EdgePro: Edge Deep Learning Model Protection via Neuron Authorization

Jinyin Chen, *Member, IEEE*, Haibin Zheng, Tao Liu, Jiawei Liu, Yao Cheng, Xuhong Zhang, and Shouling Ji, *Member, IEEE*

***Abstract*—With the development of deep learning processors and accelerators, deep learning models have been widely deployed on edge devices as part of the Internet of Things. Edge device models are generally considered as valuable intellectual properties that are worth for careful protection. Unfortunately, these models have a great risk of being stolen or illegally copied. The existing model protections using encryption algorithms are suffered from high computation overhead which is not practical due to the limited computing capacity on edge devices. In this work, we propose a light-weight, practical, and general *Edge* device model *Protection* method at neuron level, denoted as EdgePro. Specifically, we select several neurons as authorization neurons and set their activation values to locking values and scale the neuron outputs during training, where the authorization neurons, locking value, and scale factor together form the "passwords". Then, we design lock training to implement model property protection through alternately locking and releasing, which correspond to model performance preservation and encryption, respectively. EdgePro protects the model by ensuring it can only work correctly when the "passwords" are met, at the cost of encrypting and storing the information of the "passwords" instead of the whole model. Extensive experimental results indicate that EdgePro can work well on the task of protecting models on different datasets. The inference time increase of EdgePro is only 60% of state-of-the-art methods, and the accuracy loss is less than 1%. Additionally, EdgePro is robust against adaptive attacks including fine-tuning, reverse engineering, and pruning, which makes it more practical in real-world applications.**

***Index Terms*—Neural network, model protection, authorization control.**

Manuscript received 4 July 2022; revised 23 November 2023; accepted 10 February 2024. Date of publication 13 February 2024; date of current version 4 September 2024. This work was supported in part by the NSFC under Grant 62072406, and in part by Zhejiang Provincial Natural Science Foundation under Grant LDQ23F020001. *(Jinyin Chen and Haibin Zheng contributed equally to this work.) (Corresponding author: Haibin Zheng.)*

Jinyin Chen and Haibin Zheng are with the Institute of Cyberspace Security and the College of Information Engineering, Zhejiang University of Technology, Hangzhou 310023, China (e-mail: chenjinyin@zjut.edu.cn; haibinzheng320@gmail.com).

Tao Liu and Jiawei Liu are with the College of Information Engineering, Zhejiang University of Technology, Hangzhou 310023, China (e-mail: leonliu022@163.com; ljv0310@163.com).

Yao Cheng is with TÜV SÜD Asia Pacific Pte. Ltd., Singapore 189720 (e-mail: yao.cheng@tuvsud.com).

Xuhong Zhang is with the College of Control Science and Engineering, Zhejiang University, Hangzhou 310007, China (e-mail: zhangxuhong@zju.edu.cn).

Shouling Ji is with the College of Computer Science and Technology, Zhejiang University, Hangzhou 310007, China (e-mail: sji@zju.edu.cn).

EdgePro is also open sourced to facilitate future research: https://github.com/Leon022/EdgePro.

Digital Object Identifier 10.1109/TDSC.2024.3365730

I. Introduction

THE wide use of deep learning models in Internet of Things (IoT) has greatly facilitated humans in the fields of smart city, intelligent medical treatment, and industrial manufacturing [1], [2], [3], [4]. Deploying deep learning models directly on edge devices is trending thanks to the advancement of both effective and light-weight deep learning models and energy-saving deep learning processors and accelerators [5].

However, edge device models are vulnerable to illegitimate access by adversaries [6]. As shown in Fig. 1(a), when there is no protection for the edge device model, an attacker can easily access the edge device, copy the model and use or sell it for profits. In order to prevent being stolen or abused, the edge device model need to be carefully protected.

The existing work on edge device model protection can be roughly categorized into hardware-based and software-based protection. The hardware-based protections use Trusted Execution Environments (TEE) [7], [8], [9], [10] to build a trust region on the main processor, which ensures the models stored in the trust region can run safely. However, TEE is required to swap pages between secure and unprotected memory frequently, which incurs significant overhead. Other hardware-based protections that set proprietary encryption chips to store the models [11], [12]. However, they require special customization for different edge devices, which makes them difficult to be widely applied due to the large variety of IoT devices. Software-based protections use provably secure cryptographic methods to encrypt the models [13], [14], [15], [16]. The encryption and decryption theoretically ensure the security property. Unfortunately, the time cost to encrypt and dynamically decrypt a large number of model parameters on edge devices makes this line of approaches impractical [17], [18], [19]. To sum up, the encryption and decryption of previous protection methods are separated from the model prediction. We need to decipher large-scale model weights before using the model to perform predictions. It leads to potential vulnerability and high time costs for intermediate steps.

There are four challenges in edge device model protection on edge devices, which can also be seen as four requirements. The protection should 1) effectively protect models from unauthorized usage; 2) have little impact on the accuracy of the models; 3) be light-weight to run smoothly on the resource-limited edge devices; 4) be robust even if the adversaries know about the existence of the protection, i.e., robust against adaptive attacks.

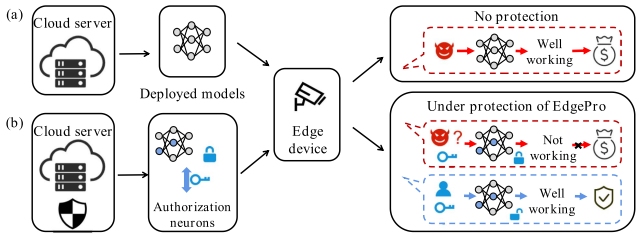


Fig. 1. Illustration of EdgePro, which locks neuron activation values to the locking values. Then, we can ensure that the model cannot work properly without knowing locking values, allowing light-weight protection of edge device models.

To overcome the above challenges, one possible solution is to add special markers to the input and train a model that can work only when special markers are part of the input. However, adding markers to input may increase the burden of input preprocessing on the edge devices. We have observed that the special markers on the input will indirectly cause the changes in neuron activation values during our exploration. In fact, the objective of model encryption can also be achieved directly by controlling the changes of neurons.

Therefore, it motivates us to propose a new *Edge* device model *Protection* method, EdgePro, from the perspective of neurons. The core idea is to leverage the activation values of a small number of neurons as markers, which are called authorization neurons. If the activation values of these neurons are not met with the locking values during the model inference, it is considered an unauthorized model inference, and vice versa, as shown in Fig. 1(b). Note that EdgePro can also be deployed on cloud servers to provide protection for deep models. However, the original intention of EdgePro is to implement lightweight protection for edge devices with limited computing resources. Therefore, we mainly consider protection scenarios on edge devices in this paper.

Specifically, we randomly select authorization neurons to guarantee their unpredictability, then lock their activation values as special markers during training, and scale the activation values of each layer. To realize it, we design lock training. During the training, we control the activation state of neurons by alternating locking and releasing, where locking is for model performance preservation and releasing is for encryption. Therefore, we can guarantee that neurons in the model have distinct activation states when authorization neurons are locked or not. Afterward, during the inference, to infer the input using EdgePro-trained models, EdgePro only requires to set the activation values of authorization neurons to the locking values. No other runtime cost is introduced, which is light-weight and practical for running on edge devices. Moreover, the impact of EdgePro on accuracy is negligible by ensuring the model converges during the training. The inference time increase of EdgePro is only 60% of state-of-the-art (SOTA) methods, and the accuracy loss is less than 1%. We evaluate the robustness of EdgePro against adaptive attacks, the experimental results of which show strong robustness against reverse engineering, model pruning and model fine-tuning. Last but not least, experiments on graph datasets show that EdgePro is general.

To summarize, the contributions of this paper are as follows,

- As far as we know, from the perspective of neurons, we first time propose a light-weight method, EdgePro, which can protect the edge device models from unauthorized usage from the perspective of neurons.
- The experimental results demonstrate that EdgePro can protect the models well. The accuracy loss of the EdgePro-trained model is only around 1%, and the inference time increase of EdgePro is only 60% of other SOTA methods.
- EdgePro is further evaluated under three adaptive attacks, e.g., reverse engineering, model pruning and model fine-tuning, the results of which demonstrate strong robustness against these adaptive attacks.

II. Related Works

A. Hardware Protection of Edge Device Models

An important way to protect the edge device models on edge devices is hardware root-of-trust [20], [21]. They suggested that a complete edge device model should be implemented in the TEE to protect the confidentiality and integrity of models. Nakai et al. [9] extended TEE from Intel's SGX to ARM's TrustZone, which is more suitable for edge devices. Due to the limited storage of TEE, Gangal et al. [22] partitioned an edge device model and encapsulated only some layers in SGX powered TEE. However, the model parameters stored in unprotected memory are still easy to be stolen, and adversaries can build a complete model through model reverse engineering [23], [24]. Another idea is to explore building custom secure neural network accelerators [19], [25], [26]. They used the Physical Unclonable Function and Processing-In-Memory to ensure that the model can be decrypted only for the authorized devices.

In addition, some work [27], [28], [29] also propose to protect the model parameters by confusing the storage of the model. Cammarota et al. [27] proposed HANN, a hardware-assisted model protection approach. They obfuscated the weights of the model based on a secret key that is stored in a trusted hardware device. Users can use the model only if they can provide the trusted key device. Goldstein et al. [28] proposed a solution based on hardware root of trust and public key cryptography infrastructure, which defends against model theft during model distribution and deployment/execution via light-weight, keyed model obfuscation scheme. Similarly, Hashemi et al. [29] provided provable model security by creating input obfuscation in TEE using a custom data encoding strategy based on matrix masks.

B. Software Protection of Edge Device Models

The software protection of edge device models is realized in two forms: encryption algorithm and intellectual property protection. In intellectual property protection, Tang et al. [30] proposed a serial number-based model protection method, which uses the knowledge distillation to assign a serial number to the customer (student) model, and the customer model can be used normally only if the correct serial number is input. Chen and Wu et al. [31] designed an adversarial example-based transformation

module to provide empowered inputs. When an unauthorized user provides input to the model, it is perturbed by adversarial perturbations, resulting in poor performance. Fan et al. [32] proposed embedding a specific passport layer in the model, which can paralyze the functionality of the neural network if unauthorized use, or maintain its functionality if verified. Zhang et al. [33] also proposed a passport-aware normalization paradigm for model protection. A new passport-aware branch was added and trained along with the model. The model performance can be maintained only if the correct passport is provided, otherwise it will drop significantly [33]. Alam et al. [34] utilized s-boxes with cryptographic properties to lock parameters of a DNN model without causing significant increase in inference time and model scales. Pyone et al. [34] used block pixel shuffling with a key as a preprocessing technique to input images, and the protected model was built by training with such preprocessed images.

On the research of encryption algorithm for edge devices, Lu et al. [35] developed a secure query scheme with high communication efficiency in the fog environment, to ensure that both cloud and edge devices can use it for privacy protection. Fiore et al. [14] developed a multi-bond homomorphic authenticator from the perspective of data outsourcing, which is suitable for resource-constrained devices. In addition, there are some methods to encrypt the model and input on edge devices based on Yao's Garbled Circuits [15], [16]. However, the existing encryption algorithms still face the problems of high computational cost, complex decryption processes and low efficiency. This motivates us to propose a new light-weight model protection method, which can protect the model by binding the inputs and outputs.

