AdvParams: An Active DNN Intellectual Property Protection Technique via Adversarial Perturbation Based Parameter Encryption

MINGFU XUE, (Senior Member, IEEE), ZHIYU WU, YUSHU ZHANG, (Member, IEEE), JIAN WANG, AND WEIQIANG LIU, (Senior Member, IEEE)

Mingfu Xue, Yushu Zhang, and Jian Wang are with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China
Zhiyu Wu is with the Polytechnic Institute, Zhejiang University, Hangzhou 310058, China
Weiqiang Liu is with the College of Electronic and Information Engineering, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China

CORRESPONDING AUTHOR: MINGFU XUE (mingfu.xue@nuaa.edu.cn)

This work was supported by the National Natural Science Foundation of China under Grants 61602241 and 62022041, and in part by the CCF-NSFOCUS Kun-Peng Scientific Research Fund under Grant CCF-NSFOCUS 2021012.

ABSTRACT The construction of Deep Neural Network (DNN) models requires high cost, thus a well-trained DNN model can be considered as intellectual property (IP) of the model owner. To date, many DNN IP protection methods have been proposed, but most of them are watermarking based verification methods where model owners can only verify their ownership passively after the copyright of DNN models has been infringed. In this article, we propose an effective framework to actively protect the DNN IP from infringement. Specifically, we encrypt a small number of model’s parameters by perturbing them with well-crafted adversarial perturbations. With the encrypted parameters, the accuracy of the DNN model drops significantly, which can prevent malicious infringers from using the model. After the encryption, the positions of encrypted parameters and the values of the added adversarial perturbations form a secret key. Authorized user can use the secret key to decrypt the model on Machine Learning as a Service, while unauthorized user cannot use the model. Compared with the existing DNN watermarking methods which passively verify the ownership after the infringement occurs, the proposed method can prevent infringement in advance. Moreover, compared with few existing active DNN IP protection methods, the proposed method does not require additional training process of the model, thus introduces low computational overhead. Experimental results show that, after the encryption, the test accuracy of the model drops by 80.65%, 81.16%, and 87.91% on Fashion-MNIST (DenseNet), CIFAR-10 (ResNet), and GTSRB (AlexNet) datasets, respectively. Moreover, the proposed method only needs to encrypt an extremely low number of parameters. The proportion of the encrypted parameters in all the model’s parameters is as low as 0.000205%. Experimental results also indicate that, the proposed method is robust against model fine-tuning attack, model pruning attack, and the adaptive attack where attackers know the detailed steps of the proposed method and all the parameters of the encrypted model.

INDEX TERMS Artificial intelligence security, deep neural networks, intellectual property protection, active authorization control, adversarial perturbation

I. INTRODUCTION

Deep Neural Networks (DNNs) are extensively deployed in commercial applications. Many companies use their trained high-performance DNN models to provide services for the public, which is called Machine Learning as a Service (MLaaS) [1]. The users can only obtain the model’s predictions, but have no access to the internal parameters. Training a DNN model with high accuracy is a costly and time-consuming task. However, some malicious infringers may illegally duplicate or abuse the well-trained model, and obtain benefits from it, which greatly infringes the intellectual property (IP) of the model owner. IP protection for DNN models is an emerging problem.
Most of the existing DNN IP protection methods are watermarking based verification methods [2]–[5]. In these watermarking based verification methods, the DNN model is embedded with watermark, which can be extracted to verify the ownership of the model [6]. However, these DNN watermarking methods are passive verification methods, which can not prevent the unauthorized usage of the DNN models, and can only be applied after the models have already been pirated.

To date, a few active DNN IP protection methods [7]–[9] have been proposed. In these methods, only authorized users can use the DNN models normally, while the functionality of the models will be deteriorated seriously for unauthorized users. Pyone et al. [7] use the secret key to preprocess the training images, then they train the model with these preprocessed training images. The model only works when the input sample is preprocessed by using the secret key. Chen and Wu [8] utilize a data transformation module to implement the function of access control. However, the works [7] and [8] both require to train the model from scratch, which is a time-consuming process. Moreover, in the works [7] and [8], each input sample needs to be transformed before being input into the model, which introduces high computational overhead, especially when there are a substantial amount of input samples. Fan et al. [9] embed the passports to the DNN model so that the model’s performance will be deteriorated significantly if the passports are not presented.

The implementation of the work [9] is time-consuming since it requires to embed the passports into multiple layers of DNN model by retraining the model. Besides, attackers can compute the hidden parameters by conducting the reverse-engineering attack.

An intuition to realize the active authorization control is to make the model be dysfunctional by encrypting the model’s parameters. However, a trained model usually contains a tremendously large number of parameters, which makes the computational overhead of parameter encryption unacceptable and also substantially increases the decryption time. Thus, it is necessary to search for a few parameters that lead to the most significant performance deterioration of the model. In this paper, we propose an adversarial perturbation based method to actively protect the DNN IP with negligible computational overhead. We utilize the gradient of the model’s loss function to find the parameters that have the greatest impact on the model’s performance, then modify the selected parameters with adversarial perturbations. With an extremely low number of encrypted parameters, the model will be dysfunctional significantly. After the encryption process, a secret key will be generated. The encrypted model is deployed on MLaas (cloud platform) to provide prediction service. Only the authorized user can provide the secret key to the cloud platform and the model will be decrypted remotely, while without the secret key, the model will output wrong predictions. The threat model will be discussed in Section III-A.

The main contributions of this work are three-fold:

- This paper proposes an active DNN IP protection method based on adversarial perturbation, which can prevent the infringement of DNN IP in advance. In order to achieve active authorization control, the parameters that have the greatest impact on the performance of the model are selected by utilizing the gradient of the model’s loss function, and the adversarial perturbations will be added to these selected parameters. With an extremely low number of encrypted parameters, the model’s accuracy will be significantly decreased. After the encryption process, a secret key containing the positions of the encrypted parameters and the values of the added perturbations will be generated. With the secret key, the authorized user can decrypt the model in MLaas and obtain high accuracy.

- The comparison with related works are presented in Section II (discussion) and Section IV-F (experimental comparison). Compared with the existing DNN watermarking based verification methods, the proposed method can actively prevent unauthorized usage of the model, thus protects the DNN IP from infringement in advance. Besides, the proposed method does not require to retrain the model. Compared with few existing active DNN IP protection works, which require to retrain the model [7]–[10], or need to preprocess the input images before inputting them into the model [7], [8], or require the support of hardware devices [10], the proposed method is low-cost and more practical in the commercial applications. Moreover, the proposed method only needs to encrypt an extremely low number of parameters, which makes the change of the model’s parameters negligible, and significantly reduces the decryption time and the storage overhead of the secret key.

- Experimental results show that, the test accuracy of the encrypted model are only 10.36%, 10.86% and 6.94% on Fashion-MNIST [11] dataset (DenseNet [12] model), CIFAR-10 [13] dataset (ResNet [14] model), and GTSRB [15] dataset (AlexNet [16] model), respectively, which demonstrate the effectiveness of the proposed method. The number of the encrypted parameters is as low as 23 (Fashion-MNIST), while the proportion of the encrypted parameters is as low as 0.000205% of all the parameters of the model (GTSRB). Besides, for malicious attackers, it’s difficult to detect the encrypted parameters, as the encrypted weights are all in a normal range and the change of the distribution of the weights is negligible. Further, the proposed method is demonstrated to be robust against model fine-tuning attack, model pruning attack, and the strong adaptive attack where the attacker knows the detailed steps of the proposed method and all the parameters of the encrypted model.

