Computers, Materials & Continua

Tech Science Press

Doi:10.32604/cmc.2025.064620

ARTICLE

Active Protection Scheme of DNN Intellectual Property Rights Based on Feature Layer Selection and Hyperchaotic Mapping

Xintao Duan[1,2,*], **Yinhang Wu**[1], **Zhao Wang**[1] and **Chuan Qin**[3]

[1]College of Computer and Information Engineering, Henan Normal University, Xinxiang, 453000, China
[2]Henan Provincial Key Laboratory of Educational AI and Personalized Learning, Henan Normal University, Xinxiang, 453000, China
[3]Optical-Electrical and Computer Engineering, University of Shanghai for Science and Technology, Shanghai, 200093, China

*Corresponding Author: Xintao Duan. Email: duanxintao@htu.edu.cn

Received: 20 February 2025; Accepted: 19 May 2025; Published: 30 July 2025

ABSTRACT: Deep neural network (DNN) models have achieved remarkable performance across diverse tasks, leading to widespread commercial adoption. However, training high-accuracy models demands extensive data, substantial computational resources, and significant time investment, making them valuable assets vulnerable to unauthorized exploitation. To address this issue, this paper proposes an intellectual property (IP) protection framework for DNN models based on feature layer selection and hyper-chaotic mapping. Firstly, a sensitivity-based importance evaluation algorithm is used to identify the key feature layers for encryption, effectively protecting the core components of the model. Next, the L1 regularization criterion is applied to further select high-weight features that significantly impact the model's performance, ensuring that the encryption process minimizes performance loss. Finally, a dual-layer encryption mechanism is designed, introducing perturbations into the weight values and utilizing hyperchaotic mapping to disrupt channel information, further enhancing the model's security. Experimental results demonstrate that encrypting only a small subset of parameters effectively reduces model accuracy to random-guessing levels while ensuring full recoverability. The scheme exhibits strong robustness against model pruning and fine-tuning attacks and maintains consistent performance across multiple datasets, providing an efficient and practical solution for authorization-based DNN IP protection.

KEYWORDS: DNN IP protection; active authorization control; model weight selection; hyperchaotic mapping; model pruning

1 Introduction

Deep neural networks (DNNs) have achieved remarkable success across a wide range of tasks and have been extensively adopted in commercial applications over the past decade [1]. The development of high-performance DNN models typically demands access to large-scale datasets, substantial computational resources, and highly skilled personnel, rendering such models valuable intellectual property assets [2]. However, these models are increasingly vulnerable to unauthorized exploitation, including model stealing and reverse engineering, which pose significant threats to intellectual property rights. Consequently, protecting the intellectual property of DNNs has emerged as a critical research focus, and various techniques have been proposed to safeguard models against infringement.

Current intellectual property protection technologies for deep neural networks (DNNs) can be categorized into passive verification and active defense approaches [3], both of which exhibit inherent limitations.

Passive verification methods primarily rely on embedding watermarks into models for subsequent ownership authentication [4,5]. Extensive research has been devoted to developing various watermarking techniques [6,7]. However, these approaches suffer from two major limitations: (1) Their reactive nature only allows post-infringement forensics, offering no means to proactively prevent unauthorized usage; and (2) the watermark embedding process typically requires full model retraining, leading to considerable computational and time overhead [8,9]. Active protection technologies, although capable of enabling access control through encryption, present their own challenges. Most existing solutions require retraining the entire network for parameter encryption, which entails significant encryption costs for already-trained DNN models. Furthermore, certain parameter encryption schemes demand complete encryption of all model parameters, similarly demanding considerable encryption overhead. These constraints pose substantial challenges for practical implementation in large-scale commercial models.

The fundamental dilemma of existing methodologies lies in their inability to balance security and efficiency. Passive approaches appear computationally efficient due to their encryption-free nature, yet critically lack proactive defense capabilities. Conversely, active methods enable real-time protection at the expense of practicality, as global parameter encryption and mandatory retraining impose prohibitive operational constraints. More critically, both paradigms fail to address the synergistic optimization between encryption granularity and algorithmic efficiency-excessive encryption squanders computational resources whereas simplistic encryption becomes vulnerable to reverse-engineering attacks. This paper proposes a dynamic fine-grained encryption framework that achieves dual optimization of security and efficiency through feature layer selection and 4D hyper-chaotic mapping. The main contributions of this paper include the following:

- A novel active protection scheme for DNNs is proposed, which safeguards DNN models through a critical weight selection strategy and a dual encryption mechanism. This approach eliminates the need to retrain the DNN model or encrypt all model parameters.
- A DNN critical weight selection strategy is designed to protect DNN IP by encrypting only a small subset of parameters that have the most significant impact on model performance.
- A novel dual encryption strategy for critical weights is developed, leveraging a 4D Lorenz hyperchaotic map and an amplitude recovery mask to encrypt the positional information and values of critical weights, respectively. This ensures the protection of key model parameters.

2 Related Work

Passive verification. Uchida et al. [10] proposed for the first time a generalized framework for embedding watermarks in deep network models by embedding watermarks into model parameters via a parameter regularizer without compromising the performance of the model. Chen et al. [11] embedded unique fingerprints in the model parameters, and the model owner can verify the ownership of the model by identifying the embedded fingerprints. Wang et al. [12] inserted a separate neural network in DNN using selective weights for watermarking and changed the neural network to be used only in the training phase and watermark verification phase. However, references [10–12] all of these schemes embed watermarking directly in the internal weights of the model, which is easy to be detected and attacked. Adi et al. [13] proposed a black-box watermarking-based IP protection method for DNNs, which can be applied to most classification tasks and can be well combined with various learning algorithms. Le Merrer et al. [14] proposed a method to construct a watermarking algorithm using antagonistic samples using the zero-bit watermarking algorithm with trigger sets. This method only alters the sample labels without altering the original data and requires very few queries to extract the watermark. Li et al. [15] utilized frequency domain image watermarking to generate triggers to construct DNN watermarks, and this black box watermarking scheme can be effective in resisting fraudulent claims attacks due to the high stealthiness of frequency domain watermarking. References' [13–15] methods have broader application prospects, but the black-box watermarking method modifies the training dataset of the model, which inevitably affects the accuracy of the model, and may not be suitable for application in some domains with higher accuracy requirements. Existing studies on DNN IP protection primarily rely on passive verification mechanisms. These approaches often require substantial resources during the initial deployment phase and can only verify model ownership retrospectively, after an infringement has occurred. Moreover, they offer no effective defense against adversarial behaviors such as unauthorized usage or model tampering. As such, passive verification schemes are increasingly insufficient for modern IP protection demands. In response, active DNN IP protection aims to proactively prevent and detect misuse, attracting growing attention and emerging as a key research direction.

Active DNN Intellectual Property Protection: Several proactive DNN protection schemes have emerged in recent research. Pyone et al. [16] proposed an input preprocessing framework where authorized users must apply identical image transformations to utilize the DNN model effectively. Ren et al. [17] introduced a model-locking mechanism that generates correct inference results exclusively for users possessing specific authentication markers. Luo et al. [18] implemented hierarchical authorization through Laplacian mechanism-based output perturbations with varying intensity levels. Tian et al. [19] developed a selective encryption algorithm for IP protection while enabling tiered service functionalities. However, these methods universally require model retraining, incurring substantial encryption overhead. Hardware-assisted solutions have also been explored: Pan et al. [20] leveraged Physical Unclonable Functions (PUFs) to generate device-bound secret keys for weight permutation and diffusion-based encryption. Fan et al. [21] devised a passport-based protection strategy demonstrating strong resilience against model modification and ambiguity attacks. Chakraborty et al. [22] proposed a trusted device-embedded framework where DNNs serve as key-dependent functions, restricting model accessibility to authorized hardware owners. Although these hardware-based approaches achieve active protection, they still demand model retraining and often involve labor-intensive key embedding processes. Parameter encryption schemes present alternative solutions. Zhou et al. [23] created an access-controlled framework preventing unauthorized users from obtaining functional DNN models. Inspired by Zhu et al. [24] and Li et al.'s [25] chaotic image encryption, Lin et al. [26] developed a chaos mapping-based weight obfuscation method that scrambles convolutional/full-connected layer kernels into chaotically disordered states through positional ambiguity. While eliminating retraining requirements, these parameter-level approaches necessitate full-model encryption, resulting in prohibitive computational costs that hinder commercial deployment. Some more novel approaches have been proposed [27–29], but again the network needs to be retrained. All of the above works perform active authorization control on DNN models, so that only legitimate users can use the services provided by the model properly with a specific secret key, and illegitimate users cannot obtain the correct functionality of the model.

