To Appear in the 45th IEEE Symposium on Security and Privacy, May 20-23, 2024.

CORELOCKER: Neuron-level Usage Control

Zihan Wang[†,§], Zhongkui Ma[†], Xinguo Feng[†], Ruoxi Sun[§], Hu Wang[‡], Minhui Xue[§], Guangdong Bai[†✉]

[†]*The University of Queensland, Australia*
[§]*CSIRO's Data61, Australia*
[‡]*The University of Adelaide, Australia*

***Abstract*—The growing complexity of deep neural network models in modern application domains necessitates a complex training process that involves extensive data, sophisticated design, and substantial computation. The trained model inherently encapsulates the intellectual property owned by the model developer (or the *model owner*). Consequently, safeguarding the model from unauthorized use by entities who obtain access to the model (or the *model controllers*), *i.e.,* preserving the fundamental rights and proprietary interests of the model owner, has become a critical necessity.**

In this work, we propose CORELOCKER, employing the strategic extraction of a small subset of significant weights from the neural network. This subset serves as the *access key* to unlock the model's complete capability. The extraction of the key can be customized to varying levels of utility that the model owner intends to release. Authorized users with the access key have full access to the model, while unauthorized users can have access to only part of its capability. We establish a formal foundation to underpin CORELOCKER, which provides crucial lower and upper bounds for the utility disparity between pre- and post-protected networks. We evaluate CORELOCKER using representative datasets such as Fashion-MNIST, CIFAR-10, and CIFAR-100, as well as real-world models including VggNet, ResNet, and DenseNet. Our experimental results confirm its efficacy. We also demonstrate CORELOCKER's resilience against advanced model restoration attacks based on fine-tuning and pruning.

1. Introduction

Deep neural networks (DNNs) have seen remarkable success in various fields, yet developing a high-quality model often demands substantial resources. This includes sophisticated architectural design, extensive high-quality data, meticulous fine-tuning, and optimization [1], and substantial computational power [2], [3]. Taking GPT-3 [4] as an example, it consists of 175 billion parameters and takes 355 GPU-years and $4.6M for a single training run [5]. A model thus represents a valuable intellectual property (IP), and transforms into a treasure of its developer. For instance, a recent study by Fortune [6] shows that ChatGPT has attracted 100 million active users two months after its launch, and earns $80 million per month for OpenAI.

In contrast to the traditional deployment of DNN models within the server or cloud under the direct control of the model owner, various scenarios, such as commercial partnerships, consulting services, and on-device inference, entail the transfer of the model to an external party, referred to as the *model controller*. Nevertheless, once the model is handed over to the model controller, the owner loses control over it. Indeed, a recent study [7] on 1,468 mobile apps uncovers that 41% of them fail to secure their DNN models against on-device model inference attacks, allowing the attacker to extract all model parameters through reverse engineering. Consequently, unethical controllers may exploit the obtained model for unscrupulous competition or unauthorized subletting, posing financial losses for the model owners. Moreover, malicious controllers can abuse the model to facilitate the generation of adversarial examples [8]–[10] to attack the model owner's legitimate services.

Two lines of research are seemingly potential to alleviate this challenge, including *passive methods* and *active methods*. Passive methods involve embedding a watermark [11]–[16] or signature [17], [18] into the model, which manifests only when certain inputs are given to the model, enabling the owner to claim the ownership of the model. Such methods often fail to prevent unauthorized usage after the model's exposure, and the incentives for theft remain. On the other hand, active methods introduce keyed data and keyed neurons. The former integrates a secret key into the training data during the preprocessing and trains a model to operate only with *key-preprocessed* inputs [19], [20]. This strategy entails retraining models for each key, rendering it extremely time-consuming and unsuitable for pre-trained models. The latter embeds the key into neurons, which undermines the utility of the entire model unless the key is known by the model controller for neutralization [21], [22]. Nevertheless, these approaches entail meticulous perturbation generation and specialized hyperparameter selection when determining the key, and the key has been shown to be detectable and removable through out-of-distribution value detection [23], [24], pruning [25] and fine-tuning [26].

Our work. In this work, we explore a novel defense paradigm that takes a step forward from conventional parameter perturbation approaches, focusing instead on the fundamental structural bedrock that determines a neural network's functionality. Our work endows the model owner

with the capability to tailor the model into a low-utility version, which can be fully restored after authorization. This capability holds broad applicability in online services such as machine learning as a service (MLaaS) [27], and on-device model deployment, where model owners strategically offer lower-utility models to entice users toward purchasing full- or higher-utility versions. Real-world examples include cutout.pro [28] and together.ai [29], which provide models with free low utility options or varying capabilities at different price points. Specifically, we aim to answer the research question of *how to degrade a model's performance to a lower utility level while ensuring that the full utility can be efficiently restored by authorized controllers?*

We propose CORELOCKER, which locks a minimal subset of neurons from the pre-protected neural network (denoted as f^*), parameterized by $0 < \alpha < 1$, leading to a subnetwork of f^* with partial or none utility (which is thus denoted as f^α). We formalize an ideally-unusable network (denoted as f^0) which conducts random inference, and in the extreme requirement where the capability of the network needs to be fully hidden, the resulting f^α should be proximate to f^0 in its performance, *i.e.,* $f^* \gg f^\alpha \approx f^0$ if we abuse f^- to denote the performance of themselves. CORELOCKER aims for a *training data-agnostic* and *retraining-free* process by directly operating on off-the-shelf pre-trained networks, making it well-suited for seamless integration across diverse neural network architectures.

CORELOCKER's solution stems from an intuitive yet compelling insight: if the performance of a neural network is disproportionately dependent on a specific small subset of critical weights, removing these weights is likely to have the potential to incapacitate the network. In particular, it employs an efficient *selective weight removal*, based on the intrinsic attributes of a neural network, including the ℓ_1-*norm* of weights and the *scaling factors* of batch normalization layers, as key criteria for its weight selection (***property of efficient extraction***). These selected weights, referred to as the *access key*, are few in number and are handled through the secure channel to ensure the model's original utility is readily recoverable upon restoration (***property of efficient authorized restoration***). However, the removal of the key makes it extremely complex to restore the original network (***property of complex unauthorized restoration***). As shown in a recent study [30], it requires millions of years for an IBM Summit supercomputer to crack a multi-layer neural network.

We provide a robust formal foundation for CORELOCKER's neuron-level usage control, which establishes both lower and upper bounds of the proximity and disparity among f^*, f^α, and f^0. Our formalization captures two essential characteristics of neural networks, including the *distribution preservation* during training [31], [32] and the *impact concentration* of weights. This formalization enables the use of probabilistic approaches to analyze the behavior of the trained network, by highlighting that the weights of a trained network preserve the probability distribution established during initialization [31], [32]. The impact concentration underscores that *the performance of a neural network is largely reliant on a crucial subset of weights*. This concept is seemingly the dual of the principle behind various pruning studies [33]–[36] that the majority of neurons can be pruned with negligible impact. Despite the pruning principle formally proved by a recent study [34], establishing a formal proof for impact concentration is more challenging, as it has to handle duplication and correlation among neurons. We for the first time address it in this work through deriving the bounds of the gaps among the outputs of f^* and f^α layer by layer (see Section 4).

We conduct both qualitative and practical evaluations to investigate the performance of CORELOCKER. The qualitative analysis corroborates our theoretical framework, empirically revealing the relationship between the depth and width of a target network and the output disparity among f^* and f^α. The practical study illustrates the effectiveness of CORELOCKER on protecting real-world models. In particular, we apply CORELOCKER to three commonly-used datasets, along with prevalent deep neural network architectures such as ResNet-164 [37]. With a global extraction ratio set as low as 0.05, the performance of all models diminishes to the level of random guessing (*e.g.,* 1% for CIFAR-100) across all datasets. Furthermore, we expose CORELOCKER to attacks based on fine-tuning [26] and pruning [25], showcasing its resilience against advanced adversarial strategies.

Contributions. Our main contributions in this work are listed as follows.

- **A new research problem.** We establish a crucial research problem of AI model usage control, which requires a neuron-level lock of the model's utility while ensuring that its full utility can be efficiently restored for authorized use with an access key.
- **A generic defense paradigm.** CORELOCKER is the first *practical* active schema for neuron-level usage control, distinguished by its lightweight, data-agnostic, and retraining-free attributes. It efficiently identifies and extracts the access key out of the target model.
- **A formal framework and theoretical analysis.** We establish a formal foundation for the theoretical analysis of model usage control. It for the first time formalizes several key characteristics of weights in a neural network, and provides crucial bounds that guarantee the efficacy of CORELOCKER, an independent confirmation of provable guarantees of AI model usage.
- **An empirical evaluation.** We implement CORELOCKER and evaluate it on representative datasets and real-world models. Our results confirm its efficacy. For example, with a mere 0.05 extraction ratio, it efficiently and consistently degrades the performance of all tested models to the level of random guessing. We also demonstrate that CORELOCKER remains robust against advanced adversarial strategies, such as fine-tuning attacks and pruning attacks.

Notations. The notations used throughout this paper are listed as follows. Lower-case Latin letters, *e.g.,* a, i, u, and x, denote variables and upper-case Latin letters denote constants like C, M, and N, or matrices like S and W. The calligraphic font represents a set, *e.g.,* $\mathcal{I}$. Greek letters are used as hyperparameters. Bold lower-case letters refer to the vector, such as $\boldsymbol{a}$, $\boldsymbol{x}$, $\boldsymbol{y}$. $\boldsymbol{x}_i$ is the i-th element of the vector $\boldsymbol{x}$. $\mathbf{0}$ or $\mathbf{1}$ represent a vector/matrix that has all its elements equal to zero or one. We use W_i to represent the i-th row of the matrix W and $W_{i,j}$ the entry/element of the i-th row and j-th column. W^T is the matrix transpose of W. The norm of a matrix refers to the operator norm, *i.e.,* $\|A\|_n = \sup\{A\boldsymbol{x} \mid \|\boldsymbol{x}\|_n \leq 1, \boldsymbol{v} \in \mathbb{R}^n, A \in \mathbb{R}^{m\times n}, n > 0\}$.

2. Problem Formulation

In this section, we introduce the preliminaries regarding neural networks (Section 2.1) to facilitate the understanding of our work. We then define the threat model (Section 2.2) and the problem of neuron-level usage control (Section 2.3).

2.1. Neural Networks

A neural network can be conceptualized as a composite of linear and nonlinear (activation) functions, with an architecture containing multiple hidden layers. The activation function is denoted as $\phi(\cdot) : \mathbb{R} \to \mathbb{R}$.

The standard *fully-connected neural network* $f^*(\boldsymbol{x}) : \mathbb{R}^{N^{(0)}} \to \mathbb{R}^{N^{(M)}}$ with M layers can be structured as

$$W^{*,(M)}\phi\left(\cdots W^{*,(2)}\phi\left(W^{*,(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right) + \boldsymbol{b}^{(2)}\right) + \boldsymbol{b}^{(M)},$$

where the input layer is denoted as the 0-th, and the M-th represents the output layer[1]. For each layer $m \in [M]$, the associated weight matrix is expressed as $W^{*,(m)}$. The network receives an input $\boldsymbol{x}$, which generates a corresponding output $\boldsymbol{y}^*$. It is posited that for any given input $\boldsymbol{x}$, $\|\boldsymbol{x}\|_2 \leq 1$. The term $\boldsymbol{y}^{*,(m)}$ signifies the output of the m-th layer, with $\boldsymbol{y}^{*,(0)} = \boldsymbol{x}$ and $\boldsymbol{y}^{*,(M)} = \boldsymbol{y}^*$. The notation $N^{(m)}$ represents the count of neurons in the m-th layer, with N being the maximum number among the neuron counts across all layers, *i.e.,* $N = \max\{N^{(1)}, N^{(2)}, \cdots, N^{(M)}\}$.

The *convolutional neural network* is transformed into a specialized form of a fully connected network in this work. Specifically, it involves processing an input tensor $X \in \mathbb{R}^{c\times p\times p}$ and producing an output tensor $Y \in \mathbb{R}^{c'\times p\times p}$. The convolutional filters, denoted as $F^*_{s,t} \in \mathbb{R}^{q\times q}$, are associated with the s-th ($s \in [c']$) output channel and the t-th ($t \in [c]$) input channel, where c and c' are the numbers of input and output channels, and p denotes the shape of the input and output tensor. We establish that these convolutional filters can be converted into weight matrices W^*. The details of this transformation are further explored in Section 4.2.2.