This paper is organized as follows. The related works are reviewed in Section II. The proposed adversarial perturbation based method is elaborated in Section III. The experimental results are presented and analyzed in Section IV. This paper is concluded in Section V.
II. RELATED WORK

In this section, the DNN watermarking methods and the few active DNN IP protection methods are reviewed. The comparison between the proposed method and the related DNN IP protection works is also presented.

**DNN Watermarking Methods.** Uchida et al. [2] embed watermark into a DNN model by changing the internal parameters of the model, and the embedded watermark can be extracted to verify the ownership of the model. This method can only be applied in the while-box scenario, in which the model’s parameters are publicly accessible [2]. However, in the commercial applications, DNN models are deployed in black-box scenarios in most cases, where the verifier has no access to the internal parameters of the pirated model. Rouhani et al. [3] propose an IP protection framework which can work in black-box scenario. They embed the watermark into the output layer of DNN model, and select some input samples as the input key set. With the presence of the input key set, the watermark can be extracted.

There are some works [17, 18] utilizing DNN backdoor as the watermark to protect the IP of DNN model. Adi et al. [17] utilize the overparameterization of the DNN to embed backdoor into the model to verify the ownership of the model in the black-box scenario. Guo and Potkonjak [18] propose a backdoor-based watermarking method, which can work under black-box scenario. They use model owner’s signature (trigger) to generate backdoor instances and embed the backdoor as watermark into the model. The watermarked model will output specific labels for the images carrying the signature at inference stage, so as to verify the ownership of the model.

There are also some works [5, 19–21] utilizing adversarial examples to protect the IP of DNN model. Merrer et al. [5] use adversarial examples to fine-tune the model. The model will output specific predictions when the adversarial examples are presented. Thus, the owner can use adversarial examples to query the marked model and verify the ownership. Cao et al. [19] utilize the adversarial examples near the classification boundary as the fingerprint of the model. Since the classification boundary of a model is unique, the predictions of the model on these adversarial examples are unique and can be used to verify whether the suspicious model is a pirated model. Lukas et al. [20] extract conferrable adversarial examples from a DNN model as the fingerprint to defend against model stealing attack. The similar accuracy of the pirated model and the original model on these adversarial examples can be used to verify the ownership of the model. Zhao et al. [21] generate a set of adversarial examples as model fingerprints for IP protection. These model fingerprints can be used to verify the ownership of suspicious models remotely. Compared with these adversarial examples based DNN fingerprinting/watermarking methods [5, 19–21], the proposed method has the following advantages and differences. First, these methods [5, 19–21] are used to verify the ownership of model after infringement occurs (i.e., passive verification). On the contrary, the proposed method can actively prevent the model from unauthorized usage (i.e., active authorization control). Second, these methods [5, 19–21] directly generate adversarial examples from the model as the model fingerprint or watermark key set. In contrast, the proposed method searches and perturbs the most significant parameters of the model to make the model dysfunctional.

Except verifying the ownership of the model, there are few works aiming to authenticate the integrity of the model. Guan et al. [22] embed a reversible watermark into DNN model for integrity authentication. They utilize entropy-based pruning technique to construct the integer host sequence. Then, the integer host sequence that contains watermark information is embedded into the weights of the model. Any slight modification to the model will cause the extracted watermark information to change dramatically [22].

Although most existing works focus on protecting the classification models, there are also few watermarking schemes focusing on protecting the IP of the image processing models. Zhang et al. [23] embed the watermark into the output images of image processing model via spatial invisible watermarking method. If the attacker uses these output images to train a substitute model, the watermark can be extracted from the output images of the substitute model.

**Active DNN IP Protection Methods.** Chen and Wu [8] implement the function of access control by training an anti-piracy model, which will be dysfunctional for unauthorized users. With the transformation module, users can generate the authorized input so as to use the model normally. Pyone et al. [7] use a secret key to preprocess the training images, in which the pixels of these training images will be shuffled. The preprocessed images will be used to train the model, and the trained model only works when the pixels of the input images are shuffled with the secret key. Fan et al. [9] design some passports and embed them into the model. Without these passports, the model will output wrong predictions, thus prevents the unauthorized usage of the model. Lin et al. [24] change the positions of the model’s weights to make these weights chaotic, thus deteriorates the performance of the model. After purchasing the secret key, the authorized user can restore the original positions of the model’s parameters, thus makes the model work properly. Several hardware based DNN IP protection works have also been proposed. Chen et al. [25] propose an on-device IP protection method, named DeepAttest. The pre-defined fingerprint is encoded in the target DNN and later be extracted with the support of the hardware platform to verify the legitimacy of the DNN program. Only authorized DNN program is allowed to run on the target hardware device [25]. Chakraborty et al. [10] propose a key-dependent loss function to train an obfuscated model. Then, they store the key into a hardware device. Without the hardware device, the trained model can not work properly.

The comparison between the proposed method and the existing DNN IP protection works, including the DNN...
watermarking methods and the few active authorization control methods are discussed as follows.

**Comparison With the DNN Watermarking Methods.** The proposed method achieves active authorization control where only authorized users are allowed to use the model. Compared with the existing DNN watermarking methods [2]–[5], [18]–[23], which only work after the infringement occurs, the proposed method can actively prevent unauthorized users from using the model, thus protect the DNN IP in advance. Moreover, most of the DNN watermarking methods retrain the model to embed the watermark, which changes the parameters completely. In contrast, the implementation of the proposed method only needs to encrypt an extremely small amount of parameters.

**Comparison With the Existing Active DNN IP Protection Methods.** The proposed method encrypts the parameters of a trained model with well-crafted adversarial perturbations. Compared with the existing active DNN IP protection methods [7]–[10], [25] which require to retrain the model [7]–[9], [10], or need the support of hardware devices [10], [25], the proposed method requires low computational resources and does not need the hardware support, thus is more feasible in realistic commercial applications. Moreover, compared with the work [24], which significantly modifies the model parameters by exchanging the positions of a substantial amount of parameters, the proposed method only needs to encrypt an extremely low number of parameters, and the weights of the encrypted parameters are all in a normal range, which makes these encrypted parameters difficult to be detected.

## III. THE PROPOSED METHOD

### A. THREAT MODEL

The goal of the defender is to ensure the utility of the model for authorized users, while preventing the unauthorized users from using the model. The application scenario of the proposed method is based on the increasingly popular business paradigm called Machine Learning as a Service (MLaaS) [1], where the DNN model is deployed as a cloud service and the users can only use the provided application programming interfaces (APIs) to obtain the prediction of the model. This MLaaS (cloud service) scenario is also widely used in related works [17], [26]–[28]. Under this application scenario, the internal parameters of the online model are not publicly accessible. In this paper, the model is encrypted and deployed on a cloud platform. As shown in Figure 1, the unauthorized users cannot use the model, as the inference accuracy of the encrypted model is extremely low. For the authorized users, they can provide the secret key to MLaaS and the model will be decrypted remotely. Then, the authorized users can use the decrypted model for classification. The prediction of the model is returned to the authorized user through APIs. The authorized users have no access to the internal parameters of the decrypted model, thus the authorized users cannot redistribute or resell the model on the marketplace.