III. Threat Model

Our objective is to design a light-weight model protection method that can protect models from unauthorized use in an untrusted environment. For adversaries, they aim to gain benefits from the stolen models deployed on edge devices. They can fully access the memory or execution environment on edge devices at any time, which may be from the actual user, malicious third-party software installed on the device, or a malicious or infected operating system. More demanding, we assume the adversaries know the existence of EdgePro. They may design specific adaptive attacks to crack EdgePro by using state-of-the-art techniques, e.g., model fine-tuning, reverse engineering and model pruning. Specifically, the adaptive attacks are detailed below.

- *Model Fine-tuning Attack:* In practice, for an adversary who lacks training data and intends to break EdgePro, one of the easiest ways is to fine-tune the stolen model [36], [37], [38]. Model fine-tuning attack is an intentional attack performed by an adversary who tries to invalidate the authorization neurons, which can be regarded as one of the most threatening attacks on EdgePro. In general, fine-tuning can produce a new model with less extra training data based on the stolen model. In this way, the new model can inherit the performance of the stolen model, but also forget the previous training information [39].
- *Reverse Engineering Attack:* The purpose of these attacks is to fit the structure and parameters of the target model as much as possible, which builds an avatar and continuously queries [40]. When the model trained by EdgePro is unauthorized, the more realistic the attacker fits it, the worse the performance of the extracted model will be. EdgePro can naturally defend against these attacks. Therefore, instead of directly using them, we designed reverse engineering specifically for EdgePro to infer its passwords. The adversary may perform reverse engineering to find multiple authorization neurons, thereby obtaining knowledge and details of EdgePro, such as the position of authorization neurons or locking values.
- *Model Pruning Attack:* Model pruning [41], [42], [43] is a technique to reduce the computational overhead of executing a neural network and still keep the performance of the original model by removing redundant neurons. An adversary may prune and aim to remove the authorization neurons embedded in the model to invalidate EdgePro. Ideally, the adversary may obtain a model with high classification accuracy, and use the model normally after pruning.

IV. Methodology

EdgePro consists of two stages, i.e., "passwords" determination and lock training. The overview of EdgePro is presented in Fig. 2. The "passwords" of EdgePro consist of three parts: authorization neurons, locking values and scale factors. Before lock training, the protector needs to determine these three parts, as shown in Fig. 2(i). The second stage is the lock training, as shown in Fig. 2(ii). First, an obfuscated dataset is created for lock training. Then, during the lock training, by alternately locking and releasing authorization neurons, while adjusting different training objectives (i.e., encryption and performance preservation), EdgePro helps the model adapt to the locking of authorization neurons and eventually converge. Finally, the EdgePro-trained model will not work properly if the activation values of authorization neurons do not reach the locking values in the inference.

A. "Passwords" Determination

EdgePro embeds activation authorization mechanisms, i.e., specific neurons need to meet corresponding activation values, to make sure it will be extremely difficult for unauthorized usage of stolen models. Therefore, these neurons, which we define as authorization neurons, will be part of the "passwords" in the model. In addition to authorization neurons, the "passwords" of the model are also composed of the authorization neuron activation values, which we define as locking values, and the scale factors of neuron activation values in each layer.

In order to ensure unpredictability, we consider that the choice of "passwords" should be random, without any strategies. Therefore, EdgePro will randomly select a small number of neurons in each layer as authorization neurons. We use ρ to express the

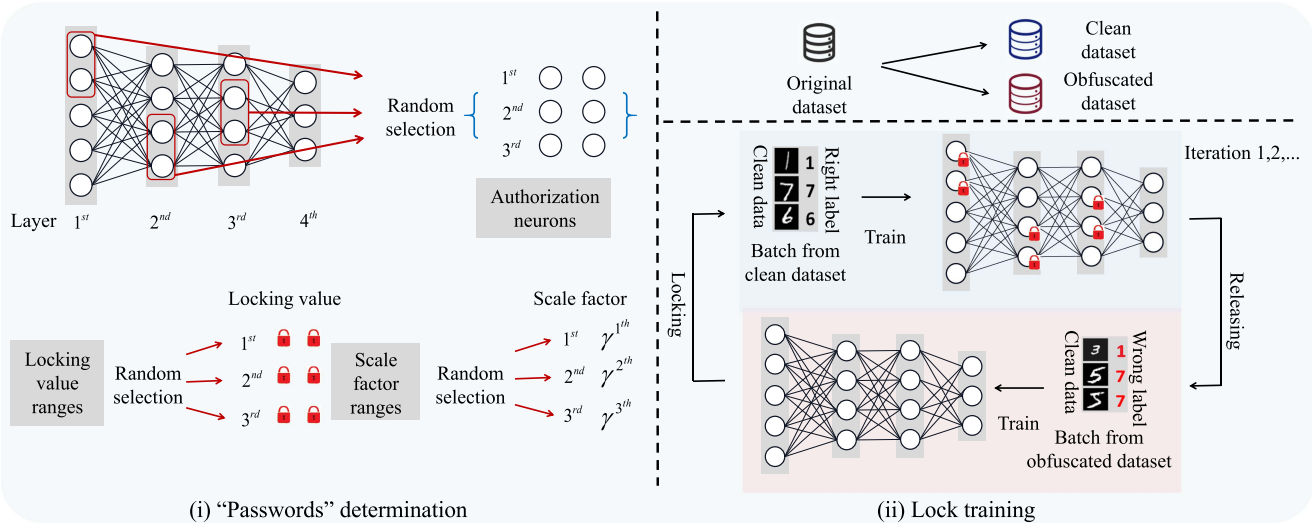


Fig. 2. Overview of EdgePro, it includes two parts: "passwords" determination and lock training. "Passwords" determination is responsible for selecting authorization neurons, locking values and scale factors. Lock training is responsible for training preparation and lock training for different objectives.

proportion of neuron selection authorized by each layer. In view of the learnability of neural network [44], [45], when ρ is small, the impact on model performance after authorized is limited (we verified the impact of ρ ratio on model performance in Section V-G.1). This is the same for the locking values and scale factors, we randomly select them from specific ranges. As for the specific range, we prove in experiments that an appropriate range will not affect the performance of the model.

Assume $\mathcal{N}$ is the set of all neurons in the kth layer, $\mathcal{A}^k$ is the set of authorization neurons in the kth layer, and $\mathcal{V}^k$ is the set of locking values corresponding to authorization neurons. When an input example is fed to the model, the ith neuron activation value α_i^{k+1} in the $(k+1)$th layer can be expressed as:

$$\alpha_i^{k+1} = \gamma_{k+1} \cdot \left(Relu \left(\left(\sum_{j|j\in\mathcal{N}\wedge j\notin\mathcal{A}^k} w_{j,i}^k \cdot \alpha_j^k + \beta_i^k \right) + \left(\sum_{j|j\in\mathcal{N}\wedge j\in\mathcal{A}^k} v_j^k \right) \right) \right) \tag{1}$$

where α_j^k represents the jth neuron activation value in the kth layer, $w_{j,i}$ corresponds to the weight between the ith neuron and the jth neuron, and β_i^k represents the bias. Here we use $Relu$ as the activation function for demonstration because of its non-negative property. γ_{k+1} is the scale factor of the $(k+1)$th layer. (1) means that when α_j^k is not an authorization neuron, its activation value will be calculated by weight. When α_j^k is an authorization neuron, EdgePro will discard its original activation value and replace it with a locking value v_j^k, regardless of any input. Note that in (1) we just take ReLU as an example to show the calculation process of α. ReLU can be replaced by any other activation function, such as Tanh.

B. Lock Training

1) Training Preparation: During training preparation, EdgePro needs to build the training dataset. EdgePro divides the original training dataset $D_{ori} = (X_{ori}, Y_{ori})$ into two equal parts, clean dataset $D_c = (X_c, Y_c) = \{(x_i, y_i)|_{i=1,2,\ldots,N_c}\}$ and obfuscated dataset $D_o = (X_o, Y_o) = \{(x_j, y'_j)|_{j=1,2,\ldots,N_o}\}$, where each sample x_j in the obfuscated dataset is assigned a wrong label y'_j, $X_c \cap X_o = \emptyset$, $X_c \cup X_o = X_{ori}$, N_c and N_o are the total number of the samples in D_c and D_o, respectively. Finally, we build the training dataset for EdgePro, represented as $D = D_c \cup D_o$.

2) Lock Training Process: In lock training, each iteration contains two batches of training. First, EdgePro samples a batch from the clean dataset D_c, and sets the activation values of authorization neurons to the locking values for training. In this training, the output of each layer will be scaled by scale factors. According to the requirement of model performance preservation, we can get the locking objective of EdgePro as follows:

$$\Theta = \arg\min_{\Theta} \sum_{(x_i,y_i)\in D_c} L(R(g(\Theta; x_i)), y_i) \tag{2}$$

where Θ is the mode parameter updated during the training, $g(\cdot;\cdot)$ is the function to calculate the model output while the model is locked. $L(\cdot,\cdot)$ is the loss function, (x_i, y_i) from D_c means the sample x_i and its ground truth y_i. $R(\cdot)$ operates the activation process of $g(\cdot;\cdot)$, which sets the activation values of authorization neurons to the locking values and scales the output based on (1). The setting of some activation values to a fixed value is similar to Dropout, and the scaling is similar to the Layer Normalization.

Then EdgePro releases the authorization neurons and samples a batch from the obfuscated dataset for training. According to the requirement of model encryption (i.e., restricting the unauthorized model to work properly), we can get the releasing

Algorithm 1: Lock Training Algorithm.

Input: Original dataset D_{ori}, number of epochs K, authorization operation $R(\cdot)$, batch size N_{bs}.
Output: EdgePro-trained model parameter Θ.

Initialize $k = 1$, divide D_{ori} into D_c and D_o according to Section 4.2.1.
while $k < K$ **do**
  for *iteration* 1,2,... **do**
    Sampling N_{bs} examples form D_c.
    for (x_i, y_i) *in batch* **do**
      Update Θ based on Eq. (2).
    end
    Sampling N_{bs} examples form D_o.
    for (x_j, y'_j) *in batch* **do**
      Update Θ based on Eq. (3).
    end
  end
  $k = k + 1$
end

objective of EdgePro as follows:

$$\Theta = \arg\min_{\Theta} \sum_{(x_j, y'_j) \in D_o} L(h(\Theta; x_j), y'_j) \tag{3}$$

where $h(\cdot;\cdot)$ is the function to calculate the model output while the model is unlocked, which shares with $g(\cdot;\cdot)$ the same parameter obtained during optimization. (x_j, y'_j) from D_o means the sample x_j and its wrong label y'_j.