1. A component-wise function applied to a vector operates on each element individually.

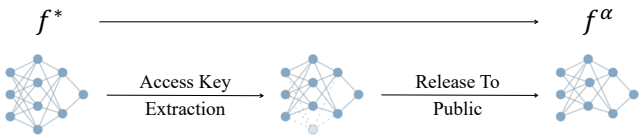


Figure 1: An illustration of the CORELOCKER workflow.

2.2. Threat Model

In this section, we outline the scope of CORELOCKER, and define the attacks CORELOCKER aims to defend against, in terms of the information available to them and the operations they can conduct on the obtained model.

Scope. The attacker CORELOCKER takes into consideration aims to obtain the complete neural network model. The possession of the model by them opens avenues for various potential abuses, such as monetization through unauthorized commercial services, or facilitating the generation of adversarial examples [8], [9] against the model owner's legitimate services. The focus of CORELOCKER is to establish a formally assured neuron-level access key generation, aiming to achieve the three fundamental properties outlined in Section 1. With access keys, existing techniques like hardware-assisted [38] and TEE-based management [39] can offer high-assurance or revocable usage control.

Attacker capabilities. The attacker has access to weight parameters of the model that is under the control of the model controller, either via a public cloud platform or due to information breaches caused by malicious malware infection or insider sources. We assume the access is *white-box*, which is in favor of the attacker. Thus, the attacker is also aware of the network architecture employed in training the model. This assumption makes the threat model pragmatic, as industrial applications typically adopt published DNN architectures, which have demonstrated high modeling capabilities. The attacker owns limited training data that is of identical distribution as the model's training data, as otherwise, they can train a competitive model on their own.

The attacker may detect the manipulation of the protection mechanism on the network and counteract their negative effects [23], [24]. We consider the attacker's strategy of recovering the utility of the protected model through fine-tuning [26] or model pruning [25], which intend to adjust or remove the parameters that are manipulated. Once the model functionality recovers, the attacker adapts it for intended applications.

2.3. Neuron-level Usage Control

Given a target neural network f^*, the objective of CORELOCKER is to locate a sub-network f^α, where α is the retention ratio, such that f^α becomes less functional without the extracted access key. In extreme requirements where the network's functionality needs to be completely hidden, the resulted f^α should become non-functional in the absence of the extracted access key. Figure 1 provides a brief overview of this workflow.

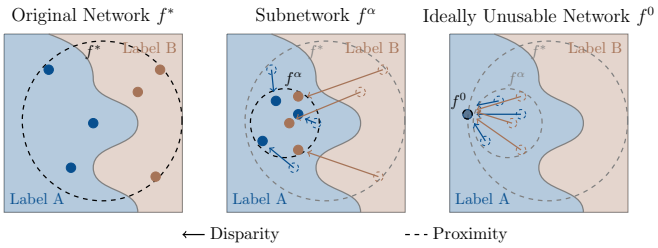


Figure 2: Illustration of the proximity and disparity among f^*, f^α, and f^0.

Ideally unusable network f^0. CORELOCKER is designed to meet the extreme requirement that f^α loses all its utility. To represent this as a reference model, we define an ideally unusable network, denoted as f^0, by setting all weights of f^* to zeros, such that f^* transforms into a constant function, *i.e.,* $f^* = f^0 = \boldsymbol{c}$, where $\boldsymbol{c}$ is a constant vector depending on the bias of the last layer of f^*. When the test dataset is evenly distributed, meaning that each class contains an equal number of samples, the constant nature of f^0's output implies that it will correctly identify only the class that matches this constant output. Therefore, the accuracy of f^0 is effectively $1/N^{(M)}$, with $N^{(M)}$ being the total number of classes. An illustration is displayed in Figure 2.

3. Our Approach: CORELOCKER

The core of CORELOCKER is to extract the access key from f^*. In this section, we detail this process.

3.1. Access Key Extraction

CORELOCKER extracts the access key by zeroing out a certain number of weights by the extraction ratio $1-\alpha$ for each weight matrix $W^{(m)}$ for fully-connected networks f^* and filters $F^*_{s,t}$ for convolutional networks. The retain ratio α describes the ratio of retained weights or filters. After this extraction process, the resulting network becomes a subnetwork of f^* by retain ratio α, denoted by f^α. We denote the weight or the filter after extracting as $W^{(m)}$ or $F^{(m)}$, compared to the original ones $W^{*,(m)}$ or $F^{*,(m)}$. In this work, there is no extraction in input and output layers, *i.e.,* $W^{(1)} = W^{*,(1)}$ and $W^{(M)} = W^{*,(M)}$. Specifically, the CORELOCKER's access key extraction process is defined as follows.

Definition 1 (Access Key Extraction)**.** *The access key extraction refers to zeroing out the weight on the weight matrices* W^*[2] *according to the indicator set* $\mathcal{I}$*, which contains the indicator* (i,j) *of the extracted weights* $W^*_{i,j}$*. To determine* $\mathcal{I}$*, each entry* $W^*_{i,j}$ *of* W^* *is listed in ascending order based on the specific extraction indicator with the total number of* D*. By setting an extraction threshold* D^α *(*$\alpha \in [0,1]$*) such that* $1 \le D^\alpha \le D$*, we let* $\mathcal{I} := \{(i_k, j_k) \mid D^\alpha \le k \le D, k \in \mathbb{Z}\}$ *to set the indices of weights to be extracted, where* k *means the entry* $W^*_{i,j}$ *is the* k*-th one in the ordering.*

2. The superscript for the layer index is omitted when referring to each layer unambiguously.

Remark. This definition aptly applies to both weights and filters, given their conceptual similarity shown by Han *et al.* [40]. For a convolutional neural network, the ℓ_1 norm of the filter, specifically the sum of the absolute values of the filter weights, serves as the extraction indicator.

Given the objective of rendering f^α dysfunctional, a natural question might arise: *how many weights need to be extracted to achieve a utility that is close enough to that of* f^0*?* To answer this question, the underlying intuition is that even minor alterations within a neural network can initiate a *butterfly effect*, where these changes, though small, build up and magnify, significantly impacting the network's behavior. Adopting this perspective sets a clear goal: to identify a sufficiently small extraction ratio $1-\alpha$ such that

$$\|f^\alpha - f^0\| \le \epsilon,$$

where ϵ represents a small constant and $\|\cdot\|$ denotes a measurement to quantify the utility difference between pre- and post-protected models, *e.g.,* the accuracy of a neural network on a specified dataset.

3.2. Extraction Criteria

The efficacy of CORELOCKER's extraction procedure relies on the precise selection of weight subsets, whose removal is expected to notably affect the model's performance. Given this context, it is a logical first step to extract weights linked to larger layer outputs. This is based on the insight that such weights in a neural network are typically considered more impactful due to their ability to significantly amplify inputs, thereby having a substantial influence on the model's predictions (discussed in Section 4.3). Therefore, we adopt the following metrics to assess the significance of weights in the extraction strategy.

ℓ_1-norm. The absolute value of each weight can be regarded as an indicator of the weight's importance, and for filters, this is measured by the sum of the absolute kernel weights, *i.e.,* the ℓ_1-norm [40], [41]. Specifically, for ℓ_1-norm based weight extraction, the process of extracting n filters from the i-th convolutional layer is succinctly described as follows.

1) to compute the sum of absolute kernel weights for each filter $F^*_{s,t}$,
2) to arrange the filters in ascending order based on the sum value,
3) to remove the n filters with the highest sum values, along with their associated feature maps, and
4) to remove the filters in the subsequent convolutional layer that are linked to these removed feature maps.

Filters that are removed during this process are then cataloged in the access key. Figure 3 intuitively shows that filters with a higher ℓ_1-norm value capture more detailed features. Filters with a lower sum of weights tend to capture a narrower range of features, whereas those with a higher

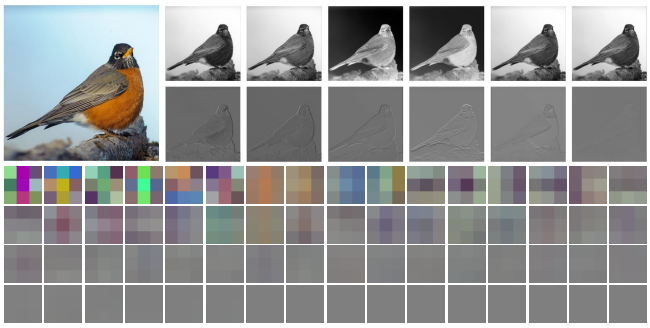


Figure 3: Visualization of feature maps (the top and bottom six) and corresponding filters (all 64 filters) from the first convolutional layer of a VggNet, sorted by filters' ℓ_1-norm.

weight sum exhibit a stronger capability in feature capture. This distinction underscores the efficacy of the proposed indicator.

Scaling factors. To further reduce the computational complexity of calculating the ℓ_1-norm across all weight matrices, we propose an alternative approach by extracting filters (or neurons in fully-connected layers) associated with higher scaling factors, as such filters are often linked to larger layer outputs. This strategy is inspired by the prevalent use of batch normalization (BN) layers following convolutional layers, which incorporate channel-specific scaling and shifting parameters [42]–[44]. Specifically, let y_{in} and y_{out} be the input and output of a BN layer, and the BN layer executes a transformation defined by the equation $y_{out} = \gamma(y_{in} - \mu)/\sqrt{\sigma^2 + \epsilon} + \beta$, where μ and σ are the mean and standard deviation values of inputs over a mini-batch, γ and β are trainable parameters (scale and shift) which provide the possibility of linearly transforming normalized values back to any scales [42], [44]. The γ parameters in BN layers can serve as effective scaling factors. The underlying rationale of this approach aligns with the ℓ_1-norm method in which it can be merged with the adjacent linear transformation. By adopting this metric, we can estimate the importance of filter weights without any direct calculations, rendering it particularly suitable for increasingly larger models.

4. Theoretical Analysis

In this section, we establish a formal foundation for the theoretical analysis of neuron-level usage control. Section 4.1 lists two assumptions about the Lipschitz property of activation functions and distribution of weights, which are used in the following theoretical proofs. Section 4.2 discusses the properties of weights and outputs of a neural network by taking weight matrices as random matrices to estimate operator norms. Section 4.3 and Section 4.4 provide the theoretical guarantee for CORELOCKER, forming the performance dynamics (Figure 4a) and disparity bounds (Figure 4b) as the extraction ratio increases.

Theoretical framework architecture. Theorem 1 in Section 4.3 reveals that selectively extracting a small subset of high-value weights can result in f^α nearing f^0, by analyzing the performance of f^α from the standpoint of output variances. Section 4.4 explores the range of local variations in output between f^α and f^*, detailing both the lower (Theorem 2) and upper bounds (Theorems 3 and 4) of these variations.

4.1. Assumptions

This section illustrates two assumptions about activation functions and weights of the neural network for the theoretical proofs.

Assumption 1 (Activation Function)**.** *The activation function ϕ satisfies the Lipschitz property that for any two values x_1 and x_2 in the domain of ϕ, $|\phi(x_1) - \phi(x_2)| \geq L|x_1 - x_2|$ holds true, where L represents a positive constant known as the Lipschitz constant. Additionally, ϕ is monotonic and $\phi(0) = 0$.*

Note that, for $\phi(0) = 0$, a linear transformation before the activation can be applied, allowing any activation function to satisfy this requisite. The properties defined in Assumption 1 are satisfied by most of those widely-used activation functions, such as ReLU and Tanh. In this work, we set a common Lipschitz constant L for all activation functions in a neural network.

Assumption 2 (Weight Distribution)**.** *The weights in the m-th layer are independently and identically distributed (i.i.d.), following a sub-Gaussian distribution* $\mathrm{subG}(\sigma^2)$*, where σ represents the standard deviation, and each weight $w_{ij}^{(m)}$ is bounded by $|w_{ij}^{(m)}| \leq C_w$, where C_w is the maximum of these bounds across all layers.*

The distribution of the weights is determined by the initialization, commonly a sub-Gaussian distribution, and affected by the training process. For example, Kaiming initialization is based on $\mathcal{N}(0, \frac{2}{n})$ [45], where n is the neuron number of the layer. As has been established in previous studies [31], [32], after the training processes, the weights still follow a sub-Gaussian distribution, which approaches the initialized distribution.