Note that, in practice, the network traffic is usually encrypted to secure the queries of API [29]. However, the proposed method is totally different from the API encryption. The proposed method provides authorization control for DNN models by only perturbing a small set of important parameters of the model. In this way, only the authorized users can use the model normally and unauthorized users will only obtain low inference accuracy. Moreover, encrypting the parameters of model deployed in the cloud can effectively prevent attacks from the internal employees. In other words, even if the internal employees steal the models, they cannot decrypt the model without the secret key. However, only encrypting the queries of API cannot prevent the model from these internal attacks, as the deployed model is the clean one without any protection. Besides, how to encrypt the API and make the queries more secure is not in the scope of this paper and this area (i.e., IP protection for DNN model). Nevertheless, the API query encryption method can be used with the proposed method together to further improve the security of DNN model.

### B. OVERALL FLOW

The overall flow of the proposed method is illustrated in Figure 2. The proposed method includes the following three parts:

- **Parameter encryption.** The process of parameter encryption includes three steps: First, a small amount of labeled images are sampled from the training set of the DNN model. The set of the sampled images is referred as the encryption set. Second, the model owner randomly selects some convolutional layers and fully-connected layers from the model, which is referred as the encrypted layer set. With the encryption set, the gradient of the model’s loss function will be calculated. According to the calculated gradient, the adversarial perturbation is iteratively added on the weights of the selected layers. In each iteration, there is only one hidden layer that will be selected for encryption, and only one weight of the selected layer will be perturbed with
adversarial perturbation. In each iteration, the encryption set will be input into the model, and the model’s loss function will be calculated. For each selected layer, if the predefined maximum number of iteration has been reached or the value of loss function is higher than the pre-defined encryption threshold, this layer will not be encrypted any more, and another layer in the encrypted layer set will be selected for encryption.

- **Secret key generation.** In the encryption process, the positions of the encrypted parameters and the values of the added adversarial perturbations will be stored. When the encryption process is terminated, the stored information will form a secret key.

- **Decryption.** With the secret key, the authorized user can remove the perturbation added to the model’s parameters and restore the model’s accuracy in the MLaaS. Note that, the proposed method only needs to encrypt an extremely small number of model’s parameters (as low as 23, which will be discussed in Section IV-B), thus the encryption is stealthy and the decryption time is negligible.

In the subsequent sections, the above three parts will be elaborated separately.

### C. PARAMETER ENCRYPTION

The encryption process of the proposed method is inspired by JSMA [30] which generates adversarial examples by perturbing only a few pixels of the clean images. The difference between JSMA and the proposed method is that, JSMA [30] perturbs the pixels of a clean image to make DNN model misclassify the perturbed image, while the proposed method modifies the weights of DNN model to make the model dysfunctional.

The process of the adversarial perturbation based parameter encryption is summarized in the proposed Algorithm 1. First, a small amount of labeled images are sampled from the model’s training set, where the set of these labeled images is referred as the encryption set. The encryption set is denoted as $D_{e} = (X_{e}, Y_{e})$, where $X_{e} = \{x_{1}, \ldots, x_{N}\}, Y_{e} = \{y_{1}, \ldots, y_{N}\}$ are the sampled images and the corresponding labels, respectively, and $N$ is the number of the sampled images. Second, for a trained model, the model owner randomly selects some convolutional layers and fully-connected layers for encryption. These randomly selected layers are referred as the encrypted layers. Then, a few weights of these encrypted layers are selected by Algorithm 1, and the adversarial perturbations will be added to these selected weights. In order to decrease the number of encrypted parameters, in each iteration, only one encrypted layer will be selected for encryption, and only one weight in the layer will be perturbed.

Formally, let $\mathcal{L}$ denotes the encrypted layer set. Here, for a single layer $l \in \mathcal{L}$, we elaborate the following two issues: (i) how to select the weights that have a significant impact on the performance of the model; (ii) how to perturb the selected weight.

**How to Select the Weight.** Formally, let $F$ and $\tilde{F}$ denote the trained model and the encrypted model, respectively. For an input image $x$, the predicted label of $F$ and $\tilde{F}$ are denoted as $F(x)$ and $\tilde{F}(x)$, respectively. The loss function is denoted as $L$. For the encryption set $D_{e} = (X_{e}, Y_{e})$, the value of the loss function can be denoted as $L(F, D_{e})$. The weights of the layer $l$ can be denoted as $W_{l} = [w_{l1}, w_{l2}, \ldots]$. In each iteration, the gradient $\nabla_{W_{l}}L(F, D_{e})$ for the weights $W_{l}$ can be calculated as:

$$
\nabla_{W_{l}}L(F, D_{e}) = \left[ \frac{\partial L(F, D_{e})}{\partial w_{l1}}, \frac{\partial L(F, D_{e})}{\partial w_{l2}}, \ldots \right]
$$

The partial derivative with the highest absolute value will be searched, and the corresponding weight $w_{l}$ is considered to have the greatest impact on the model’s performance among all the weights contained in layer $l$, i.e.,

---

FIGURE 2. The overall flow of the proposed method.
then the weight $w_l$ will be selected, and perturbed with adversarial perturbation. The model’s performance will be significantly deteriorated after the weight $w_l$ is perturbed.

**Algorithm 1. Parameter Encryption**

**Input:** encrypted layer set $\mathcal{L}$ and the corresponding weights $W_l = [w_{l1}, w_{l2}, \ldots], \ (l \in \mathcal{L})$, trained model $F$, encryption set $D_e$, the maximum number of the iteration $I$, encryption threshold $T_{loss}$.

**Output:** encrypted model $\hat{F}$.

1. for $l$ in $\mathcal{L}$ do
2. $\ Mask \leftarrow [1, 1, \ldots, 1]$, where the dimension of $\ Mask$ is the same as that of $W_l$;
3. for $i = 1$ to $I$ do
4. Calculate the value of the loss function $L(F, D_c)$;
5. if $L(F, D_c) > T_{loss}$ then
6. break;
7. end if
8. Calculate the gradient $\nabla_W L(F, D_c)$;
9. $\nabla_W L(F, D_c) \leftarrow \nabla_W L(F, D_c) \odot Mask$;
10. Select the the weight $w_l$ with the highest absolute value of partial derivative;
11. $\eta_l = \theta \times \text{sign} \left( \frac{\partial L(F, D_c)}{\partial w_l} \right) \times \max(W_l) - \min(W_l)$;
12. if $w_l + \eta_l \notin [T_{l1}, T_{l2}]$ then
13. $Mask_l \leftarrow 0$;
14. end if
15. w_l $\leftarrow \text{Clip}(w_l + \eta_l)$;
16. end for
17. end for
18. return $\hat{F}$;

**How to Perturb the Selected Weight.** After the weight $w_l$ with the highest absolute value of partial derivative is selected, the value of the adversarial perturbation $\eta_l$ can be calculated as:

$$\eta_l = \theta \times \text{sign} \left( \frac{\partial L(F, D_c)}{\partial w_l} \right) \times \max(W_l) - \min(W_l)$$

(3)

where $\theta$ is a hyper-parameter with low value that controls the perturbation $\eta_l$, and $\max(W_l)$ and $\min(W_l)$ are the maximum weight and the minimum weight in the layer $l$, respectively. The weight $w_l$ will be updated to $\text{Clip}(w_l + \eta_l)$. For other weights that are not being selected, these weights will not be updated and remain the same. The $\text{Clip}$ function is used to control the value of the updated weights, which can be formalized as:

$$\text{Clip}(w_l) = \begin{cases} w_l, w_l \in [T_{l1}, T_{l2}] \\
T_{l1}, w_l < T_{l1} \\
T_{l2}, w_l > T_{l2} \end{cases}$$

where $\alpha$ is a hyper-parameter, and $T_{l1}$ and $T_{l2}$ are two thresholds that control the value of the updated weights. If the updated weight $w_l + \eta_l$ exceeds the range $[T_{l1}, T_{l2}]$ (lower than $T_{l1}$, or higher than $T_{l2}$), it will be clipped into the range $[T_{l1}, T_{l2}]$.

