Privacy-preserving and verifiable convolution neural network inference and training in cloud computing

Cao, Wei; Shen, Wenting; Qin, Jing; Lin, Hao

DOI
10.1016/j.future.2024.107560

Publication date
2025

Document Version
Final published version

Published in
Future Generation Computer Systems

Future Generation Computer Systems 164 (2025) 107560

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Privacy-preserving and verifiable convolution neural network inference and training in cloud computing

Wei Cao [a], Wenting Shen [a,*], Jing Qin [b], Hao Lin [c]

[a] College of Computer Science and Technology, Qingdao University, Qingdao, 266071, China
[b] School of Mathematics, Shandong University, Jinan, 250100, China
[c] Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, Delft, 2628 XE, The Netherlands

ARTICLE INFO

Keywords:
Integrity verification
Privacy-preserving
Cloud computing
Convolutional neural network

ABSTRACT

With the rapid development of cloud computing, outsourcing massive data and complex deep learning model to cloud servers (CSs) has become a popular trend, which also brings some security problems. One is that the model stored in the CSs may be corrupted, leading to incorrect inference and training results. The other is that the privacy of outsourced data and model may be compromised. However, existing privacy-preserving and verifiable inference schemes suffer from low detection probability, high communication overhead and substantial computational time. To solve the above problems, we propose a privacy-preserving and verifiable scheme for convolutional neural network inference and training in cloud computing. In our scheme, the model owner generates the authenticators for model parameters before uploading the model to CSs. In the phase of model integrity verification, model owner and user can utilize these authenticators to check model integrity with high detection probability. Furthermore, we design a set of privacy-preserving protocols based on replicated secret sharing for both the inference and training phases, significantly reducing communication overhead and computational time. Through security analysis, we demonstrate that our scheme is secure. Experimental evaluations show that the proposed scheme outperforms existing schemes in privacy-preserving inference and model integrity verification.

1. Introduction

Recently, deep neural networks (DNNs) have become an integral part of artificial intelligence technology and are widely used in many applications such as image processing [1], speech recognition [2], fingerprint recognition [3], and machine translation [4]. However, the explosive growth of data and the rising complexity of neural network structure have presented significant challenges to local servers with limited computational resource [5]. Fortunately, with the advancement of cloud computing, outsourcing data and neural network model to cloud servers (CSs) has become a popular and feasible approach that is able to effectively mitigate the storage and computational demands on local servers [6–8].

Although cloud computing offers many benefits, there are also many security and privacy risks that are not able to be overlooked [9–11]. For instance, once outsourcing model to the CSs, model owner (MO) will relinquishe physical control over the model. This means that if the model stored in the CSs is corrupted due to uncontrollable factors, such as software/hardware failures or natural disasters [12,13], the CSs may return incorrect inference and training results. These incorrect results will lead to severe consequences in certain fields, such as face recognition [14], health monitoring [15], and financial risk assessment [16]. Hence, it is essential to verify the model's integrity to guarantee the correctness of inference and training results returned by CSs.

Furthermore, the data used for inference and training often encompasses user's private information, such as geographical location and health data. If these data is exposed to CSs, the confidentiality of the user's personal information will be compromised [17,18]. Additionally, a well-trained neural network model holds significant value for its owner due to the substantial training costs involved [19,20]. Thus, it is necessary to preserve the privacy of the user data and private model.

To preserve the privacy of the model and the user's personal information, some privacy-preserving schemes based on homomorphic encryption (HE) [21,22] and secret sharing (SS) [23,24] have been proposed. These schemes [21–24] enable CSs to provide inference and training services without obtaining the private information of user data and model. However, these schemes [21–24] are unfriendly to the devices with limited computational resources. The reasons are that most HE-based schemes entail substantial computation cost due to complex

* Corresponding author.
E-mail addresses: focuscw1996@163.com (W. Cao), shenwentingmath@163.com (W. Shen), qinjing@sdu.edu.cn (J. Qin), h.lin-3@tudelft.nl (H. Lin).

https://doi.org/10.1016/j.future.2024.107560
Received 8 March 2024; Received in revised form 9 July 2024; Accepted 12 October 2024
Available online 19 October 2024

encryption and decryption operations, while most SS-based schemes incur high communication overhead during dot product operations. Thus, how to achieve privacy-preserving with low computation and communication overheads is a significant challenge. Moreover, currently, only one existing scheme [22] supports both privacy-preserving and model integrity verification. However, this scheme [22] only considers privacy-preserving inference, but not privacy-preserving training. Furthermore, this scheme [22] requires high computation overhead to achieve privacy-preserving due to its reliance on HE technique. Therefore, how to achieve efficient privacy-preserving inference and training while ensuring model integrity is another challenge.

In this paper, we construct a novel convolutional neural network (CNN) scheme supporting model integrity verification, as well as privacy-preserving inference and training. Specifically, MO and user can verify model integrity. Three CSs are used to cooperatively execute the privacy-preserving protocols and return the inference and training results to the user and the MO. A summary of our contributions can be found below:

(1) We design a verifiable and privacy-preserving CNN inference and training scheme in cloud computing. In the proposed scheme, the model integrity can be guaranteed by verifying model integrity through MO or user. User is unable to obtain the model privacy during the model integrity verification phase. To ensure privacy protection, the user and MO employ the replicated secret sharing (RSS) technique to securely encrypt both the data and model, then outsource the encrypted data and model to three CSs for inference and training. To the best of our knowledge, this is the first work that achieves model integrity verification, as well as privacy-preserving inference and training for CNN.
(2) Different from the previous works, we generate the authenticators for the shares of vectorized parameters in the model, which are utilized to verify model integrity during the verification phase. Additionally, we design a random masking technique to blind the verification proofs generated by the CSs, thereby preventing user from inferring the real model from the verification proofs. We also develop a series of privacy-preserving protocols based on RSS technique for conducting CNN inference and training.
(3) We validate the security of the proposed scheme through security analysis and perform comprehensive experimental evaluations to evaluate its performance. The results showcase that the proposed scheme exhibits a higher detection probability of corrupted vectorized parameters compared to previous schemes. Additionally, the proposed scheme demonstrates lower communication overhead and computational time in secure inference, measuring 6.80MB and 0.098s, respectively.

Organization: The remaining sections of our paper are structured below: Related work is discussed in Section 2. Section 3 provides some preliminaries used for the proposed scheme. We define system model and threat model in Section 4. The detailed description of the proposed scheme is given in Section 5. Security analysis and experiment performance are provided in Sections 6 and 7. We discuss the proposed scheme in Section 8. Finally, our work is concluded in Section 9.