4.2. Properties of Neural Networks

This section discusses the properties of neural networks that our formal proofs rely on. We take weight matrices as random matrices because the numerous weights follow sub-Gaussian distributions after initialization and training (Assumption 2). The overall insight is to treat each layer as a composition of linear and non-linear transformations and estimate the output range by its norm. The linear transformation is equivalent to a matrix multiplication, and non-linear transformation (activation function) is upper bounded by the Lipschitz property.

The following properties focus on providing bounds for the weight matrix $\|W^{*,(m)}\|$, the output $\|\boldsymbol{y}^{*,(m)}\|_2$, and the difference between pre- and post-extraction weights $\|W^{(m)} - W^{*,(m)}\|$ for the m-th layer. Their proofs rely on

the properties of random matrices that are summarized as lemmas in Appendix A.1.

4.2.1. Properties of Fully-connected Networks. We start with exploring the properties of fully-connected networks.

Property 1 (Bounding Weight Matrix)**.** *Given a fully-connected network, its weight matrix $W^{*,(m)}$ satisfies, for any $\tau > 0$,*

$$\|W^{*,(m)}\|_2 \leq \sqrt{N} + C_s K_s^2(\sqrt{N} + \tau),$$

with a probability of at least $1 - 2e^{-\tau^2}$, where C_s is a universal constant and $K_s = \max_i \|W_i^{*,(m)}\|_2$.

Proof. By Lemma 3 and Assumption 2,

$$\begin{aligned}&\mathbb{P}\{s_1(W^{*,(m)}) \leq \sqrt{N^{(m)}} + C_s K_s^2(\sqrt{N^{(m-1)}} + \tau)\}\\ \geq& 1 - 2e^{-\tau^2}.\end{aligned}$$

Given $N = \max\{N^{(1)}, N^{(2)}, \cdots, N^{(M)}\}$,

$$\begin{aligned}s_1(W^{*,(M)}) \leq& \sqrt{N^{(m)}} + C_s K_s^2(\sqrt{N^{(m-1)}} + \tau)\\ \leq& \sqrt{N} + C_s K_s^2(\sqrt{N} + \tau).\end{aligned}$$

□

Property 2 (Bounding Output)**.** *Given a fully-connected network, its output $\boldsymbol{y}^{*,(m)}$ satisfies, for any constant $\tau > 0$,*

$$\|\boldsymbol{y}^{*,(m)}\|_2 \leq (L\lambda)^m \|\boldsymbol{x}\|_2,$$

with a probability of at least $(1 - 2e^{-\tau^2})^m$, where $\lambda = \sqrt{N} + C_s K_s^2(\sqrt{N} + \tau)$. C_s is a universal constant and K_s depends on weight matrices.

Proof. We denote $s_1(A)$ as the maximum singular value of A and set $\lambda = \sqrt{N} + C_s K_s^2(\sqrt{N} + \tau)$ as the upper bound of the maximum singular value of all weight matrices by Property 1. By Lipschiz property of the activation function ϕ and $\|A\boldsymbol{x}\| = s_1(A)\|\boldsymbol{x}\|$,

$$\begin{aligned}\|\boldsymbol{y}^{*,(m)}\|_2 =& \|\phi(W^{*,(m)}\boldsymbol{y}_k^{*,(m-1)})\|_2\\ =& \|\phi(W^{*,(m)}\boldsymbol{y}_k^{*,(m-1)}) - \phi(0)\|_2\\ \leq& L\|W^{*,(m)}\boldsymbol{y}^{*,(m-1)} - 0\|_2\\ =& L\|W^{*,(m)}\boldsymbol{y}^{*,(m-1)}\|_2\\ =& L s_1(W^{*,(m)})\|\boldsymbol{y}^{*,(m-1)}\|_2\\ \leq& L\lambda\|\boldsymbol{y}^{*,(m-1)}\|_2\\ \leq& (L\lambda)^2\|\boldsymbol{y}^{*(m-2)}\|_2\\ &\cdots\\ \leq& (L\lambda)^m\|\boldsymbol{x}\|_2.\end{aligned}$$

Then we calculate the probability that this inequality holds by Property 1,

$$\begin{aligned}&\mathbb{P}\{\|\boldsymbol{y}^{*,(m)}\|_2 \leq (L\lambda)^m\|\boldsymbol{x}\|_2\}\\ =& \prod_{k=1}^{m} \mathbb{P}\{\|W^{*,(k)}\boldsymbol{y}^{*,(k-1)}\|_2 \leq s_1(W^{*,(k)})\|\boldsymbol{y}^{*,(k-1)}\|_2\}\\ \geq& (1 - 2e^{-\tau^2})^m.\end{aligned}$$

□

Property 3 (Bounding Difference of Weight Matrices)**.** *Give a fully-connected network with its post-extraction network, $\|W^{(m)} - W^{*,(m)}\|_2$ satisfies, for any constant $\mu > 0$,*

$$\|W^{(m)} - W^{*,(m)}\|_2 \geq \mu,$$

with a probability of at most $\frac{6C_e C_w \sqrt{D_w}}{\mu}$, where D_w is the maximum number of extracted weights across all layers. C_e and C_w are universal constants.

Proof. Omitting the layer index, the non-zero entries of $W - W^*$ are $\{W_{i,j} - W_{i,j}^* \mid (i,j) \in \mathcal{I}\}$. Then, according to Lemma 2, for any $w = W_{i,j} - W_{i,j}^*$ we have

$$\begin{aligned}\mathbb{E}|w|^2 \leq& \mathbb{E}|W_{i_D,j_D}^*|^2\\ =& C_w^2 \frac{(D+1)D}{(D+2)(D+1)}\\ \leq& C_w^2 \frac{(2D)^2}{D^2}\\ =& 4C_w^2,\end{aligned}$$

$$\begin{aligned}\mathbb{E}|w|^4 \leq& \mathbb{E}|W_{i_D,j_D}^*|^4\\ =& C_w^4 \frac{(D+3)(D+2)(D+1)D}{(D+4)(D+3)(D+2)(D+1)}\\ \leq& C_w^4 \frac{(2D)^4}{D^4}\\ =& 16C_w^4.\end{aligned}$$

The expected norm of W and W^* can be obtained by Lemma 3,

$$\begin{aligned}&\mathbb{E}\|W - W^*\|_2\\ \leq& C_e[(D_w \cdot 4C_w^2)^{\frac{1}{2}} + (D_w \cdot 4C_w^2)^{\frac{1}{2}} + (D_w^2 \cdot 16C_w^4)^{\frac{1}{4}}]\\ =& 6C_e C_w \sqrt{D_w}.\end{aligned}$$

By Markov's inequality, for any $\mu > 0$,

$$\mathbb{P}\{\|W - W^*\|_2 \geq \mu\} \leq \frac{\mathbb{E}\|W - W^*\|_2}{\mu} \leq \frac{6C_e C_w \sqrt{D_w}}{\mu}.$$

□

4.2.2. Properties for Convolutional Networks. This section first illustrates the form conversion from a convolutional layer to a fully-connected layer and then lists the properties of convolutional networks by considering the filter weight matrices.

Conversion from convolutional layer to fully-connected layer. A convolutional layer can be converted to a fully-connected layer based on the fact that they both apply linear transformations [34], [46]–[48]. Considering that the input for a convolutional layer is a multi-channel image, we will use a three-dimensional tensor representing channel, width, and height as the input in subsequent sections.

For simplicity, we let the input tensor X and the output tensor Y of the m-th convolutional layer differ in their channel dimension but share the same width and height dimensions. We then define the matrix $W^{*,(m)}$ transformed from

the convolutional m-th layer with filters $F_{s,t}^{*,(m)} \in \mathbb{R}^{q\times q}$, which is for the s-th output channel and the t-th input channel. Following [34], [48], this matrix is formalized as $W^{*,(m)} \in \mathbb{R}^{p^2c'\times p^2c}$ to facilitate the equation $\mathrm{vec}(Y) = W^{*,(m)}\mathrm{vec}(X)$, with $\mathrm{vec}(\cdot)$ representing the vectorization function. Specifically,

$$W^* = \begin{bmatrix} B_{1,1} & \cdots & B_{1,c} \\ \vdots & \ddots & \vdots \\ B_{c',1} & \cdots & B_{c',c} \end{bmatrix},$$

where $B_{s,t}$ is a double block circulant matrix as follows

$$\begin{bmatrix} \mathrm{circ}(K_{s,t,1,:}) & \mathrm{circ}(K_{s,t,2,:}) & \cdots & \mathrm{circ}(K_{s,t,p,:}) \\ \mathrm{circ}(K_{s,t,p,:}) & \mathrm{circ}(K_{s,t,1,:}) & \cdots & \mathrm{circ}(K_{s,t,p-1,:}) \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{circ}(K_{s,t,2,:}) & \mathrm{circ}(K_{s,t,3,:}) & \cdots & \mathrm{circ}(K_{s,t,1,:}) \end{bmatrix},$$

$$K_{s,t} = \begin{bmatrix} F_{s,t}^* & \mathbf{0}_{q\times(p-q)} \\ \mathbf{0}_{(p-q)\times q} & \mathbf{0}_{(p-q)\times(p-q)} \end{bmatrix},$$

and $\mathrm{circ}(\cdot)$ is the circular matrix of a vector (Definition 2).

We describe the properties of convolutional networks below, which pertain to the size and number of filters. Owing to a proof scheme similar to that for fully-connected networks, we have deferred the corresponding proofs to Appendix A.2.

Property 4 (Bounding Weight Matrix)**.** *Given a convolutional neural network, its weight matrix $W^{*,(m)}$ satisfies, for any constant $\tau > 0$,*

$$\|W^{*,(m)}\|_2 \le C^2\lambda,$$

with a probability of at least $(1-2e^{-\tau^2})^{C^2}$, where $\lambda = (\sqrt{Q}+C_sK_s^2(\sqrt{Q}+\tau))$ and C is the maximum number of channels in each convolutional layer. Q is the maximum size of all filters. C_s is a universal constant and K_s depends on kernel weight matrices of filters.

Property 5 (Bounding Output)**.** *Given a convolutional neural network, the output $\boldsymbol{y}^{*,(m)}$ satisfies, for any constant $\tau > 0$*

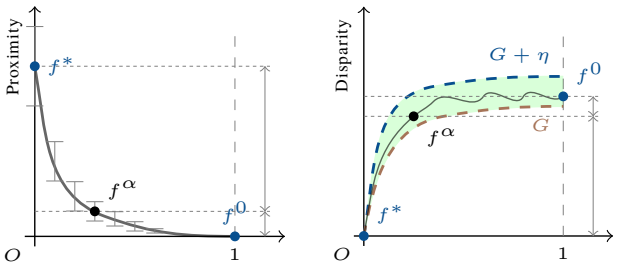
$$\|\boldsymbol{y}^{*,(m)}\|_2 \le (LC^2\lambda)^m\|\boldsymbol{x}\|_2,$$

with a probability of at least $(1-2e^{-\tau^2})^{C^2m}$, where $\lambda = \sqrt{Q}+C_sK_s^2(\sqrt{Q}+\tau)$ and Q is the maximum size $q\times q$ of all filters. C and C_s are universal constants and K_s depends on kernel weight matrices of filters.

Property 6 (Bounding Difference of Weight Matrices)**.** *Give a convolutional network with its post-extraction network, $\|W^{(m)} - W^{*,(m)}\|_2$ satisfies, for any constant $\mu > 0$,*

$$\|W^{(m)} - W^{*,(m)}\|_2 \ge \mu,$$

with a probability of at most $\frac{6Q^2C_eC_w\sqrt{D_w}}{\mu}$ where D_w is the maximum number of extracted filters in each layer and Q is the maximum size of all filters. C_e and C_w are universal constants.



(a) Bounded Output Variance (b) Bounded Output Disparity

Figure 4: The output variance in a neural network is bounded by the variance of its weights and biases. The range of the network's output disparity post-extraction $\|\boldsymbol{y}-\boldsymbol{y}^*\|_2$ is bounded by G and $G+\eta$, where $\eta = \mu L^m\lambda^{m-1}\|\boldsymbol{x}\|_2$.