The process of lock training is shown in Algorithm IV-B2, which alternately trains model parameters according to (2) and (3). EdgePro performs iterative training until the model converges on the clean dataset. Note that we do not do anything with the obfuscated dataset. The overall training objective of EdgePro can be obtained by combining (2) and (3), as follows:

$$\Theta = \arg\min_{\Theta} \left(\sum_{(x_i, y_i) \in D_c} L(R(g(\Theta; x_i)), y_i) + \sum_{(x_j, y'_j) \in D_o} L\left(h\left(\Theta; x_j\right), y'_j\right) \right) \tag{4}$$

The ways of utilizing those parameters differ in those two functions (i.e., $g(\cdot;\cdot)$ and $h(\cdot;\cdot)$). The alternate training impacts the optimization of the parameters rather than optimizing different parameters. The training objective in (4) is for the model to achieve the highest classification accuracy when the activation values of authorization neurons are set to the locking values, and vice versa. In this way, we embed the active authorization mechanism in the EdgePro-trained model. We only need to encrypt and store the authorization neurons, their locking values, and scale factors, instead of the entire model weights to protect the model.

C. Theoretical Analysis of EdgePro

We can get the authorized model $f(\Theta_{auth}; x) = g(\Theta; x)$ and unauthorized model $f(\Theta_{unau}; x) = h(\Theta; x)$, where the weights Θ_{unau} of the unauthorized model can be converted to the weights Θ_{auth} of the authorized model through the passwords. The purpose of model property protection is to ensure that the authorized model works properly, while the unauthorized model cannot. Taking the classification model as an example, the purpose of model property protection can be expressed as:

$$\begin{cases} f(\Theta_{auth}; x_i) = y_i \\ f(\Theta_{unau}; x_i) \neq y_i \end{cases} \tag{5}$$

where x_i is the test input and y_i is its corresponding ground truth.

Then, we relax the constraints, i.e., $\forall x_i$, $\exists \Theta_{unau}$, so that $f(\Theta_{unau}; x_i) \neq y_i$ holds, and $\exists \Theta_{auth}$, so that $f(\Theta_{auth}; x_i) = y_i$ holds. Although the expected property protection is that the unauthorized model predicts all test inputs incorrectly, the prediction results close to random guesses are sufficient in practice. Thus, we further relax the constraints. Taking classification accuracy as an example, we define that the performance of the unauthorized model should be close to random prediction, i.e., the accuracy is close to $\frac{1}{N} \times 100\%$, where N is the number of classes. Note that there is a premise that the classification accuracy of the authorized model is close to that of the normally trained model. Then, we get *Theorem* 1 as follows.

Theorem 1: For a test input x_i, if the classification accuracy of the unauthorized model satisfies $\left|\frac{N_{unau}|_{f(\Theta_{unau}; x_i) = y_i}}{N} - \frac{1}{N}\right| \leq \delta_{unau}$ and $\left|\frac{N_{auth}|_{f(\Theta_{auth}; x_i) = y_i}}{N} - acc_{normal}\right| \leq \delta_{auth}$, then the property protection of EdgePro is valid. $N_{unau}|_{f(\Theta_{unau}; x_i) = y_i}$ represents the number of test inputs correctly classified by the unauthorized model, $N_{auth}|_{f(\Theta_{auth}; x_i) = y_i}$ represents the number of test inputs correctly classified by the authorized model, $|\cdot|$ means absolute value, acc_{normal} is the accuracy of the normally trained model, δ_{unau} and δ_{auth} represent a small positive value.

Proof: To effectively protect model property, we need to ensure that the inequality holds. We minimize the difference, i.e., $\min \left|\frac{N_{unau}|_{f(\Theta_{unau}; x_i) = y_i}}{N} - \frac{1}{N}\right|$ and $\min \left|\frac{N_{auth}|_{f(\Theta_{auth}; x_i) = y_i}}{N} - acc_{normal}\right|$. To ensure that the unauthorized model does not work properly, we minimize its classification accuracy, i.e., maximize the misclassification result, as follows.

$$\begin{aligned} &\min \left|\frac{N_{unau}|_{f(\Theta_{unau}; x_i) = y_i}}{N} - \frac{1}{N}\right| \\ \Rightarrow\ &\min \frac{N_{unau}|_{f(\Theta_{unau}; x_i) = y_i}}{N} \times 100\% \\ \Rightarrow\ &\max \frac{N_{unau}|_{f(\Theta_{unau}; x_i) \neq y_i}}{N} \times 100\% \\ \Rightarrow\ &\arg\min_{\Theta} \sum_{(x_j, y'_j) \in D_o} L(f(\Theta_{unau}; x_j), y'_j) \end{aligned} \tag{6}$$

To ensure that the authorization model works properly, we maximize its classification accuracy, as follows.

$$\min \left|\frac{N_{auth}|_{f(\Theta_{auth}; x_i) = y_i}}{N} - acc_{normal}\right|$$

$$\Rightarrow \max \frac{N_{auth}|_{f(\Theta_{auth};x_i)=y_i}}{N} \times 100\%$$

$$\Rightarrow \arg\min_{\Theta} \sum_{(x_i,y_i)\in D_c} L(R(f(\Theta_{auth};x_i)),y_i) \quad (7)$$

Then, we conclude that the locking objective in (2) and the releasing objective in (3) satisfy the conditions of Theorem 1, i.e., the overall training objective in (4) can be derived from Theorem 1. Note that the smaller the lower bounds of δ_{unau} and δ_{auth} are, the better the property protection effect of EdgePro is.

We preliminarily test the performance of EdgePro on the NVIDIA Jetson Nano edge device [46]. We deploy three models on the edge device, i.e., the original LeNet-5 model trained on the MNIST dataset without protection, the model protected by Deepsecure [15], and the model protected by EdgePro. We conclude that the increased memory consumption and time cost of LeNet-5 protected by Deepsecure (i.e., +158% and +126%) are much higher than those protected by EdgePro (i.e., +0% and +24%). EdgePro provides lightweight and efficient protection of deep models in edge devices, making it naturally adaptable to "edge".

V. Experiments

A. Experimental Setup

We will introduce datasets, models, baselines, and configuration information. We refer to previous works [31], [32], [34] to determine the experimental settings.

Datasets and Models:

1) MNIST [47][1] is a general image classification dataset which contains 70,000 handwritten gray-scale digital images with size of 28x28, ranging from 0 to 9 (10 classes).
2) CIFAR-10 [48][2] is a general image classification dataset which contains 60,000 RGB color images with size of 32x32 in 10 classes. Each pixel includes three RGB values, with an integer value in [0, 255].
3) CIFAR-100 [48][3] is a general image classification dataset which contains 60,000 RGB color images with size of 32x32 in 100 classes, which contains 50,000 training images and 10,000 testing images.
4) Tiny-ImageNet [49][4] is a computer vision dataset containing 200 classes. Each class has 500 training examples, 50 testing examples and 50 valid examples.

For each dataset, we use different network architectures for experiments. On MNIST, we adopt LeNet [47] and MLP. We adopt ResNet [50], VGG [51] for CIFAR-10, and ResNet [50], VGG [51], DenseNet [52] for CIAFR-100. On Tiny-ImageNet, experiments are implemented on DenseNet [52], SENet [53] and ShuffleNet [54]. Model configurations and experiment parameter setups are summarized in Table I, which records the learning rate, batch size, and training run epoch used for each dataset.

TABLE I
Model Configurations and Experiment Parameter Setups

Datasets	Models	Learning rate	Batch size	Epoch
MNIST	LeNet-1 LeNet-5 MLP	0.01	64	20
CIFAR-10	ResNet-18 ResNet-50 VGG-16	0.01	64	40
CIFAR-100	ResNet-101 VGG-19 DenseNet-121	0.001	128	100
Tiny-ImageNet	DenseNet-121 SeNet ShuffleNet	0.001	128	200

Evaluation Metrics: The metrics used in the experiments are defined as follows:

- Neuron locking accuracy (acc_{nl}): $acc_{nl} = \frac{n_{nl}}{N}$, where n_{nl} is the number of examples correctly classified by the model when it is authorized, N is the total number of examples.
- Neuron unlocking accuracy (acc_{nu}): $acc_{nu} = \frac{n_{nu}}{N}$, where n_{nu} is the number of examples correctly classified by the model when the model is not authorized, N is the total number of examples.

The larger the gap between acc_{nl} and acc_{nu} means the better the protective effect of EdgePro.

Baselines: We implement and compare four SOTA methods with EdgePro to evaluate their performance, including Password Normalization (PN) [33], Deep-Lock [34], AntiP [31], and LIE [55]. All baselines are advanced protection methods for authorized use of models, and they are configured according to the performance setting reported in the respective papers.

Platform: We leverage a platform with the following setup: CPU is Intel XEON 6240 2.6 GHz x 18 C, GPU is Tesla V100 32GiB, the Memory is DDR4-RECC 2666 16GiB, the operating system is Ubuntu 16.04, the programming language is Python 3.6.0, and the deep learning framework is PyTorch-1.4.0.

B. Effectiveness of EdgePro

In this section, we focus on the evaluation results of EdgePro when model is authorized and not authorized.

Implementation Details: 1) We evaluate EdgePro in two scenarios, including normal training and EdgePro training. In normal training, we train the model normally then test the model accuracy. In EdgePro traing, we set $\rho = 5$, which means 5% neurons in each layer are selected as the authorization neurons, and the each locking value v will be randomly selected in the range $\mathcal{V} = (0, 1)$. The scale factors γ also will be randomly selected in the range (0.2, 1). 2) We train the model until the specified training epoch is reached, or the model loss is below 1e-4. To mitigate non-determinism, we repeated the experiment for 5 times and reported the average results.

Results and Analysis: The results of which are shown in Table II. We can see that EdgePro-trained models achieve high acc_{nl} and low acc_{nu}. The low acc_{nu} values indicate that when the activation values of the authorization neurons in the

[1]MNIST can be downloaded at *http://yann.lecun.com/exdb/mnist/*

[2]CIFAR-10 can be downloaded at *https://www.cs.toronto.edu/\;kriz/ cifar.html*

[3]CIFAR-100 can be downloaded at *https://www.cs.toronto.edu/\;kriz/ cifar.html*

[4]Tiny-ImageNet can be downloaded at *http://cs231n.stanford.edu/tiny-imagenet-200.zip*

TABLE II
EVALUATION RESULTS OF EDGEPRO ON DIFFERENT DATASETS AND MODEL ARCHITECTURES, INCLUDING THE TEST ACCURACY FOR THE NORMALLY TRAINED MODEL, AND FOR EDGEPRO-TRAINED MODELS

MNIST			CIFAR-10			CIFAR-100			Tiny-ImageNet		
Models	Normal	EdgePro	Models	Normal	EdgePro	Models	Normal	EdgePro	Models	Normal	EdgePro
LeNet-1	98.48%	97.60% (17.67%)	ResNet-18	87.28%	87.10% (11.71%)	VGG-19	73.77%	70.52% (1.00%)	DenseNet-121	55.47%	55.10% (0.35%)
LeNet-5	100.00%	99.29% (10.01%)	ResNet-50	89.05%	88.72% (7.05%)	ResNet-101	75.98%	75.11% (0.68%)	SENet	56.80%	55.82% (0.61%)
MLP	98.88%	98.79% (12.16%)	VGG-16	89.10%	88.92% (13.15%)	DenseNet-121	74.71%	74.20% (1.03%)	ShuffleNet	56.20%	55.45% (0.40%)

For EdgePro, the values without brackets in the table represent acc_{nl}, and the values with brackets represent acc_{nu}.