2. Related work

In recent years, numerous privacy-preserving neural network inference or training schemes have been proposed. These schemes are mainly based on differential privacy (DP) [25,26], [27,28] HE [21, 29], [30,31] and secure multiparty computation (MPC) techniques (oblivious transfer (OT) [32], garbled circuit (GC) [33,34], and SS [35–37]). Specifically, DP is commonly utilized to protect data privacy in DNN training phase by introducing noise to data or weights. Yuan et al. [27] designed a CNN-based medical image analysis scheme using DP technique, which protects the privacy of patient by adding noise during the training process. Ma et al. [28] proposed a DP-based Rényi-differentially private-generative adversarial network (GAN), which achieves privacy protection by adding noise to the value of the exchanged loss function during the training process. However, the above-mentioned DP-based schemes may suffer from a reduction in the accuracy of results due to the addition of noise. HE enables computation to be performed on ciphertext and produces the same result as if the computation is performed directly on the plaintext after decryption. Kim et al. [30] constructed a privacy-preserving CNN inference scheme based on fully homomorphic encryption (FHE). This scheme introduces a more efficient way for evaluating convolutions with FHE, significantly enhancing inference efficiency. Wang et al. [31] proposed HT2ML, which combines the advantages of HE and trusted execution environment to address data privacy concerns. However, HE-based schemes generally incur significant computational overhead due to the heavy encryption and decryption operations involved in HE technique. Recently, there has been a surge of research into MPC-based privacy-preserving schemes. MPC allows the evaluation of functions based on inputs from each party without revealing their individual inputs. In the MPC techniques, SS is commonly used in privacy-preserving neural network schemes. Yang et al. [38] designed a privacy-preserving GAN scheme based on SS, effectively addressing the issue of privacy leakage that arises during model training and image generation processes in GAN. Liu et al. [23] constructed an SS-based CNN inference scheme, which achieves face detection while protecting the privacy of the user's data and model. However, SS-based schemes incur high communication overhead in dot product operations, which reduces the efficiency of inference and training for neural networks that involve a large number of dot product operations. Furthermore, some mixed-protocol schemes for neural network inference have been proposed [39–42], which implement different operations by converting between different cryptographic techniques. However, the mixed-protocols schemes require complex conversions between different cryptographic techniques. Based on the above analysis, it is essential to design a privacy-preserving scheme with low computational and communication overheads while guaranteeing accuracy.

The above schemes can achieve privacy protection. However, they do not consider the problem of model integrity verification. The corruption of model integrity may significantly affect the correctness of inference and training results. To verify the correctness of inference result provided by server, Ghodsi et al. constructed SafetyNets [43], which is an interactive proof-based verifiable computing scheme designed for DNN. Based on quadratic matrix program, Weng et al. [44] put forward pvCNN, a zero-knowledge succinct non-interactive argument of knowledge scheme. In this scheme, the correctness of CNN output results can be verified. Duan et al. [24] proposed PVDLI, which verifies the correctness of inference result by injecting verifiable data into the input. These schemes [24,43,44] can support secure inference. Nevertheless, the above schemes [24,43,44] primarily focus on ensuring the correctness of inference result from the server and may fail to detect subtle changes in the model parameters. The subtle parameter changes may lead to incorrect inference results for certain inputs while other inputs can be processed correctly [45]. To tackle the above problem, Kuttichira et al. [46] introduced a Bayesian Compromise Detection (BCD) scheme that can verify the model integrity by generating sensitive samples. Yin et al. [47] developed a DNN fragile watermarking framework called fragile trigger generation (FTG), which detects the subtle changes in model parameters using fragile watermarking algorithm. While these schemes [46,47] can guarantee model integrity, they cannot support user data and model privacy protection. Xu et al. [22] presented a privacy-preserving and verifiable inference scheme called SecureDL. However, this scheme [22] only considers privacy-preserving inference and does not support privacy-preserving training. In addition, the scheme [22] requires high computation overhead to achieve privacy protection due to the use of HE technique. Thus, it is necessary

Table 1
Comparison of the proposed scheme with major related work.

Scheme	Secure inference	Secure training	Model integrity verification
PVDLI [24]	✓	×	×
pvCNN [44]	✓	×	×
BCD [46]	×	×	✓
FTG [47]	×	×	✓
SecureDL [22]	✓	×	✓
Ours	✓	✓	✓

to develop a privacy-preserving and verifiable scheme for inference and training that supports model integrity verification while achieving efficient data and model privacy protection.

Table 1 provides the comparison of the proposed scheme with existing major related works. From Table 1, it can be observed that only the proposed scheme simultaneously supports secure inference, secure training, and model integrity verification.

3. Preliminaries

This section introduces the basic knowledge required for the proposed scheme, including bilinear map, Computational Diffie–Hellman (CDH) problem, Discrete Logarithm (DL) problem, Replicated Secret Sharing, and Convolutional Neural Network.

3.1. Bilinear map

Consider two multiplicative cyclic groups G_1 and G_2 with the same prime order p. Assuming that g is G_1's generator. A bilinear map $e : G_1 \times G_1 \to G_2$ exhibits the subsequent characteristics:

(1) Bilinearity: for any g_1, g_2 belongs to G_1, and a, b belong to Z_p^*, satisfying $e\left(g_1^a, g_2^b\right) = e(g_1, g_2)^{ab}$.
(2) Non-degeneracy: $e(g, g) \neq 1$.
(3) Computability: it is efficient to calculate $e(g_1, g_2)$ for any g_1, g_2 belongs to G_1.

3.2. Computational Diffie–Hellman problem

For elements a and $b \in Z_p^*$, provided with inputs g, g^a, and $g^b \in G_1$, the output is $g^{ab} \in G_1$. The CDH assumption in G_1 holds if it is computationally infeasible to solve the CDH problem in G_1.

3.3. Discrete logarithm problem

For $a \in Z_p^*$, provided with inputs g and $g^a \in G_1$, the output is a. The DL assumption in G_1 holds if it is computationally infeasible to solve the DL problem in G_1.

3.4. Replicated secret sharing [48]

In RSS with n ($n \geq 3$) parties, the secret is divided into n shares and is distributed to n parties respectively. The number of shares is equal to the number of parties. The secret can be recovered when the number of parties is $t+1$, where t satisfies the conditions that $1 \leq t < n$ and $n = 2t + 1$ [49]. For secure multiplication, each party needs to interact with t other parties. As a result, the communication overhead of each party is t in secure multiplication. Thus, the communication overhead of each party increases with the number of parties n. Based on the relationship between n and t, it can be observed that as the number of shares n increases, t also increases, leading to higher the communication overhead for each party. To minimize communication overhead, the proposed scheme selects $t = 1$ and $n = 3$.

The following illustrates a replicated secret sharing scheme with three parties.

A secret $x \in Z_{2^\lambda}$ is randomly divided into three shares x_1, x_2, x_3, with the constraint that $x_1 + x_2 + x_3 = x \in Z_{2^\lambda}$. Each of the three parties P_1, P_2, and P_3 holds a pair of shares, denotes as (x_1, x_2), (x_2, x_3), and (x_3, x_1) respectively. Such a sharing of x is denoted as $[\![x]\!] := (x_1, x_2, x_3)$. To simplify the notation, the pair of shares held by party P_k ($k \in [1, 3]$) is represented as $[\![x]\!]_k$, *i.e.*, P_1 holds $[\![x]\!]_1 := (x_1, x_2)$, P_2 holds $[\![x]\!]_2 := (x_2, x_3)$ and P_3 holds $[\![x]\!]_3 := (x_3, x_1)$.