**D. SECRET KEY GENERATION**

Figure 3 illustrates the composition of the secret key, which is generated after the parameter encryption. Specifically, in each iteration, after the weight is added with adversarial perturbation, the encryption set $D_e$ will be input into the model, and the value of the loss function $L(F, D_e)$ will be calculated. If $L(F, D_e) > T_{loss}$, the encryption for the selected layer $l$ will be terminated, and another layer in the encrypted layer set $\mathcal{L}$ will be selected for encryption, where $T_{loss}$ is the pre-defined encryption threshold. Note that, the encryption process will be terminated after only a few iterations, as the modification of each selected parameter leads to a significant deterioration in the accuracy of the model, which makes the value $L(F, D_e)$ of the loss function rapidly increase and be higher than the threshold $T_{loss}$.

**FIGURE 3.** An example of the secret key.

where $\alpha$ is a hyper-parameter, and $T_{l1}$ and $T_{l2}$ are two thresholds that control the value of the updated weights. If the updated weight $w_l + \eta_l$ exceeds the range $[T_{l1}, T_{l2}]$ (lower than $T_{l1}$, or higher than $T_{l2}$), it will be clipped into the range $[T_{l1}, T_{l2}]$. With the $\text{Clip}$ function, the encrypted weights will be controlled into a normal range, thus makes these encrypted weights difficult to be detected by potential attackers. Further, if $w_l + \eta_l \notin [T_{l1}, T_{l2}]$, after the weight $w_l$ has been updated to $\text{Clip}(w_l + \eta_l)$, in the following iterations, the weight $w_l$ will not be updated any more.

In each iteration, after the weight is added with adversarial perturbation, the encryption set $D_e$ will be input into the model, and the value of the loss function $L(F, D_e)$ will be calculated. If $L(F, D_e) > T_{loss}$, the encryption for the selected layer $l$ will be terminated, and another layer in the encrypted layer set $\mathcal{L}$ will be selected for encryption, where $T_{loss}$ is the pre-defined encryption threshold. Note that, the encryption process will be terminated after only a few iterations, as the modification of each selected parameter leads to a significant deterioration in the accuracy of the model, which makes the value $L(F, D_e)$ of the loss function rapidly increase and be higher than the threshold $T_{loss}$.

**E. DECRYPTION**

The decryption process is summarized in the proposed Algorithm 2. Assume there are $K$ layers, $l_1, l_2, \ldots, l_K$, contained in
the encrypted model $\tilde{F}$. For each layer $l \in \{l_1, l_2, \ldots, l_K\}$, the weights of the layer $l$ is denoted as $\tilde{W}_l = [\tilde{w}_{l1}, \tilde{w}_{l2}, \ldots]$. With the secret key $\mathcal{K} = \{(p_l, v_l)|l \in \mathcal{L}, t = 1, \ldots, n_l\}$, the perturbations added to the encrypted model $\tilde{F}$ can be removed, and the original high accuracy of the trained model $F$ will be restored. Specifically, for a layer $l \in \{l_1, l_2, \ldots, l_K\}$, the positions of the encrypted parameters can be determined according to the secret key $\mathcal{K}$. Let $p$ denotes the position of an encrypted parameter of the layer $l$, the added perturbation can be removed, as follows:

$$w_{lp} = \tilde{w}_{lp} - v_p$$

where $\tilde{w}_{lp}$ is the weight of the encrypted parameter, and $v_p$ is the added perturbation on $\tilde{w}_{lp}$.

Algorithm 2. Decryption

Output: secret key $\mathcal{K} = \{(p_l, v_l)|l \in \mathcal{L}, t = 1, \ldots, n_l\}$, the encrypted model $\tilde{F}$ and it’s weights $\tilde{W}_l$ of layer $l$, ($l \in \{l_1, l_2, \ldots, l_K\}$).

Output: original trained model $F$.

1: for $i = 1$ to $K$ do
2: \hspace{1em} if $l_i \in \mathcal{L}$ then
3: \hspace{2em} $l \leftarrow l_i$;
4: \hspace{1em} for $j = 1$ to $n_l$ do
5: \hspace{2em} $p \leftarrow p_j$;
6: \hspace{2em} $v \leftarrow v_j$;
7: \hspace{2em} $w_{lp} \leftarrow \tilde{w}_{lp} - v$;
8: \hspace{1em} end for
9: end if
10: end for
11: return $F$;

IV. EXPERIMENTS

In this section, the experiment is conducted to evaluate the proposed method. First, the experimental setup is introduced in Section IV-A. Then, the experimental results are presented and analyzed in Section IV-B. Besides, the parameter discussion is presented in Section IV-C. Further, the robustness of the proposed method against three attacks is evaluated in Section IV-D. The comparison with cryptographic encryption is presented in Section IV-E. The proposed method is compared with the related works in Section IV-F.

A. EXPERIMENTAL SETUP

A.1 DATASETS

The datasets used in the experiment are Fashion-MNIST [11], CIFAR-10 [13] and GTSRB [15]. For all the three datasets, the number of the encryption images contained in the encryption set $D_e$ is 300, i.e., $|D_e| = 300$. For Fashion-MNIST [11], CIFAR-10 [13], and GTSRB [15] datasets, the proportions of the encryption images of all the training images are only 0.5% (300/60,000), 0.6% (300/50,000), and 0.77% (300/39,208), respectively. The loss functions used in the three datasets are all cross-entropy loss, and the encryption threshold $T_{loss}$ is set to be 15, 12 and 12 for Fashion-MNIST [11], CIFAR-10 [13] and GTSRB [15] datasets, respectively. The hyper-parameter $\theta$ (defined in Section III-C) is set to be 0.07 for all the three datasets, and the hyper-parameter $\alpha$ (defined in Section III-C) is set to be 0.05, 0.05 and 0.03 for Fashion-MNIST [11], CIFAR-10 [13] and GTSRB [15] datasets, respectively.

A.2 DNN MODELS

The model trained on the Fashion-MNIST [11] dataset is the DenseNet model [12]. The DenseNet model is trained on the training set of Fashion-MNIST for 20 epochs with the SGD optimizer [31]. The learning rate and batch size are set to be 0.1 and 128, respectively. The DenseNet model [12] contains 46 convolutional or fully-connected layers. The maximum number $I$ of the iteration is set to be 18 for the DenseNet model. In the experiment, only 6 layers of DenseNet model are selected for encryption by Algorithm 1.

The model trained on the CIFAR-10 [13] dataset is the ResNet model [14]. The ResNet model is trained on the training set of CIFAR-10 for 200 epochs with the Adam optimizer [31]. The batch size is set to be 128. Initially, the learning rate for ResNet model is set to be 0.1, and the CosineAnnealingLR function [32] is used to decrease the learning rate in the training process. The ResNet model [14] contains 58 convolutional or fully-connected layers. The maximum number $I$ of the iteration is set to be 18 for the ResNet model. In the experiment, only 14 layers of ResNet model are selected for encryption by the proposed Algorithm 1.