EdgePro-trained model do not meet the locking values, the model accuracy is reduced to the level of random guessing. This reflects that EdgePro discourages illegal users from using models by reducing model performance. Regarding the acc_{nl}, compared with the original accuracy of the normal trained model, the EdgePro-trained model has a minor accuracy loss of 1.63% on average. The accuracy loss is understandable and maybe inevitable because EdgePro locks some of the neurons in the model. Overall, EdgePro has protective effects on the four datasets and for the twelve models.

C. Complexity Comparison of EdgePro

In this section, we focus on how much extra time cost is introduced by EdgePro in inferring phases.

Implementation Details: 1) We consider that training is offline and usually a one-time process. Therefore, the server has sufficient training time, and we concern about the time cost of the model in the inferring phase. We do not compare with the hardware method because hardware encryption is compatible with software encryption and EdgePro is basically a software-based protection as well. 2) EdgePro selected 5% of the neurons in the model as authorization neurons, then we measure the time cost for the model to infer 1000 examples records. Meanwhile, we measure the time cost of the baselines for comparison. For encryption protection method, Deep-Lock, the decryption also needs to be counted in.

Results and Analysis: The results can be found in Table III. In this experiment, we can see that for the inference phase time cost, EdgePro only takes 36.5% more than the normal process, while baselines take 164.2% more time. This is because the EdgePro authorization only needs to activate a small number of neurons to reach the locking values, which reduces the time overhead. This means that EdgePro introduces minimal overhead in the inference phase compared to baselines, which can improve model performance and user experience. Additionally, we observe that the EdgePro on large models add more time overhead. To this this problem, we consider that for large models, e.g., VGG-19, EdgePro can reduce the extra overhead by using a small ρ, e.g., using $\rho = 1$. Last but not the least, it cannot be ignored that EdgePro hardly increases the parameter size of the model, i.e., compared to the massive storage of deep models, EdgePro only introduces a small extra space for storing the "passwords". This facilitates the deployment of EdgePro on resource-constrained edge devices.

TABLE III
COMPARISON OF EDGEPRO'S TIME COST WITH NORMAL TRAINING AND BASELINES

Datasets	Models	Methods	Size(bit)	Time(s)
MNIST	LeNet-5	Normal	243K	0.50
		PN	282K	0.69
		Deep-Lock	486K	0.73
		AntiP	252K	0.63
		LIE	243K	0.65
		EdgePro	**243K**	**0.62**
CIFAR-10	ResNet-18	Normal	42.68M	0.76
		PN	44.60M	2.06
		Deep-Lock	85.36M	1.80
		AntiP	42.70M	1.26
		LIE	42.68M	1.44
		EdgePro	**42.68M**	**0.90**
CIFAR-100	VGG-19	Normal	174.08M	0.86
		PN	186.47M	3.70
		Deep-Lock	348.17M	3.46
		AntiP	174.10M	**1.76**
		LIE	174.08M	2.28
		EdgePro	**174.08M**	1.83
Tiny-ImageNet	DenseNet-121	Normal	27.78M	1.04
		PN	29.02M	1.90
		Deep-Lock	55.55M	1.76
		AntiP	27.82M	1.40
		LIE	27.78M	1.66
		EdgePro	**27.78M**	**1.39**

D. Robustness Comparison of EdgePro

The attacker may try to launch adaptive attacks against model protection methods if they know the existence of model protection methods. In this section, we compare the robustness between EdgePro and baselines against model fine-tuning attack. Going a step further, we evaluate the robustness of EdgePro under two adaptive attacks designed specifically for EdgePro (as analyzed in our threat model in Section III).

1) Model Fine-Tuning Attack: Fine-tuning could be an intuitive way for an adversary to remove authorization neurons in an EdgePro-trained model with a small amount of data. Therefore we compare the protection effect of EdgePro with baselines under fine-tuning attack.

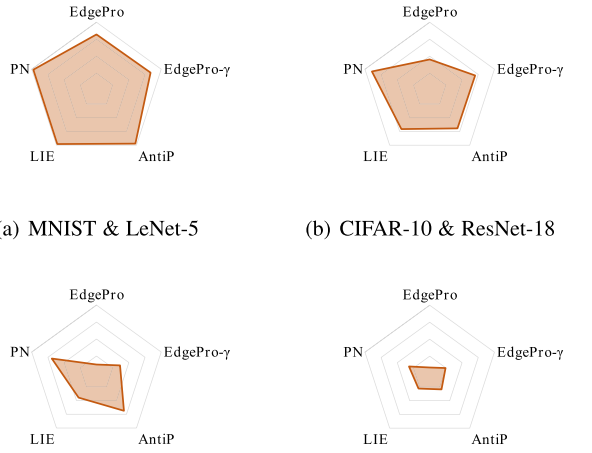
Implementation Details: 1) We reserve 10% test data for fine-tuning EdgePro-trained models. In particular, the fine-tuning epoch is 10 for the MNIST and CIFAR-10 datasets, and 20 for the CIFAR-100 and Tiny-ImageNet datasets. 2) We compare the robustness of baselines: PN [33], LIE [55], and AntiP [31]. We do not consider Deep-Lock [34] because it cannot be fine-tuning

TABLE IV
ROBUSTNESS EVALUATION OF EDGEPRO AGAINST THE FINE-TUNING ATTACK UNDER DIFFERENT DATA PERCENTAGE, MEASURED BY acc_{nu}

Datasets	Models	Data Percentage									
		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
MNIST	LeNet-5	10.20%	10.23%	10.48%	10.73%	10.88%	13.80%	11.28%	11.80%	12.78%	14.63%
CIFAR-10	ResNet-18	10.13%	10.18%	10.25%	10.43%	10.73%	12.13%	12.30%	12.85%	13.33%	13.55%
CIFAR-100	VGG-19	0.80%	0.90%	0.95%	0.98%	0.98%	1.03%	1.08%	1.15%	1.23%	1.29%
Tiny-ImageNet	DenseNet-121	0.38%	0.50%	0.55%	0.65%	0.75%	0.78%	0.88%	0.98%	1.05%	1.15%

TABLE V
ROBUSTNESS EVALUATION OF EDGEPRO AGAINST THE FINE-TUNING ATTACK UNDER DIFFERENT DATA DOMAINS, MEASURED BY acc_{nu}

Source Datasets	Target Datasets	acc_{nu}
ImageNet-200	Tiny-ImageNet	0.85%
Tiny-ImageNet	ImageNet-200	0.73%



(a) MNIST & LeNet-5 (b) CIFAR-10 & ResNet-18

(c) CIFAR-100 & VGG-19 (d) Tint-ImageNet & DenseNet-121

Fig. 3. Robustness evaluation against the fine-tuning attack.

after encryption. 3) In addition, we consider a new scenario: the scale factors are leaked, i.e., the adversary knows scale factors γ used by each layer of EdgePro. We define this scenario as "EdgePro-γ" to measure the robustness of EdgePro. 4) We use acc_{nu} as a measure of model robustness. The lower the acc_{nu} of the fine-tuned model, the more robust the method is. 5) We then evaluate the robustness of EdgePro against fine-tuning attacks under different data percentages and data domains. The experimental results are shown in Tables IV and V. In the fine-tuning experiments of a similar domain, ImageNet-200 contains 200 classes sampled from ImageNet [49], most of the labels overlap with Tiny-ImageNet, and the number of samples is close to that of Tiny-ImageNet.

Results and Analysis: The experimental results can be observed using the radar chart as shown in Fig. 3. Compared to baselines, EdgePro is the most robust on the four datasets. On complex datasets, EdgePro is more robust, e.g., on the CIFAR-100 and Tiny-ImageNet datasets, EdgePro's acc_{nu} is only about 25% of the baselines. Unlike baselines that aim to perturb the input or perturb the hidden layer output, EdgePro introduces scale factors into the model. In this way, EdgePro not only affects the classification layer of the model, but also controls the feature extraction effect of the model. When the model is not authorized, the scale factors corrupt the output of the model in each layer, thereby enhancing the robustness of EdgePro. Additionally, comparing different datasets, we find that the models (VGG-19 and DenseNet-121) on large datasets (CIFAR-100 and Tiny-ImageNet) are more robust than the models (LeNet-5 and ResNte-18) on small datasets (MNIST and CIFAR-10). We speculate the reason is that training on large datasets is more difficult than on small datasets, so the impact of fine-tuning is limited under the condition of a certain amount of data (10% of the test data). For "EdgePro-γ", it can be seen that when the attacker masters the scale factors of the model, "EdgePro-γ" is less robust than EdgePro, but still better than baselines. "EdgePro-γ" is still able to control model accuracy to an unusable level and has the effect of preventing illegal use of the model.

Moreover, EdgePro can effectively protect model property against fine-tuning attacks, even if the attacker collects a large amount of data for fine-tuning. For instance, in Table IV, when the data collected by the attacker increases by 9 times (i.e., from 10% to 100%), the acc_{nu} value only increases by 0.42 times on average. EdgePro is also robust against fine-tuning attacks even if the attacker adopts data from similar domains for fine-tuning. For instance, in Table V, when the attacker uses Tiny-ImageNet and ImageNet-200 as similar domain data for fine-tuning, the acc_{nu} value still meets the requirements of property protection, i.e., the model protected by EdgePro cannot work properly in an unauthorized state. We speculate that the weights of the unauthorized model affect the initialization of fine-tuning training. Fine-tuning is to slightly modify the model weights under the limited training cost. Therefore, when the model is unauthorized, it cannot work properly by modifying the weights in a small range.

2) Reverse Engineering Attack: We first evaluate the performance of EdgePro by implementing existing reverse attacks [56], [57] to extract model information, including the original target model without protection and the unauthorized model protected by EdgePro. Then, we evaluate the robustness of EdgePro under reverse engineering attack which is an attack method specifically designed for EdgePro. Finally, to protect the authorized model more reliably, we conduct experiments by adding noise to the output layer.