Most existing active DNN IP protection methods require either model retraining or specialized hardware support, leading to significant overhead in terms of time and computational resources. In contrast, the method proposed in this paper achieves IP protection by encrypting only a small subset of critical weights from selected feature layers. This design significantly reduces both computational and time costs, while maintaining high efficiency in both encryption and decryption processes. Unlike the chaotic weighting approach proposed by Lin et al. [26], which is limited to encrypting square-shaped parameter regions, our method supports encryption over arbitrarily shaped parameter subsets. Furthermore, by applying a dual-stage parameter selection strategy, our approach minimizes the number of parameters to be encrypted, thereby enhancing overall efficiency.

3 The Proposed Algorithm

3.1 Problem Statement

The overall concept of the proposed scheme is illustrated in Fig. 1. The DNN model, developed by the model owner at substantial computational and time costs, must be protected to prevent unauthorized exploitation. As depicted in the figure, both authorized users and potential attackers can access the encrypted model via a public cloud API. However, only authorized users possess the trusted decryption key provided by the model owner. Upon decryption, authorized users can fully restore the model and utilize its services as intended. In contrast, attackers, lacking the correct key, are restricted to an encrypted version of the model with significantly degraded inference accuracy. The goal of our scheme is to prevent unauthorized entities from benefiting from high-performance models, while ensuring that legitimate users experience no degradation in model accuracy or functionality after decryption.

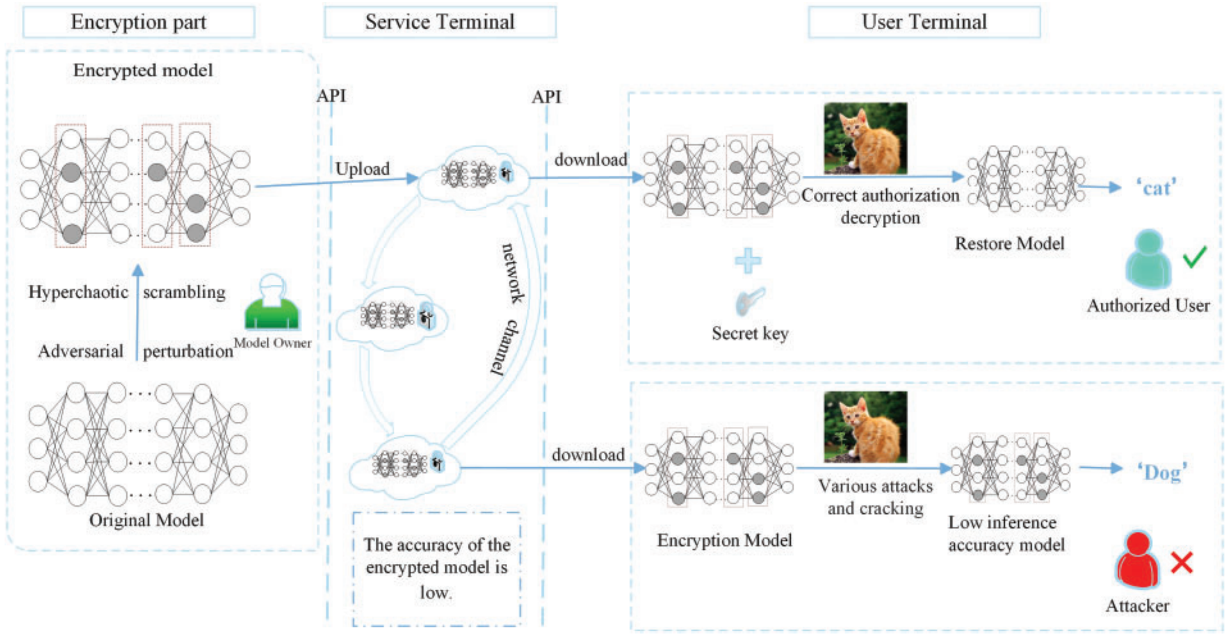

Figure 1: Overall flowchart of DNN IP protection

3.2 Model Encryption

During DNN model training, each convolutional layer of the network works together to optimize the model performance after a certain number of iterations. However, the role played by each convolutional layer is different, i.e., each convolutional layer has a different degree of influence on the model performance. Therefore, before selecting the weights for encryption, the model owner first sorts all the feature layers according to their importance according to the feature layer importance discrimination algorithm (Algorithm 1), sets a parameter p, which represents the proportion of the number of feature layers selected for encryption to the total number of feature layers, and then processes the selected encrypted layers to get the weights that have the greatest effect on the model, and then the weight positions are scrambled using a 4D Lorentz hyperchaotic system, and an Amplitude recovery maskis added to the weights for further encryption.

3.2.1 Structured Obfuscation

According to Algorithm 1, firstly sort all feature layers by importance, and add different degrees of disturbance to the weight of each feature layer. After each disturbance, calculate the change in model accuracy, obtain the accuracy drop value drop, and compare it with the drop threshold th. If it is greater than the threshold th, the feature layer will be marked as the layer to be encrypted, and the drop value drop will be retained. The initial threshold is set to 0.1. After all feature layers are selected, sort the feature layers by the drop value to obtain the feature layers to be encrypted. Here, the disturbance range and threshold th can be reasonably adjusted according to the task calculation amount and calculation time. In order to obtain a more accurate ranking of the importance of the convolutional layer, this paper selects a larger disturbance range and a smaller th value.

Algorithm 1: Feature layer importance evaluation

```
Input: $Threshold : th = 0.1$
1: $Trained\ DNN\ model : F$
Output: $Layers\ to\ be\ encrypted : En$
2: function Discrimination() /* Core evaluation procedure */
3: for $L_i \in F$ do /* Iterate through all layers */
4:     $pe = 0.1, drop = 0$
5:     while $pe > 1 \times 10^{-10}$ do /* Continue until perturbation becomes negligible */
6:         $L_i.weight.data += pe$ /* Apply additive weight perturbation */
7:         $drop = \text{ACC}_{\text{original}} - \text{ACC}_{\text{drop}}$ /* Calculate accuracy degradation */
8:         if $drop > th$ then /* Check significance threshold */
9:             Save $L_i$, drop in $En$ and break /* Record sensitive layer */
10:        end if
11:        $pe *= 0.1$ /* Exponentially reduce perturbation magnitude */
12:    end while
13: end for
14: $argsort(drop) \rightarrow En$ /* Sort layers by sensitivity descending */
15: return $En$ /* Return security-critical layer indices */
```

In this algorithm, the outer loop runs n times while the inner loop runs 10 times, resulting in a time complexity of O(n×10). The algorithm only retains the feature layer that has the most significant impact on model performance during each iteration, ultimately producing just a sorted list of feature layer importance. Therefore, its space complexity is O(1).

After selecting the target layer for encryption, a weight importance evaluation algorithm is applied to identify the most critical weights. Specifically, the parameter dimensions of a convolutional layer are C*N*K*K, where C, N, and K denote the number of input channels, output channels, and kernel size, respectively (typically 3*3 or 5*5). Due to structural differences across layers, the functional significance of individual kernels and their associated weights varies. Drawing inspiration from model pruning techniques, where less significant weights are removed to improve efficiency, we assess weight importance using the L1-norm. Since convolutional weights are multi-dimensional, they are first flattened into a one-dimensional array for ranking. The L1-norm values are then used to partially sort the weights. Based on a predefined encryption ratio, a selection function identifies the most impactful weights and their corresponding indices. The result is a list of selected weights and their positions, which defines the final encryption target.

