10000–10010. PMLR (2021)
46. Tramèr, F., Zhang, F., Juels, A., Reiter, M.K., Ristenpart, T.: Stealing machine learning models via prediction {APIs}. In: 25th USENIX security symposium (USENIX Security 16). pp. 601–618 (2016)
47. Voulimeneas, A., Vinck, J., Mechelinck, R., Volckaert, S.: You shall not (by) pass! practical, secure, and fast pku-based sandboxing. In: Proceedings of the Seventeenth European Conference on Computer Systems. pp. 266–282 (2022)
48. Wang, B., Gong, N.Z.: Stealing hyperparameters in machine learning. In: 2018 IEEE symposium on security and privacy (SP). pp. 36–52. IEEE (2018)
49. Wang, Z., Ma, Z., Feng, X., Sun, R., Wang, H., Xue, M., Bai, G.: Corelocker: Neuron-level usage control. In: 2024 IEEE Symposium on Security and Privacy (SP). pp. 222–222. IEEE Computer Society (2024)
50. Warden, P.: Speech commands: A dataset for limited-vocabulary speech recognition. arXiv preprint arXiv:1804.03209 (2018)
51. Wikipedia: Amazon alexa. https://en.wikipedia.org/wiki/Amazon_Alexa (2025), accessed: 07, Jan., 2025

52. Xu, H., Su, Y., Zhao, Z., Zhou, Y., Lyu, M.R., King, I.: Deepobfuscation: Securing the structure of convolutional neural networks via knowledge distillation. arXiv preprint arXiv:1806.10313 (2018)
53. Xu, M., Liu, J., Liu, Y., Lin, F.X., Liu, Y., Liu, X.: A first look at deep learning apps on smartphones. In: The World Wide Web Conference. pp. 2125–2136 (2019)
54. Xue, M., Wu, Z., Zhang, Y., Wang, J., Liu, W.: Advparams: An active dnn intellectual property protection technique via adversarial perturbation based parameter encryption. IEEE Transactions on Emerging Topics in Computing **11**(3), 664–678 (2022)
55. Yang, P., Chen, J., Hsieh, C.J., Wang, J.L., Jordan, M.: Ml-loo: Detecting adversarial examples with feature attribution. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 34, pp. 6639–6647 (2020)
56. Yu, C., Chen, J., Xue, Y., Liu, Y., Wan, W., Bao, J., 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)
57. Yu, H., Yang, K., Zhang, T., Tsai, Y.Y., Ho, T.Y., Jin, Y.: Cloudleak: Large-scale deep learning models stealing through adversarial examples. In: NDSS. vol. 6, p. 3 (2020)
58. Zhou, M., Park, A., Zheng, W., Shi, E.: Piano: Extremely simple, single-server pir with sublinear server computation. In: 2024 IEEE Symposium on Security and Privacy (SP). pp. 55–55. IEEE Computer Society (2023)
59. Zhou, T., Luo, Y., Ren, S., Xu, X.: Nnsplitter: an active defense solution for dnn model via automated weight obfuscation. In: International Conference on Machine Learning. pp. 42614–42624. PMLR (2023)
60. Zhou, T., Ren, S., 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)

A Proof of Theorem 1

Proof. (sketch) If a random function F is used instead of PRF, the DP-guarantee is exactly (ϵ, δ), where δ is the failing probability of the mechanism. A DP-adversary $\mathcal{A}$ may perform differently in the real-random case and in the PRF-case. This $\mathcal{A}$ can be used to construct a PRF distinguisher $\mathcal{B}$: $\mathcal{B}$ simulates the DP-game [6] for $\mathcal{A}$ using its own oracle in the PRF-security game [16], and $\mathcal{B}$ simply outputs what $\mathcal{A}$ outputs. But for any efficient $\mathcal{B}$, the probability of $\mathcal{B}$'s performance change is upper-bounded by the security guarantee of PRF, i.e., δ_{PRF}. Due to the symmetry of DP, the advantage change of $\mathcal{A}$ is bounded by two times the advantage of $\mathcal{B}$, i.e, the failing probability against $\mathcal{A}$ is bounded by $\delta + 2 \cdot \delta_{\mathsf{PRF}}$. □