Implementation Details: 1) We preliminarily verify the protection effect of EdgePro on MNIST dataset by executing MAZE (a logit-query attack) [56] and DFMS-HL (a label-query attack) [57] to extract the target model without authorization, as

TABLE VI
EXTRACTION RESULTS OF DEEP MODELS WITH/WITHOUT EDGEPRO PROTECTION, WHERE acc IS THE CLASSIFICATION ACCURACY OF THE ORIGINAL MODEL WITHOUT PROTECTION, acc_{nu} IS THE ACCURACY OF THE MODEL WITH EDGEPRO PROTECTION WHEN UNAUTHORIZED, AND acc_{ext} IS THE ACCURACY OF THE EXTRACTED MODEL

Datasets & Models	Protections	Attacks	acc_{ext}
MNIST & LeNet-5	without (acc=100.00%)	MAZE	100.00%
		DFMS-HL	100.00%
	EdgePro (acc_{nu}=10.40%)	MAZE	**9.85%**
		DFMS-HL	**9.45%**

shown in Table VI. MAZE and DFMS-HL are configured according to the best performance setting reported in the respective papers. 2) To better evaluate the performance of EdgePro, we specially designed a reverse engineering attack against it, i.e., password inference. In this attack scenario, the attacker knows all the knowledge of the unauthorized target model except the password (i.e., authorized neuron positions, lock values, scaling factors). The attacker's goal is not to extract the model structure or parameters, but to extract the password. Therefore, we adopt the depth-first traversal strategy to extract passwords. The end condition of the traversal algorithm is that the model accuracy reaches the threshold or the search time exceeds the maximum setting. 3) Considering the high complexity of the traversal algorithm, we experiment on the EdgePro-trained model using the MNIST dataset. In the fourth and fifth layers of the LeNet-5 model, which have 120 and 84 neurons, respectively, we select one or two or three neurons as authorization neurons. 4) As shown in the Table VII, "AZ neurons" means the "passwords" and "4:29:0.7" refers to that EdgePro selects the 29th neuron in the 4th layer as the authorization neuron, and the locking value size is $v = 0.7$. "RE neurons" refers to the adversary finding the neurons that can crack the EdgePro. We set a running time for the reverse engineering attack. When the time exceeds 48 hours (172,800.00 seconds), the attack will terminate with a "Timeout". We record the passwords obtained by reverse engineering, and the model accuracy acc_{nu}^{re}, where the model is authorized by the password obtained through reverse engineering. 5) We set up two scenarios for the adversary based on the adversary's knowledge. "All" means the adversary not only knows the EdgePro, but also knows which layer the authorization neurons are in. "Half" means the adversary knows the EdgePro method, but does not know the location of the authorization neurons. We assume that in both scenarios, the adversary knows the scale factors γ of EdgePro. 6) For authorized models, the attacker performs model extraction based on MAZE [56]. Then, we implement two protection strategies, including only EdgePro protection, and both EdgePro and GONE [58] (a protection method by adding noise to the output layer). The classification accuracy of the extracted model is recorded as acc_{nl}^{ext}, as shown in Table VIII.

Results and Analysis: We conclude that EdgePro effectively protects deep models against existing reverse engineering attacks [56], [57]. For instance, in Table VI, the functions of the target model are completely extracted, i.e., the extracted model realizes 100% classification accuracy. When the target model is protected by EdgePro, the model extracted by the attacker is invalid, i.e., the classification result is close to random guessing (acc_{ext} <10%). Therefore, instead of directly using well-known model extraction attacks, we designed reverse engineering specifically for EdgePro to infer passwords.

Table VII shows the password inference results and attack time cost in each scenario. When the number of authorization neurons is one, that is approximately 0.5% of the total neurons, the adversary may crack EdgePro. The adversary just needs to find the neuron that has a similar effect as the authorization neuron rather than infer the exact authorization neuron, e.g., in Table VII acc_{nu}^{re} reaches 97.90% when "passwords" are "4:1:0.5". When we increase the number of authorization neurons to two or three, the time cost increases drastically and even exceeds the set time (172,800 s). This reflects that the robustness of EdgePro increases significantly as the number of neurons increases, e.g., when the number of authorized neurons goes from 1 to 2, the average time of reverse engineering attacks increases by 45.6 times.

When the attacker reverses part of the authorization neurons, it still cannot destroy the protection ability of EdgePro. For instance, in Table VII, when the number of authorization neurons reaches three, reverse engineering requires a lot of time for inference. Under time constraints, "AZ neurons" and "RE neurons" are more different as the number of authorization neurons increases, which leads to invalid authorization using the reverse-derived password. For a model with a large number of authorization neurons, cracking EdgePro costs far more than training a model from scratch. In addition, in the real scene, the adversary also needs to reverse the scale factor γ of each layer, which will further increase the cost of cracking EdgePro. Compare with both scenarios, "All" and "Half", EdgePro always protects the model.

For authorized models, increasing the number of authorized neurons cannot perform effective protection. Therefore, we plan to add noise to the output layer for stronger protection. Table VIII shows the protection results of combining EdgePro and GONE [58]. Their protective abilities stack and do not conflict. For instance, when the number of authorized neurons is three, EdgePro effectively prevents model extraction when the model is unauthorized (acc_{nu}^{re} ≈10%). When the model is authorized, GONE plays a protective role (acc_{nl}^{ext}=11.85%).

3) Model Pruning Attack: We evaluate the robustness of EdgePro under model pruning attack [59] which is specially designed for EdgePro.

Implementation Details: 1) We consider three basic metrics for model pruning: Average Activation (AvgAct), Grad-CAM [60], and Layer-wise Relevance Propagation (LRP) [61]. Those metrics can measure the importance of neurons. 2) We use 10% of the test data to stimulate the EdgePro-trained model (where ρ=5, $\mathcal{V}$=(0, 1)) and record the three metrics of each neuron in the model and order by metrics. Then we iteratively prune the neurons of the model according to the ascending order. Finally, we feed the remaining data to the model to compute acc_{nu}. 3) In Table IX, we adapt different pruning rates, where "acc_{nu}-P10%" represents the acc_{nu} after 10% of the neurons are pruned. We use acc_{nu} as a measure of robustness. The lower

TABLE VII
EVALUATING THE ROBUSTNESS OF EDGEPRO UNDER REVERSE ENGINEERING ATTACK BY acc_{nu}^{re}, AND TIME

Datasets & Models	AZ neurons	acc_{nu}	acc_{nl}	Knowledge	RE neurons	acc_{nu}^{re}	Time(s)
MNIST & LeNet-5	4:29:0.7	10.70%	99.18%	All	4:1:0.5	97.90%	16.32
				Half	4:2:0.6	98.10%	28.50
	5:76:0.2	10.42%	99.25%	All	5:76:0.1	98.63%	1,053.64
				Half	5:76:0.1	98.39%	1,666.22
	4:29:0.7+5:76:0.2	10.20%	99.48%	All	4:29:0.7+5:76:0.7	96.25%	48,065.76
				Half	4:29:0.7+5:76:0.7	96.20%	80,748.32
	4:29:0.7+5:76:0.2+5:38:0.5	10.40%	99.15%	All	4:29:0.6+5:76:0.3+5:38:0.3	11.45%	172,800.00
				Half	4:29:0.5+5:76:0.5+5:31:0.5	10.60%	172,800.00

"AZ neurons" means the "passwords" and "4:29:0.7" refers to that EdgePro selects the 29th neuron in the 4th layer as the authorization neuron, and the locking value size is V = 0.7. "RE Neurons" refers to the adversary finding the neurons that can crack the edgepro.

TABLE VIII
EXPERIMENTAL RESULTS OF IMPLEMENTING DIFFERENT DEFENSE STRATEGIES FOR THE LENET-5 MODEL ON THE MNIST DATASET

Datasets & Models	AZ neurons	Protection	acc_{nu}^{re}	acc_{nl}^{ext}
MNIST & LeNet-5	4:29:0.7+5:76:0.2	EdgePro	96.25%	94.35%
		EdgePro+GONE	96.65%	**12.60%**
	4:29:0.7+5:76:0.2 +5:38:0.5	EdgePro	**11.45%**	96.10%
		EdgePro+GONE	**11.90%**	**11.85%**

the acc_{nu} of the pruned model, the more robust the EdgePro is. Note that an excessively high pruning rate will also cause a drop in accuracy, leading to the false impression of EdgePro protection. Therefore, we perform fine-tuning after pruning based on previous work [59].

Results and Analysis: Table IX shows the results of EdgePro on four datasets. Generally, the pruning rate does not exceed 20%. In this case, the acc_{nu}-P are all very low in Table IX, which indicates that the pruning can not crack EdgePro. For instance, when the pruning rate is less than 20%, the average acc_{nu} value is 9.85% on all datasets, which is about $\frac{1}{8}$ of acc_{nl}, making the model unusable. As the pruning rate increases, EdgePro still effectively performs property protection on CIFAR-10, CIFAR-100, and Tiny-ImageNet datasets. For instance, when the pruning rate is 100%, the average acc_{nu} value is 13.12% on the three datasets, which is about $\frac{1}{6}$ of acc_{nl}. This shows that the authorization neurons in the model are concealed. The adversary cannot detect authorization neurons by stimulating neurons, and also cannot obtain the property to use the model by pruning the authorization neurons. However, on the MNIST dataset, as the pruning rate increases, the acc_{nu} value also increases. We speculate that because the LeNet-5 model has few weights, the fine-tuning operation after pruning is close to retraining a new model. Similar to the experimental results of model fine-tuning attack, EdgePro shows stronger robustness on complex datasets (such as CIFAR-10, CIFAR-100, and Tiny-ImageNet) and large models (such as ResNet-18, VGG-19, and DenseNet-121). This is because large models have more authorization neurons. Comparing the three pruning metrics, none of them can break EdgePro. This reflects our selection of authorization neurons is sufficient, EdgePro is resistant to model pruning attacks.

E. EdgePro Case Study on Graph Dataset

The deep learning models deployed on edge devices are diverse, and there are not only for image tasks. In this section, we discuss the generality of EdgePro, e.g., whether EdgePro has a protection effect on node classification tasks. Since graph-level anomaly detection has been a promising means in many different domains [62], [63], such as transportation, energy, and factory, it is necessary to protect graph neural network (GNN) [64], [65] on edge devices.

Implementation Details: Specifically, we select a small graph dataset Cora and a large graph dataset PubMed.

1) *Cora* [66][5] consists of 2,708 scientific publications classified into one of seven classes. The citation network consists of 5,429 links. Each publication in the dataset is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary. The dictionary consists of 1,433 unique words.
2) *PubMed* [67][6] consists of 19,717 scientific publications from PubMed database pertaining to diabetes classified into one of three classes. The citation network consists of 44,338 links. Each publication in the dataset is described by a weighted word vector from a dictionary which consists of 500 unique words.

For each dataset, we use three models, GCN [68], SGC [65] and GAT [69] to train. The learning rates are all 0.01, and a total of 100 epochs are trained. For GCN model, hyperparameters are chosen as follows: 0.5 (dropout rate of first and last layer), 5×10^{-4} (L2 regularization at first layer) and 128 (number of units for each hidden layer). The SGC model and the GCN model keep the same hyperparameter settings. For a two-layer GAT model, the first layer consists of 8 attention heads computing 8 features each. The second layer is used for classification: a single attention head that computes 128 features, followed by ReLU activation. For the "passwords" in GNN, we randomly select 10% of neurons as authorization neurons, set the locking value to any number between 0 and 1, and set the scale factor to any number between 0.5 and 2.

Results and Analysis: Table X shows the results on two datasets. As we expected, EdgePro also has a protection effect on graph datasets, and the accuracy of the model when unauthorized tends to be close to random guessing. For example, on the Cora dataset of GAT model, the acc_{nl} is 5.7 times larger than acc_{nu}. This shows that EdgePro is general and competent for different task scenarios.