The model trained on the GTSRB [15] dataset is the AlexNet model [16]. The AlexNet model is trained on the training set of GTSRB for 200 epochs with the Adam optimizer [33]. The learning rate and batch size are set to be 0.003 and 128, respectively. The AlexNet model [16] contains 8 convolutional or fully-connected layers. The maximum number $I$ of the iteration is set to be 108 for the AlexNet model. In the experiment, only 4 layers of AlexNet model are selected for encryption by Algorithm 1.

A.3 METRICS

Accuracy drop $A_d$. This metric represents the drop of the model’s accuracy after parameter encryption. Let $A_o$ represent the original accuracy of the trained model, and $A_p$ represent the accuracy of the encrypted model. Then, the accuracy drop is $A_d = A_o - A_p$ [34]. A large value of $A_d$ indicates that the proposed method can effectively deteriorate the model’s performance. The larger the value of $A_d$, the more effective the proposed method is.

B. EXPERIMENTAL RESULTS

The accuracy drop of the model caused by parameter encryption is presented in Table 1. The values of accuracy drop $A_d$ are as high as 80.65%, 81.16% and 87.91% for Fashion-MNIST [11], CIFAR-10 [13] and GTSRB [15] datasets, respectively. For GTSRB, the accuracy of the model is reduced from 94.85% to 6.94%, which indicates that the model’s performance has been significantly degraded. Besides, for
Fashion-MNIST and CIFAR-10, the proposed method can reduce the model’s accuracy to around 10%. Since both the two datasets contain 10 classes, the model’s accuracy is reduced to a random guess (i.e., close to 10%). In conclusion, the above experimental results demonstrate the effectiveness of the proposed method.

Let \( n_e \) and \( n_{all} \) denote the number of the encrypted parameters and the number of all the parameters of the model, respectively. For the three datasets, the proportion of the encrypted parameters of all the model’s parameters (i.e., \( n_e/n_{all} \)) are presented in Table 2. For the three datasets, the numbers of the encrypted parameters are all in an extremely low level, which are only 23, 47 and 48 for Fashion-MNIST [11], CIFAR-10 [13] and GTSRB [15], respectively. The reason is that, the proposed method utilize the Clip function to control the values of the modified weights, which ensures that these encrypted weights all fall in the pre-defined range \([T_{l1}, T_{l2}]\) (as discussed in Section III-C).

The distribution of the weights of DNN model before and after the parameter encryption is illustrated in Figure 5. As shown in Figure 5, for all the three datasets, the change of the mean value \( \mu \) and variance value \( \sigma \) of the distribution of weights is negligible after the encryption, which means the change of the distribution of weights of DNN model is negligible. The above experimental results indicate that, the encrypted parameters are concealed for the potential attackers, as these encrypted parameters are extremely hard to be detected by analyzing the changes of weights.

### C. PARAMETER DISCUSSION

In this section, we evaluate and discuss the impact of three factors on the performance of the proposed method, which are the encryption threshold \( T_{loss} \), the encrypted layers, and the number \( n_e \) of the encrypted parameters.

#### C.1 DIFFERENT SETTINGS OF ENCRYPTION THRESHOLD \( T_{loss} \)

The performance of the proposed method with different encryption threshold \( T_{loss} \) is evaluated. In the experiment, the values of \( T_{loss} \) are set to be 1~15, and for each \( T_{loss} \), the test...
The accuracy of the encrypted model is calculated, as shown in Figure 6. When $T_{loss} = 1$, the test accuracies are 61.28%, 62.95%, and 69.34% on Fashion-MNIST [11], CIFAR-10 [13], and GTSRB [15] datasets, respectively. It can be seen that, the test accuracy of the encrypted model decreases significantly with the increase of the encryption threshold $T_{loss}$. Besides, as shown in Figure 6, when the encryption threshold $T_{loss}$ reaches 12, the test accuracy is reduced to a stable low value (around 10%, 10%, and 6% on Fashion-MNIST, CIFAR-10, and GTSRB datasets, respectively). Moreover, the higher the value of $T_{loss}$, the more parameters need to be encrypted. Thus, $T_{loss} = 12$ can be considered to have the optimal performance on the trade-off between the encryption time and decreasing the accuracy of the model.

C.2 PERFORMANCE WITH DIFFERENT SETTINGS OF ENCRYPTED LAYER

The performance of the proposed method with different encrypted layers is evaluated in this section. In each experiment, there is only one layer that is selected for encryption, i.e., $|L| = 1$, where $L$ is the encrypted layer set. After the encryption, the test accuracy of the model is calculated. The test accuracy of the encrypted model with different encrypted layers are presented in Figure 7. As shown in Figure 7(a), for Fashion-MNIST, when only encrypt the layer 1, the test accuracy is effectively reduced from 91.01% to 10.75%. Moreover, when only encrypt the layer 2, the layer 4, and the layer 16, the test accuracy of the encrypted model is only 19.79%, 20.76%, and 22.67%, respectively. As shown in Figure 7(b), for CIFAR-10, the test accuracy is significantly decreased from 92.02% to 9.63% by encrypting the layer 22. Moreover, when other layers are selected for encryption, the test accuracy can also decrease to as low as around 10%. For GTSRB, when the layer 1 is encrypted, the test accuracy of DNN model is effectively reduced from 94.85% to 12.64%.

**Figure 5.** The distribution of the weights of DNN model before and after encryption. Figures 5(a)–(c) represent the distribution of weights of the original trained model for Fashion-MNIST, CIFAR-10 and GTSRB, respectively. Figures 5(d)–(f) represent the distribution of weights of the encrypted model for Fashion-MNIST, CIFAR-10 and GTSRB, respectively. For all the three datasets, the change of the mean value $\mu$ and variance value $\sigma$ of the distribution of weights is negligible after the encryption.

**Figure 6.** The performance of the proposed method with different encryption threshold $T_{loss}$. 

---

*Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.*
Besides, as shown in Figure 7(c), when encrypt the layer 2~8 separately, the test accuracy of the model is hardly decreased, which means the impact of the layer 2~8 on the test accuracy of the model is insignificant for GTSRB. In conclusion, encrypting the layer 1, the layer 22, and the layer 1 can be considered to have the optimal performance in terms of decreasing the accuracy of the model for Fashion-MNIST [11], CIFAR-10 [13] and GTSRB [15] datasets, respectively. Besides, the experimental results also indicate that, the proposed method can effectively decrease the model’s accuracy by only encrypting a single layer of the model.

C.3 DIFFERENT SETTINGS OF THE NUMBER OF ENCRYPTED PARAMETERS

The impact of different settings of the number of encrypted parameters on the performance of the proposed method is evaluated in this section. Note that, the experimental setup in this section is different from that in Section IV-A, where in this section, all layers of DNN model is selected for encryption, while in Section IV-A, only a few layers are selected for encryption. The test accuracy of the encrypted model with different settings of the number $n_e$ of encrypted parameters is presented in Figure 8. As $n_e$ rises, the test accuracy decreases significantly for all the three datasets. Moreover, for Fashion-MNIST and CIFAR-10, the test accuracy stabilizes to a random guess (around 10%) when $n_e$ reaches 16 and 34, respectively. For GTSRB, when $n_e$ reaches 49, the test accuracy stabilizes to an extremely low value (around 6%). It can be seen that, the test accuracy of the model has already stabilized to a low value after $n_e$ reaches 16, 34, and 49 for Fashion-MNIST [11], CIFAR-10 [13] and GTSRB [15] datasets, respectively. The above experimental results indicate that, by encrypting an extremely low number of encrypted parameters, the proposed method can achieve an excellent encryption effect.