4.3. Performance Dynamics

We start with exploring the performance dynamics of f^α as the retention ratio decreases, specifically focusing on the variance $\mathbb{V}$ of the network outputs. The following theorem hints that the extraction of weights with larger absolute values causes f^α to approach f^0 with only a minimal extraction ratio $1-\alpha$.

Theorem 1 (Bounding Output Variance)**.** *Given a fully connected neural network with an input $\boldsymbol{x}$,*

$$\mathbb{V}(\boldsymbol{y}^{*,(M)}) \le \|\boldsymbol{x}\|_2^2(L^2N)^m\prod_{m=1}^{M}(\sigma_w^{(m)})^2 + L^2N(\sigma_b^{(M)})^2 + L^2\sum_{m=1}^{M-1}\{N(\sigma_b^{(m)})^2\prod_{l=m+1}^{M}[L^2N(\sigma_w^{(l)})^2]\},$$

where $\sigma_w^{(m)}$ and $\sigma_b^{(m)}$ are the standard deviations of weights and biases in the m-th layer.

Proof. For the 1-st hidden layer, we have $W_i^{*,(1)}\boldsymbol{x}+b_i^{(1)}$ of linear transformation for one neuron. Given that all weights and biases are i.i.d,

$$\begin{aligned}\mathbb{V}(W_i^{*,(1)}\boldsymbol{x}+b_i^{(1)}) =& \mathbb{V}(\sum_{j=1}^{N^{(1)}}W_{i,j}^{*,(1)}x_j + b_i^{(1)}) \\ =& \mathbb{V}(\sum_{j=1}^{N^{(1)}}W_{i,j}^{*,(1)}x_j) + \mathbb{V}(b_i^{(1)}) \\ =& \sum_{j=1}^{N^{(1)}}x_j^2\mathbb{V}(W_{i,j}^{*,(1)}) + \mathbb{V}(b_i^{(1)}) \\ =& (\sigma_w^{(1)})^2\sum_{j=1}^{N^{(1)}}x_j^2 + (\sigma_b^{(1)})^2 \\ =& (\sigma_w^{(1)})^2\|\boldsymbol{x}\|_2^2 + (\sigma_b^{(1)})^2.\end{aligned}$$

By the Lipschitz property of the activation function, we have

$$y_i^{*,(1)} = \phi(W_i^{*,(1)}\boldsymbol{x}+b_i^{(1)}) \le L(W_i^{*,(1)}\boldsymbol{x}+b_i^{(1)}),$$

and then,

$$\begin{aligned}\mathbb{V}(y_i^{*,(1)}) \leq & \mathbb{V}[L(W_i^{*,(1)}\boldsymbol{x} + b_i^{(1)})] \\ \leq & L^2\mathbb{V}(W_i^{*,(1)}\boldsymbol{x} + b_i^{(1)}) \\ \leq & L^2[(\sigma_w^{(1)})^2\|\boldsymbol{x}\|_2^2 + (\sigma_b^{(1)})^2].\end{aligned}$$

Hence, we have

$$\begin{aligned}\mathbb{V}(\boldsymbol{y}^{*,(1)}) = & \sum_{j=1}^{N^{(1)}} \mathbb{V}(y_i^{*,(1)}) \\ \leq & L^2 N^{(1)}[(\sigma_w^{(1)})^2\|\boldsymbol{x}\|_2^2 + (\sigma_b^{(1)})^2].\end{aligned}$$

Similar for other layers,

$$\mathbb{V}(\boldsymbol{y}^{*,(m)}) \leq L^2 N^{(m-1)}[(\sigma_w^{(m)})^2\|\boldsymbol{y}^{(m-1)}\|^2 + (\sigma_b^{(m)})^2],$$

and it concludes the theorem by iterative applying the inequalities. □

Remark: *Theorem 1 establishes that the upper bound of the output variance in a neural network is determined by the variances of its weights and biases. It suggests a direct relationship where zeroing out larger-magnitude weights contributes to a significant reduction in the dynamic range of the output (smaller output variance), since altering larger-magnitude weights directly affects the variance of the weights and, consequently, the variance of outputs from each layer. Additionally, the function f^α approaches f^0 following the extraction of larger-magnitude weights. This occurs because, after the extraction of those weights, the variance of the outputs primarily depends on the variance of the biases, and the outputs tend to converge towards the biases of the last layer, as described by $\boldsymbol{y} = W^{(M)}\boldsymbol{y}^{(M-1)} + \boldsymbol{b}^{(M)}$.*

4.4. Theoretical Disparity Bounds

The range of the network's output disparity $\|\boldsymbol{y} - \boldsymbol{y}^*\|_2$ is estimated in this section. Importantly, we establish a lower bound, denoted as G, for $\|\boldsymbol{y} - \boldsymbol{y}^*\|_2$ to guarantee the effectiveness of our proposed methodology. This lower bound G is tailored to the specific network and its inputs. Additionally, we demonstrate that the upper bound for $\|\boldsymbol{y} - \boldsymbol{y}^*\|_2$ is high with a high probability, indicating that the disparity is significant.

Disparity quantity G. For the m-th layer, the disparity quantity G is defined as

$$G^{(m)} = \|\boldsymbol{y}^{(m)} - \phi(W^{(m)}\boldsymbol{y}^{*,(m-1)})\|_2,$$

where it serves as a lower bound of $\|\boldsymbol{y}^{(m)} - \boldsymbol{y}^{*,(m)}\|_2$. Specifically, $G^{(m)}$ represents the norm of the discrepancy between the m-th layer's output after extraction, and the output obtained by feeding the $(m-1)$-th layer output of the original network into the extracted layer. Both lower and upper bounds of output discrepancies for each layer m will be established based on this measure $G^{(m)}$.

We utilize inductive proof to prove the case of $\|\boldsymbol{y}^{(m)} - \boldsymbol{y}^{*,(m)}\|_2$, extending up to the final output layer. This inductive process allows us to systematically assess how alterations in each layer, due to weight extraction, propagate through the network and ultimately manifest in the output layer. In our proof, we omit the bias terms in this section as it can be regarded as a variable within the homogeneous function perspective.

4.4.1. Lower Bound of Disparity. We first establish a lower bound of $\|\boldsymbol{y}^{(m)} - \boldsymbol{y}^{*,(m)}\|_2$, which signifies a largely inevitable error margin between $\boldsymbol{y}^{(m)}$ and $\boldsymbol{y}^{*,(m)}$ across each layer. Although extracting those weights that are opposite numbers may lead to $\boldsymbol{y}^{(m)}$ and $\boldsymbol{y}^{*,(m)}$ being identical, such an occurrence is rare in practical applications. To further enhance the established disparity lower bound, we also provide the estimated lower bound of $G^{(m)}$ with high probability.

Theorem 2 (Generic Lower Bound)**.** *When extracting weights with the highest absolute values, we have*

$$\|\boldsymbol{y}^{(m)} - \boldsymbol{y}^{*,(m)}\|_2 \geq G^{(m)}, \tag{1}$$

where $G^{(m)} \geq \mu$ for any constant $\mu > 0$, with a probability at most $[N^{m-1}(D_w C_w)^m]/\mu$. D_w is the maximum number of weights extracted from each layer and C_w is a universal constant.

Proof. We first prove $\|\boldsymbol{y}^{(m)} - \boldsymbol{y}^{*,(m)}\|_2 \geq G^{(m)}$, and then provide a lower bound of $G^{(m)}$.

Considering the monotonicity of ϕ, we only need to prove each entry of the inputs of the ϕ satisfying $|W_i^{(m)}\boldsymbol{y}^{(m-1)} - W_i^{*,(m)}\boldsymbol{y}^{*,(m-1)}| \geq |W_i^{(m)}\boldsymbol{y}^{(m-1)} - W_i^{(m)}\boldsymbol{y}^{*,(m-1)}|$ for the i-th neuron[3]. By extracting from the highest positive weights of the m-th layer, we have $\boldsymbol{y}^{(m-1)} \leq \boldsymbol{y}^{*,(m-1)}$ and $W_i^{(m)}\boldsymbol{y}^{(m-1)} \leq W_i^{(m)}\boldsymbol{y}^{*,(m-1)}$[4]. Then, $W_i^{(m)}\boldsymbol{y}^{(m-1)} \leq W_i^{(m)}\boldsymbol{y}^{*,(m-1)} \leq W_i^{*,(m)}\boldsymbol{y}^{*,(m-1)}$ and it naturally leads to $\|\boldsymbol{y}^{(m)} - \boldsymbol{y}^{*,(m)}\|_2 \geq G^{(m)}$. Conversely, with extraction from the lowest negative weights, $\boldsymbol{y}^{(m-1)} \geq \boldsymbol{y}^{*,(m-1)}$ and $W_i^{(m)}\boldsymbol{y}^{(m-1)} \geq W_i^{(m)}\boldsymbol{y}^{*,(m-1)} \geq W_i^{*,(m)}\boldsymbol{y}^{*,(m-1)}$, upholding the above inequality.

Next, we explore the lower bound of $G^{(m)}$. Again, by considering the extraction from the highest positive weights, we establish that for $i \in [N^{(k)}]$, $\mathbb{E}|\phi(W^{(1)}\boldsymbol{x}) - \boldsymbol{y}^{(1)}| \leq D_w C_w$ and then

$$\begin{aligned}\mathbb{E}|\phi(W^{(2)}\boldsymbol{y}^{*,(1)}) - \boldsymbol{y}^{(2)}| \leq & D_w C_w \cdot \mathbb{E}|\phi(W^{(1)}\boldsymbol{x}) - \boldsymbol{y}^{(1)}| \\ \leq & N(D_w C_w)^2.\end{aligned}$$

Similarly, for the m-th layer,

$$\mathbb{E}|\phi(W^{(m)}\boldsymbol{y}^{*,(m-1)}) - \boldsymbol{y}^{(m)}| \leq N^{m-1}(D_w C_w)^m,$$

where C_w is an upper bound of the extracted weight or the sum of weights of each extracted filter. By Markov inequality, we have $G^{(m)} \geq \mu$ for any constant $\mu > 0$, with a probability at most $[N^{m-1}(D_w C_w)^m]/\mu$. The case of extracting from the lowest negative weights is similar. □

3. The inequality of two vectors means each pair of their elements satisfies this inequality.

4. This holds when $\boldsymbol{y}^{(m-1)} \geq \boldsymbol{y}^{*,(m-1)} \geq 0$, satisfied by modifying the biases for a positive output in an equivalent activation function.

Theorem 2 applies to both fully-connected and convolutional neural networks. In convolutional networks, our extraction targets the filter as a whole. Extracting a filter with either the maximum or minimum sum of weights effectively mirrors the process of extracting multiple positive or negative weights in W^*, as the sum of these weights is either predominantly positive or negative.

4.4.2. Upper Bound of Disparity. We start our analysis on fully-connected networks. Theorem 3 below establishes the upper bound of $\|\boldsymbol{y}^{(m+1)} - \boldsymbol{y}^{*,(m+1)}\|_2$.

Theorem 3 (Upper Bound for Fully-connected Network)**.** *When extracting weights with the highest absolute values, for any constants $\tau > 0$ and $\mu > 0$,*

$$\|\boldsymbol{y}^{(m)} - \boldsymbol{y}^{*,(m)}\|_2 \leq G^{(m)} + \mu L^m \lambda^{m-1} \|\boldsymbol{x}\|_2, \tag{2}$$

with a probability of at least $(1 - \frac{6C_eC_w\sqrt{D_w}}{\mu})(1 - 2e^{-\tau^2})^{m-1}$, where $\lambda = \sqrt{N} + C_s K^2(\sqrt{N} + \tau)$. D_w is the maximum number of extracted weights in each layer and C_e and C_w are universal constants.