After calculating the weight information to be encrypted, use the 4D Lorenz Hyperchaotic Map to scramble the weight position. The Eq. (1) is as follows:

$$\begin{cases} w_1' & = \alpha_1(w_2 - w_1) + w_2 w_3 \\ w_2' & = \gamma_1 w_1 - w_2 - w_1 w_3 + w_4 \\ w_3' & = w_1 w_2 - \beta_1 w_3 \\ w_4' & = \rho_1 w_4 - w_1 w_3 \end{cases} \tag{1}$$

where, w_1, w_2, w_3, w_4 are state variables that changes over time. w_i' denotes the derivative of time t. The parameters $\alpha_1, \beta_1, \gamma_1, \rho_1$ are the system parameters of the control system behaviour, which affect the dynamic characteristics of the system. When the parameter values are $\alpha_1 = 10, \beta_1 = 8/3, \gamma_1 = 28$ and $1.52 \leq \rho_1 \leq 0.06$, the system exhibits hyperchaotic behavior. In this paper, $\rho_1 = 1.0$ is taken. α_1 regulates the coupling strength between w_1 and w_2. γ_1 controls the strength of the response of w_2 to w_1, β_1 determines the decay rate of w_3, ρ_1 controls the growth rate of w_4. We use the Runge-Kutta-Fehlberg method (RKF45) to integrate the differential equations of the 4D Lorenz system and calculate four chaotic sequences X, Y, Z, and W with the same dimension as the encrypted parameters. The RKF45 method controls the error in the solution process by adaptively adjusting the step size. It is an explicit Runge-Kutta method with two different orders, usually fourth and fifth. At each time step, RKF45 calculates two estimates of the fifth and fourth order, and uses the difference between them to estimate the error to determine whether the step size needs to be adjusted to achieve the set accuracy. The following is the Runge-Kutta table in the general form of the RKF45 algorithm (Eq. (2)):

$$\begin{cases} k_1 = f\left(t_i, y_i\right) \\ k_2 = f\left(t_i + \frac{h}{4}, y_i + \frac{k_1}{4}\right) \\ k_3 = f\left(t_i + \frac{3h}{8}, y_i + \frac{3hk_1}{32} + \frac{9hk_2}{32}\right) \\ k_4 = f\left(t_i + \frac{12h}{13}, y_i + \frac{1932hk_1}{2197} - \frac{7200hk_2}{2197} + \frac{7296hk_3}{2197}\right) \\ k_5 = f\left(t_i + h, y_i + \frac{439hk_1}{216} - 8hk_2 + \frac{3680hk_3}{513} - \frac{845hk_4}{4104}\right) \\ k_6 = f\left(t_i + \frac{h}{2}, y_i - \frac{8hk_1}{27} + 2hk_2 - \frac{3544hk_3}{2565} + \frac{1859hk_4}{4104} - \frac{11hk_5}{40}\right) \end{cases} \tag{2}$$

where, t_i is the current time step, y_i is the estimated value of the current step, $f(t_i, y_i)$ represents the ordinary differential equation, which can usually be defined as $d_y/d_t = f(t, y)$, $k_1, k_2, k_3, k_4, k_5, k_6$ are intermediate values, and h is the set step size. The step size set in this paper is $h = 0.01$. Then, the linear combination of derivative terms can be obtained to calculate the Runge-Kutta approximation of the fourth-order y_{i+1}^4 and the fifth-order y_{i+1}^4 (Eq. (3)):

$$\begin{cases} y_{i+1}^{(5)} = y_i + h\left(\frac{16k_1}{135} + \frac{6656k_3}{12825} + \frac{28561k_4}{56430} - \frac{9k_5}{50} + \frac{2k_6}{55}\right) \\ y_{i+1}^{(4)} = y_i + h\left(\frac{25k_1}{216} + \frac{1408k_3}{2565} + \frac{2197k_4}{4104} - \frac{k_5}{5}\right) \end{cases} \tag{3}$$

The error estimate can be obtained by taking the difference between the two approximations (Eq. (4)):

$$E_i = y_{i+1}^{(5)} - y_{i+1}^{(4)} \tag{4}$$

Adjust the step size according to the error estimate to meet the predetermined error tolerance. If the error exceeds the tolerance, reduce the step size; if the error is within the tolerance, increase the step size appropriately. Next, sort the subscripts of the X, Y, Z, and W sequence elements in ascending (or descending) order according to their size, and obtain four new subscript sequences, as shown in the Eq. (5):

$$\begin{cases} Sh\{I_x, I_y, I_z, I_w\} = \xi(x), \xi(y), \xi(z), \xi(w) \\ x, y, z, w = \Phi(m_1, m_2, m_3, m_4) \end{cases} \tag{5}$$

where, Sh is a dictionary used to store new sequences, Φ is a hyperchaotic map, m_1, m_2, m_3, m_4 are the sizes of the encrypted areas, and the ξ function is used to rearrange the X, Y, Z, and W sequences. Use the sequences I_x, I_y, I_z, I_w in the dictionary Sh to confuse the parameter dimensions that need to be encrypted. Then use Sh to rearrange the parameters that need to be encrypted and generate a decryption key. The final position of the original parameter is determined by the four chaotic sequences, achieving highly nonlinear obfuscation of the encryption layer. The obfuscation equation (Eq. (6)) can be expressed as:

$$\begin{pmatrix} i' \\ j' \\ k' \\ p' \end{pmatrix} = \begin{pmatrix} l_x(i) \\ l_y(j) \\ l_z(k) \\ l_w(p) \end{pmatrix} \tag{6}$$

where, i', j', k', p' are the positions after parameter confusion, and i, j, k, p are the positions before parameter confusion. Original parameter at position i, j, k, p migrates to i', j', k', p' creating interleaved scattering across all dimensions simultaneously, this architecture ensures that even partial parameter leakage reveals no structural patterns.

In this design, even if an attacker obtains a model with scrambled parameter positions, they cannot achieve high inference accuracy without the correct decryption key. The obfuscated model remains resistant to brute-force attacks, effectively preventing the recovery of its original performance. This mechanism thus provides strong protection for DNN intellectual property, making it difficult for unauthorized users to exploit the model. Furthermore, since the chaotic sequence used for scrambling is deterministically generated from fixed initial conditions, it does not need to be transmitted along with the model. Only the initial parameters are required to regenerate the same sequence during decryption, significantly reducing communication overhead.

3.2.2 Amplitude Recovery Mask Weights

In order to further improve the security of the encryption parameters, we add a small perturbation to the selected encryption weight value to further increase the difficulty for attackers to crack. Specifically, the loss function is recorded as Loss, the weight of the convolution layer l is recorded as $W_l = [W_{l1}, W_{l2}, W_{l3}, \ldots]$, where $W_{li}, i \in (1, 2, 3, \ldots)$ is the convolution kernel parameter, the model input is recorded as x, and the output result is recorded as $\hat{y}$, The gradient obtained during back propagation is Eq. (7):

$$\nabla W_l Loss(x, \hat{y}) = \left[\frac{\partial Loss(x, \hat{y})}{\partial W_{l1}}, \frac{\partial Loss(x, \hat{y})}{\partial W_{l2}}, \ldots \right] \tag{7}$$

And calculate the parameter with the largest absolute value of the partial derivative (Eq. (8)):

$$\left|\frac{\partial \mathrm{Loss}(x,\hat{y})}{\partial W_{lt}}\right| = \max\left[\left|\frac{\partial \mathrm{Loss}(x,\hat{y})}{\partial W_{l1}}\right|, \left|\frac{\partial \mathrm{Loss}(x,\hat{y})}{\partial W_{l2}}\right|, \ldots\right] \tag{8}$$

After determining the encrypted weight W_{lt}, the Amplitude recovery maskper of the weight can be defined as Eq. (9):

$$per = \mu \times \mathrm{sign}\left[\frac{\partial \mathrm{Loss}(x,\hat{y})}{\partial w_{lt}}\right] \times [\max(W_l) - \min(W_l)] \tag{9}$$

where, μ is a hyperparameter with a very small value, which is used to control the perturbation per within a very small range. $\max(W_l)$ and $\min(W_l)$ are the maximum and minimum values of the weight of the encryption layer, respectively. The encryption weight W_{lt} is updated to $(W_{lt} + per)$, and the unselected weight value will not be updated. The $Crop()$ function is used to control the newly generated weight within a reasonable range so that it will not be detected by the attacker due to unreasonable weights. The public display is as follows (Eq. (10)):

$$\begin{gathered}\mathrm{Crop}(W_{lt} + per) = \begin{cases} W_{lt} + per, & W_{lt} \in [H_{l1}, H_{l2}] \\ Anew(W_{lt} + per), & W_{lt} < H_{l1} \\ Anew(W_{lt} + per), & W_{lt} > H_{l2} \end{cases} \\ H_{l1} = \min(W_l) + \varepsilon \times [\max(W_l) - \min(W_l)], \\ H_{l2} = \max(W_l) - \varepsilon \times [\max(W_l) - \min(W_l)]\end{gathered} \tag{10}$$

where, ε is a hyperparameter, H_{l1} and H_{l2} are two thresholds that control the range of weights after perturbation. Their function is to make the encrypted weights within a reasonable range after updating, and not too large or too small. The function $Anew(W_l t + per)$, will perform two steps. In order to ensure that the original weights can be correctly decrypted, before updating the weights, the perturbation value per must be updated to per', and then the weights are updated to W_{lt}. If the value of $(W_l t + per)$ happens to be in the interval $[H_{l1}, H_{l2}]$, the weights are not processed and are directly updated to the encrypted model. Otherwise, the parameters are updated according to the $Anew(W_l t + per)$ function. The definition of the $Anew(W_l t + per)$ function is as follows Eq. (11):

$$Anew(W_{lt} + per) : \begin{cases} per' = H_{l1} - W_{lt}, & W_{lt} < H_{l1} \\ W'_{lt} = H_{l1}, & W_{lt} < H_{l1} \\ per' = H_{l2} - W_{lt}, & W_{lt} > H_{l2} \\ W'_{lt} = H_{l2}, & W_{lt} > H_{l2} \end{cases} \tag{11}$$

where, per' and W'_{lt} are the updated perturbation values and weights. $per' = H_{l1} - W_{lt}, W_{lt} < H_{l1}$ Recalibrate perturbation to reach lower threshold, $W'_{lt} = H_{l1}, W_{lt} < H_{l1} Crop()$ weight to minimum allowable value. $per' = H_{l2} - W_{lt}, W_{lt} > H_{l2}$ Adjust perturbation to hit upper threshold, $W'_{lt} = H_{l2}, W_{lt} > H_{l2} Crop()$ weight to maximum allowable value.