(1) **Secure Secret Reconstruction Protocol** Two scenarios for secret reconstruction are considered: revealing the secret x to a single party or revealing it to all parties. In the first case, suppose that x is only disclosed to a single party P_k ($k \in [1, 3]$). Each party P_k ($k \in [1, 3]$) can reconstruct the secret x using the shares x_k and x_{k+1} they hold and the share x_{k-1} from P_{k-1}. In the second case, suppose that x is disclosed to all parties P_k ($k \in [1, 3]$). Each party P_k ($k \in [1, 3]$) is capable of reconstructing x by using the shares x_k and x_{k+1} they hold and the share x_{k-1} from P_{k-1}.
(2) **Secure Addition (SecAdd) Protocol** Two scenarios for SecAdd are considered: the addition of two values $[\![x]\!]$ and $[\![y]\!]$, represented as $[\![z]\!] = [\![x]\!] + [\![y]\!]$, and the addition of a value $[\![x]\!]$ with a constant c, represented as $[\![z]\!] = [\![x]\!] + c$. In the first case, $[\![z]\!] = [\![x]\!] + [\![y]\!] = (x_1 + y_1) + (x_2 + y_2) + (x_3 + y_3)$. Let $z_1 := x_1 + y_1$, $z_2 := x_2 + y_2$, and $z_3 := x_3 + y_3$. Each party P_k ($k \in [1, 3]$) calculates $z_k = x_k + y_k$ and $z_{k+1} = x_{k+1} + y_{k+1}$ locally based on the shares they hold. Thus, each party P_k ($k \in [1, 3]$) holds the shares (z_k, z_{k+1}). In the second case, $[\![z]\!] = [\![x]\!] + c = x_1 + c + x_2 + x_3$. The parties P_1 and P_3 who hold the share x_1 need to calculate $z_1 = x_1 + c$ locally. Meanwhile, the parties P_1, P_2, and P_3 who hold the shares x_2 and x_3 calculate $z_2 = x_2$ and $z_3 = x_3$. Therefore, each party P_k ($k \in [1, 3]$) holds the shares (z_k, z_{k+1}). During the phase of SecAdd, each party P_k ($k \in [1, 3]$) only needs to locally calculate the addition of the shares they hold. Subtraction can be performed in a similar way to addition.
(3) **Secure Multiplication (SecMul) Protocol** Two scenarios for SecMul are considered: the multiplication of two values $[\![x]\!]$ and $[\![y]\!]$, represented as $[\![z]\!] = [\![x]\!] \cdot [\![y]\!]$, and the multiplication of a value $[\![x]\!]$ and a constant c, represented as $[\![z]\!] = [\![x]\!] \cdot c$. In the first case, as shown in the Eq. (1), $[\![z]\!] = [\![x]\!] \cdot [\![y]\!] = (x_1y_1 + x_1y_2 + x_2y_1) + (x_2y_2 + x_2y_3 + x_3y_2) + (x_3y_3 + x_3y_1 + x_1y_3)$. Let $z'_1 := x_1y_1 + x_1y_2 + x_2y_1$, $z'_2 := x_2y_2 + x_2y_3 + x_3y_2$ and $z'_3 := x_3y_3 + x_3y_1 + x_1y_3$. Each party P_k ($k \in [1, 3]$) can calculate z'_k locally based on the shares they hold. In RSS, each party should hold two shares. To meet this requirement, each party P_k ($k \in [1, 3]$) needs to transmit the share they hold to P_{k-1}. This transmission process is termed re-sharing. To protect the privacy of share during the transmission process, each party P_k ($k \in [1, 3]$) needs to generate a random value r_k to blind the share z'_k held by themselves, *i.e.*, $z_k = z'_k + r_k$, satisfying $r_1 + r_2 + r_3 = 0$. Specifically, P_{k-1} and P_k share a pseudorandom number generator in advance, allowing them to generate the same random value $a_{k-1,k}$. Similarly, P_k and P_{k+1} perform the same operation to generate $a_{k,k+1}$. Consequently, the parties P_1, P_2, and P_3 respectively hold the random values $(a_{3,1}, a_{1,2})$, $(a_{1,2}, a_{2,3})$, and $(a_{2,3}, a_{3,1})$. Each party P_k ($k \in [1, 3]$) can then calculate $r_k = a_{k,k+1} - a_{k-1,k}$ [48]. As a result, the parties P_1, P_2, and P_3 respectively hold the shares (z_1, z_2), (z_2, z_3), and (z_3, z_1). In the second case, each party P_k ($k \in [1, 3]$) calculates $z_k = x_k \cdot c$ and $z_{k+1} = x_{k+1} \cdot c$ based on the shares they hold locally. Thus, the parties P_1, P_2, and P_3 respectively hold the shares (z_1, z_2), (z_2, z_3), and (z_3, z_1).

$$\begin{aligned}[\![z]\!] &= [\![x]\!] \cdot [\![y]\!] \\ &= (x_1 + x_2 + x_3) \cdot (y_1 + y_2 + y_3) \\ &= (x_1y_1 + x_1y_2 + x_2y_1) + (x_2y_2 + x_2y_3 + x_3y_2) \\ &\quad + (x_3y_3 + x_3y_1 + x_1y_3)\end{aligned} \tag{1}$$

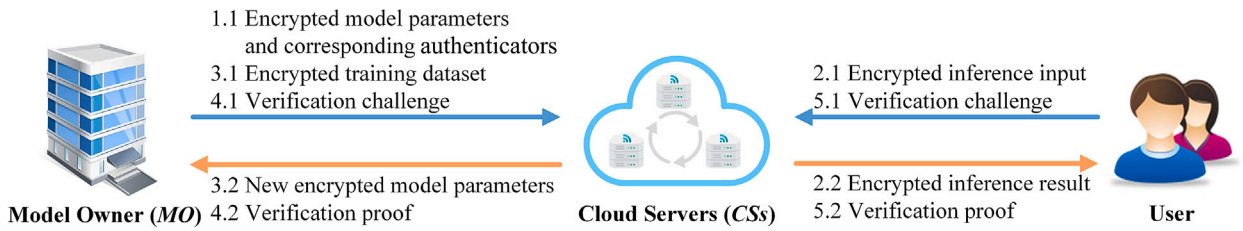

Fig. 1. System model.

(4) **Secure Dot Product (SecDot) Protocol** Suppose there are two vectors $[\![\vec{x}]\!] = ([\![x^{(0)}]\!], [\![x^{(1)}]\!], \ldots, [\![x^{(n-1)}]\!])$ and $[\![\vec{y}]\!] = ([\![y^{(0)}]\!], [\![y^{(1)}]\!], \ldots, [\![y^{(n-1)}]\!])$, where $[\![x^{(j)}]\!]$ and $[\![y^{(j)}]\!]$ ($j \in [0, n-1]$) respectively denote the j-th element of the vectors $[\![\vec{x}]\!]$ and $[\![\vec{x}]\!]$. Each element $[\![x^{(j)}]\!]$ of the vector $[\![\vec{x}]\!]$ is divided into three shares $x_1^{(j)}$, $x_2^{(j)}$, $x_3^{(j)}$, satisfying $x^{(j)} = x_1^{(j)} + x_2^{(j)} + x_3^{(j)}$. Same operations are performed for the vector $[\![\vec{y}]\!]$. These shares are distributed among three parties P_k ($k \in [1,3]$) in the form of $\{(x_1^{(j)}, x_2^{(j)}), (x_2^{(j)}, x_3^{(j)}), (x_3^{(j)}, x_1^{(j)})\}$ and $\{(y_1^{(j)}, y_2^{(j)}), (y_2^{(j)}, y_3^{(j)}), (y_3^{(j)}, y_1^{(j)})\}$. The dot product of $[\![\vec{x}]\!]$ and $[\![\vec{y}]\!]$ can be calculated as follows:

$$
\begin{aligned}
[\![z]\!] &= [\![\vec{x}]\!] \cdot [\![\vec{y}]\!] = \sum_{j=0}^{n-1} ([\![x^{(j)}]\!] \cdot [\![y^{(j)}]\!]) \\
&= \sum_{j=0}^{n-1} ({x_1}^{(j)} + {x_2}^{(j)} + {x_3}^{(j)}) \cdot ({y_1}^{(j)} + {y_2}^{(j)} + {y_3}^{(j)}) \\
&= \sum_{j=0}^{n-1} ({x_1}^{(j)}{y_1}^{(j)} + {x_1}^{(j)}{y_2}^{(j)} + {x_2}^{(j)}{y_1}^{(j)}) + \\
&\quad \sum_{j=0}^{n-1} ({x_2}^{(j)}{y_2}^{(j)} + {x_2}^{(j)}{y_3}^{(j)} + {x_3}^{(j)}{y_2}^{(j)}) + \\
&\quad \sum_{j=0}^{n-1} ({x_3}^{(j)}{y_3}^{(j)} + {x_3}^{(j)}{y_1}^{(j)} + {x_1}^{(j)}{y_3}^{(j)})
\end{aligned}
\tag{2}
$$