Proof. First,

$$\begin{aligned}
&\|\boldsymbol{y}^{(m)} - \boldsymbol{y}^{*,(m)}\|_2 \\
=&\|\boldsymbol{y}^{(m)} - \phi(W^{(m)}\boldsymbol{y}^{*,(m-1)}) \\
&+ \phi(W^{(m)}\boldsymbol{y}^{*,(m-1)}) - \boldsymbol{y}^{*,(m)}\|_2 \\
\leq&\|\boldsymbol{y}^{(m)} - \phi(W^{(m)}\boldsymbol{y}^{*,(m-1)})\|_2 \\
&+ \|\phi(W^{(m)}\boldsymbol{y}^{*,(m-1)}) - \boldsymbol{y}^{*,(m)}\|_2 \\
=&G^{(m)} + \|\phi(W^{(m)}\boldsymbol{y}^{*,(m-1)}) - \boldsymbol{y}^{*,(m)}\|_2.
\end{aligned}$$

Considering the activation function ϕ's Lipschitz property,

$$\begin{aligned}
&\|\phi(W^{(m)}\boldsymbol{y}^{*,(m-1)}) - \boldsymbol{y}^{*,(m)}\|_2 \\
=&\|\phi(W^{(m)}\boldsymbol{y}^{*,(m-1)}) - \phi(W^{*,(m)}\boldsymbol{y}^{*,(m-1)})\|_2 \\
\leq&L\|W^{(m)}\boldsymbol{y}^{*,(m-1)} - W^{*,(m)}\boldsymbol{y}^{*,(m-1)}\|_2 \\
=&L\|(W^{(m)} - W^{*,(m)})\boldsymbol{y}^{*,(m-1)}\|_2 \\
\leq&L\|W^{(m)} - W^{*,(m)}\|_2\|\boldsymbol{y}^{*,(m-1)}\|_2.
\end{aligned}$$

By the upper bounds derived in Property 2 and 3,

$$\begin{aligned}
&L\|W^{(m)} - W^{*,(m)}\|_2\|\boldsymbol{y}^{*,(m-1)}\|_2 \\
\leq&L\mu(L\lambda)^{m-1}\|\boldsymbol{x}\|_2 \\
=&\mu L^m\lambda^{m-1}\|\boldsymbol{x}\|_2,
\end{aligned}$$

where $\lambda = \sqrt{N} + C_s K^2(\sqrt{N} + \tau)$.

Last, for calculating the probability that this inequality holds with Property 2 and 3,

$$\begin{aligned}
&\mathbb{P}\{\|\boldsymbol{y}^{(m)} - \boldsymbol{y}^{*,(m)}\|_2 \leq G^{(m)} + \eta^{(m)}\} \\
=&\mathbb{P}\{\|W^{(m)} - W^{*,(m)}\| \leq \mu\} \\
&\cdot \mathbb{P}\{\|\boldsymbol{y}^{*,(m-1)}\| \leq (L\lambda)^{m-1}\|\boldsymbol{x}\|_2\} \\
\geq&(1 - \frac{6C_eC_w\sqrt{D_w}}{\mu})(1 - 2e^{-\tau^2})^{m-1}.
\end{aligned}$$

□

Convolutional neural networks. For convolutional neural networks (CNNs), a different upper bound is established by factoring in the convolutional filters. This distinction arises since the CORELOCKER extracts entire filters in CNNs rather than individual weights. Note that in such case, the weight matrix W^* is composed of double block circulant matrices $B_{s,t}$, as detailed in Section 4.2.2.

Theorem 4 (Upper Bound for Convolutional Network)**.** *When extracting weights with the highest absolute values, for any constants $\tau > 0$ and $\mu > 0$,*

$$\|\boldsymbol{y}^{(m)} - \boldsymbol{y}^{*,(m)}\|_2 \leq G^{(m)} + \mu L^m \lambda^{m-1} \|\boldsymbol{x}\|_2, \tag{3}$$

with a probability of at least $(1 - \frac{6Q^2C_eC_w\sqrt{D_w}}{\mu})(1 - 2e^{-\tau^2})^{C^2(m-1)}$, where $\lambda = C^2[\sqrt{Q} + C_s K_s^2(\sqrt{Q} + \tau)]$. D_w is the maximum number of extracted filters in each layer, C is the maximum number of filters, and C_e and C_w are universal constants.

Proof. Similar to Theorem 3,

$$\begin{aligned}
&\|\boldsymbol{y}^{(m)} - \boldsymbol{y}^{*,(m)}\|_2 \\
\leq&G^{(m)} + L\|W^{(m)} - W^{*,(m)}\|_2\|\boldsymbol{y}^{*,(m-1)}\|_2.
\end{aligned}$$

By Properties 5 and 6,

$$\begin{aligned}
&L\|W^{(m)} - W^{*,(m)}\|_2\|\boldsymbol{y}^{*,(m-1)}\|_2 \\
\leq&L\mu(L\lambda)^{m-1}\|\boldsymbol{x}\|_2 \\
=&\mu L^m\lambda^{m-1}\|\boldsymbol{x}\|_2,
\end{aligned}$$

where $\lambda = C^2[\sqrt{Q} + C_s K_s^2(\sqrt{Q} + \tau)]$.

Last, for calculating the probability that this inequality holds with Properties 5 and 6,

$$\begin{aligned}
&\mathbb{P}\{\|\boldsymbol{y}^{(m)} - \boldsymbol{y}^{*,(m)}\|_2 \leq G^{(m)} + \eta^{(m)}\} \\
=&\mathbb{P}\{\|W^{(m)} - W^{*,(m)}\| \geq \mu\} \\
&\cdot \mathbb{P}\{\|\boldsymbol{y}^{*,(m-1)}\| \leq (L\lambda)^{m-1}\|\boldsymbol{x}\|_2\} \\
\geq&(1 - \frac{6Q^2C_eC_w\sqrt{D_w}}{\mu})(1 - 2e^{-\tau^2})^{C^2(m-1)}.
\end{aligned}$$

□

We contend that there is mutual reinforcement among the established theorems. Specifically, the established upper bounds do not scale indefinitely, in alignment with Theorem 1 where f^α tends towards f^0 as the extraction ratio increases. This is linked to the fact that $\sqrt{D_w}$, which is solely related to the extraction process, increases as α decreases. However, this increase progresses at a progressively slower pace, consistent with the decrease in the output's variance.

Remark: *Theorems 2 to 4 establish both lower and upper bounds of the disparity among f^*, f^α, and f^0. The formal analysis demonstrates that as the extraction ratio increases, the disparity also increases with a high probability. This finding is further corroborated by empirical results, as elaborated soon in Section 5. Overall, Section 4 establishes a strong formal foundation for CORELOCKER's neuron-level usage control, ensuring that CORELOCKER's strategy offers strong guarantees.*

5. Experimental Evaluation

This section validates CORELOCKER with qualitative analysis (Section 5.1) and practicality evaluation (Section 5.2). The qualitative analysis aims to substantiate the alignment between theoretical analysis and practicality evaluation in order to examine the effects of various factors on a neural network, such as network depth (number of hidden layers) and width (number of neurons in each hidden layer). The practicality evaluation presents the results of applying CORELOCKER on commonly used large networks, *e.g.,* ResNet-164 [37].

5.1. Qualitative Analysis

In this section, we conduct a qualitative analysis of the CORELOCKER mechanism, aiming to complement our theoretical analysis in Section 4.4 with real-world demonstrations. Specifically, we compare model performance after weight extraction with multiple models of different widths and depths.

5.1.1. Experimental Setup. We conduct the analysis on fully-connected neural networks (FCNs). Specifically, a total of 80 FCNs are constructed with different structures, *i.e.,* the number of hidden layers is set from 4 to 13 (with a step of 1 layer), and the number of neurons in each layer is set from 1,000 to 8,000 (with a step of 1,000 neurons). The FCNs are trained on the MNIST dataset [49] using Adam optimizer [50] with a learning rate of 0.001, and a batch size of 128. Each network has 784 input neurons and 10 output neurons. After 50 epochs of training, these FCNs achieve 98.6% test accuracy on average. The weights are then extracted based on the ℓ_1-norm indicator with a fixed global extraction ratio of 0.002%.

5.1.2. Results. Figure 5 illustrates the analysis results of the accuracy dropping across networks with varying depth and width settings. Networks with larger width and depth are more susceptible to utility loss upon extraction. For example, after weight extraction, the performance of most networks with more than 6 hidden layers and more than 3,000 neurons in each layer decreases by more than 50%, and networks with more than 9 hidden layers and more than 5,000 neurons in each layer demonstrate very low accuracy, where most of them drop to a mere 10%.

The analysis results further demonstrate that the model performance after weight extraction approaches $1/N^{(M)}$ (*e.g.,* 10% when N=4,000, M=12), which is consistent with Theorem 1. This indicates that the performance of f^{α} approaches f^{0} even when small amounts of weights are extracted.

It is also notable that the depth of the model has a more pronounced impact on the decrease in accuracy compared to its width, highlighted by the fact that introducing an additional layer yields a similar effect to the expansion of width by 1,000 units. Such analysis result is consistent with our analysis in Theorem 2 and Theorem 3, where the depth has an exponential effect on the disparity bounds. Additionally, such results empirically validate the insights of our theorems. This empirical validation offers a novel viewpoint for analyzing the behavior of trained networks and lays the groundwork for CORELOCKER's neuron-level usage control.

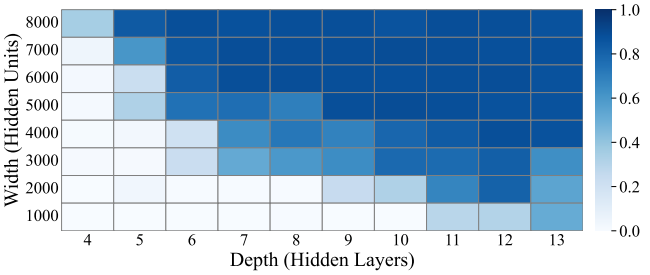


Figure 5: Accuracy dropping across networks with varying depth and width settings. Networks with greater width and depth are more susceptible to utility loss upon extraction.

5.2. Practicality Evaluation

To establish that CORELOCKER is practical to protect real-world models, we apply it to prevalent DNN architectures. We first demonstrate that CORELOCKER is capable of degrading the model performance with a low extraction ratio, then analyze its potential capability to enable fine-grained customization of access keys.

5.2.1. Experimental Setup. We consider several prevalent DNN architectures, including VGG-19 [51], DenseNet-40 [52], and ResNet-164 [37] trained on three different datasets. All networks are trained with a batch size of 256 for 120 epochs. Given that CORELOCKER is the first work in its domain, we propose a random weight extraction method as the baseline to demonstrate the effectiveness of our model. For a fair comparison, we use the same extraction ratio in CORELOCKER and baseline.

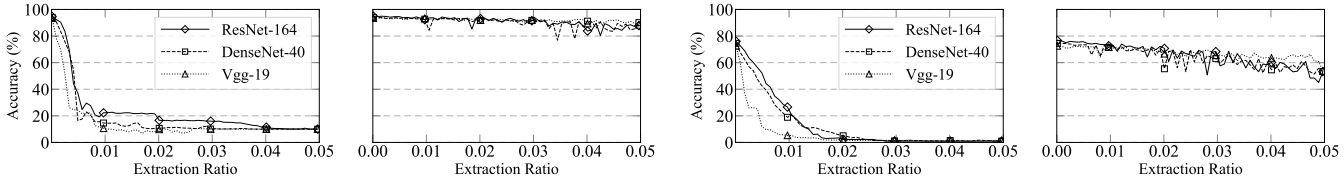
Our evaluation is conducted with three commonly used datasets: Fashion-MNIST [53], CIFAR-10, and CIFAR-100 [54]. Specifically, the Fashion-MNIST dataset comprises 60,000 training images and 10,000 test images. These images are 28×28 grayscale representations, distributed across 10 classes. CIFAR-10/100 both have 50,000 training images and 10,000 test images of 32 $\times$ 32, except that CIFAR-10 includes 10 classes while CIFAR-100 has 100 classes.

5.2.2. Results. We report the top-1 test accuracy of original models, random extraction protected models, and CORELOCKER protected models with different extraction indicators when the global extraction ratio is set as 0.05 in Table 1. From the experimental results, CORELOCKER successfully degrades all target models' inference accuracy to a level of random guessing (*i.e.,* $1/N^{(M)}$). Specifically, for Fashion-MNIST and CIFAR-10 datasets, the top-1 accuracy of all target models post-extraction declines significantly to 10%;

TABLE 1: Performance on different models and datasets with an extraction ratio of 0.05.