3.2.3 Key Generation

The composite key in our framework comprises three security elements:

$$K = \underbrace{\{a, b, c, d\}}_{\text{Chaotic seeds}} \cup \underbrace{\{(plt, vlt)\}}_{l \in L} \cup \underbrace{\{S_l, W_l, H_l\}}_{\text{Layer metadata}} \tag{12}$$

where, $\{a, b, c, d\}$: Initial parameters of the 4D hyperchaotic map Φ, protected by AES-256 with 10^{14} possible combinations (NIST SP 800-22 validated), $\{(plt, vlt)\}$: Position-perturbation pairs where $plt = I_x(i) \| I_y(j) \| I_z(k) \| I_w(p)$ denotes the 4D permuted index, and $vlt = per'$ is the amplitude-adjusted perturbation from Eq. (11), $\{S_l, W_l, H_l\}$: Spatial dimensions of encrypted layer l for decryption alignment Key Usage Example: For a convolutional layer with $S \times W \times H \times C = 3 \times 3 \times 64 \times 128$, the key entry corresponding to position $(2, 1, 5, 33)$ would be: $K_{(2,1,5,33)} = \left(I_x(2) \| I_y(1) \| I_z(5) \| I_w(33),\ 0.127\right)$, where 0.127 is the calibrated perturbation bounded by $[H_{l1}, H_{l2}]$. Authorized users decrypt through:

$$W_{original} = \text{ARMDecode}(W_{encrypted} - vlt) \circ \Phi^{-1}(a, b, c, d) \tag{13}$$

While key transmission relies on established protocols (e.g., TLS 1.3 + RSA-4096), we mitigate interception risks through:

- Dynamic key rotation: Regenerate $\{a, b, c, d\}$ hourly via PUF (Physically Unclonable Functions)
- Key fragmentation: Distribute K across multiple secure enclaves (SGX/TEE)

The encryption process of the encryption algorithm proposed in this article is shown in Algorithm 2.

Algorithm 2: Encryption

Input: $Encrypted\ layer\ :\ L\ Corresponding\ weights\ :\ W_l = [W_{l1}, W_{l2}, W_{l3}, \ldots], (l \in L)\ Trained\ DNN\ model : F\ Chaotic\ System : \Phi$

Output: $Encrypted\ DNN\ model :\ \hat{F}$

```
1: function Encryption() /* Core encryption pipeline */
2: $x, y, z, w \leftarrow \Phi(LayerShape(L)[:4], a, b, c, d)$ /* Generate chaos sequences */
3: $I_x, I_y, I_z, I_w \leftarrow argsort(x, y, z, w)$ /* Get permutation indices by sorting chaotic values */
4: $S, W, H \leftarrow GetArea(L)$ /* Extract spatial dimensions (S:slice, W:width, H:height) */
5: for $i, j, k, p \in [range(S.shape[0]), range(S.shape[1]), range(S.shape[2]), range(S.shape[3])]$ do
6:     $L_z \leftarrow exchange(L[i, j, k, p], L[I_x(i), I_y(j), I_z(k), I_w(p)])$ /* Scramble weights */
7: end for
8: for $P_{valid} \in W_l$ do /* Process each weight parameter */
9:     $P_{ARM}, key_{ARM} \leftarrow ARMencode(P_{valid}, key)$ /* Apply amplitude recovery mask */
10:    $P'_{ARM} \leftarrow Crop(W_{lt} + per, key_{ARM})$
11:    $P' \leftarrow ARMDecode(P'_{ARM})$
12:    $L_z \leftarrow P'$
13: end for
14: $\hat{F} \leftarrow L_z$ /* Build final encrypted model */
15: $K \leftarrow [key, (a, b, c, d), (S, W, H)]$
16: return $\hat{F}$ /* Return encrypted model and secret key */
```

The overall time complexity of the algorithm is $O(nlogn) + O(n) + O(m) = O(nlogn + m)$, dominated by the sorting operation with $O(nlogn)$, while chaotic sequence generation and region computation (each $O(n)$) as well as weight amplitude recovery ($O(m)$) contribute to the remaining costs. The space required for generating chaotic sequences is O(n). The GetArea(L) operation returns a region whose size correlates with the input layer L's dimensions, assuming the returned region occupies O(n) space. In the weight scrambling operation, no new arrays are created—it merely swaps the positions of weight elements, resulting in O(1) space complexity. The amplitude recovery mask also demonstrates O(1) space complexity. Consequently, the total space complexity is O(n).

3.3 Model Decryption

Since the chaotic scrambling method selected is reversible, the decryption process is essentially the inverse process of encryption. As long as the transmitted key is correct, the original model can be quickly restored. First, subtract the added Amplitude recovery maskvalue to restore the encrypted weight value, and then generate the required chaotic sequence based on the initial value of the chaotic sequence, and perform chaotic scrambling in reverse order to restore the original position of the model and obtain the original DNN model. The decryption equation is as follows (Eq. (14)):

$$\begin{pmatrix} i \\ j \\ k \\ p \end{pmatrix} = \begin{pmatrix} l_x^-(i') \\ l_y^-(j') \\ l_z^-(k') \\ l_w^-(p') \end{pmatrix} \tag{14}$$

where, i', j', k', p' are the positions after parameter confusion, and i, j, k, p are the positions before parameter confusion. And $l_x^-, l_y^-, l_z^-, l_w^-$ are the inverse sequences of the encrypted chaotic sequence.

4 Experimental Results and Analysis

4.1 Experimental Setup

Our method is evaluated on an NVIDIA GeForce RTX 4060 Laptop GPU with CUDA12.2 and CUDNN11.6. The CUDA Toolkit and cuDNN libraries compatible with PyTorch1.13+cu116 are installed, and PyTorch is configured to support GPU acceleration. In addition, Vscode is installed as a code editor for Python3.9.13. In this paper, four datasets, namely CIFAR-10 [30], CIFAR-100 [30], Fashion-MNIST [31], and ImageNet [32], are selected as experimental datasets. First, seven classification networks, namely EfficientNet [33], MobileNetV2 [34], MobileNetV3 [34], ResNet-18 [35], ResNet-50 [35], Shuffle-Net [36], and VGG-16 [37], are selected for evaluation on the above four datasets.

4.2 Feature Layer Importance Judgment

This paper introduces a feature layer importance evaluation algorithm to prioritize layers with significant influence on model performance, thereby narrowing the encryption scope. In our initial analysis, we assessed all feature layers including convolutional, pooling, and fully connected layers and observed that convolutional layers have the most critical impact on overall model accuracy. Subsequent experiments focused on evaluating the relative importance of each convolutional layer, with the results presented in Fig. 2. We trained ResNet-18 and VGG-16 on the CIFAR-10 dataset, achieving over 90% classification accuracy. Each convolutional layer was indexed as i. Where $i \in (1, 2, 3, \ldots, n)$, denotes the total number of convolutional layers. In total, 19 and 13 convolutional layers were evaluated for ResNet-18 and VGG-16, respectively.