Each party P_k ($k \in [1,3]$) locally calculates the k-th sum $\sum_{j=0}^{n-1} ({x_k}^{(j)}{y_k}^{(j)} + {x_k}^{(j)}{y_{k+1}}^{(j)} + {x_{k+1}}^{(j)}{y_k}^{(j)}) := z_k$ of Eq. (2) and transmits z_k to P_{k-1}. Thus, P_1 holds the shares (z_1, z_2), P_2 holds the shares (z_2, z_3), and P_3 holds the shares (z_3, z_1). It is observed that only one communication round is required for n-dimensional dot product of RSS. In contrast, SS requires n communication rounds for n-dimensional dot product due to the requirement of performing n secret multiplications.

3.5. Convolutional neural network

The CNN employs multiple convolutional layers to extract features from input data, resulting in a more abstract and higher-level representation of input data. Each linear layer is able to be expressed as $\boldsymbol{y} = \boldsymbol{W} \cdot \boldsymbol{x}$, where $\boldsymbol{W}$ is weight matrix and $\boldsymbol{x}$ is input. Common layers in CNN include the convolutional layer, fully connected layer, ReLU layer, batch normalization layer, and max pooling layer.

4. System model and threat model

4.1. System model

As depicted in Fig. 1, system model contains three distinct entities: model owner (MO), cloud servers (CSs), and user.

- MO: The MO possesses a well-trained neural network model and has limited computational resources. The MO outsources the computationally intensive secure inference and secure training services to the CSs. To guarantee model integrity, MO requires to produce the authenticators for the shares of vectorized parameters in the model. During the secure training phase, MO transmits the encrypted training dataset to CSs for model training. Then the MO receives the new encrypted model parameters from the CSs and decrypts them to obtain the new plaintext model parameters. The MO can verify model integrity by transmitting verification challenge to CSs and verifying the correctness of verification proofs returned by CSs.
- CSs: The CSs, consisting of three cloud servers, have powerful computational capability and storage resource. Each of the three CSs plays the same role, responsible for storing the model, providing secure inference and training services, and generating the verification proofs of the model. CSs perform the secure inference on the encrypted inference input sent from the user and return encrypted inference result to user. In addition, CSs train the model on the encrypted training dataset sent from the MO and return the new encrypted model parameters to the MO. When the CSs receive the verification challenge from MO or user, they produce and transmit the verification proofs to MO or user.
- User: The user is the entity that requires the CSs to provide secure inference service. During secure inference phase, the user transmits the encrypted inference input to the CSs. The user is capable of decrypting the encrypted inference result from CSs and obtaining the plaintext inference result. User can check model integrity to ensure the correct of inference result returned by the CSs. Specifically, user submits a verification challenge to CSs. After receiving the verification proofs from CSs, user can verify the correctness of proofs.

Remark. Similar to existing RSS-based schemes [42,50,51], the proposed scheme delegates computationally intensive tasks on three CSs. This is due to the fact that in RSS technique, using three parties incurs the minimum communication overhead [49]. Consequently, we employ RSS with three parties as the encryption method and utilize three CSs in the proposed scheme.

4.2. Threat model

Similar to the existing schemes [23,38,52,53], in the threat model, the CSs are assumed to be the honest but curious entities. They faithfully execute the protocols, but may be curious about model and input data. Hence, it is necessary to preserve the privacy of the model and data from being leaked to the CSs. Assuming that the CSs are independent and non-colluding, and they do not intentionally delete or modify the model. They only share the necessary information during the execution of the protocols, and do not disclose any additional information to each other. This is a realistic assumption because cloud server providers are reputable companies (such as Google Cloud and Alibaba Cloud) that are not willing to compromise their reputation through corrupt operations [54]. However, the model stored in the CSs may be corrupted due to uncontrollable factors, such as software/hardware failures or natural disasters. Therefore, it is crucial to verify model integrity. Furthermore, user is also an honest but curious entity. He correctly uploads the encrypted inference input to the CSs,

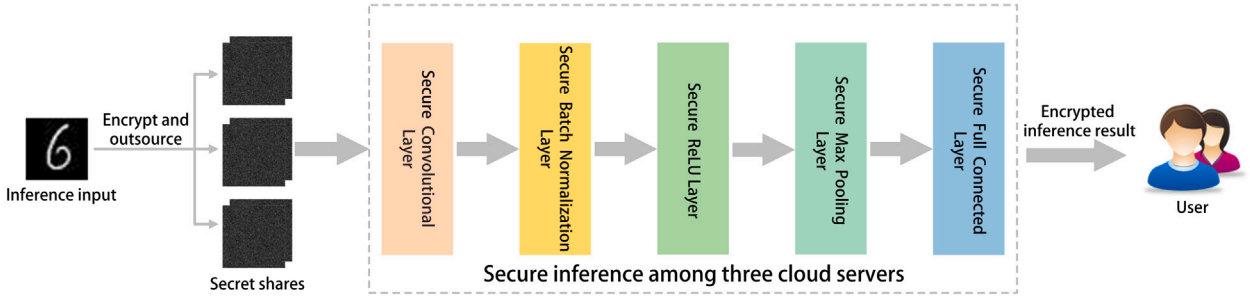

Fig. 2. Secure inference of convolution neural network.

but is curious about the model privacy of MO. He may attempt to obtain MO's real model by repeatedly challenging the same vectorized parameters in the phase of model integrity verification. Thus, it is necessary to safeguard model privacy, preventing any potential leakage to user. MO is the honest entity that correctly uploads the encrypted model parameters, the corresponding authenticators, and the encrypted training dataset to the CSs.

5. The proposed scheme

This section provides a detailed description of the proposed scheme.

5.1. Initialization

Suppose there are L linear layers in the neural network. The model parameters of the l-th layer are set as $W^{(l)}$, where $l \in [1, L]$.

(a) The MO divides the parameters $W^{(l)}$ of the l-th layer into three shares $W_1^{(l)}$, $W_2^{(l)}$, $W_3^{(l)}$, where $W^{(l)} = W_1^{(l)} + W_2^{(l)} + W_3^{(l)}$ ($l \in [1, L]$). These three shares are distributed as $[\![W^{(l)}]\!]_k = (W_k^{(l)}, W_{k+1}^{(l)})$

$$= \begin{pmatrix} (w_{1,1_k}^{(l)}, w_{1,1_{k+1}}^{(l)}) & \cdots & (w_{1,s_k}^{(l)}, w_{1,s_{k+1}}^{(l)}) \\ \vdots & \ddots & \vdots \\ (w_{n,1_k}^{(l)}, w_{n,1_{k+1}}^{(l)}) & \cdots & (w_{n,s_k}^{(l)}, w_{n,s_{k+1}}^{(l)}) \end{pmatrix}$$

$$= \begin{pmatrix} [\![w_{1,1}^{(l)}]\!]_k & \cdots & [\![w_{1,s}^{(l)}]\!]_k \\ \vdots & \ddots & \vdots \\ [\![w_{n,1}^{(l)}]\!]_k & \cdots & [\![w_{n,s}^{(l)}]\!]_k \end{pmatrix} (l \in [1, L], k \in [1, 3]).$$

Note: n and s can be different for different linear layers.