	Fashion-MNIST			CIFAR-10			CIFAR-100		
	Dense-40	ResNet-164	Vgg-19	Dense-40	ResNet-164	Vgg-19	Dense-40	ResNet-164	Vgg-19
Original	94.62%	94.88%	93.50%	94.04%	94.71%	93.52%	74.53%	76.02%	72.60%
$Random_{avg}$	91.17%	88.24%	82.30%	87.41%	85.23%	88.31%	57.52%	62.79%	61.13%
CORELOCKER[1]	10.01%	10.00%	9.72%	10.00%	10.01%	10.11%	1.04%	1.01%	1.02%
CORELOCKER[2]	10.00%	10.01%	10.00%	10.03%	10.00%	10.00%	1.00%	1.02%	1.00%

[1] CORELOCKER with the ℓ_1-norm as the extraction indicator.
[2] CORELOCKER with the scale factors as the extraction indicator.



(a) CORELOCKER (left) versus random extraction (right).

(b) CORELOCKER (left) versus random extraction (right).

Figure 6: Top-1 accuracy on different extraction ratios across three neural networks on CIFAR-10 (a) and CIFAR-100 (b).

while for CIFAR-100 which includes 100 classes, the target models post-extraction reaches a mere 1% top-1 accuracy. In contrast, random extraction only causes a limited accuracy drop. Specifically, for the DenseNet-40 and ResNet-164 models trained on the Fashion-MNIST dataset, the observed average accuracy reductions are 3.45% and 6.64% respectively, taken from five trials. This analysis confirms that CORELOCKER is capable of significant performance degradation even with quite a low global extraction ratio, *e.g.,* 0.05, which indicates that CORELOCKER can provide model usage control through neuron-level access key extraction. We also provide the top-3 accuracy in Figure 7, all target models post-extraction decline significantly to $3/N^{(M)}$, *i.e.,* all models reached an accuracy of 30.00% and 3.00% respectively on the CIFAR-10/100 datasets. further confirming the efficacy of the proposed approach.

To further examine whether CORELOCKER can provide a fine-grained model utility protection through customized access key extraction, we conduct more analysis with different global extraction ratios ranging from 0.0001 to 0.05. In this experiment, we adopt the scale factors as the extraction indicated and compare the top-1 accuracy of CORELOCKER and random extraction, as demonstrated in Figure 6. As shown in Figure 6a, on the CIFAR-10 dataset, three networks, *i.e.,* DenseNet-40, ResNet-164, and VGG-19, exhibit a consistent decrease in model accuracy with respect to increasing weight extraction ratios (from 0.0001 to 0.05). Similar results are exhibited on CIFAR-100 dataset (as shown in Figure 6b). Such a consistent decrease in model accuracy is crucial to our model access control solution, potentially enabling customized access key generation with respect to specific utility expectations in demo models that are publicly accessible to unauthorized users.

Overall, our experimental results confirm that model owners can regulate the model's utility level by adjusting the key extraction volume. For example, given a desired utility (accuracy) level of 50 – 55%, a smaller extraction ratio of 0.0050 could be performed on the ResNet model. We postpone the detailed statistics on CORELOCKER's capability for granular utility control to Table 5 in Appendix B.1. To ensure the practicality of CORELOCKER, we also apply it across other widely recognized architectures such as transformers [55] and recurrent neural networks [56]. The experimental results outlined in Tables 6 and 7 demonstrate that our approach is broadly applicable, grounded in the essential property of impact concentration within neural networks.

> **Remark:** *Our results reaffirm that* CORELOCKER *can degrade a model's utility with fine-grained customization concerning specific utility requirements for demo models. In practice, the model owner can establish a mapping between the extraction ratio and model utility based on the theoretical bounds provided in Section 4. Specifically, for a given model and desired utility level, variables like neuron count (N) and layer number (M) are directly available as the model's inherent attributes. The remaining universal constants within these bounds can be estimated using specific results from a few random extraction ratio experiments, thus enabling a direct mapping.*

5.3. Resilience against Attacks

In addition to its effectiveness in key extraction, CORELOCKER is also desired to be resilient against potential attacks. This includes scenarios where an adversary might identify and neutralize the weights CORELOCKER has altered, or restore the model's original accuracy through additional attacks, such as model fine-tuning with limited data. In

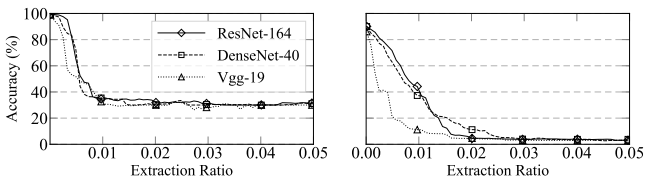


Figure 7: Performance (top-3 accuracy) of CORELOCKER on CIFAR-10 (left) and CIFAR-100 (right) datasets.

TABLE 2: Accuracy recovered by fine-tuning attack with different fine-tuning data ratios.

Dataset	Model	Fine-tuning Data	
		5%	10%
CIFAR-10	DenseNet-40	▲17.23 ± 4.26	▲23.90 ± 6.09
	ResNet-164	▲15.98 ± 8.47	▲25.09 ± 4.11
	Vgg-19	▲15.08 ± 9.21	▲29.52 ± 9.05

such cases, the attacker could repurpose the recovered model for their desired uses. This section assesses the resilience of CORELOCKER, emphasizing its effectiveness in thwarting those advanced attacks, including fine-tuning attacks [26] and model pruning attacks [25].

Against fine-tuning. As large models are trained on large-scale datasets collected from different sources to achieve satisfactory performance, the attacker might collect datasets from the internet that are part of the actual training data of the protected model. Such a fact raises the probability that the attacker may adopt a fine-tuning attack against the protected model to reconstruct the extracted weights [26].

To evaluate the resilience of CORELOCKER against fine-tuning attacks, we consider different sizes of datasets available to the attackers, specifically 5% and 10% of the training data utilized in target models for simplicity. We assessed the resilience of all three models trained on the CIFAR-10 dataset. Those models are protected by CORELOCKER with the scale factor as the extraction indicator, with a 0.05 extraction ratio (exhibits an average accuracy of 10.01% post-protection). The accuracy recovered (denoted as ▲) by the attacker is demonstrated in Table 2. The table establishes a general trend: the accuracy recovery tends to increase when a larger portion of the dataset is employed for fine-tuning. These results are based on three trials, wherein data is randomly resampled for each trial. Due to this random selection process, and the fact that certain data may have a more substantial effect on the fine-tuning of the model than others, the resulting variances can be explained. Overall, our method demonstrates sustained resilience even when subjected to fine-tuning with 10% of the training data, a scenario that is often considered unrealistic in most practical situations.

Model pruning attack. The model pruning attack is an advanced attack known as the most effective attack against encryption-based or noise-based model protection methods [21], [25]. It operates by pruning the parameters that are encrypted or corrupted to restore the model performance, assuming that these protection methods randomly encrypt or corrupt some of the parameters to degrade the model performance. We follow the same attack setup as [21], and test the resilience of CORELOCKER using up to 40% pruning ratio. As shown in Table 3, the evaluated models are trained on the Fashion-MNIST dataset, protected by CORELOCKER with the scale factor as the extraction indicator, with a 0.05 extraction ratio (models exhibit an average of 10.00% post-protection). Clearly, even when we adopt a 40% pruning ratio, our methodology exhibits a completely robust defense against such attacks, *i.e.,* less than 1% accuracy recovery on average. This further elevates the significance of our methodology, providing the insight that our strategy does not merely change the model "superficially" but instead enacts a fundamental disruption. The reason for this is the important parameters have been identified and have already been excluded by our algorithm, and such an attack will not be effective to ours. Such resilience is not just an incremental improvement but a paradigm shift in ensuring model utility protection.

TABLE 3: Accuracy recovered by pruning attack with different pruning ratios.

Dataset	Model	Pruning Ratio	
		20%	40%
Fashion-MNIST	DenseNet-40	▲0.00 ± 0.34	▲1.04 ± 0.81
	ResNet-164	▲0.00 ± 0.20	▲0.52 ± 0.82
	Vgg-19	▲0.00 ± 0.07	▲0.00 ± 0.42

6. Related Work

Model IP protection. Model IP protection aims to protect the ownership of intellectual property of the model from being abused. As discussed earlier, existing passive model protection approaches like watermarking [12], [15], [57] may have limitations in preventing unauthorized usage after the model's exposure. In response, proactive protection strategies such as model encryption have been introduced. They typically necessitate decoding the model at runtime for inference, which may still leave the decoded model susceptible to attacks [58]. Specifically, Chakraborty *et al.* [38] leverage secure hardware support to propose a key-dependent back-propagation algorithm for training a DNN with obscured weight space. This obfuscation ensures that only authorized users with access to trusted hardware and an embedded key can use the model effectively; unauthorized extraction and deployment of the model by attackers, particularly on different devices, lead to a significant decline in model accuracy. Besides the reliance of hardware modifications, this technique may fall short in universal applicability to pre-trained models. Similarly, Fan *et al.* [59] propose a method to protect the model IP by integrating a passport layer into the deep neural network, specifically designed to counter ambiguity attacks. Other approaches integrate a secret key into the training data during prepossessing and train models to operate only with key-preprocessed inputs [19], [20].

TABLE 4: Configuration comparison of existing methods.

	Data Access	Additional Training	Hardware Support
Chakraborty *et al.* [38]	●	●	●
Chen *et al.* [20]	●	●	○
Fan *et al.* [59]	●	●	○
Pyone *et al.* [19]	●	●	○
Xue *et al.* [21]	●	○	○
Zhou *et al.* [22]	●	○	○
CORELOCKER (Ours)	○	○	○

* ● (or ○) refers to require (or do not require) the condition.

Recent research seeks to mitigate unauthorized access by integrating adversarial perturbations [21], [22]. The challenge of deploying such approaches lies in the restricted availability of secure memory, necessitating that only a limited number of modifications can be made. As such, existing approaches either utilize the original testing data to create perturbations [21], or set perturbations as an optimization objective using reinforcement learning [22].

As a summary, Table 4 lists the comparison between the configurations of our method to those of existing model IP protection strategies, in terms of data access, additional training, and hardware support. CORELOCKER sets itself apart from other methods that typically require original data for encryption or adversarial perturbation generation [19]–[22], [38], [59], as well as those requiring additional training [19], [20] or hardware support [38]. It is also worth highlighting the efficiency of CORELOCKER in dynamic scenarios where the access key needs to be regenerated when the model undergoes significant updates. Unlike existing approaches that typically rely on expensive retraining or fine-tuning, CORELOCKER's lightweight and data-free nature minimizes the overhead for regeneration.

Validation of lottery ticket hypothesis. We draw insight from the proof frameworks that establish the existence of "lottery tickets" to obtain the essential properties of neural networks. The Lottery Ticket Hypothesis posits the intriguing presence of "winning ticket" sub-networks within a randomly initialized network, which – when trained in isolation – can reach or exceed the original network's test accuracy. This hypothesis was first proposed in [60] and has since attracted great interest. Subsequent research endeavors, such as those by Zhou *et al.* [61] and Ramanujan *et al.* [62], have theoretically corroborated the existence of these sub-networks, demonstrating their potential to perform well without the conventional training of weights. More recent findings by Zhang *et al.* [63] suggest that sub-networks capable of high performance can be identified within pre-trained models.

Neural network pruning. Neural network pruning, a widely utilized technique for model compression, enables the deployment of models on devices with constrained resources. Over time, numerous pruning methods have been proposed, demonstrating that it's feasible to decrease the parameter count in neural network models by as much as 90% while incurring only a minimal loss in performance [33], [35], [36]. Works by LeCun *et al.* [33] and Hassibi *et al.* [35] studied the efficiency of network pruning based on second derivative conditions. Other lines of network pruning focus on the magnitude of the weights [40]. Other pruning techniques remove neurons with zero activation [64], or other measures of redundancy [65]. Recent methodologies in the field have been promoting the pruning of entire convolutional channels to achieve more significant performance enhancements [44], [66]–[68]. These pruning techniques provide further insight, that the performance of a neural network is largely reliant on a crucial subset of weights.