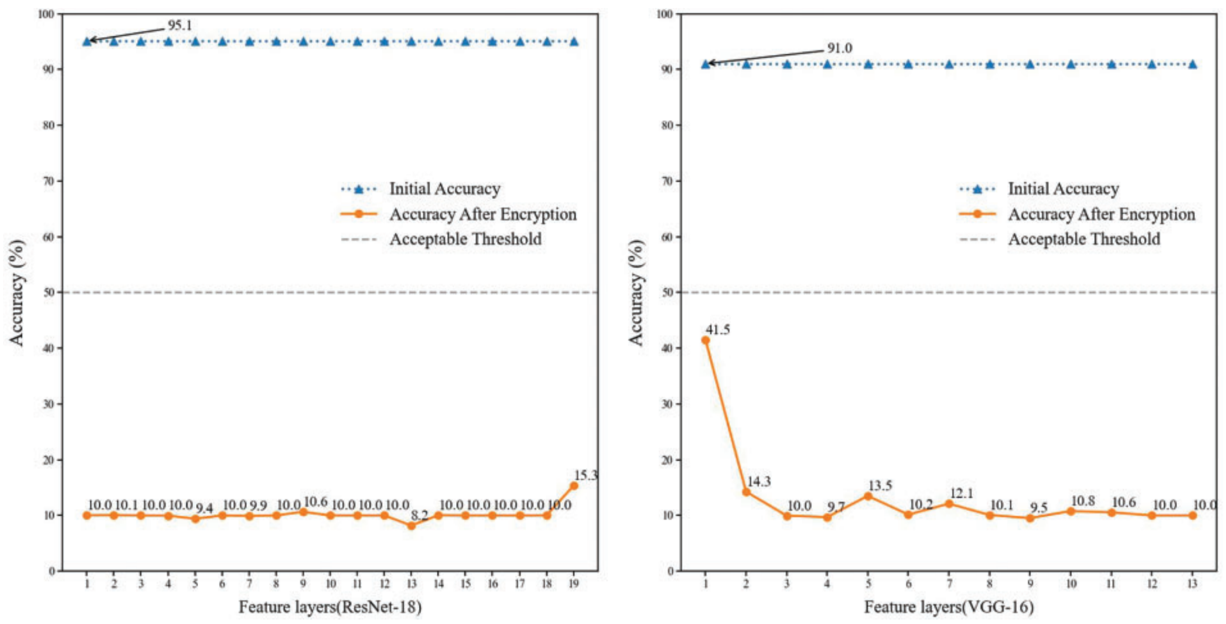

Figure 2: Analysis of the importance of feature layers

After selecting the layers to be encrypted, the number of encrypted weights is further reduced by arranging the L1 norm of each layer. The encryption ratio of each layer is set to 0.5% per layer. Different numbers of layers are selected for encryption according to the importance of the feature layer, and the performance changes of the model are observed. As shown in Fig. 3, in ResNet-18, only 0.5% of the parameters (184) of a convolutional layer need to be encrypted to reduce the model performance to 61.57%. Encrypting about 550 parameters of three layers can reduce the model accuracy to 19.59%, accounting for 0.016% and 0.049% of the total parameters, respectively. For the VGG-16 network, only 84 parameters of one layer need to be encrypted to reduce the model accuracy to 65.34%, and about 6600 parameters of three layers need to be encrypted to reduce the model accuracy to 43.95%, accounting for 0.0006% and 0.049% of the total parameters, respectively.

4.3 Encryption Effects

Based on the above methodology, the encryption layer ratio $\delta = i/n$ is set to 0.8, and the encryption ratio within each selected layer is set to 0.01. All subsequent experiments adopt these fixed parameters. Table 1 summarizes the encryption results across four datasets, where seven classification networks are evaluated per dataset. For each model, we report the original accuracy, the accuracy after encryption, and the accuracy after decryption. Notably, for datasets with 10 categories such as CIFAR-10 and Fashion-MNIST, the encrypted model accuracy drops to approximately 10.00%, close to random guessing. For CIFAR-100 and ImageNet-200 (a subset with 200 classes), encryption accuracy falls to around 1.00% and 0.50%, respectively—also near the expected random baseline. These results demonstrate that the proposed method is highly stable across various datasets and architectures, exhibiting strong robustness.

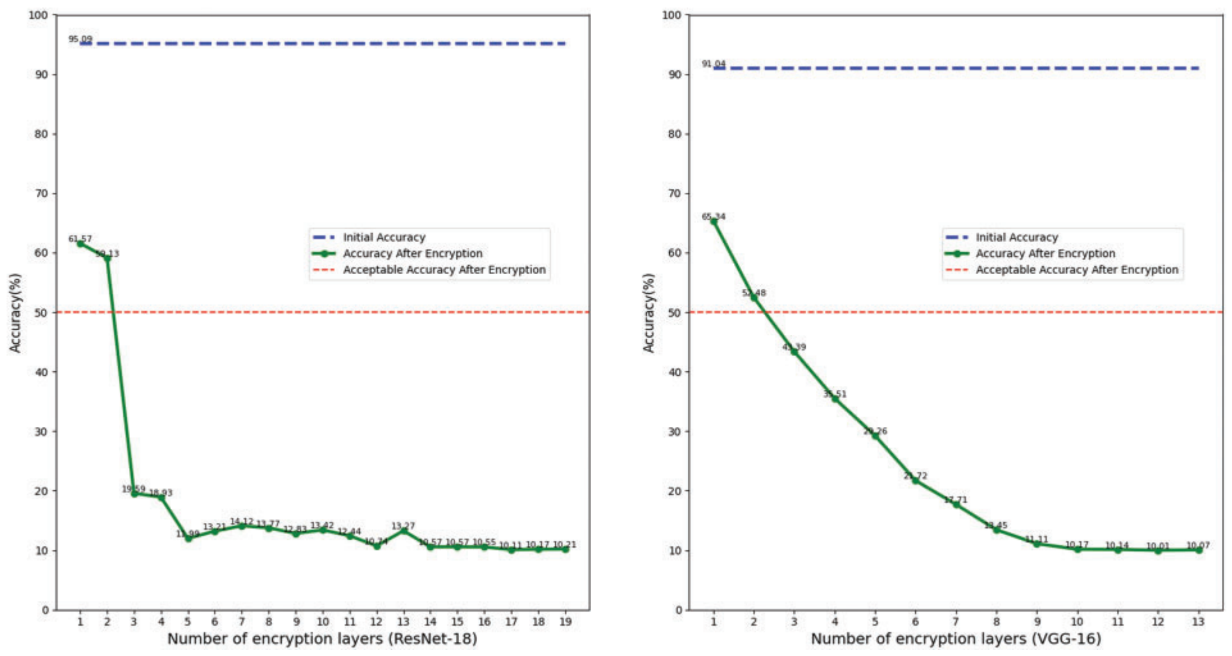

Figure 3: Selection of encryption feature layers initial accuracy represents the original accuracy of the model, accuracy after encryption represents the accuracy of the model after encryption, and acceptable accuracy after encryption represents the acceptable degree of accuracy reduction after encryption

Table 1: Encryption effects of different networks

Dataset	Type	Efficient	MobileV2	MobileV3	Res-18	Res-50	Shuffle	VGG-16
Cifar-10	Original	94.34%	95.13%	95.09%	95.09%	96.44%	96.26%	91.04%
	Encryption	9.92%	10.85%	10.59%	10.00%	10.00%	9.98%	13.43%
	Restored	94.34%	95.13%	95.09%	95.09%	96.44%	96.26%	91.04%
Cifar-100	Original	84.18%	81.37%	81.55%	80.59%	82.75%	82.86%	80.26%
	Encryption	0.87%	0.98%	0.88%	1.00%	1.00%	1.03%	1.21%
	Restored	84.18%	81.36%	81.55%	80.59%	82.75%	82.86%	80.26%
Fashion-MNIST	Original	94.57%	92.68%	92.91%	92.33%	91.56%	90.48%	92.05%
	Encryption	10.12%	10.03%	9.86%	10.52%	10.00%	10.00%	10.49%
	Restored	94.56%	92.68%	92.92%	92.33%	91.56%	90.48%	86.68%
Imagenet	Original	94.51%	86.02%	89.1%	84.31%	89.18%	86.85%	82.54%
	Encryption	0.52%	0.50%	0.45%	0.76%	0.43%	0.53%	0.39%
	Restored	94.51%	86.02%	89.11%	84.31%	89.18%	86.85%	82.54%

Furthermore, due to the reversibility of the proposed encryption scheme, model performance after decryption is nearly identical to the original. As shown in Table 1, the fluctuation in accuracy is typically within 0.01%, with only a few exceptional cases (highlighted in bold) showing minimal deviation. This indicates that the scheme introduces negligible performance loss, making it suitable for most DNNs unless extreme precision is required. Additionally, Table 2 reports the storage overhead of encrypted model files. The file sizes before and after encryption remain nearly unchanged, confirming the method's lightweight design.