(b) The MO undertakes the subsequent procedures to produce public/private key pair and public parameters:

- Chooses a bilinear pair $e : G_1 \times G_1 \to G_2$, where G_1 and G_2 are two distinct multiplicative cyclic groups with a prime order p.
- Chooses a generator g of group G_1 and generates independent generators u_j ($j \in [1, s]$) of G_1.
- Selects three cryptographic hash functions $H_1 : \{0,1\}^* \to G_1$, $H_2 : Z_{2^\lambda} \to Z_p^*$ and $H_3 : G_1 \to Z_p^*$.
- Chooses a random number $sk \in Z_p^*$ as private key and calculates $pk = g^{sk}$ as public key.
- Publishes public parameters $(G_1, G_2, H_1, H_2, H_3, p, g, u_1, u_2, \ldots, u_s, pk)$.

Note: The generator g is used to generate the public key pk in the *Initialization* phase and is also used to verify the model integrity in the *Proof Verification* phase. The independent generators u_j ($j \in [1, s]$) are employed to generate the authenticators in the *Authenticator Generation* phase and to mask the aggregated model parameter in the *Proof Generation* phase.

5.2. Authenticator generation

For each $[\![W^{(l)}]\!]_k = \begin{pmatrix} [\![w_{1,1}^{(l)}]\!]_k & \cdots & [\![w_{1,s}^{(l)}]\!]_k \\ \vdots & \ddots & \vdots \\ [\![w_{n,1}^{(l)}]\!]_k & \cdots & [\![w_{n,s}^{(l)}]\!]_k \end{pmatrix} = \begin{pmatrix} [\![\vec{w}_1^{(l)}]\!]_k \\ [\![\vec{w}_2^{(l)}]\!]_k \\ \vdots \\ [\![\vec{w}_n^{(l)}]\!]_k \end{pmatrix}$ ($k \in [1, 3], l \in [1, L]$), the MO calculates a authenticator $\langle \sigma_i^{(l)} \rangle_k$ for each share of vectorized parameter $[\![\vec{w}_i^{(l)}]\!]_k$ ($i \in [1, n]$) of $[\![W^{(l)}]\!]_k$ by the following equation,

$$\langle \sigma_i^{(l)} \rangle_k = (H_1(i \parallel V) \cdot \prod_{j=1}^{s} u_j^{H_2([\![w_{i,j}^{(l)}]\!]_k)})^{sk}, \tag{3}$$

where V is the version number of the model. To ensure security, it is necessary to utilize s independent generators u_j to generate authenticators $\langle \sigma_i^{(l)} \rangle_k$ because each vectorized parameter $[\![\vec{w}_i^{(l)}]\!]_k$ contains s model parameters $[\![w_{i,j}^{(l)}]\!]_k$. Finally, the MO sends the shares $\{[\![W^{(l)}]\!]_k\}_{k\in[1,3],l\in[1,L]}$ and corresponding authenticators $\{\langle \sigma_i^{(l)} \rangle_k\}_{k\in[1,3],l\in[1,L],i\in[1,n]}$ to the corresponding cloud server CS_k. Protocol 1 demonstrates the process of authenticator generation.

Protocol 1: Authenticator Generation

Input: The hash functions H_1 and H_2, the version number V, and the model parameters' shares $[\![w_{i,j}^{(l)}]\!]_k$ ($k \in [1, 3], l \in [1, L], i \in [1, n], j \in [1, s]$).

Output: The authenticators $\langle \sigma_i^{(l)} \rangle_k$ ($k \in [1, 3], l \in [1, L], i \in [1, n]$).

```
for k ∈ [1, 3] do
    for l ∈ [1, L] do
        for i ∈ [1, n] do
            MO computes
            ⟨σ_i^(l)⟩_k = (H_1(i ∥ V) · ∏_{j=1}^{s} u_j^{H_2([[w_{i,j}^(l)]]_k)})^{sk}.
        end
    end
end
MO holds ⟨σ_i^(l)⟩_k (k ∈ [1, 3], l ∈ [1, L], i ∈ [1, n]).
```

5.3. Challenge generation

To verify the integrity of the model stored in CSs, MO or user transmits the verification challenge to each CS_k ($k \in [1, 3]$). Specifically, MO or user randomly chooses a set I containing m elements, where $I \subseteq [1, n]$. MO or user generates a random element $v_i \in Z_p^*$ for each $i \in I$. Then, MO or user transmits the verification challenge $\{i, v_i\}_{i\in I}$ to each CS_k ($k \in [1, 3]$).

5.4. Proof generation

After receiving the verification challenge $\{i, v_i\}_{i\in I}$, CS_k ($k \in [1, 3]$) generates the corresponding verification proof as follows: $\langle \sigma^{(l)} \rangle_k =$

$\prod_{i\in I}(\langle\sigma_i^{(l)}\rangle_k)^{v_i}$ and $\langle\mu_j^{(l)}{}'\rangle_k = \sum_{i\in I} v_i \cdot H_2([\![w_{i,j}^{(l)}]\!]_k)$, where $l \in [1,L]$ and $j \in [1,s]$. To prevent user from inferring the real model, the cloud server CS_k ($k \in [1,3]$) utilizes random masking technique to mask $\langle\mu_j^{(l)}{}'\rangle_k$ ($l \in [1,L], j \in [1,s]$) by picking a random value $\langle r\rangle_k \in Z_p^*$ and computes $\langle t_j\rangle_k = u_j^{\langle r\rangle_k}$. Then, the cloud server CS_k ($k \in [1,3]$) calculates $\langle\mu_j^{(l)}\rangle_k = \langle\mu_j^{(l)}{}'\rangle_k + \langle r\rangle_k \cdot H_3(\langle t_j\rangle_k)$ ($l \in [1,L], j \in [1,s]$). Finally, the cloud server CS_k ($k \in [1,3]$) transmits the verification proofs $P_k = \{\langle\sigma^{(l)}\rangle_k, \langle\mu_j^{(l)}\rangle_k, \langle t_j\rangle_k\}_{l\in[1,L],j\in[1,s]}$ to MO or user.

5.5. Proof verification

After receiving the verification proofs, MO or user verifies the integrity of the model through the following:

$$\prod_{k=1}^{3} e(\langle\sigma^{(l)}\rangle_k, g) = \prod_{k=1}^{3} e(\prod_{i\in I} H_1(i \parallel V)^{v_i} \cdot \prod_{j=1}^{s} (u_j^{\langle\mu_j^{(l)}\rangle_k} \cdot \langle t_j\rangle_k^{-H_3(\langle t_j\rangle_k)}), pk), \tag{4}$$

where $l \in [1,L]$. If the Eq. (4) holds, it means that the model stored in the CSs is intact; otherwise, it is not.