[5]Cora can be downloaded at *http://www.cs.umd.edu/ sen/lbc-proj/LBC.html*
[6]PubMed can be downloaded at *https://linqs-data.soe.ucsc.edu/public/Pubmed-Diabetes.tgz*

TABLE IX
EVALUATING THE ROBUSTNESS OF EDGEPRO AGAINST MODEL PRUNING ATTACK UNDER DIFFERENT PRUNING RATES, MEASURED BY acc_{nu}, WHERE acc_{nu}-P10% REPRESENTS THE acc_{nu} AFTER PRUNING 10% NEURONS

Datasets & Models	Metrics	acc_{nu}	acc_{nl}	acc_{nu}-P10%	acc_{nu}-P20%	acc_{nu}-P30%	acc_{nu}-P40%	acc_{nu}-P50%	acc_{nu}-P60%	acc_{nu}-P70%	acc_{nu}-P80%	acc_{nu}-P90%	acc_{nu}-P100%
MNIST & LeNet-5	AvgAct	11.10%	97.90%	13.64%	23.50%	43.55%	56.21%	60.43%	62.34%	88.52%	97.27%	97.27%	97.89%
	GradCAM			9.65%	10.53%	13.44%	15.22%	35.69%	46.27%	82.45%	95.82%	97.40%	97.72%
	LRP			11.25%	15.66%	22.43%	38.54%	46.95%	56.05%	77.14%	93.41%	95.51%	97.93%
CIFAR-10 & ResNet-18	AvgAct	9.50%	87.61%	16.32%	22.03%	22.14%	22.14%	22.36%	22.53%	23.12%	24.50%	24.73%	26.44%
	GradCAM			16.10%	21.60%	21.60%	21.35%	21.62%	20.00%	22.17%	23.73%	24.12%	26.87%
	LRP			13.24%	14.11%	15.35%	16.48%	17.76%	18.17%	19.38%	22.12%	24.53%	26.69%
CIFAR-100 & VGG-19	AvgAct	1.00%	73.77%	2.44%	4.97%	4.15%	3.82%	4.61%	2.63%	3.49%	4.41%	5.03%	4.95%
	GradCAM			1.35%	1.36%	1.36%	1.39%	1.44%	1.47%	2.15%	2.96%	3.84%	4.93%
	LRP			4.66%	6.73%	5.47%	5.89%	5.89%	5.98%	5.46%	6.12%	6.21%	4.89%
Tiny-ImageNet & DenseNet-121	AvgAct	0.35%	55.10%	3.22%	6.45%	5.36%	5.27%	6.28%	2.50%	4.61%	7.53%	7.55%	7.72%
	GradCAM			2.63%	3.62%	3.56%	3.74%	3.79%	3.80%	4.01%	4.98%	5.54%	7.81%
	LRP			3.25%	8.00%	7.31%	7.67%	7.71%	6.80%	7.69%	7.72%	7.75%	7.82%

TABLE X
EVALUATING THE EDGEPRO ON TWO GRAPH DATASETS: CORA AND PUBMED

Cora			PubMed		
Model	Normal	EdgePro	Model	Normal	EdgePro
GCN	83.40%	83.00% (14.40%)	GCN	84.20%	84.00% (29.80%)
SGC	80.60%	80.00% (15.00%)	SGC	83.60%	83.20% (34.60%)
GAT	89.00%	89.00% (14.80%)	GAT	83.20%	80.40% (32.40%)

The values without and with brackets in the table represent acc_{nl} and acc_{nu}.

F. Effect of Authorization Neuron Selection on EdgePro

EdgePro guarantees unpredictability by randomly selecting authorization neurons. In this section, we discuss the effect of different selection strategies rather than random selection on EdgePro.

Implementation Details: 1) According to the works [70], [71], we use several indicators to describe the importance of a neuron, including its activation value, activation frequency, weight value, Grad-CAM [60], and LRP [61]. 2) We use 10% of the test data to stimulate the neurons in the pre-trained model, and then rank the neurons according to different indicators. Our intuition is that the change of low importance neurons has less impact on the performance of the model, and is more suitable to be selected as authorization neurons. Therefore, after ranking the neurons, we give these neurons corresponding weights, and perform selection according to their weights. The neurons with lower importance will be given greater weight, which means that the probability of being selected is higher. 3) We take **r**andom **n**euron **r**anking (**RNR**) selection as the baseline, and design five neuron ranking selection strategies. The training parameters for each strategy are the same as in Section V-B. The details of five strategies are as follows:

- *Activation Value Ranking (AVR):* AVR counts the cumulative activation value of each neuron using a batch of data, and ranks the neurons by the cumulative activation values.
- *Activation Frequency Ranking (AFR):* AFR counts the times that neurons are activated (activation value $\alpha > 0$) using a batch of data as input, and ranks the neurons according to the number of times.
- *Weight Value Ranking (WVR):* In each layer, WVR counts the cumulative weight values of each neuron connection, and then sorts the cumulative weight values in descending order as the neuron importance ranking.
- *Grad-CAM Ranking (GCR):* We design the Grad-CAM ranking (GCR), using the Grad-CAM technique to rank neurons in the convolutional layers. Grad-CAM does not rank the neurons in linear layers. Therefore we rank them in a random manner.
- *Layer-wise relevance propagation Ranking (LRPR):* LRPR distributes the output correlation backward through the pre-trained model, and determines the contribution ranking of neurons to classification.

Results and Analysis: Table XI shows the results of different ranking strategies. Comparing the six ranking strategies, they all show good model protection with slight differences, for example, on the CIFAR-100 dataset, acc_{nl} average reaches 72.99% ($\pm$0.45%) between different strategies. All six different strategies have the effect of protecting models. This means that the selection of authorization neurons does not depend on specific selection strategies, because the model can adapt to a small number of authorization neurons through training. Therefore, it is appropriate to randomly select authorization neurons, which can fully ensure the unpredictability of EdgePro.

G. Analysis of Parameter Sensitivity

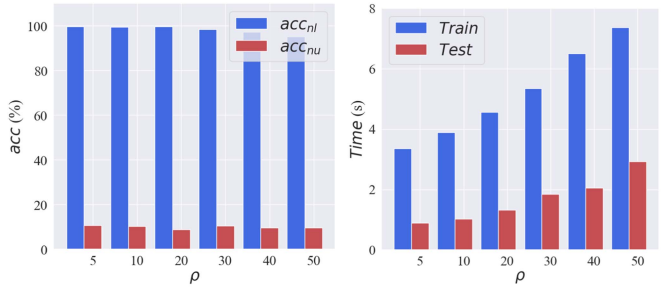
1) The Effect of Authorization Neuron Ratio: We provide an analysis of the effect about authorization neuron numbers by comparing model accuracy (both acc_{nl} and acc_{nu}) and training time cost.

Implementation Details: 1) We conduct experiments on the LeNet-5 model using 50,000 training data and 10,000 test data on the MNIST dataset and the ResNet-18 model using 50,000 training data and 10,000 test data on the CIFAR-10 dataset. 2) We divide 6 different values: ρ=$\{5, 10, 20, 30, 40, 50\}$. We record the acc_{nl} and acc_{nu} of EdgePro after running 20 epochs and compute the average training time in 1 epoch. Finally, we record the testing time.

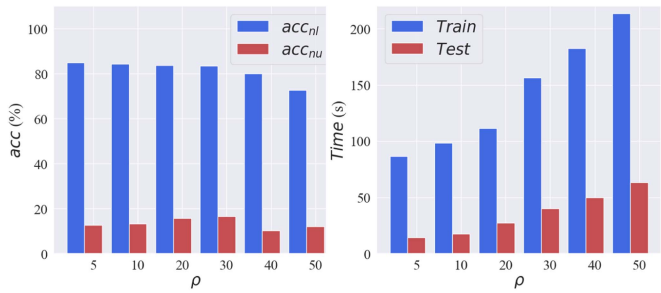
TABLE XI
EVALUATION RESULTS OF EDGEPRO ON DIFFERENT RANKING STRATEGIES, I.E., RNR, AVR, AFR, WVR, GCR, LRPR

Dataset	Model	Method					
		RNR	AVR	AFR	WVR	GCR	LRPR
MNIST	LeNet-5	99.29% (10.01%)	99.60% (10.70%)	98.77% (9.80%)	99.64% (10.51%)	98.91% (11.30%)	99.11% (12.16%)
CIFAR-10	ResNet-18	87.10% (11.71%)	87.01% (8.20%)	87.44% (10.31%)	86.97% (11.47%)	86.72% (9.22%)	87.02% (9.27%)
CIFAR-100	VGG-19	73.52% (1.00%)	73.44% (0.69%)	72.81% (1.27%)	73.03% (0.93%)	72.49% (1.33%)	72.62% (1.03%)
Tiny-ImageNet	DenseNet-121	55.10% (0.35%)	55.12% (0.41%)	54.85% (0.40%)	55.23% (0.38%)	55.07% (0.50%)	54.59% (0.38%)

The values without and with brackets in the table represent acc_{nl} and acc_{nu}.

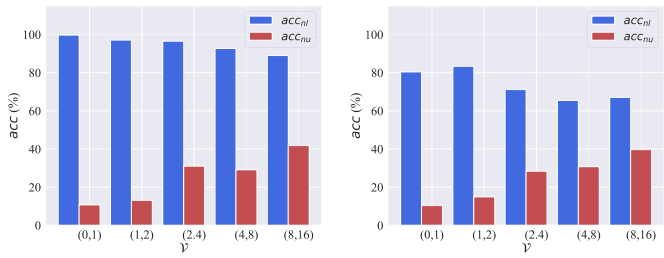


(a) MNIST



(b) CIFAR-10

Fig. 4. Results of accuracy and time cost under different numbers of authorization neurons on MNIST and CIFAR-10.



(a) MNIST (b) CIFAR-10

Fig. 5. Results of accuracy under different locking value sizes on LeNet-5 of MNIST and ResNet-18 of CIFAR-10.

Results and Analysis: Fig. 4 shows the results. As the authorization neuron ratio ρ increases, the training cost of EdgePro also increases. For instance, in Fig. 4(b) compared with $\rho = 10$, when $\rho = 50$, not only does acc_{nl} drop by 13.8%, but the training time per epoch is also increased by 3.6 times. And in simple models, such as LeNet-5, the impact of ρ on the accuracy is not obvious. In fact, $\rho = 5$ is also sufficient for the VGG-19 model with 50,782 neurons. Therefore, it does not require many authorization neurons to achieve the protection purpose. However, in consideration of the robustness against the reverse engineering attack, the neurons should not be too few, e.g., 1 or 2 neurons are shown to be vulnerable in Section V-D.

Takeaways. It would be beneficial to enhance the model encryption while ensuring we do not compromise on time efficiency. Thus, we recommend 10% of the neurons for models in comprehensive consideration of effectiveness, efficiency, and robustness.

2) The Effect of Locking Value: We provide an analysis of the effect of locking value range $\mathcal{V}$ and perform the experiments on the LeNet-5 of MNIST and the ResNet-18 of CIFAR-10.