Table 2: The storage space occupied by the model weights before and after encryption

	Shuffle	MobileV2	MobileV3	Res-18	Res-50	Efficient	VGG-16
Original (KB)	208038	8972	16659	43755	92218	21194	524626
Encryption (KB)	208041	8973	16660	43756	92219	21195	524628
Extra storage (%)	0.0014	0.0111	0.0601	0.0029	0.0011	0.0047	0.0004

4.4 Model Decryption Time

For a user-oriented network model, service response time is a criterion for measuring user experience. Therefore, the encryption algorithm should have a high decryption efficiency while protecting the intellectual property rights of the model owner to ensure a good user experience. Table 3 shows the decryption time of some encrypted feature layers of three networks, ResNet-18, VGG-16, and MobileNetV2, respectively, under the CIFAR-100 and CIFAR-10 datasets, as well as the accuracy changes of the encrypted model at this layer. The encryption ratio of the CIFAR-100 dataset used by ResNet-18 is set to 3%. It can be seen that the decryption model takes very little time, all within 1 second, but it is worth noting that the decryption time is affected by many factors such as the hardware conditions of the test device and the operating status of the device, and is also affected by the number of encrypted weights of the model. In different scenarios, the decryption time will vary. The accuracy of the model after encryption remains at a low level. After decryption, the model accuracy is basically not lost, which can well meet the various service needs of users.

Table 3: Decryption time of different networks

Network	Layer name	Layer shape	Decryption time (s)	Encryption accuracy	Decryption accuracy
ResNet-18 (Original accuracy: 80.%)	Layer1.0.conv1	[64, 3, 7, 7]	0.24	2.0%	80.6%
	Layer2.0.conv1	[128, 64, 3, 3]	0.51	31.0%	80.6%
	Layer2.0.conv2	[128, 128, 3, 3]	0.69	36.3%	80.6%
	Layer2.0.downsample.0	[128, 64, 1, 1]	0.11	35.1%	80.6%
VGG-16 (Original accuracy: 91.0%)	Features.[0]	[64, 3, 3, 3]	0.01	32.7%	91.0%
	Features.[2]	[64, 64, 3, 3]	0.21	31.6%	91.0%
	Features.[5]	[128, 64, 3, 3]	0.37	24.2%	91.0%
	Features.[7]	[128, 128, 3, 3]	0.71	39.3%	91.0%
MobileV2 (Original accuracy: 95.1%)	Features.0.0	[32, 3, 3, 3]	0.01	15.1%	95.1%
	Features.1.0	[32, 1, 3, 3]	0.01	15.8%	95.1%
	Features.2.0	[96, 16, 1, 1]	0.02	24.6%	95.1%
	Features.3.1	[144, 1, 3, 3]	0.02	26.3%	95.1%

4.5 Parameter Distribution Analysis Attacks

The attacker can infer which weights are encrypted based on the weight distribution of the model or the variance of the model weights. For this reason, this paper adds an adversarial perturbation to the encrypted weights so that the weight distribution and variance of the encrypted model remain basically unchanged. As shown in Fig. 4, this is an encrypted convolutional layer from ResNet-18 and VGG-16. The two figures on the left are the weight distributions of ResNet-18 and VGG-16 before encryption, and the two figures on the right are the weight distributions of ResNet-18 and VGG-16 after encryption. The two vertical lines represent the mean and variance of the model, respectively. It can be seen that the mean and variance changes before and after encryption are both in the range of 10^{-6}. These experimental results show that the encrypted weights are invisible to attackers, and it is difficult for attackers to determine the location of the encrypted layer and the encrypted weights by analyzing the abnormality of the weights, that is, it is difficult to detect anomalies for these encrypted layers by analyzing weight changes.

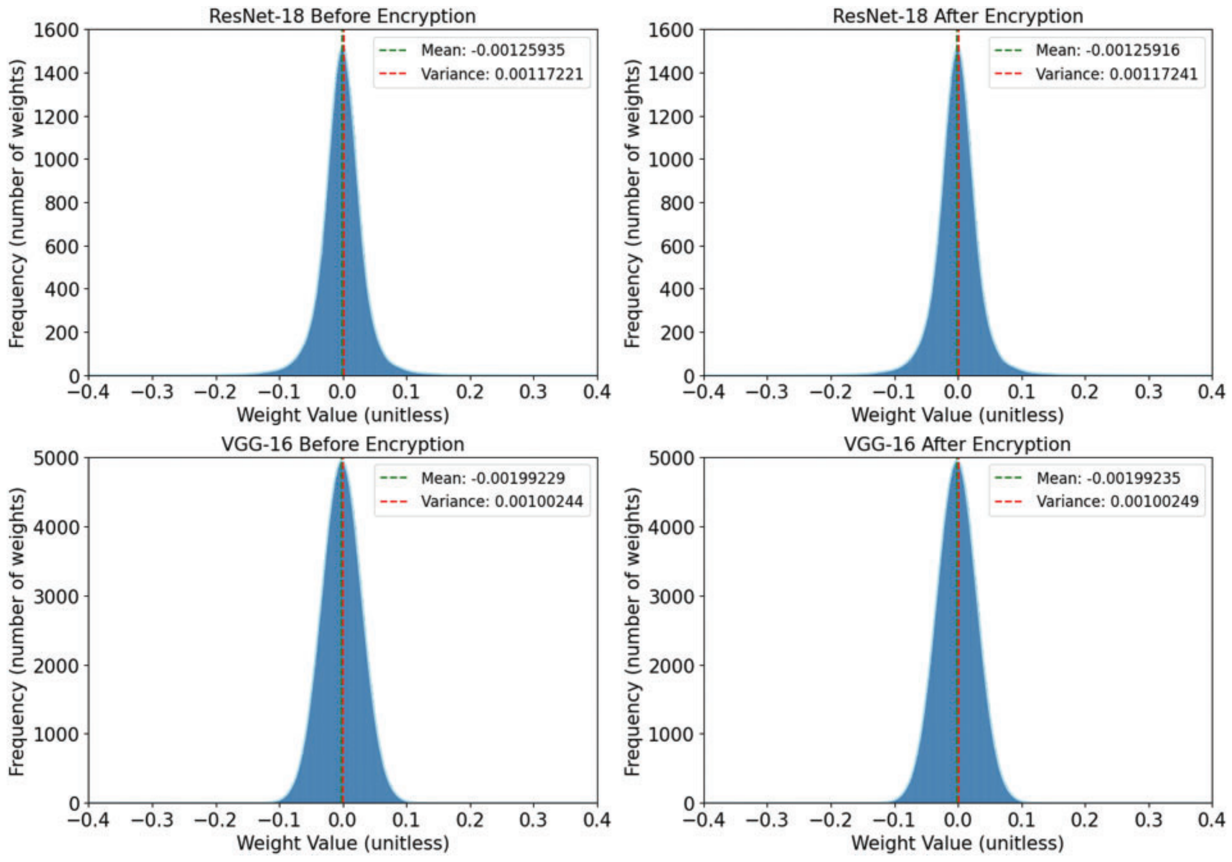

Figure 4: Weight distribution histogram mean represents the mean of the model weights, and variance represents the variance of the model weights

4.6 Model Fine-Tuning Attack

4.6.1 Fine-Tuning with Limited Data

Since the encryption model can be obtained from the IoT system by all devices, an attacker can fine-tune the encryption model to invalidate the encryption and thus obtain the services of the original model. In the experiment, ShuffleNet, MobileNetV2, and MobileNetV3 use the CIFAR-10 dataset, while EfficientNet, ResNet-18, ResNet-50, and VGG-16 use the CIFAR-100 dataset for testing. 10% of the data in the test sets of the two datasets are selected for model fine-tuning attacks, and the remaining 90% of the data in the test sets are used as validation sets. The learning rate lr is set to 0.00001 to evaluate the change in the inference accuracy of the model when it is fine-tuned. The results are shown in Table 4. From the results, even after 100 epochs of reasoning, the reasoning accuracy of the fine-tuned encrypted model remains at a low level. Among the three models fine-tuned on CIFAR-10, only MobileNetV2 has a reasoning accuracy of 28.24%, while the accuracy of ShuffleNet and MobileNetV3 is only 11.47% and 14.82%, respectively. It is very close to the random selection accuracy of CIFAR-10 of 10%. The highest accuracy of the four networks on CIFAR-100 is only 18.31%, and the four models of ShuffleNet, EfficientNet, ResNet-50, and VGG-16 have already achieved the best fine-tuning accuracy before reaching 100 epochs, which means that even if the number of epochs continues to increase, the accuracy of the model will not be improved, which further illustrates the robustness of the encrypted model to fine-tuning attacks. Since the training cost of the entire original model is high and the attacker cannot have the private dataset of the model owner, fine-tuning the encrypted model with a small dataset cannot obtain the original accuracy of the model. Fine-tuning the model with a large dataset is equivalent to retraining the model, the fine-tuning attack loses its meaning.