7. Conclusion

In this paper, CORELOCKER emerges as a pioneering solution for neuron-level model usage control in neural networks. It stands out for its practicality, being lightweight, data-agnostic, and retraining-free, and efficiently manages access key extraction. The method is underpinned by a solid theoretical analysis that introduces a formal framework and crucial boundaries for model usage control. Empirically, CORELOCKER demonstrates effectiveness in degrading model performance and exhibits robustness against advanced adversarial strategies like fine-tuning and pruning attacks. This research marks a significant advancement in neural network model protection and usage control.

Acknowledgments

We would like to thank our shepherd and the anonymous reviewers from IEEE S&P 2024 for their insightful comments. This work is partially supported by Australian Research Council Discovery Projects (DP230101196, DP240103068). Minhui Xue is supported by CSIRO – National Science Foundation (US) AI Research Collaboration Program.

References

[1] N. Tajbakhsh, J. Y. Shin, S. R. Gurudu, R. T. Hurst, C. B. Kendall, M. B. Gotway, and J. Liang, "Convolutional neural networks for medical image analysis: Full training or fine tuning?" *IEEE Transactions on Medical Imaging*, vol. 35, no. 5, pp. 1299–1312, 2016.

[2] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[3] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for modern deep learning research," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 34, no. 09, 2020, pp. 13 693–13 696.

[4] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," in *Advances in neural information processing systems (NeurIPS)*, 2020, pp. 1877–1901.

[5] L. Chuan, "OpenAI's GPT-3 language model: A technical overview," https://lambdalabs.com/blog/demystifying-gpt-3, 2023.

[6] C. Taylor, "Chatgpt creator openai earnings: $80 million a month, $1 billion annual revenue, $540 million loss: Sam altman," https://fortune.com/2023/08/30/chatgpt-creator-openai-earnings-80-million-a-month-1-billion-annual-revenue-540-million-loss-sam-altman/, 2023.

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

[8] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *International Conference on Learning Representations (ICLR)*, 2015.

[9] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial examples: Attacks and defenses for deep learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 9, pp. 2805–2824, 2019.

[10] R. Sun, M. Xue, G. Tyson, T. Dong, S. Li, S. Wang, H. Zhu, S. Camtepe, and S. Nepal, "Mate! Are you really aware? An explainability-guided testing framework for robustness of malware detectors," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2023.

[11] B. Darvish Rouhani, H. Chen, and F. Koushanfar, "Deepsigns: An end-to-end watermarking framework for ownership protection of deep neural networks," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2019, p. 485–497.

[12] Y. Uchida, Y. Nagai, S. Sakazawa, and S. Satoh, "Embedding watermarks into deep neural networks," in *Proceedings of ACM on International Conference on Multimedia Retrieval (ICMR)*, 2017, p. 269–277.

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

[14] Z. Wang, O. Byrnes, H. Wang, R. Sun, C. Ma, H. Chen, Q. Wu, and M. Xue, "Data hiding with deep learning: A survey unifying digital watermarking and steganography," *IEEE Transactions on Computational Social Systems*, 2023.

[15] A. B. Kahng, J. Lach, W. H. Mangione-Smith, S. Mantik, I. L. Markov, M. Potkonjak, P. Tucker, H. Wang, and G. Wolfe, "Watermarking techniques for intellectual property protection," in *Proceedings of the 35th Annual Design Automation Conference (DAC)*, 1998, pp. 776–781.

[16] S. Wang, S. Abuadbba, S. Agarwal, K. Moore, R. Sun, M. Xue, S. Nepal, S. Camtepe, and S. Kanhere, "PublicCheck: Public watermarking verification for deep neural networks," in *44th IEEE Symposium on Security and Privacy (IEEE S&P)*, 2023.

[17] H. Chen, B. D. Rouhani, and F. Koushanfar, "Blackmarks: Blackbox multibit watermarking for deep neural networks," *arXiv preprint arXiv:1904.00344*, 2019.

[18] Y. Lao, W. Zhao, P. Yang, and P. Li, "Deepauth: A DNN authentication framework by model-unique and fragile signature embedding," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2022, pp. 9595–9603.

[19] A. Pyone, M. Maung, and H. Kiya, "Training dnn model with secret key for model protection," in *2020 IEEE 9th Global Conference on Consumer Electronics (GCCE)*, 2020, pp. 818–821.

[20] M. Chen and M. Wu, "Protect your deep neural networks from piracy," in *2018 IEEE International Workshop on Information Forensics and Security (WIFS)*, 2018, pp. 1–7.

[21] M. Xue, Z. Wu, Y. Zhang, J. Wang, and W. Liu, "AdvParams: An active DNN intellectual property protection technique via adversarial perturbation based parameter encryption," *IEEE Transactions on Emerging Topics in Computing*, vol. 11, no. 3, pp. 664–678, 2023.

[22] T. Zhou, Y. Luo, S. Ren, and X. Xu, "Nnsplitter: An active defense solution for dnn model via automated weight obfuscation," in *Proceedings of the 40th International Conference on Machine Learning (ICML)*. JMLR, 2023.

[23] H. Lim, S.-D. Roh, S. Park, and K.-S. Chung, "Robustness-aware filter pruning for robust neural networks against adversarial attacks," in *2021 IEEE 31st International Workshop on Machine Learning for Signal Processing (MLSP)*, 2021, pp. 1–6.

[24] J. Chen, Y. Li, X. Wu, Y. Liang, and S. Jha, "Robust out-of-distribution detection for neural networks," in *The AAAI-22 Workshop on Adversarial Machine Learning and Beyond*, 2022.

[25] K. Liu, B. Dolan-Gavitt, and S. Garg, "Fine-pruning: Defending against backdooring attacks on deep neural networks," in *International Symposium on Research in Attacks, Intrusions, and Defenses (RAID)*. Springer, 2018, pp. 273–294.

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

[27] M. Ribeiro, K. Grolinger, and M. A. Capretz, "Mlaas: Machine learning as a service," in *IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2015, pp. 896–902.

[28] "cutout.pro," https://www.cutout.pro, 2018.

[29] "together.ai," https://www.together.ai, 2023.

[30] N. Lin, X. Chen, H. Lu, and X. Li, "Chaotic weights: A novel approach to protect intellectual property of deep neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 7, pp. 1327–1339, 2021.

[31] A. Jacot, F. Gabriel, and C. Hongler, "Neural tangent kernel: Convergence and generalization in neural networks," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[32] Y. Bai and J. D. Lee, "Beyond linearization: On quadratic and higher-order approximation of wide neural networks," in *International Conference on Learning Representations (ICLR)*, 2020.

[33] Y. LeCun, J. Denker, and S. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems (NeurIPS)*, 1989, pp. 598–605.

[34] X. Qian and D. Klabjan, "A probabilistic approach to neural network pruning," in *International Conference on Machine Learning (ICML)*. PMLR, 2021, pp. 8640–8649.

[35] B. Hassibi and D. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," in *Advances in Neural Information Processing Systems (NeurIPS)*, 1992, pp. 164–171.

[36] G. Thimm and E. Fiesler, "Evaluating pruning methods," in *International Symposium on Artificial Neural Networks (ISANN)*, 1995, pp. 20–25.

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

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

[39] Z. Sun, R. Sun, C. Liu, A. R. Chowdhury, L. Lu, and S. Jha, "Shadownet: A secure and efficient on-device model inference system for convolutional neural networks," in *2023 IEEE Symposium on Security and Privacy (IEEE S&P)*, 2023, pp. 1596–1612.

[40] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

[41] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710*, 2016.

[42] N. Bjorck, C. P. Gomes, B. Selman, and K. Q. Weinberger, "Understanding batch normalization," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[43] M. Lin, R. Ji, Y. Wang, Y. Zhang, B. Zhang, Y. Tian, and L. Shao, "Hrank: Filter pruning using high-rank feature map," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 1529–1538.

[44] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2017, pp. 2755–2763.

[45] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2015, pp. 1026–1034.

[46] P. Petersen and F. Voigtlaender, "Equivalence of approximation by convolutional neural networks and fully-connected networks," *Proceedings of the American Mathematical Society*, vol. 148, no. 4, pp. 1567–1581, 2020.

[47] W. Ma and J. Lu, "An equivalence of fully connected layer and convolutional layer," *arXiv preprint arXiv:1712.01252*, 2017.

[48] H. Sedghi, V. Gupta, and P. M. Long, "The singular values of convolutional layers," *arXiv preprint arXiv:1805.10408*, 2018.

[49] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[50] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, 2015.

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

[52] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 4700–4708.

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

[54] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[55] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017.

[56] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[57] J. Guo and M. Potkonjak, "Watermarking deep neural networks for embedded systems," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2018, pp. 1–8.

[58] H. Chabanne, J.-L. Danger, L. Guiga, and U. Kühne, "Side channel attacks for architecture extraction of neural networks," *CAAI Transactions on Intelligence Technology*, vol. 6, no. 1, pp. 3–16, 2021.

[59] L. Fan, K. W. Ng, and C. S. Chan, "Rethinking deep neural network ownership verification: Embedding passports to defeat ambiguity attacks," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[60] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *International Conference on Learning Representations (ICLR)*, 2019.

[61] H. Zhou, J. Lan, R. Liu, and J. Yosinski, "Deconstructing lottery tickets: Zeros, signs, and the supermask," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019, pp. 3597–3607.

[62] V. Ramanujan, M. Wortsman, A. Kembhavi, A. Farhadi, and M. Rastegari, "What's hidden in a randomly weighted neural network?" in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 11 893–11 902.

[63] Y. Zhang, M. Lin, Y. Zhong, F. Chao, and R. Ji, "Lottery jackpots exist in pre-trained models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

[64] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," *arXiv preprint arXiv:1607.03250*, 2016.

[65] S. Srinivas, A. Subramanya, and R. Venkatesh Babu, "Training sparse neural networks," in *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017, pp. 138–145.

[66] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 6071–6079.

[67] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," in *International Conference on Learning Representations (ICLR)*, 2017.

[68] J.-H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2017.

[69] R. Vershynin, *High-dimensional probability: An introduction with applications in data science*. Cambridge University Press, 2018, vol. 47.

[70] H. A. David and H. N. Nagaraja, *Order statistics*. John Wiley & Sons, 2004.

[71] R. Latała, "Some estimates of norms of random matrices," *Proceedings of the American Mathematical Society*, vol. 133, no. 5, pp. 1273–1282, 2005.

Appendix A. Supporting Lemmas & Supplementary Proofs

In this section, we present the lemmas required to prove the primary theorems in Section 4.

A.1. Supporting Definitions and Lemmas

The circular matrix of a vector is given as follows.

Definition 2. *For a vector* $\boldsymbol{a} = (a_1, \cdots, a_n)^T \in \mathbb{R}^n$*, its circular matrix is*

$$\mathrm{circ}(\boldsymbol{a}) = \begin{bmatrix} a_1 & a_2 & \cdots & a_n \\ a_n & a_1 & \cdots & a_{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_2 & a_3 & \cdots & a_1 \end{bmatrix}.$$

The following lemma is to estimate the bound of a random matrix norm, where $s_1(\cdot)$ is the biggest singular value and we use $\|A\boldsymbol{x}\|_2 = s_1(A)\|\boldsymbol{x}\|_2$ to estimate the norm of a random matrix, *e.g.,* $\|W^*\boldsymbol{y}^*\|_2$.

Lemma 1 (Singular value bound of sub-Gaussian random matrices [69])**.** *Given a matrix* $A \in \mathbb{R}^{m\times n}$ *whose rows* A_i *are independent, mean zero, sub-gaussian isotropic random vectors in* $\mathbb{R}^n$*, for any* $\tau \geq 0$*,*

$$\mathbb{P}\left\{s_1(A) \leq \sqrt{m} + C_s K_s^2(\sqrt{n} + \tau)\right\} \geq 1 - 2e^{-\tau^2},$$

where C_s *is a universal constant and* $K_s = max_i\|A_i\|_2$*.*

The following lemma about order statistics is utilized to estimate each entry of $\mathbb{E}\|W - W^*\|_2$.