D. ROBUSTNESS OF THE PROPOSED METHOD

In this section, the robustness of the proposed method against three attacks are evaluated, which are model fine-tuning attack, model pruning attack, and adaptive attack. In the adaptive attack, the attacker knows the detailed steps of the proposed method.

D.1 ROBUSTNESS AGAINST MODEL FINE-TUNING ATTACK

By using model fine-tuning [35], an attacker tries to remove the perturbations of the encrypted parameters and restore the original accuracy of the model. In the experiment, 10% of the model’s test set is used as the attacker’s private dataset to fine-tune the encrypted model, and the accuracy of the fine-tuned model is evaluated with the rest 90% of the test set. The learning rates for fine-tuning are set to be 0.000001, 0.00001 and 0.00001 in Fashion-MNIST [11], CIFAR-10 [13] and GTSRB [15] datasets, respectively. The encrypted model is fine-tuned for 100 epochs and the test accuracy of the model is calculated for every 10 epochs. For the three datasets, the test accuracy of the encrypted model after model fine-tuning attack is shown in Table 3. The experimental results indicate that, even the encrypted model is fine-tuned for 100 epochs, the accuracy of the encrypted model still remains at a low level (51.14%, 43.80%, and 61.66% for Fashion-MNIST [11], CIFAR-10 [13] and GTSRB [15] datasets, respectively) after fine-tuning attack, which means that the proposed method is robust against...
model fine-tuning attack. The reasons that the attacker is assumed to use 10% of test data and a small learning rate (e.g., 0.00001) to fine-tune the model are as follows. First, the cost of fine-tuning the model on a large amount of data is close to that of training the model from scratch. Therefore, considering the training cost, the adversary will not use a large amount of data to fine-tune the encrypted model. Second, it is difficult for individuals to obtain large scale and high-quality labeled datasets, as it usually costs a company a lot of money to collect and label the images. Third, in deep learning area, the common practice of fine-tuning is to utilize a small set of data for fine-tuning in order to reduce the training time [36, 37]. Lastly, using a very small learning rate to fine-tune the model is also the common practice of fine-tuning in order to prevent destroying the learned features of pre-trained models [36, 37].

Moreover, we also consider the worst scenario, where the attacker utilizes a large amount of data and a large learning rate to fine-tune the encrypted model. Specifically, $p\%$ of the model’s test set is used to fine-tune the encrypted model, and the accuracy of the fine-tuned model is evaluated with the rest $1-p\%$ of the test set, where $p$ is set to be 20, 30 and 40, respectively. The learning rates for fine-tuning are set to be 0.1, 0.1 and 0.003 in Fashion-MNIST [11], CIFAR-10 [13] and GTSRB [15] datasets respectively. These learning rates are the same as that used in the training stage. The encrypted model is fine-tuned for 100 epochs. With different settings of $p$, the test accuracy of the encrypted model after model fine-tuning attack is shown in Table 4. The experimental results show that, when the encrypted model is fine-tuned for 100 epochs on 40% of test data, the accuracy of the encrypted model is 78.45%, 83.19%, and 77.94% for Fashion-MNIST [11], CIFAR-10 [13] and GTSRB [15] datasets, respectively. Note that, the above experiments settings (large learning rate and large amount of data) are only used to evaluate the performance of the proposed method under the worst scenario. In practice, the attacker usually will not use such large amount of data and large learning rate (e.g., 0.1) to fine-tune the model, in order to reduce the training cost. If the attacker uses large amount of data and large learning rate to fine-tune the model, fine-tuning the model is close to train the model from scratch. In other words, if the amount of data and learning rate are both large, it is equivalent to discard the encrypted model and train a new model. In fact, when 10% of test data is used to fine-tune the model with a small learning rate (e.g., 0.00001), the accuracy of the encrypted model is only 51.14%, 43.80% and 61.66% on Fashion-MNIST, CIFAR-10 and GTSRB, respectively.

D.2 ROBUSTNESS AGAINST MODEL PRUNING ATTACK
An attacker may use model pruning [38] to prune the encrypted parameters. In the experiment, the pruning rate is set to be 10%~90%, and the experimental results are presented in Table 5. It can be seen that, for all the three datasets, the test accuracy of the encrypted model is still lower than 20% after model pruning attack, which demonstrate the robustness of the proposed method against model pruning attack. The reason is that, the weights of the encrypted parameters are all in a normal range. To remove the encrypted parameters, other parameters that are not encrypted will also be pruned at the same time. Thus, even if some of the encryption parameters are pruned, the test accuracy is still at a low level.

D.3 ROBUSTNESS AGAINST ADAPTIVE ATTACK
In this section, the worst-case scenario is considered, where the malicious attacker knows the detailed steps of the proposed method, and attempts to conduct the powerful adaptive attack to remove the added adversarial perturbations without the secret key. In practice, the internal parameters of DNNs are not publicly accessible, which means the attacker doesn’t have access to the encrypted parameters of the DNN model. However, in this experiment, we assume that the attacker has the knowledge of all the internal parameters of the encrypted model, which is the worst-case scenario. Only the key is unknown to the attacker.
The attacker attempts to search for the potential encrypted parameters by following the steps described in Section III-C, and calculates the value of the perturbation with Equation (3), then removes the added perturbations of these parameters to decrypt the model. The test accuracy of the encrypted model after the adaptive attack is shown in Table 6. The test accuracy of the model still remains at a low level (56.28%, 62.03%, and 63.92% for Fashion-MNIST [11], CIFAR-10 [13] and GTSRB [15] datasets, respectively) after the adaptive attack. Overall, under the adaptive attack where the attackers know the mechanism of the proposed method and the internal parameters of the encrypted model, the proposed method is still robust. The reason is that, the proposed method encrypts the model by searching and modifying the important parameters that have significant impact on the inference performance of the model. After the model is encrypted, those important parameters that have significant impact on the performance will change. In this way, if the attacker tries to generate the secret key from the encrypted model through the proposed method, the searched important parameters are not the same as the ones in the model before encryption. As a result, even if the attacker completely follows the steps of the proposed method, he cannot generate the correct key. Hence, the attacker cannot recover the model’s original parameters.