Table 4: Changes in model accuracy under fine-tuning attack

Epo	Shuffle	MobileV2	MobileV3	Res-18	Res-50	Efficient	VGG-16
10	9.85%	13.27%	8.62%	5.50%	7.73%	13.93%	7.47%
20	11.66%	16.28%	9.85%	6.90%	8.55%	16.17%	12.32%
30	11.37%	19.34%	9.57%	7.53%	9.45%	17.62%	13.50%
40	11.37%	21.25%	10.64%	8.22%	9.64%	18.05%	13.38%
50	11.40%	23.01%	12.06%	8.30%	9.87%	18.15%	13.38%
60	11.16%	24.73%	12.72%	8.60%	9.71%	18.31%	13.37%
70	11.25%	25.47%	13.32%	8.71%	10.05%	18.25%	13.38%
80	11.41%	26.44%	13.32%	8.90%	9.91%	18.25%	13.37%
90	11.35%	27.35%	14.07%	9.12%	9.94%	18.01%	13.38%
100	11.47%	28.24%	14.82%	9.20%	9.81%	18.15%	13.37%

4.6.2 Increase the Fine-tuning Ratio

To further verify the model's resistance to fine-tuning attacks, we assume that the attacker has a large number of annotated images for fine-tuning the model. Use a larger learning rate, set the learning rate lr to 0.001, and set a dataset ratio p, $p \in (0.2, 0.3, 0.4, 0.5, 0.6)$. Use the same dataset for 100 epochs of attacks. As shown in Table 5, the model inference accuracy of the 7 networks after 100 epochs of attacks on the training datasets with different proportions. It can be seen that when $p = 0.2$, the inference accuracy of several networks is still below 50%, and even when $p = 0.6$, the model inference accuracy of the four networks tested using CIFAR-100 after 100 epochs of attacks is still below 50%, and the model accuracy of the model after 100 epochs of attacks on ShuffleNet is only 57.62%, which is still a big gap compared to the original accuracy of the model of 96.26%, which fully demonstrates the robustness of the proposed algorithm in resisting model fine-tuning attacks.

Table 5: Accuracy of model fine-tuning attack under different training ratios

Net	$p = 0.2$	$p = 0.3$	$p = 0.4$	$p = 0.5$	$p = 0.6$
Shuffle	44.50%	50.37%	53.31%	55.96%	57.62%
MobileV2	46.80%	48.22%	52.78%	57.24%	56.60%
MobileV3	37.45%	41.70%	44.75%	48.32%	50.90%
Res-18	30.40%	37.72%	40.13%	40.36%	45.72%
Res-50	29.20%	32.97%	37.58%	38.06%	44.90%

(Continued)

Table 5 (continued)

Net	p = 0.2	p = 0.3	p = 0.4	p = 0.5	p = 0.6
Efficient	30.06%	36.18%	42.43%	44.66%	47.95%
VGG-16	13.31%	13.51%	15.95%	18.60%	19.55%

4.7 Model Pruning Attack

The attacker may prune the encrypted area of the model by pruning the model to obtain the original accuracy of the model. In this experiment, CIFAR-10 was used to perform pruning attacks on three networks, EfficientNet, MobileNetV2, and ResNet-18, while MobileNetV2, ResNet-50, and VGG-16 used the CIFAR-100 dataset. The pruning rate of the model was set from 0.0 to 0.7, where 0.0 represents the accuracy of the encrypted model and 0.7 represents a pruning rate of 70%. The results are shown in Table 6. Regardless of the pruning rate, the reasoning accuracy of the encrypted model remains at an extremely low level, close to the random selection accuracy. When the pruning rate is 0.5, several networks achieve the best pruning effect. If the pruning rate is further increased, the model reasoning accuracy is even lower. After pruning, the model reasoning accuracy is even lower than that after encryption. This shows that the model pruning attack not only does not prune the important weights of the encrypted model, but also prunes some less important weights that are not encrypted, resulting in a further reduction in the reasoning accuracy of the model. This shows that the algorithm we proposed does not continuously encrypt a series of adjacent weights, but selectively encrypts model weights, which can effectively resist model pruning attacks.

Table 6: Changes in model accuracy under pruning attack

Pruning rate	Efficient	MobileV2	Res-18	MobileV3	Res-50	VGG-16
0.0	9.92%	10.15%	10.00%	0.88%	1.00%	1.21%
0.1	9.62%	11.52%	10.18%	0.99%	1.00%	1.24%
0.2	10.34%	11.11%	10.22%	0.88%	1.00%	1.34%
0.3	9.60%	9.82%	10.22%	0.95%	1.00%	1.38%
0.4	10.50%	10.12%	10.52%	0.96%	0.98%	1.08%
0.5	9.86%	10.08%	10.08%	0.99%	0.61%	1.22%
0.6	9.78%	10.14%	10.00%	0.93%	1.00%	1.10%
0.7	9.65%	9.97%	10.00%	0.93%	1.00%	1.17%

4.8 Ablation Experiments

We performed ablation experiments and the results are shown in Table 7. The first row is the accuracy of the original model, the second row represents the accuracy of the encryption model, and the third row is the result of our random selection of the same number of weight encryption models.

4.9 Comparison with SOTA Methods

We compare the proposed method with existing active DNN IP protection schemes. As shown in Table 8, we give the encryption effects of these schemes on Fashion-MNIST and cifar-10. The scheme proposed by Pyone et al. [16] requires image preprocessing when training and using the model, which undoubtedly consumes a lot of time, and retraining also consumes a lot of computing resources. The scheme proposed by Zhou et al. [23] also requires the full parameters of the cryptographic model, which cannot meet the needs of commercial applications. For work [21], attackers may find out the information of the model encryption weights through reverse engineering of the model. The scheme of work [22] not only requires retraining the model, but also requires the support of hardware equipment, which will also consume a lot of costs in commercial applications. Compared with previous work, the scheme proposed in this paper not only does not require retraining the model, but also does not require hardware equipment support, and decryption only takes a very short time, which can well meet the needs of commercial applications.

Table 7: Ablation experiments

	Shuffle	**MobileV2**	**MobileV3**	**Res-18**	**Res-50**	**Efficient**	**VGG-16**
Original	94.34%	95.13%	95.09%	95.09%	96.44%	96.26%	91.04%
Encryption	9.92%	10.85%	10.59%	10.00%	10.00%	9.98%	13.43%
Random	91.55%	83.21%	76.04%	94.48%	94.86%	93.95%	90.36%

Table 8: Comparison of different methods

Work	**Dataset**	**Authorized accuracy**	**Unauthorized accuracy**	**ACC drop**	**Require additional training**	**Require hardware support**
Encryption -based [16]	MNIST	–	–	–	Yes	No
	cifar-10	92.26%	20.01%	72.25%		
NNSplitter module -based [23]	MNIST	93.71%	11.17%	82.54%	Yes	No
	cifar-10	93.07%	11.17%	81.90%		
Passports-based [21]	MNIST	–	–	–	Yes	No
	cifar-10	90.89%	10.00%	80.89%		
Hardware device -based [22]	MNIST	89.93%	10.05%	79.88%	Yes	Yes
	cifar-10	89.54%	9.37%	80.17%		
The proposed method	MNIST	92.33%	10.52%	81.81%	No	No
	cifar-10	95.09%	10.00%	85.09%		

Note: '–' indicates that the cited work did not conduct the corresponding experiments.

5 Conclusion

This paper presents a Model Active Protection Framework that secures DNN models at the source by encrypting only a small subset of critical weights. These encrypted weights remain statistically indistinguishable from normal values, making their locations hard to detect. The framework ensures efficient decryption with minimal computational overhead, enabling practical deployment. Experimental results show that, after encryption, model accuracy drops to near-random levels across seven classification networks and four benchmark datasets, effectively preventing unauthorized exploitation. Moreover, the scheme demonstrates strong resistance to common attacks such as model fine-tuning and pruning, underscoring its robustness. In summary, the proposed method offers a lightweight, resilient, and practical solution for protecting the integrity and confidentiality of DNN models in real-world scenarios.

Acknowledgement: Thank all the members who have contributed to this work with us.

Funding Statement: This work was supported in part by the National Natural Science Foundation of China under Grant No. 62172280, in part by the Key Scientific Research Projects of Colleges and Universities in Henan Province, China under Grant No. 23A520006, and in part by Henan Provincial Science and Technology Research Project under Grant No. 222102210199.

Author Contributions: The authors confirm the following contributions to this article: Research concept and design: Xintao Duan and Yinhang Wu; Experimental, analytical, and interpretive results: Yinhang Wu and Zhao Wang; Drafted by Xintao Duan. Chuan Qin reviewed the results and approved the final version of the manuscript. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: Data openly available in a public repository.

Ethics Approval: Not the study included human or animal subjects.

Conflicts of Interest: The authors declare no conflicts of interest to report regarding the present study.