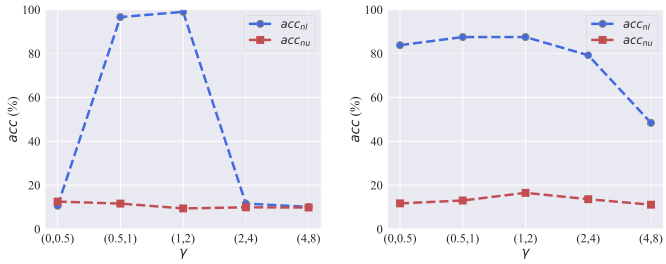
Implementation Details: 1) We divide five locking value ranges: $\mathcal{V}$=$\{(0,1),(1,2),(2,4),(4,8),(8,16)\}$. In each experiment, lock values will be randomly selected from the range. 2) We record the acc_{nl} and acc_{nu} of EdgePro after running 20 epochs.

Results and Analysis: Fig. 5 shows the acc_{nl} and acc_{nu} of EdgePro-trained model under different locking value ranges $\mathcal{V}$. As $\mathcal{V}$ increases, the acc_{nl} values demonstrate a decreasing trend. When $\mathcal{V} = (8,16)$, acc_{nl} is reduced by 16.3% compared with $\mathcal{V} = (0,1)$. Since EdgePro randomly selects authorization neurons, it may select some neurons that are irrelevant to the classification task. When the lock values of these irrelevant neurons are too large, the classification accuracy of the model will be affected, resulting in the reduction of acc_{nl}. Meanwhile, too large locking values will also lead to excessive learning of the characteristics of authorization neurons during model training, resulting in the easy discovery of authorization neurons.

Takeaways. We consider $\mathcal{V}$=(0, 1) or $\mathcal{V}$=(1, 2) to be appropriate, at this time authorization neurons not only has little impact on model performance, but also are concealment.

3) The Effect of Scale Factor: We also provide an analysis of the effect of scale factors γ by comparing the acc_{nl} and acc_{nu}.

Implementation Details: 1) We choose MNIST with LeNet-5 and CIFAR-10 with ResNet-18 for experiments. In view of the complex structure of ResNet, we multiply the output of each block by the scale factors instead of the output of each layer. 2) We divide five scale factor ranges of different sizes:



(a) MNIST (b) CIFAR-10

Fig. 6. Results of accuracy under different scale factors on LeNet-5 of MNIST and ResNet-18 of CIFAR-10.

TABLE XII
RESULTS OF SETTING THE SCALE FACTOR IN DIFFERENT POSITIONS OF RESNET-18 MODEL ON CIFAR-10 DATASET, MEASURED BY acc_{nu} AND acc_{nl}

Datasets & Models	Position of scale factor settings							
	none		output layer		random block		each block	
	acc	acc	acc	acc	acc	acc	acc	acc
CIFAR-10 & ResNet-18	13.63%	86.40%	14.53%	87.36%	12.57%	86.91%	12.61%	78.43%

$\gamma \in \{(0, 0.5), (0.5, 1), (1, 2), (2, 4), (4, 8)\}$. As with the procedure for determining the locking value, we will also randomly select the scale factors from a range before starting the lock training. 3) To verify the necessity of the scale factor, we record the results of setting scale factors at different positions, including four aspects, i.e., not setting the scale factor, only setting the scale factor in the output layer, setting the scale factor for each block, setting the scale factor to a randomly selected block.

Results and Analysis: Fig. 6 shows the acc_{nl} and acc_{nu} of EdgePro. It can be observed that inappropriate scale factors will affect the performance of the model. For instance, on the MNIST dataset, both $\gamma \in$(0, 0.5) and $\gamma \in$(2, 4) will make the model unusable. They scale the activation of neurons to an abnormal degree, resulting in the model can not be trained. Additionally, compared with LeNet-5 scaling all layer outputs, only scaling block outputs in ResNet-18 will increase the tolerance of the model for scale factors. This reflects that EdgePro does not need to add scale factors to all layers of the model. It is sufficient to select some layers to add scale factors in a complex model. Adding scale factors in too many layers will affect the performance of the model. Therefore, for complex models, e.g., ResNet and DenseNet based models, EdgePro can add scale factors after each block instead of each layer.

Additionally, EdgePro's protection is not sensitive to the position of the scale factor setting. Even if the scale factor is not set, EdgePro still performs protection. For instance, in Table XII, after setting the scale factor at different positions, the mean value of acc_{nu} is 13.24%, and its variance is 1.25E-04. When the scale factor is none, the acc_{nu} value is 13.63%. We speculate that protection without the scale factor is equivalent to that with a scaling factor of 1.

Takeaways. We consider scale factor $\gamma \in (0.5, 2)$ to be appropriate. Because our observation from Fig. 6 indicates that accuracy results are optimized when within the appropriate range (i.e., $\gamma \in (0.5, 2)$).

VI. FUTURE WORKS AND DISCUSSIONS

In this section, we will discuss the future work of EdgePro. Theoretically, as long as there are neurons in the model, EdgePro ensures the security of the model through authorization at the neuron level. In the experiments, we have explored and proved the effectiveness of EdgePro in image classification and node classification tasks. In the future, it would be interesting to extend EdgePro to more tasks, e.g., semantic segmentation, object detection/tracking, and text classification tasks. In addition, generative adversarial networks also is one of our objectives.

EdgePro provides a light-weight encryption method for deep models, but how to effectively protect the "password" is still a security issue. Once the "password" is obtained by an attacker, EdgePro will lose its ability to protect deep models. Thus, we consider using classical password encryption algorithms [72] to convert these plaintext passwords into ciphertext passwords in the future. Only when the model processes the input data, the ciphertext is temporarily decrypted to plaintext to ensure that the model works properly. If an attacker obtains the ciphertext passwords, it will not compromise EdgePro's protection ability.

VII. CONCLUSION

We introduce a new protection method to the models on the edge devices, which is named EdgePro. Different from the existing methods, we embed the specific markers into part of the model neurons for the purpose of being light-weight. Only when the specific authorization neurons are locked to the locking values, the EdgePro-trained model can work correctly. EdgePro replaces the encryption and storage of the entire model with the information of authorization neurons. Our experiments show that the EdgePro is effective, light-weight, and robust against adaptive attacks including fine-tuning, reverse engineering and model pruning. Note that EdgePro can protect not only deep models running on edge devices, but also deep models on cloud servers. However, one of the main contributions of EdgePro is to protect deep models under limited computing resources and time costs. Thus, we mainly consider edge computing scenarios.

REFERENCES

[1] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, "Industrial Internet of Things: Challenges, opportunities, and directions," *IEEE Trans. Ind. Inform.*, vol. 14, no. 11, pp. 4724–4734, Nov. 2018.
[2] A. Gulati, G. S. Aujla, R. Chaudhary, N. Kumar, M. S. Obaidat, and A. Benslimane, "DiLse: Lattice-based secure and dependable data dissemination scheme for social Internet of Vehicles," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 6, pp. 2520–2534, Nov./Dec. 2021.
[3] R. A. Khalil, N. Saeed, M. Masood, Y. M. Fard, M.-S. Alouini, and T. Y. Al-Naffouri, "Deep learning in the Industrial Internet of Things: Potentials, challenges, and emerging applications," *IEEE Internet Things J.*, vol. 8, no. 14, pp. 11016–11040, Jul. 2021.
[4] M. Chehab and A. Mourad, "LP-SBA-XACML: Lightweight semantics based scheme enabling intelligent behavior-aware privacy for IoT," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 1, pp. 161–175, Jan./Feb. 2022.
[5] T. Chen et al., "DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," *ACM SIGARCH Comput. Archit. News*, vol. 42, no. 1, pp. 269–284, 2014.
[6] W. Hua, Z. Zhang, and G. E. Suh, "Reverse engineering convolutional neural networks through side-channel information leaks," in *Proc. 55th ACM/ESDA/IEEE Des. Automat. Conf.*, 2018, pp. 1–6.

[7] F. Mo et al., "DarkneTZ: Towards model privacy at the edge using trusted execution environments," in *Proc. 18th Int. Conf. Mobile Syst. Appl. Serv.*, 2020, pp. 161–174.

[8] J. Jang et al., "PrivateZone: Providing a private execution environment using ARM trustzone," *IEEE Trans. Dependable Secure Comput.*, vol. 15, no. 5, pp. 797–810, Sep./Oct. 2018.

[9] T. Nakai, D. Suzuki, and T. Fujino, "Towards trained model confidentiality and integrity using trusted execution environments," in *Proc. Int. Conf. Appl. Cryptogr. Netw. Secur.*, 2021, pp. 151–168.

[10] D. Evtyushkin, J. Elwell, M. Ozsoy, D. Ponomarev, N. A. Ghazaleh, and R. Riley, "Flexible hardware-managed isolated execution: Architecture, software support and applications," *IEEE Trans. Dependable Secure Comput.*, vol. 15, no. 3, pp. 437–451, May/Jun. 2018.

[11] S. Huang, H. Jiang, X. Peng, W. Li, and S. Yu, "Secure XOR-CIM engine: Compute-in-memory SRAM architecture with embedded XOR encryption," *IEEE Trans. Very Large Scale Integration Syst.*, vol. 29, no. 12, pp. 2027–2039, Dec. 2021.

[12] P. Zuo, Y. Hua, L. Liang, X. Xie, X. Hu, and Y. Xie, "Sealing neural network models in secure deep learning accelerators," 2020, *arXiv: 2008.03752*.

[13] R. Xu, J. Joshi, and C. Li, "NN-EMD: Efficiently training neural networks using encrypted multi-sourced datasets," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 4, pp. 2807–2820, Jul./Aug. 2022.

[14] D. Fiore, A. Mitrokotsa, L. Nizzardo, and E. Pagnin, "Multi-key homomorphic authenticators," *IET Inf. Secur.*, vol. 13, no. 6, pp. 618–638, 2019.

[15] B. D. Rouhani, M. S. Riazi, and F. Koushanfar, "DeepSecure: Scalable provably-secure deep learning," in *Proc. 55th Annu. Des. Automat. Conf.*, 2018, pp. 1–6.

[16] M. Ball, B. Carmer, T. Malkin, M. Rosulek, and N. Schimanski, "Garbled neural networks are practical.," *IACR Cryptol. ePrint Arch.*, vol. 2019, 2019, Art. no. 338.

[17] W. Trappe, R. Howard, and R. S. Moore, "Low-energy security: Limits and opportunities in the Internet of Things," *IEEE Secur. Privacy*, vol. 13, no. 1, pp. 14–21, Jan./Feb. 2015.

[18] J. Bryzek, "Roadmap for the trillion sensor universe," in *Proc. iNEMI Spring Member Meeting Webinar*, 2013, pp. 1–5.

[19] U. Guin, A. Singh, M. Alam, J. Canedo, and A. Skjellum, "A secure low-cost edge device authentication scheme for the Internet of Things," in *Proc. IEEE 31st Int. Conf. VLSI Des. 17th Int. Conf. Embedded Syst.*, 2018, pp. 85–90.