**E. COMPARISON WITH CRYPTOGRAPHIC ENCRYPTION METHOD**

In this section, the proposed method is compared with Advanced Encryption Standard (AES) [39], which is a traditional cryptographic encryption method. In the experiment, AES under CBC mode [39] is implemented to encrypt all the model parameters, i.e., \( n_e = n_{all} \), where the size of the key for AES is set to 256 bits. In the proposed method, only a few model parameters are encrypted, i.e., \( n_e < < n_{all} \). The number of the encrypted parameters \( n_e \) and the number of all the parameters \( n_{all} \) are presented in Table 2. The experiment is conducted on Google Colab, where a Tesla K80 GPU and two Intel Xeon CPUs are allocated. Let \( t_e \) and \( t_d \) denote the encryption time and decryption time, respectively. The encryption time \( t_e \) and decryption time \( t_d \) are presented in Table 7. The experimental results show that, the encryption time for the proposed AdvParams is as low as 21.69 s, while for AES, the encryption time is as high as 18735.39 s. Besides, for AdvParams, the decryption time is as low as 0.0041 s, which is negligible, while the time cost of decryption for AES is much higher than that of the proposed method, especially on the GTSRB dataset (19187.77 s). Note that, for the proposed method, the encryption time is much higher than the decryption time. The reason is that, for encryption process, the proposed method requires to iteratively search suitable parameters and generate corresponding perturbations. In contrast, for decryption process, the proposed method directly recovers those parameters according to the information (positions of perturbed parameters and corresponding perturbations) stored in the secret key. The experimental results indicate that, when all the parameters of the model are encrypted by AES, the time cost of encryption is unacceptable and the decryption time is also very long, especially when the model contains a tremendously large number of parameters. On the contrary, by encrypting an extremely low number of parameters, the time cost of the proposed method is negligible. For example, for the GTSRB dataset, the decryption time of AES is 2,863,846 times that of our method, and the encryption time of AES is 863.78 times that of our method. Moreover, in the proposed method, the perturbations added to the encrypted parameters are very small, which make the parameters encryption imperceptible and difficult to detect. However, the parameters encrypted through cryptographic encryption method are very abnormal (garbled characters), which makes the parameters of model very suspicious and cannot be loaded and deployed.

Even if lightweight encryption method is used instead of advanced encryption methods (like AES) to encrypt the model, it still has the following limitations. First, considering the enormous number of parameters, decrypting the model through lightweight cryptographic method is a time-consuming process. In comparison, the proposed method only encrypts a small number of parameters through slight perturbation. The proposed method can quickly recover the parameters with the secret key. In this way, the time cost of the proposed method is negligible. Second, when the model is encrypted by lightweight encryption method, all parameters of the model are transformed to garbled characters, which are not numbers and totally different from the normal parameters. In this way, the encrypted model becomes very suspicious and cannot be loaded and deployed. In contrast, the proposed method only selects a small number of parameters to add well-crafted perturbations. The generated perturbations are very small, which makes the perturbed parameters very imperceptible and hard to detect. Meanwhile, the perturbed model can still be loaded and deployed, while the inference accuracy of

**TABLE 6. The test accuracy of the encrypted model after adaptive attack.**

| Dataset                | Test accuracy |
|------------------------|---------------|
| Fashion-MNIST (DenseNet) | 56.28%        |
| CIFAR-10 (ResNet)       | 62.03%        |
| GTSRB (AlexNet)         | 63.92%        |

**TABLE 7. The time cost for the proposed method and AES.**

| Dataset                | Ours  | AES   |
|------------------------|-------|-------|
|                        |       |       |
|                       | \( t_e \) | \( t_d \) | \( t_e \) | \( t_d \) |
| Fashion-MNIST (DenseNet) | 25.08 s | 0.0055 s | 360.60 s | 366.74 s |
| CIFAR-10 (ResNet)       | 36.46 s | 0.0041 s | 954.99 s | 973.39 s |
| GTSRB (AlexNet)         | 21.69 s | 0.0067 s | 18735.39 s | 19187.77 s |

1https://colab.research.google.com
TABLE 8. Comparison between the proposed method and existing active DNN IP protection works.

| Works                      | Dataset   | Accuracy Authorized usage | Accuracy Unauthorized usage | Accuracy drop $A_d$ | Require additional training | Require hardware support |
|----------------------------|-----------|----------------------------|-----------------------------|---------------------|-----------------------------|--------------------------|
| Encryption-based [7]       | CIFAR-10  | 92.26%                     | 20.01%                      | 72.25%              | Yes                         | No                       |
| Transformation module-based [8] | Fashion-MNIST | 92.55%                     | 1.55%                       | 91.00%              | Yes                         | No                       |
| Passports-based [9]       | CIFAR-10  | 90.61%                     | 0.78%                       | 89.83%              | No                          | No                       |
| Hardware device-based [10] | Fashion-MNIST | 89.93%                     | 10.05%                      | 79.88%              | Yes                         | No                       |
| The proposed method       | Fashion-MNIST | 91.01%                     | 10.36%                      | 80.65%              | No                          | No                       |
|                           | CIFAR-10  | 92.02%                     | 10.86%                      | 81.16%              |                             |                          |
|                           | GTSRB     | 94.85%                     | 6.94%                       | 87.91%              |                             |                          |

perturbed model is very low. Third, compared with advanced encryption method (e.g., AES), the lightweight encryption method is less secure [40], [41]. For example, it is indicated that the lightweight encryption methods are not secure against related-key attacks [41].

F. COMPARISON WITH EXISTING ACTIVE DNN IP PROTECTION WORKS

In this section, the proposed method is compared with the existing active DNN IP protection works [7]–[10], in which, only the work [7] is the encryption based method.

The comparison results are presented in Table 8. As shown in Table 8, for authorized users, the test accuracy of the five works are all in a high level (around 90%). For the three datasets, the accuracy drop $A_d$ of the proposed method is as high as 87.91%, which is higher than works [7], [9], [10], and similar to work [8]. To realize the function of authorization control, the proposed method only needs to perturb a small amount of parameters of a trained model. However, for the works [7]–[10], the implementations of these related works are time-consuming as these works require to retrain the DNN model, and the original parameters of the model will be changed significantly. Besides, in the works [7], [8], the input images need to be transformed before being input into the model, which introduce high computational overhead especially when there are a substantial amount of input images. Moreover, the work [10] requires the support of specific hardware devices, which is costly in commercial applications. In addition, for the work [9], attackers can conduct reverse-engineering attack to obtain the hidden parameters of DNN model. Thus, compared with the four active DNN IP protection works, the proposed method requires low computational overhead as the proposed method does not require to retrain the DNN models, which also makes the modification of model’s parameters negligible. Moreover, the proposed method is robust against strong attacks (as discussed in Section IV-D), and does not require the support of hardware, which is low-cost and feasible in the realistic commercial applications.

V. CONCLUSION

This paper proposes an active DNN IP protection method based on adversarial perturbations. The proposed method realizes the function of authorization control, which makes the DNN model only work for authorized users, thus can prevent the infringement in advance. Compared with the existing active DNN IP protection works, the proposed method only needs to encrypt an extremely small number of parameters, and does not require additional training or hardware support, which is low-cost and more practical in the commercial applications. For unauthorized users, the accuracy drops are as high as 80.65%, 81.16%, and 87.91% for Fashion-MNIST [11], CIFAR-10 [13] and GTSRB [15] datasets, respectively. Besides, the number of the encrypted parameters is as low as 23, and the weights of the encrypted parameters are all in a normal range, which makes the encrypted parameters difficult to be detected by malicious attackers. In addition, the proposed method is robust against model fine-tuning attack, model pruning attack, and the adaptive attack. In the future, we will explore the IP protection methods for distributed learning scenarios.