References

1. Malhotra R, Singh P. Recent advances in deep learning models: a systematic literature review. Multimed Tools Appl. 2023;82(29):44977–5060. doi:10.1007/s11042-023-15295-z.
2. Peng S, Chen Y, Xu J, Chen Z, Wang C, Jia X. Intellectual property protection of DNN models. World Wide Web. 2023;26(4):1877–911. doi:10.1007/s11280-022-01113-3.
3. Li Z, Hu C, Zhang Y, Guo S. How to prove your model belongs to you: a blind-watermark based framework to protect intellectual property of DNN. In: Proceedings of the 35th Annual Computer Security Applications Conference; 2019 Dec 9–13; San Juan, Puerto Rico. p. 126–37.
4. Furukawa R, Sakazawa S. Generation management of white-box DNN model watermarking. In: 2023 IEEE 12th Global Conference on Consumer Electronics (GCCE); 2023 Oct 10–13; Nara, Japan. p. 792–3.
5. Chen H, Zhang W, Liu K, Chen K, Fang H, Yu N. Speech pattern based black-box model watermarking for automatic speech recognition. In: ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP); 2022 May 22–27; Singapore. p. 3059–63.
6. Li Y, Wang H, Barni M. A survey of deep neural network watermarking techniques. Neurocomputing. 2021;461:171–93. doi:10.1016/j.neucom.2021.07.051.
7. Ogundokun RO, Abikoye CO, Kumar Sahu A, Akinrotimi AO, Babatunde AN, Sadiku PO, et al. Enhancing security and ownership protection of neural networks using watermarking techniques: a systematic literature review using PRISMA. In: Multimedia watermarking. 1st ed. Singapore: Springer; 2024. p. 1–28. doi:10.1007/978-981-99-9803-6_1.
8. Zhang J, Chen D, Liao J, Zhang W, Feng H, Hua G et al. Deep model intellectual property protection via deep watermarking. IEEE Trans Pattern Anal Mach Intell. 2021;44(8):4005–20. doi:10.1109/tpami.2021.3064850.
9. Ye Z, Zhang X, Feng G. Deep neural networks watermark via universal deep hiding and metric learning. Neural Comput Appl. 2024;36(13):7421–38. doi:10.1007/s00521-024-09469-5.
10. Uchida Y, Nagai Y, Sakazawa S, Satoh S. Embedding watermarks into deep neural networks. In: Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval; 2017 Jun 6–9; Bucharest, Romania. p. 269–77.
11. Chen H, Rohani BD, Koushanfar F. Deepmarks: a digital fingerprinting framework for deep neural networks. arXiv:1804.03648. 2018.
12. Wang J, Wu H, Zhang X, Yao Y. Watermarking in deep neural networks via error back-propagation. Electron Imaging. 2020;32(4):1–9. doi:10.2352/issn.2470-1173.2020.4.mwsf-022.
13. Adi Y, Baum C, Cisse M, Pinkas B, Keshet J. Turning your weakness into a strength: watermarking deep neural networks by backdooring. In: 27th USENIX Security Symposium (USENIX Security 18); 2018 Aug 15–17; Baltimore, MD, USA. p. 1615–31.

14. Le Merrer E, Perez P, Trédan G. Adversarial frontier stitching for remote neural network watermarking. Neural Comput Appl. 2020;32(13):9233–44. doi:10.1007/s00521-019-04434-z.
15. Li M, Zhong Q, Zhang LY, Du Y, Zhang J, Xiang Y. Protecting the intellectual property of deep neural networks with watermarking: the frequency domain approach. In: 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom); 2020 Dec 29–2021 Jan 1; Guangzhou, China. p. 402–9.
16. Pyone A, Maung M, Kiya H. Training DNN model with secret key for model protection. In: 2020 IEEE 9th Global Conference on Consumer Electronics (GCCE); 2020 Oct 13–16; Kobe, Japan. p. 818–21.
17. Ren G, Wu J, Li G, Li S, Guizani M. Protecting intellectual property with reliable availability of learning models in AI-based cybersecurity services. IEEE Trans Dependable Secure Comput. 2022;21(2):600–17. doi:10.1109/tdsc.2022.3222972.
18. Luo Y, Feng G, Zhang X. Hierarchical authorization of convolutional neural networks for multi-user. IEEE Signal Process Lett. 2021;28:1560–4. doi:10.1109/lsp.2021.3100307.
19. Tian J, Zhou J, Duan J. Hierarchical services of convolutional neural networks via probabilistic selective encryption. IEEE Trans Serv Comput. 2021;16(1):343–55. doi:10.1109/tsc.2021.3136601.
20. Pan Q, Dong M, Ota K, Wu J. Device-bind key-storageless hardware AI model IP protection: a PUF and permute-diffusion encryption-enabled approach. arXiv:2212.11133. 2022.
21. Fan L, Ng KW, Chan CS, Yang Q. DeepIPR: deep neural network ownership verification with passports. IEEE Trans Pattern Anal Mach Intell. 2022;44(10):6122–39. doi:10.1109/tpami.2021.3088846.
22. Chakraborty A, Mondai A, Srivastava A. Hardware-assisted intellectual property protection of deep learning models. In: 2020 57th ACM/IEEE Design Automation Conference (DAC); 2020 Jul 20–24; San Francisco, CA, USA. p. 1–6.
23. Zhou T, Luo Y, Ren S, Xu X. NNSplitter: an active defense solution for DNN model via automated weight obfuscation. In: ICML'23: International Conference on Machine Learning; 2023 Jul 23–29; Honolulu, HI, USA. p. 42614–24.
24. Zhu H, Zhang X, Yu H, Zhao C, Zhu Z. An image encryption algorithm based on compound homogeneous hyper-chaotic system. Nonlinear Dyn. 2017;89(1):61–79. doi:10.1007/s11071-017-3436-y.
25. Li Z, Peng C, Li L, Zhu X. A novel plaintext-related image encryption scheme using hyper-chaotic system. Nonlinear Dyn. 2018;94(2):1319–33. doi:10.1007/s11071-018-4426-4.
26. Lin N, Chen X, Lu H, Li X. Chaotic weights: a novel approach to protect intellectual property of deep neural networks. IEEE Trans Comput-Aided Des Integr Circuits Syst. 2020;40(7):1327–39. doi:10.1109/tcad.2020.3018403.
27. Mohseni A, Moaiyeri MH, Amirany A, Rezayati MH. Protecting the intellectual property of binary deep neural networks with efficient spintronic-based hardware obfuscation. IEEE Trans Circuits Syst I: Regul Pap. 2024;71(7):3146–56. doi:10.1109/tcsi.2024.3397925.
28. Xue M, Wu Y, Zhang LY, Gu D, Zhang Y, Liu W. SSAT: active authorization control and user's fingerprint tracking framework for DNN IP protection. ACM Trans Multimed Comput Commun Appl. 2024;20(10):1–24. doi:10.1145/3679202.
29. Li P, Huang J, Wu H, Zhang Z, SecureNet Qi C. Proactive intellectual property protection and model security defense for DNNs based on backdoor learning. Neural Netw. 2024;174(3):106199. doi:10.1016/j.neunet.2024.106199.
30. Krizhevsky A, Hinton G. Learning multiple layers of features from tiny images. Technical Report. Toronto, ON, Canada: University of Toronto; 2009. [cited 2025 Mar 30]. Available from: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.
31. Xiao H, Rasul K, Vollgraf R. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. arXiv:1708.07747. 2017.
32. Deng J, Dong W, Socher R, Li LJ, Li K, Fei-Fei L. ImageNet: a large-scale hierarchical image database. In: 2009 IEEE Conference on Computer Vision and Pattern Recognition; 2009 Jun 20–25; Miami, FL, USA. p. 248–55.
33. Tan M, Le Q. EfficientNet: rethinking model scaling for convolutional neural networks. In: International Conference on Machine Learning; 2019 Jun 9–15; Long Beach, CA, USA. p. 6105–14.

34. Sandler M, Howard A, Zhu M, Zhmoginov A, Chen LC. MobileNetV2: inverted residuals and linear bottlenecks. In: Proceedings of the 2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR); 2018 Jun 18–23; Salt Lake City, UT, USA. [cited 2025 Mar 30]. Available from: https://arxiv.org/abs/1801.04381.
35. He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. In: Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition; 2016 Jun 27–30; Las Vegas, NV, USA. p. 770–8.
36. Zhang X, Zhou X, Lin M, Sun J. ShuffleNet: an extremely efficient convolutional neural network for mobile devices. In: Proceedings of the 2018 IEEE Conference on Computer Vision and Pattern Recognition; 2018 Jun 18–23; Salt Lake City, UT, USA. p. 6848–56.
37. Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. arXiv:1409.1556. 2014.