In the following, an example of challenge-verification process is given. Suppose that the size of the 4-th layer is 10×2. It means that the 4-th layer has 10 vectorized parameters, and each vectorized parameter has 2 model parameters. In the *Challenge Generation* phase, if MO/user wants to verify the integrity of the 3-rd, 5-th and 6-th vectorized parameters of the 4-th layer, it sends the verification challenge $\{3, v_3, 5, v_5, 6, v_6\}$ to each CS_k ($k \in [1,3]$), where $\{3,5,6\}$ are the indexes of challenged vectorized parameters and $\{v_3, v_5, v_6\}$ are the random values chosen by MO/user. After receiving the verification challenge $\{3, v_3, 5, v_5, 6, v_6\}$, based on the stored vectorized parameters and the corresponding authenticators, each CS_k ($k \in [1,3]$) respectively calculates the corresponding verification proof as follows: $\langle\sigma^{(4)}\rangle_k = (\langle\sigma_3^{(4)}\rangle_k^{v_3} \cdot \langle\sigma_5^{(4)}\rangle_k^{v_5} \cdot \langle\sigma_6^{(4)}\rangle_k^{v_6})$, $\langle\mu_1^{(4)}{}'\rangle_k = v_3 \cdot H_2([\![w_{3,1}^{(4)}]\!]_k) \cdot v_5 \cdot H_2([\![w_{5,1}^{(4)}]\!]_k) \cdot v_6 \cdot H_2([\![w_{6,1}^{(4)}]\!]_k)$ and $\langle\mu_2^{(4)}{}'\rangle_k = v_3 \cdot H_2([\![w_{3,2}^{(4)}]\!]_k) \cdot v_5 \cdot H_2([\![w_{5,2}^{(4)}]\!]_k) \cdot v_6 \cdot H_2([\![w_{6,2}^{(4)}]\!]_k)$. To protect the privacy of $\langle\mu_1^{(4)}{}'\rangle_k$ and $\langle\mu_2^{(4)}{}'\rangle_k$, each CS_k ($k \in [1,3]$) picks a random value $\langle r\rangle_k$ and computes $\langle t_1\rangle_k = u_1^{\langle r\rangle_k}$ and $\langle t_2\rangle_k = u_2^{\langle r\rangle_k}$. Then each CS_k ($k \in [1,3]$) calculates $\langle\mu_1^{(4)}\rangle_k = \langle\mu_1^{(4)}{}'\rangle_k + \langle r\rangle_k \cdot H_3(\langle t_1\rangle_k)$ and $\langle\mu_2^{(4)}\rangle_k = \langle\mu_2^{(4)}{}'\rangle_k + \langle r\rangle_k \cdot H_3(\langle t_2\rangle_k)$. Finally, each CS_k ($k \in [1,3]$) transmits the verification proofs $P_k = \{\langle\sigma^{(4)}\rangle_k, \langle\mu_1^{(4)}\rangle_k, \langle\mu_2^{(4)}\rangle_k, \langle t_1\rangle_k, \langle t_2\rangle_k\}$ to the MO/user. After receiving the verification proofs, MO/user verifies the integrity of the model through the Eq. (4).

5.6. Secure inference

The proposed scheme can offer inference service to the user. To ensure inference input privacy, the user divides inference input into three shares and sends them to three CSs respectively. The inference result can be obtained by aggregating the shares of the inference result returned from the CSs. Fig. 2 illustrates that secure inference contains the following four processes: secure linear (SecLin) layers, secure batch normalization (SecBN) layer, secure ReLU (SecReLU) layer, and secure max pooling (SecMP) layer.

5.6.1. Secure linear layers

In the proposed scheme, both secure convolution (SecConv) layer and secure fully connected (SecFC) layer are referred to SecLin layers. The mathematical expression of the SecLin layers can be written as $\boldsymbol{y} = \boldsymbol{W}\cdot\boldsymbol{x}$, where $\boldsymbol{W}$ represents the trainable weight matrix and $\boldsymbol{x}$ represents the layer's input. Note that the bias term of the SecLin layers is set to 0 since the same shifting activation effect can also be achieved by the bias term in the SecBN layer [55].

For the SecConv layer, the linear operation is performed by scanning the input with the convolutional kernel, which is normally formulated as the weight matrix $\boldsymbol{W}$ with a fixed size. Let the convolutional kernel size be $n \times n$, for the neuron in the l-th layer, its receptive field on the $(l-1)$-th layer is $n \times n$. In other words, the (i,j)-th neuron in the l-th layer is computed as $y_{i,j}^{(l)} = \sum_{p=0}^{n-1}\sum_{q=0}^{n-1} w_{p,q}^{(l)} \cdot x_{i+p,j+q}^{(l-1)}$, where $w_{p,q}^{(l)}$ denotes the (p,q)-th weight of the convolutional kernel in the l-th layer and $x_{i,j}^{(l-1)}$ denotes the (i,j)-th input in the $(l-1)$-th layer. Therefore, the CSs need to cooperatively compute $[\![y_{i,j}^{(l)}]\!] = \sum_{p=0}^{n-1}\sum_{q=0}^{n-1} [\![w_{p,q}^{(l)}]\!] \cdot [\![x_{i+p,j+q}^{(l-1)}]\!]$.

For the SecFC layer, $w_{p,q}^{(l)}$ represents the weight that maps the p-th neuron of the $(l-1)$-th layer to the q-th neuron of the l-th layer. Thus, the value of the q-th neuron in the l-th layer is computed as $y_q^{(l)} = \sum_p w_{p,q}^{(l)} \cdot x_p^{(l-1)}$. Accordingly, the CSs need to cooperatively calculate $[\![y_q^{(l)}]\!] = \sum_p [\![w_{p,q}^{(l)}]\!] \cdot [\![x_p^{(l-1)}]\!]$.

5.6.2. Secure batch normalization layer

The secure batch normalization (SecBN) layer is commonly utilized to normalize the output of the SecLin layer, which may significantly accelerate the training of the network and improve the network's performance by reducing internal covariate shift. The calculation of SecBN layer can be expressed as $y_{p,q}^{(l)} = \gamma \cdot \frac{x_{p,q}^{(l-1)} - \epsilon}{\omega} + \delta$, where ϵ and ω respectively denote the running mean and the running variance of the training datasets, γ and δ respectively denote learnable scale and shift terms. In secure inference phase, ϵ, ω, γ and δ are pre-learned and fixed terms. The CSs need to cooperatively compute $[\![y_{p,q}^{(l)}]\!] = \frac{\gamma}{\omega} \cdot ([\![x_{p,q}^{(l-1)}]\!] - \epsilon) + \delta$.

5.6.3. Secure ReLU layer

The secure ReLU (SecReLU) layer enhances the expressive ability of the neural network by utilizing a non-linear transformation to the output of the SecLin/SecBN layer. It is calculated as $\mathrm{ReLU}(x) = \max(x,0)$, where x denotes the previous layer's output. In the scheme, $\max(x,0)$ is calculated by computing $\max(x,0) = \neg\mathrm{MSB}(x)\cdot x = (1-\mathrm{MSB}(x))\cdot x$, where $\mathrm{MSB}(x)$ denotes the most significant bit (MSB) of the two's complement of x and $\neg$ denotes the negation of $\mathrm{MSB}(x)$. Specifically, $\mathrm{MSB}(x) = 0$ when $x \geq 0$ and $\mathrm{MSB}(x) = 1$ when $x < 0$. Therefore, for the activation value of the (i,j)-th neuron of the SecReLU layer, the CSs cooperatively compute $[\![y_{i,j}^{(l)}]\!] = \max([\![x_{i,j}^{(l-1)}]\!], 0) = \neg\mathrm{MSB}([\![x_{i,j}^{(l-1)}]\!]) \cdot [\![x_{i,j}^{(l-1)}]\!] = (1 - \mathrm{MSB}([\![x_{i,j}^{(l-1)}]\!])) \cdot [\![x_{i,j}^{(l-1)}]\!]$. The method for computing MSB in ABY^3 [42] is used to calculate MSB(x).