REFERENCES

[1] M. Ribeiro, K. Grolinger, and M. A. M. Capretz, “MLaaS: Machine learning as a service,” in Proc. IEEE 14th Int. Conf. Mach. Learn. Appl., 2015, pp. 906–902.
[2] Y. Uchida, Y. Nagai, S. Sakazawa, and S. Satoh, “Embedding watermarks into deep neural networks,” in Proc. ACM Int. Conf. Multimedia Retrieval, 2017, pp. 269–277.
[3] B. D. Rouhani, H. Chen, and F. Koushanfar, “DeepSigns: An end-to-end watermarking framework for ownership protection of deep neural networks,” in Proc. 24th Int. Conf. Architectural Support Program. Lang. Operating Syst., 2019, pp. 485–497.
[4] H. Chen, B. D. Rouhani, and F. Koushanfar, “BlackMarks: Blackbox multibit watermarking for deep neural networks,” 2019, arXiv:1904.00344.
[5] E. Le Merrer, P. Perez, and G. Trédan, “Adversarial frontier stitching for remote neural network watermarking,” Neural Comput. Appl., vol. 32, no. 13, pp. 9233–9244, 2020.
[6] M. Xue, Y. Zhang, J. Wang, and W. Liu, “Intellectual property protection for deep learning models: Taxonomy, methods, attacks, and evaluations,” IEEE Trans. Artif. Intell., vol. 3, no. 6, pp. 908–923, Dec. 2022.
[7] A. Pyone, M. Maung, and H. Kiya, “Training DNN model with secret key for model protection,” in Proc. IEEE 9th Glob. Conf. Consum. Electron., 2020, pp. 818–821.
Xue et al.: AdvParams: An Active DNN Intellectual Property Protection Technique via Adversarial Perturbation based Parameter Encryption

[8] M. Chen and M. Wu, “Protect your deep neural networks from piracy,” in Proc. IEEE Int. Workshop Inf. Forensics Secur., 2018, pp. 1–7.
[9] L. Fan, K. Ng, and C. S. Chan, “Rethinking deep neural network ownership verification: Embedding passports to defeat ambiguity attacks,” in Proc. Annu. Conf. Neural Inf. Process. Syst., 2019, pp. 4716–4725.
[10] A. Chakraborty, A. Mondal, and A. Srivastava, “Hardware-assisted intellectual property protection of deep learning models,” in Proc. IEEE/ACM 57th Des. Automat. Conf., 2020, pp. 1–6.
[11] H. Xiao, K. Rasul, and R. Vollgraf, “Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms,” 2017, arXiv:1708.07747.
[12] 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. 2261–2269.
[13] A. Krizhevsky and G. Hinton, “Learning multiple layers of features from tiny images,” Univ. Toronto, Toronto, ON, Canada, Tech. Rep. 001, 2009.
[14] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2016, pp. 770–778.
[15] J. Stallkamp, M. Schlippsing, J. Salmen, and C. Igel, “The German traffic sign recognition benchmark: A multi-class classification competition,” in Proc. Int. Joint Conf. Neural Netw., 2011, pp. 1453–1460.
[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in Proc. Int. Conf. Neural Inf. Process. Syst., 2012, pp. 1106–1114.
[17] Y. Adi, C. Baum, M. Cisse, B. Pinkas, and J. Keshet, “Turning your weakness into a strength: Watermarking deep neural networks by backdooring,” in Proc. 27th USENIX Secur. Symp., 2018, pp. 1615–1631.
[18] J. Guo and M. Potkonjak, “Watermarking deep neural networks for embedded systems,” in Proc. Int. Conf. Comput.-Aided Des., 2018, pp. 1–8.
[19] X. Cao, J. Jia, and N. Z. Gong, “IPGuard: Protecting intellectual property of deep neural networks via fingerprinting the classification boundary,” in Proc. ACM Asia Conf. Comput. Commun. Secur., 2021, pp. 14–25.
[20] L. Lukas, Y. Zhang, and F. Kerschbaum, “Deep neural network fingerprinting by conferrable adversarial examples,” in Proc. 9th Int. Conf. Learn. Representations, 2021, pp. 1–17.
[21] J. Zhao, Q. Hu, G. Liu, X. Ma, F. Chen, and M. M. Hassan, “AFA: Adversarial fingerprinting authentication for deep neural networks,” Comput. Commun., vol. 150, pp. 488–497, 2020.
[22] X. Guan, H. Feng, W. Zhang, H. Zhou, J. Zhang, and N. Yu, “Reversible watermarking in deep convolutional neural networks for integrity authentication,” in Proc. 28th ACM Int. Conf. Multimedia, 2020, pp. 2273–2280.
[23] J. Zhang et al., “Model watermarking for image processing networks,” in Proc. AAAI Conf. Artif. Intell., 2020, pp. 12805–12812.
[24] N. Lin, X. Chen, H. Lu, and X. Li, “Chaotic weights: A novel approach to protect intellectual property of deep neural networks,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 40, no. 7, pp. 1327–1339, Jul. 2021.
[25] H. Chen, C. Fu, B. D. Rouhani, J. Zhao, and F. Koushanfar, “DeepAttest: An end-to-end attestation framework for deep neural networks,” in Proc. 46th Int. Symp. Comput. Archit., 2019, pp. 487–498.
[26] J. Zhang et al., “Protecting intellectual property of deep neural networks with watermarking,” in Proc. Asia Conf. Comput. Commun. Secur., 2018, pp. 159–172.
[27] M. Juntti, S. Szyller, S. Marchal, and N. Asokan, “PRADA: Protecting against DNN model stealing attacks,” in Proc. IEEE Eur. Softw. Secur. Privacy, 2019, pp. 512–527.
[28] S. Szyller, B. G. Atli, S. Marchal, and N. Asokan, “DAWN: Dynamic adversarial watermarking of neural networks,” in Proc. 29th ACM Int. Conf. Multimedia, 2021, pp. 4417–4425.
[29] W. He, D. Akhawe, S. Jain, E. Shi, and D. X. Song, “ShadowCrypt: Encrypted web applications for everyone,” in Proc. Comput. Commun. Secur., 2014, pp. 1028–1039.
[30] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, “The limitations of deep learning in adversarial settings,” in Proc. IEEE Eur. Softw. Secur. Privacy, 2016, pp. 372–387.
[31] S. Theodoridis, “Stochastic gradient descent: The LMS algorithm and its family,” in Machine Learning: A Bayesian and Optimization Perspective, S. Theodoridis, Ed., Oxford, U.K.: Academic Press, 2015, pp. 161–231.
[32] I. Loshchilov and F. Hutter, “SGDR: Stochastic gradient descent with warm restarts,” in Proc. 5th Int. Conf. Learn. Representations, 2017, pp. 1–16.
JIAN WANG received the PhD degree in computer application technology from Nanjing University, China, in 1998. From 2001 to 2003, he was a postdoctoral researcher with the University of Tokyo, Japan. From 1998 to 2004, he was an Associate Professor with Nanjing University, China. He is currently a full professor with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, China, where he was also the vice director of this college from 2010 to 2015. He has published more than 70 papers. His research interests include applied cryptography, system security, key management, security protocol, and information security.

WEIQIANG LIU (Senior Member, IEEE) received the BSc degree from the Nanjing University of Aeronautics and Astronautics (NUAA), China, and the PhD degree from the Queen’s University Belfast (QUB), U.K. He is currently a professor and the vice dean with the College of Electronic and Information Engineering, NUAA. He has published two research book and more than 180 leading journal and conference papers. His paper was selected as the Highlight Paper of IEEE TCAS-I in the 2021 January Issue and the Feature Paper of IEEE TC in the 2017 December issue. He has been awarded the prestigious Excellent Young Scholar Award by National Natural Science Foundation of China. He has served as the associate editors of the IEEE Transactions on Computers, IEEE Transactions on Circuits and Systems I: Regular Papers and IEEE Transactions on Emerging Topics in Computing, a Steering Committee member of IEEE Transactions on Very Large Scale Integration (VLSI) Systems and a guest editor of the Proceedings of the IEEE. His research interests include approximate computing, and hardware security.