Lemma 2 (Order statistics [70]). *Given n i.i.d.[5] random variables $U_1, \dots, U_n \sim \mathcal{U}(-a, a)$, we have*

$$\mathbb{E}U_{(r)}^2 = a^2 \frac{(r+1)r}{(n+2)(n+1)},$$
$$\mathbb{E}U_{(r)}^4 = a^4 \frac{(r+3)(r+2)(r+1)r}{(n+4)(n+3)(n+2)(n+1)},$$

where $1 \le r \le n$ is a constant and $U_{(1)}, \dots, U_{(n)}$ are order statistics of $U_1, \dots, U_n$.

Note that even though the above Lemma 2 is for uniform distribution, it provides an upper bound for the weights following a sub-gaussian distribution when considering r is close to n in this work.

The next lemma estimates the upper bound of a random matrix's norm based on its expected value, which is used to estimate $\mathbb{E}\|W - W^*\|_2$ by its entry's expected value.

Lemma 3 (Expected norm of a random matrix [71]). *Let B be a random matrix whose entries $B_{i,j}$ are independent mean zero random variables with finite moment. Then*

$$\begin{aligned}\mathbb{E}\|B\|_2 \le & C_e[\max_i(\sum_j \mathbb{E}B_{i,j}^2)^{\frac{1}{2}} \\ & + \max_j(\sum_i \mathbb{E}B_{i,j}^2)^{\frac{1}{2}} + (\sum_{i,j} \mathbb{E}B_{i,j}^4)^{\frac{1}{4}}],\end{aligned}$$

where C_e is a universal positive constant.

Lemma 4 (Theorem 6 in [48]). *Let $\omega = e^{\frac{2\pi i}{p}}$, with $i = \sqrt{-1}$ and denote by S the $p \times p$ matrix that embodies the discrete Fourier transform, the weight matrix W^* in a convolutional layer can be estimated by*

$$S = \begin{bmatrix} \omega^1 & \cdots & \omega^p \\ \vdots & \ddots & \vdots \\ \omega^p & \cdots & \omega^{p^2} \end{bmatrix}.$$

Given a tensor $F \in \mathbb{R}^{c \times c \times q \times q}$, we denote $K \in \mathbb{R}^{c \times c \times p \times p}$ and $W^* \in \mathbb{R}^{cp^2 \times cp^2}$ as the matrix encoding the linear transformation computed by the convolutional layer parameterized by K, as defined in Section 4.2.2. Let $P^{(u,v)}$ be the $c \times c$ matrix such that the (s,t)-th element of $P^{(u,v)}$ is equal to the (u,v)-th element of $S^T K_{s,t} S$, $u, v \in [p]$, $s, t \in [c]$, or equivalently $P_{s,t}^{(u,v)} = (S^T K_{s,t} S)_{u,v}$. Then

$$\|W^*\|_2 = \max_{u,v \in [p]} \|P^{(u,v)}\|_2.$$

A.2. Supplementary Proofs on Properties of Convolutional Networks

The following part proves an upper bound of $\|W^*\|_2$ for convolutional networks.

5. Independent and identically distributed.

A.2.1. Proof for Property 4.

Proof. We have $\|W^*\|_2 = \max_{u,v \in [p]}\{\|P^{(u,v)}\|_2\}$ by Lemma 4, and

$$P_{s,t}^{(u,v)} = (S^T K_{s,t} S)_{u,v} = \sum_{i,j \in [q]} \omega^{ui+vj} K_{s,t,i,j}.$$

Next,

$$\begin{aligned}\sum_{i,j \in [q]} \omega^{ui+vj} K_{s,t,i,j} \le & \|K_{s,t}\|_2 = \|F_{s,t}\|_2 \\ \le & \sqrt{q} + C_s K_s^2(\sqrt{q} + \tau).\end{aligned}$$

Then by Lemma 1 we have

$$\begin{aligned}\|W^*\|_2 &= \max_{u,v \in [p]}\{\|P^{(u,v)}\|_2\} \\ &\le \max_{u,v \in [p]}\{\sum_{s \in [c'], t \in [c]} \|P_{s,t}^{(u,v)}\|_2\} \\ &\le \max_{u,v \in [p]}\{cc' \|F_{s,t}\|_2\} \\ &\le cc'[\sqrt{q} + C_s K_s^2(\sqrt{q} + \tau)] \\ &\le C^2[\sqrt{Q} + C_s K_s^2(\sqrt{Q} + \tau)].\end{aligned}$$

Lastly, we calculate the probability when this inequality holds.

$$\begin{aligned}&\mathbb{P}\{\|W^*\|_2 \le C^2[\sqrt{Q} + C_s K_s^2(\sqrt{Q} + \tau)]\} \\ =& \sum_{s \in [c'], s \in [c]} \mathbb{P}\{\|F_{s,t}\|_2 \le \sqrt{q} + C_s K_s^2(\sqrt{q} + \tau)\} \\ =& (1 - 2e^{-\tau^2})^{cc'} \\ \ge& (1 - 2e^{-\tau^2})^{C^2}.\end{aligned}$$

□

Then, we can approximate the upper bounds of output for convolutional networks.

A.2.2. Proof for Property 5.

Proof. By Lipschiz constant of the activation function and the upper bound of the transformed convolutional weight matrix by Property 4,

$$\begin{aligned}&\|\boldsymbol{y}^{*,(m)}\|_2 \\ =& \|\phi(W^{*,(m)} \boldsymbol{y}_k^{*,(m-1)})\|_2 \\ =& \|\phi(W^{*,(m)} \boldsymbol{y}_k^{*,(m-1)}) - \phi(0)\|_2 \\ \le& L\|W^{*,(m)} \boldsymbol{y}^{*,(m-1)} - 0\|_2 \\ =& L\|W^{*,(m)} \boldsymbol{y}^{*,(m-1)}\|_2 \\ \le& Lcc'\lambda \|\boldsymbol{y}^{*,(m-1)}\|_2 \\ \le& (Lcc'\lambda)^2 \|\boldsymbol{y}^{*(m-2)}\|_2 \\ & \cdots \\ \le& (Lcc'\lambda)^m \|\boldsymbol{x}\|_2 \\ \le& (LC^2\lambda)^m \|\boldsymbol{x}\|_2,\end{aligned}$$

where $\lambda = \sqrt{q} + C_s K_s^2(\sqrt{q} + \tau)$. Then we calculate the probability that this inequality holds,

$$\begin{aligned}
&\mathbb{P}\{\|\boldsymbol{y}^{*,(m)}\|_2 \leq (LC^2\lambda)^m \|\boldsymbol{x}\|_2\} \\
=&\prod_{k=1}^{m} \mathbb{P}\left\{\|W^*\|_2 \leq C^2\lambda\right\} \\
\geq&(1-2e^{-\tau^2})^{C^2 m}.
\end{aligned}$$

□

Next, we give the upper bound for $\|W^{(m)} - W^{*,(m)}\|_2$

A.2.3. Proof for Property 6.

Proof. The non-zero filters F^* composing of $W^{(m)} - W^{*,(m)}$ are $\{F^*_{s,t} \mid (s,t) \in \mathcal{I}\}$. For any $F^*_{s,t}$ by Lemma 2,

$$\begin{aligned}
\mathbb{E}|\sum_{i,j\in[q]} F^*_{s,t,i,j}|^2 \leq&\mathbb{E}|\sum_{i,j\in[q]} F^*_{s_D,t_D,i,j}|^2 \\
=&q^2C_w^2\frac{(D+1)D}{(D+2)(D+1)} \\
\leq&q^2C_w^2\frac{(2D)^2}{D^2} = 4q^2C_w^2
\end{aligned}$$

and

$$\begin{aligned}
\mathbb{E}|\sum_{i,j\in[q]} F^*_{s,t,i,j}|^4 \leq&\mathbb{E}|\sum_{i,j\in[q]} F^*_{s_D,t_D,i,j}|^2 \\
=&q^4C_w^4\frac{(D+3)(D+2)(D+1)D}{(D+4)(D+3)(D+2)(D+1)} \\
\leq&q^4C_w^4\frac{(2D)^4}{D^4} \\
=&16q^4C_w^4.
\end{aligned}$$

Similar to Property 3 by Lemma 3, let D_w be the maximum number of the extracted filters, we gave

$$\begin{aligned}
\mathbb{E}\left\|W - W^*\right\|_2 =&\mathbb{E}\max_{u,v\in[p]}\{\sum_{(s,t)\in\mathcal{I},i,j\in[q]} \|F_{s,t,i,j}\|_2\} \\
\leq&6q^2C_eC_w\sqrt{D_w}
\end{aligned}$$

By Markov's inequality, for all $\mu \geq 0$,

$$\begin{aligned}
\mathbb{P}\{\|W - W^*\|_2 \geq \mu\} \leq&\frac{\mathbb{E}\|W - W^*\|_2}{\mu} \\
\leq&\frac{6q^2C_eC_w\sqrt{D_w}}{\mu} \\
\leq&\frac{6Q^2C_eC_w\sqrt{D_w}}{\mu}.
\end{aligned}$$

□

Appendix B. Additional Experiments and Statistics

B.1. Granular Utility Control

TABLE 5: Granular utility control (5% granularity) on ResNet-164 and DenseNet-40 trained on CIFAR-100.

Extraction Ratio	ResNet-164		DenseNet-40	
	Utility	Range (%)	Utility	Range (%)
0.0005	73.3%	70 – 75	70.2%	70 – 75
0.0010	71.6%	70 – 75	66.3%	65 – 70
0.0015	69.3%	65 – 70	63.5%	60 – 65
0.0020	66.6%	65 – 70	60.0%	60 – 65
0.0025	63.1%	60 – 65	55.9%	55 – 60
0.0030	61.3%	60 – 65	53.7%	50 – 55
0.0035	59.7%	55 – 60	51.5%	50 – 55
0.0040	56.3%	55 – 60	47.4%	45 – 50
0.0045	53.2%	50 – 55	43.6%	40 – 45
0.0050	51.9%	50 – 55	43.1%	40 – 45
0.0055	45.9%	45 – 50	39.3%	35 – 40
0.0060	43.9%	40 – 45	36.7%	35 – 40
0.0065	41.0%	40 – 45	34.1%	30 – 35
0.0070	35.7%	35 – 40	29.3%	25 – 30
0.0075	32.0%	30 – 35	27.2%	25 – 30
0.0080	32.2%	30 – 35	25.2%	25 – 30
0.0085	28.7%	25 – 30	25.0%	20 – 25
0.0090	27.9%	25 – 30	20.8%	20 – 25
0.0095	26.7%	25 – 30	19.5%	15 – 20
0.0100	24.4%	20 – 25	19.5%	15 – 20

B.2. Compatibility with Other Model Architectures

TABLE 6: CORELOCKER's access control capability on models with other popular architectures (BERT, CNN-LSTM, Vision Transformer). With a global extraction ratio of 0.05, all settings reached an expected utility level for access control ($Acc \approx 1/N_{class}$) post-extraction. Results confirm that CORELOCKER is model-independent, as it is built on the fundamental characteristic of impact concentration in neural networks.

Model	Architecture	Dataset	#Class	Original	CORELOCKER
BERT	Transformer	AG News	4	87.2%	25.3%
CNNLSTM	CNN, RNN	CIFAR-10	10	81.1%	10.0%
ViT	Transformer	CIFAR-100	100	81.4%	1.4%

TABLE 7: Granular utility control on Vision Transformer (ViT) trained on CIFAR-100 dataset. Model owners may regulate the model's utility level by adjusting the key extraction volume.

Extraction Ratio	Utility (Accuracy)	Utility Range (%)
0.0000	81.4%	-
0.0050	74.2%	70 – 80
0.0100	65.0%	60 – 70
0.0150	58.1%	50 – 60
0.0200	41.8%	40 – 50

Appendix C. Meta-Review

The following meta-review was prepared by the program committee for the 2024 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

C.1. Summary

The paper introduces CORELOCKER, a novel method for protecting deep neural network models from unauthorized use. It extracts a critical subset of the model's weights, effectively serving as an access key that limits full functionality to authorized users. The proposed methods are tested across various neural architectures, demonstrating its effectiveness in significantly reducing unauthorized performance while ensuring easy restoration for legitimate users.

C.2. Scientific Contributions

- Provides a Valuable Step Forward in an Established Field

C.3. Reasons for Acceptance

- This paper provides a valuable step forward in an established field. The paper proposes a novel problem setting in AI model usage control, and addresses it with a simple yet effective method.