5.6.4. Secure max pooling layer

To reduce redundancy and improve computation efficiency, secure max pooling (SecMP) layer is commonly applied to the output of the SecReLU layer. By selecting the maximum value within a sliding window, this layer is able to retain the key features of the image. In the scheme, the maximum of the secrets $([\![x_1^{(l-1)}]\!], [\![x_2^{(l-1)}]\!], \ldots, [\![x_n^{(l-1)}]\!])$ is computed using the MSB() method mentioned above, and record the index *max* of the maximum value. For example, the CSs can compare $[\![x_1^{(l-1)}]\!]$ and $[\![x_2^{(l-1)}]\!]$ by cooperatively calculating $\max([\![x_1^{(l-1)}]\!], [\![x_2^{(l-1)}]\!]) = \neg\mathrm{MSB}([\![x_1^{(l-1)}]\!] - [\![x_2^{(l-1)}]\!]) \cdot [\![x_1^{(l-1)}]\!] + \mathrm{MSB}([\![x_1^{(l-1)}]\!] - [\![x_2^{(l-1)}]\!]) \cdot [\![x_2]\!] = (1\text{-}\mathrm{MSB}([\![x_1^{(l-1)}]\!] - [\![x_2^{(l-1)}]\!])) \cdot [\![x_1^{(l-1)}]\!] + \mathrm{MSB}([\![x_1^{(l-1)}]\!] - [\![x_2^{(l-1)}]\!]) \cdot [\![x_2^{(l-1)}]\!]$.

Furthermore, inspired by [56], by moving the SecReLU layer after the SecMP layer, the number of MSB() method calls on the SecReLU layer can be significantly reduced, thus saving communication overhead. As illustrated in Fig. 3, for a 4×4 feature matrix, the number of comparisons in the SecReLU layer can be reduced from 12 to 3 by adjusting the order of the SecReLU and SecMP layers, thus reducing the number of MSB() method calls by 9.

In the following, an example is provided to illustrate the process of secure inference where data, model, and inference are all encrypted. Suppose the encrypted inference input is a 5×5 matrix $(([\![a_{1,1}^{(0)}]\!], \ldots, [\![a_{1,5}^{(0)}]\!]), \ldots, ([\![a_{5,1}^{(0)}]\!], \ldots, [\![a_{5,5}^{(0)}]\!]))$. For the first SecConv layer, assume the encrypted convolution kernel is a 2×2 matrix $(([\![w_{1,1}^{(1)}]\!], [\![w_{1,2}^{(1)}]\!]), ([\![w_{2,1}^{(1)}]\!], [\![w_{2,2}^{(1)}]\!]))$. The three CSs collaboratively compute the output of SecConv layer, resulting in a 4×4 matrix $(([\![b_{1,1}^{(1)}]\!], \ldots, [\![b_{1,4}^{(1)}]\!]), \ldots, ([\![b_{4,1}^{(1)}]\!], \ldots, [\![b_{4,4}^{(1)}]\!]))$. based on the encrypted inference input $(([\![a_{1,1}^{(0)}]\!], \ldots, [\![a_{1,5}^{(0)}]\!]), \ldots, ([\![a_{5,1}^{(0)}]\!], \ldots, [\![a_{5,5}^{(0)}]\!]))$ and the encrypted

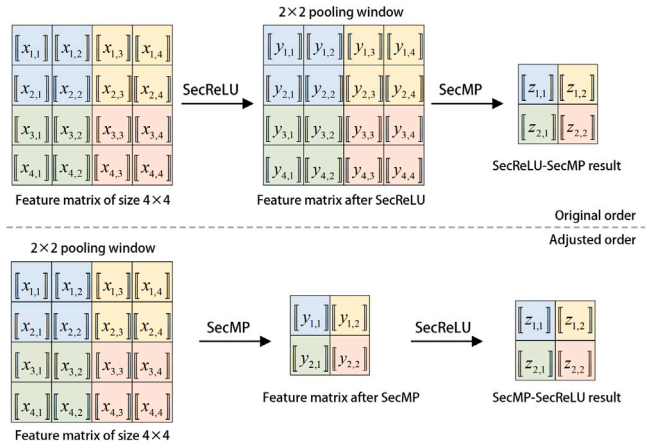

Fig. 3. An example of adjusting the order of SecReLU and SecMP layers with 2×2 pooling window.

convolution kernel $((\llbracket w_{1,1}^{(1)} \rrbracket, \llbracket w_{1,2}^{(1)} \rrbracket), (\llbracket w_{2,1}^{(1)} \rrbracket, \llbracket w_{2,2}^{(1)} \rrbracket))$. For example, the value of $\llbracket b_{1,1}^{(1)} \rrbracket$ is calculated as $\llbracket b_{1,1}^{(1)} \rrbracket = \llbracket w_{1,1}^{(1)} \rrbracket \cdot \llbracket a_{1,1}^{(0)} \rrbracket + \llbracket w_{1,2}^{(1)} \rrbracket \cdot \llbracket a_{1,2}^{(0)} \rrbracket + \llbracket w_{2,1}^{(1)} \rrbracket \cdot \llbracket a_{2,1}^{(0)} \rrbracket + \llbracket w_{2,2}^{(1)} \rrbracket \cdot \llbracket a_{2,2}^{(0)} \rrbracket$. The output of SecConv layer is utilized as the input to the second SecBN layer. The three CSs collaboratively compute the output of SecBN layer based on SecConv layer's output $((\llbracket b_{1,1}^{(1)} \rrbracket, \ldots, \llbracket b_{1,4}^{(1)} \rrbracket), \ldots, (\llbracket b_{4,1}^{(1)} \rrbracket, \ldots, \llbracket b_{4,4}^{(1)} \rrbracket))$, learnable scale γ, running variance ω, running mean ϵ and shift terms δ, resulting in another 4×4 matrix $((\llbracket c_{1,1}^{(2)} \rrbracket, \ldots, \llbracket c_{1,4}^{(2)} \rrbracket), \ldots, (\llbracket c_{4,1}^{(2)} \rrbracket, \ldots, \llbracket c_{4,4}^{(2)} \rrbracket))$. For example, the value of $\llbracket c_{1,1}^{(2)} \rrbracket$ is computed as $\llbracket c_{1,1}^{(2)} \rrbracket = \frac{\gamma}{\omega} \cdot (\llbracket b_{1,1}^{(1)} \rrbracket - \epsilon) + \delta$. For the third SecMP layer, assume the window size is 2×2. The three CSs collaboratively compute the output of SecMP layer based on SecBN layer's output $((\llbracket c_{1,1}^{(2)} \rrbracket, \ldots, \llbracket c_{1,4}^{(2)} \rrbracket), \ldots, (\llbracket c_{4,1}^{(2)} \rrbracket, \ldots, \llbracket c_{4,4}^{(2)} \rrbracket))$, resulting in a 2×2 matrix $((\llbracket d_{1,1}^{(3)} \rrbracket, \llbracket d_{1,2}^{(3)} \rrbracket), (\llbracket d_{2,1}^{(3)} \rrbracket, \llbracket d_{2,2}^{(3)} \rrbracket))$. For example, the value of $\llbracket d_{1,1}^{(3)} \rrbracket$ is calculated as $\llbracket d_{1,1}^{(3)} \rrbracket = \max((\llbracket c_{1,1}^{(2)} \rrbracket, \llbracket c_{1,2}^{(2)} \rrbracket), (\llbracket c_{2,1}^{(2)} \rrbracket, \llbracket c_{2,2}^{(2)} \rrbracket))$. The output $((\llbracket d_{1,1}^{(3)} \rrbracket, \llbracket d_{1,2}^{(3)} \rrbracket), (\llbracket d_{2,1}^{(3)} \rrbracket, \llbracket d_{2,2}^{(3)} \rrbracket))$ of SecMP layer then serves as the input to the fourth SecReLU layer. The three CSs collaboratively compute the output of SecReLU layer based on $((\llbracket d_{1,1}^{(3)} \rrbracket, \llbracket d_{1,2}^{(3)} \rrbracket), (\llbracket d_{2,1}^{(3)} \rrbracket, \llbracket d_{2,2}^{(3)} \rrbracket))$, resulting in a 2×2 matrix $((\llbracket e_{1,1}^{(4)} \rrbracket, \llbracket e_{1,2}^{(4)} \rrbracket), (\llbracket e_{2,1}^{(4)} \rrbracket, \llbracket e_{2,2}^{(4)} \rrbracket))$. For example, the value of $\llbracket e_{1,1}^{(4)} \rrbracket$ is calculated as $\llbracket e_{1,1}^{(4)} \rrbracket = \max(\llbracket d_{1,1}^{(3)} \rrbracket, 0)$. For the fifth SecFC layer, assume the encrypted weight is a 2×2 matrix $((\llbracket w_{1,1}^{(5)} \rrbracket, \llbracket w_{1,2}^{(5)} \rrbracket), (\llbracket w_{2,1}^{(5)} \rrbracket, \llbracket w_{2,2}^{(5)} \rrbracket))$. The three CSs collaboratively compute the output of SecFC layer based on $((\llbracket w_{1,1}^{(5)} \rrbracket, \llbracket w_{1,2}^{(5)} \rrbracket), (\llbracket w_{2,1}^{(5)} \rrbracket, \llbracket w_{2,2}^{(5)} \rrbracket))$ and $((\llbracket e_{1,1}^{(4)} \rrbracket, \llbracket e_{1,2}^{(4)} \rrbracket), (\llbracket e_{2,1}^{(4)} \rrbracket, \llbracket e_{2,2}^{(4)} \rrbracket))$, resulting in the final encrypted inference result $\llbracket f_1^{(5)} \rrbracket = \llbracket w_{1,1}^{(5)} \rrbracket \cdot \llbracket e_{1,1}^{(4)} \rrbracket + \llbracket w_{1,2}^{(5)} \rrbracket \cdot \llbracket e_{1,2}^{(4)} \rrbracket + \llbracket w_{2,1}^{(5)} \rrbracket \cdot \llbracket e_{2,1}^{(4)} \rrbracket + \llbracket w_{2,2}^{(5)} \rrbracket \cdot \llbracket e_{2,2}^{(4)} \rrbracket$.