[20] M. Tehranipoor and C. Wang, *Introduction to Hardware Security and Trust*. Berlin, Germany: Springer, 2011.

[21] V. Costan and S. Devadas, "Intel SGX explained," *IACR Cryptol. ePrint Arch.*, vol. 2016, no. 86, pp. 1–118, 2016.

[22] A. Gangal, M. Ye, and S. Wei, "HybridTEE: Secure mobile DNN execution using hybrid trusted execution environment," in *Proc. IEEE Asian Hardware Oriented Secur. Trust Symp.*, 2020, pp. 1–6.

[23] S. Huang, X. Peng, H. Jiang, Y. Luo, and S. Yu, "New security challenges on machine learning inference engine: Chip cloning and model reverse engineering," 2020, *arXiv: 2003.09739*.

[24] F. Regazzoni et al., "Machine learning and hardware security: Challenges and opportunities-invited talk," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Des.*, 2020, pp. 1–6.

[25] M. Isakov, L. Bu, H. Cheng, and M. A. Kinsy, "Preventing neural network model exfiltration in machine learning hardware accelerators," in *Proc. IEEE Asian Hardware Oriented Secur. Trust Symp.*, 2018, pp. 62–67.

[26] W. Li, Y. Wang, H. Li, and X. Li, "P3M: A pim-based neural network model protection scheme for deep learning accelerator," in *Proc. 24th Asia South Pacific Des. Automat. Conf.*, 2019, pp. 633–638.

[27] A. Chakraborty, A. Mondai, and A. Srivastava, "Hardware-assisted intellectual property protection of deep learning models," in *Proc. 57th ACM/IEEE Des. Automat. Conf.*, 2020, pp. 1–6.

[28] B. F. Goldstein, V. C. Patil, V. C. Ferreira, A. S. Nery, F. M. Franca, and S. Kundu, "Preventing DNN model IP theft via hardware obfuscation," *IEEE Trans. Emerg. Sel. Topics Circuits Syst.*, vol. 11, no. 2, pp. 267–277, Jun. 2021.

[29] H. Hashemi, Y. Wang, and M. Annavaram, "DarKnight: An accelerated framework for privacy and integrity preserving deep learning using trusted hardware," in *Proc. IEEE/ACM 54th Annu. Int. Symp. Microarchitecture*, 2021, pp. 212–224.

[30] R. Tang, M. Du, and X. Hu, "Deep serial number: Computational watermarking for DNN intellectual property protection," 2020, *arXiv: 2011.08960*.

[31] M. Chen and M. Wu, "Protect your deep neural networks from piracy," in *Proc. IEEE Int. Workshop Inf. Forensics Secur.*, 2018, pp. 1–7.

[32] L. Fan, K. W. Ng, and C. S. Chan, "Rethinking deep neural network ownership verification: Embedding passports to defeat ambiguity attacks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 4716–4725.

[33] J. Zhang, D. Chen, J. Liao, W. Zhang, G. Hua, and N. Yu, "Passport-aware normalization for deep model protection," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 22619–22628.

[34] M. Alam, S. Saha, D. Mukhopadhyay, and S. Kundu, "Deep-lock: Secure authorization for deep neural networks," 2020, *arXiv: 2008.05966*.

[35] R. Lu, "A new communication-efficient privacy-preserving range query scheme in fog-enhanced IoT," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 2497–2505, Apr. 2019.

[36] H.-C. Shin et al., "Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset characteristics and transfer learning," *IEEE Trans. Med. Imag.*, vol. 35, no. 5, pp. 1285–1298, May 2016.

[37] N. Pittaras, F. Markatopoulou, V. Mezaris, and I. Patras, "Comparison of fine-tuning and extension strategies for deep convolutional neural networks," in *Proc. Int. Conf. Multimedia Model.*, 2017, pp. 102–114.

[38] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" 2014, *arXiv:1411.1792*.

[39] K. Liu, B. Dolan-Gavitt, and S. Garg, "Fine-pruning: Defending against backdooring attacks on deep neural networks," in *Proc. Int. Symp. Res. Attacks Intrusions Defenses*, 2018, pp. 273–294.

[40] S. J. Oh, B. Schiele, and M. Fritz, "Towards reverse-engineering black-box neural networks," in *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*, vol. 11700. Berlin, Germany: Springer, 2019, pp. 121–144.

[41] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," 2015, *arXiv:1506.02626*.

[42] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," 2016, *arXiv:1611.06440*.

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

[44] A. Prakash, J. Storer, D. Florencio, and C. Zhang, "RePr: Improved training of convolutional filters," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 10666–10675.

[45] S. Han et al., "DSD: Dense-sparse-dense training for deep neural networks," 2016, *arXiv:1607.04381*.

[46] S. P. Baller, A. Jindal, M. Chadha, and M. Gerndt, "Deepedgebench: Benchmarking deep neural networks on edge devices," in *Proc. IEEE Int. Conf. Cloud Eng.*, San Francisco, CA, USA, 2021. pp. 20–30.

[47] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[48] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[49] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2009, pp. 248–255.

[50] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 630–645.

[51] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.

[52] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 4700–4708.

[53] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 7132–7141.

[54] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 6848–6856.

[55] W. Sirichotedumrong, T. Maekawa, Y. Kinoshita, and H. Kiya, "Privacy-preserving deep neural networks with pixel-based image encryption considering data augmentation in the encrypted domain," in *Proc. IEEE Int. Conf. Image Process.*, 2019, pp. 674–678.

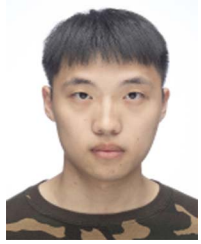
[56] S. Kariyappa, A. Prakash, and M. K. Qureshi, "MAZE: Data-free model stealing attack using zeroth-order gradient estimation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 13814–13823.

[57] S. Sanyal, S. Addepalli, and R. V. Babu, "Towards data-free model stealing in a hard label setting," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, New Orleans, LA, USA, 2022, pp. 15263–15272.

[58] H. Zheng, J. Chen, W. Shangguan, Z. Ming, X. Yang, and Z. Yang, "GONE: A generic o(1) noise layer for protecting privacy of deep neural networks," *Comput. Secur.*, vol. 135, 2023, Art. no. 103471.
[59] V. Sanh, T. Wolf, and A. M. Rush, "Movement pruning: Adaptive sparsity by fine-tuning," in *Proc. Adv. Neural Inf. Process. Syst. 33: Annu. Conf. Neural Inf. Process. Syst.*, 2020, pp. 20378–20389.
[60] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-CAM: Visual explanations from deep networks via gradient-based localization," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 618–626.
[61] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek, "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation," *PLoS One*, vol. 10, no. 7, 2015, Art. no. e0130140.
[62] Y. Wu, H.-N. Dai, and H. Tang, "Graph neural networks for anomaly detection in Industrial Internet of Things," *IEEE Internet Things J.*, vol. 9, no. 12, pp. 9214–9231, Jun. 2022.
[63] D. Wu, C. Zhang, L. Ji, R. Ran, H. Wu, and Y. Xu, "Forest fire recognition based on feature extraction from multi-view images," *Traitement du Signal*, vol. 38, no. 3, pp. 775–783, 2021.
[64] L. Zhang, P. Liu, Y. Choi, and P. Chen, "Semantics-preserving reinforcement learning attack against graph neural networks for malware detection," *IEEE Trans. Dependable Secure Comput.*, vol. 20, no. 2, pp. 1390–1402, Mar./Apr. 2023.
[65] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, "Simplifying graph convolutional networks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 6861–6871.
[66] Z. Yang, W. Cohen, and R. Salakhudinov, "Revisiting semi-supervised learning with graph embeddings," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 40–48.
[67] G. Namata, B. London, L. Getoor, B. Huang, and U. Edu, "Query-driven active surveying for collective classification," in *Proc. 10th Int. Workshop Mining Learn. Graphs*, 2012, Art. no. 1.
[68] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2016, *arXiv:1609.02907*.
[69] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *Stat*, vol. 1050, 2017, Art. no. 20.
[70] Y. Zeng, W. Park, Z. M. Mao, and R. Jia, "Rethinking the backdoor attacks' triggers: A frequency perspective," 2021, *arXiv:2104.03413*.
[71] B. Wang et al., "Neural cleanse: Identifying and mitigating backdoor attacks in neural networks," in *Proc. IEEE Symp. Secur. Privacy*, 2019, pp. 707–723.
[72] Y. Liu et al., "Design of password encryption model based on AES algorithm," in *Proc. IEEE 1st Int. Conf. Civil Aviation Saf. Inf. Technol.*, 2019, pp. 385–389.

Jinyin Chen (Member, IEEE) received BS and PhD degrees from the Zhejiang University of Technology, Hangzhou, China, in 2004 and 2009, respectively. She studied evolutionary computing in Ashikaga Institute of Technology, Japan, in 2005 and 2006. She is currently a professor with the Zhejiang University of Technology. Her research interests include artificial intelligence security, graph data mining, and evolutionary computing.

Haibin Zheng received BS and PhD degrees from the Zhejiang University of Technology, Hangzhou, China, in 2017 and 2022, respectively. He is currently a university Lecturer with the Institute of Cyberspace Security, Zhejiang University of Technology. His research interests include deep learning and artificial intelligence security.

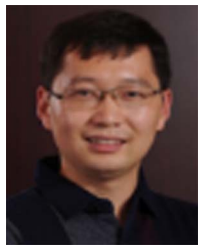


Tao Liu is currently working toward the master's degree with the college of Information engineering, Zhejiang University of Technology. His research interests include federated learning and its applications, and artificial intelligence.

Jiawei Liu is currently working toward the master's degree with the college of Information engineering, Zhejiang University of Technology. His research interests include federated learning and its applications, and artificial intelligence.

Yao Cheng received the PhD degree in computer science and technology from University of Chinese Academy of Sciences. She is currently a senior researcher with Huawei International, in Singapore. Her research interests include security and privacy in deep learning systems, blockchain technology applications, Android framework vulnerability analysis, mobile application security analysis, and mobile malware detection.

Xuhong Zhang received the BS degree in software engineering from the Harbin Institute of Technology, in 2011, the MS degree in computer science from Georgia State University, in 2013, and the PhD degree in computer engineering from the University of Central Florida, in 2017. He is an assistant professor of College of Control Science and Engineering with Zhejiang University. His research interests include distributed Big Data and AI systems, Big Data mining and analysis, data-driven security, AI, and Security.



Shouling Ji (Member, IEEE) received the BS (with Honors) and MS degrees both in computer science from Heilongjiang University, the PhD degree in electrical and computer engineering from the Georgia Institute of Technology, and the PhD degree in computer science from Georgia State University. He is a ZJU 100-Young Professor in the College of Computer Science and Technology with Zhejiang University and a Research Faculty in the School of Electrical and Computer Engineering with the Georgia Institute of Technology (Georgia Tech). His current research interests include Data-driven Security and Privacy, AI Security, and Big Data Analytics.