5.7. Secure training

The proposed scheme supports the secure training of CNN on encrypted data. The encrypted training dataset is sent from the MO to the CSs. The CSs perform the secure training based on encrypted training dataset. Specifically, the data discrepancies between the model's outputs at the SecFC layer and the target values are calculated using the softmax function and the cross-entropy loss function [57]. These misfits are then backpropagated layer by layer to the layers preceding the SecFC layer through the chain rule. This training process is repeated until the loss function converges or the predefined number of training iterations is reached. After the secure training is completed, the CSs transfers the new encrypted model parameters back to the MO. Then the MO generates the authenticators for the shares of vectorized parameters (encrypted model parameters) sent by CSs by performing the Authenticator Generation algorithm. The version number of the initial model uploaded by the MO is set as 1, and subsequent versions are incremented sequentially. Furthermore, the MO decrypts the new encrypted model parameters only when it needs the updated model.

Note: After receiving the encrypted model parameters from the CSs, the MO does not need to rebuild the new model as it does not perform inference and training locally. Furthermore, the MO does not need to decrypt the encrypted model parameters sent by the CSs every time since it generates the authenticators for the shares of the encrypted model parameters.

6. Correctness and security analysis

This section analyzes the correctness of model integrity verification, model privacy protection, the detectability of model integrity verification, and the correctness and security of the secure inference and training.

Theorem 1 (*Correctness of Model Integrity Verification*). *If the model stored in the CSs is intact, the verification proofs produced by CSs can pass the checking of MO or user.*

Proof. According to the valid verification proofs $P_k = \{\langle \sigma^{(l)} \rangle_k, \langle \mu_j^{(l)} \rangle_k, \langle t_j \rangle_k\}_{l \in [1,L], j \in [1,s]} (k \in [1,3])$ from the CS_k, the Eq. (4) in the *Proof Verification* algorithm will hold. The correctness of Eq. (4) can be derived below:

$$
\begin{aligned}
&\prod_{k=1}^{3} e(\langle \sigma^{(l)} \rangle_k, g) \\
&= \prod_{k=1}^{3} e(\prod_{i \in I} (\langle \sigma_i^{(l)} \rangle_k)^{v_i}, g) \\
&= \prod_{k=1}^{3} e(\prod_{i \in I} (H_1(i \parallel V) \cdot \prod_{j=1}^{s} u_j^{H_2(\llbracket w_{i,j}^{(l)} \rrbracket_k)})^{sk \cdot v_i}, g) \\
&= \prod_{k=1}^{3} e(\prod_{i \in I} H_1(i \parallel V)^{v_i} \cdot \prod_{j=1}^{s} u_j^{\sum_{i \in I} v_i \cdot H_2(\llbracket w_{i,j}^{(l)} \rrbracket_k)}, g^{sk}) \\
&= \prod_{k=1}^{3} e(\prod_{i \in I} H_1(i \parallel V)^{v_i} \cdot \prod_{j=1}^{s} u_j^{\langle \mu_j^{(l)\prime} \rangle_k}, pk) \\
&= \prod_{k=1}^{3} e(\prod_{i \in I} H_1(i \parallel V)^{v_i} \cdot \prod_{j=1}^{s} u_j^{\langle \mu_j^{(l)} \rangle_k - \langle r \rangle_k \cdot H_3(\langle t_j \rangle_k)}, pk) \\
&= \prod_{k=1}^{3} e(\prod_{i \in I} H_1(i \parallel V)^{v_i} \cdot \prod_{j=1}^{s} (u_j^{\langle \mu_j^{(l)} \rangle_k} \cdot \langle t_j \rangle_k^{-H_3(\langle t_j \rangle_k)}), pk)
\end{aligned}
$$

Theorem 2 (*Model Privacy Protection*). *In the proposed scheme, user is unable to obtain real model through verification proofs from the CSs.*

Proof. $P_k = \{\langle \sigma^{(l)} \rangle_k, \langle \mu_j^{(l)} \rangle_k, \langle t_j \rangle_k\}_{l \in [1,L], j \in [1,s]}$ $(k \in [1,3])$ are the verification proofs outputted by the CS_k. On the one hand, CS_k $(k \in [1,3])$ utilizes $\langle r \rangle_k$ and $\langle t_j \rangle_k$ to mask the real aggregated vectorized parameter $\langle \mu_j^{(l)\prime} \rangle_k = \sum_{i \in I} v_i \cdot H_2(\llbracket w_{i,j}^{(l)} \rrbracket_k)$ $(l \in [1,L], j \in [1,s])$ as $\langle \mu_j^{(l)} \rangle_k = \langle \mu_j^{(l)\prime} \rangle_k + \langle r \rangle_k \cdot H_3(\langle t_j \rangle_k)$, where $\langle r \rangle_k$ is randomly generated by the CS_k and kept confidential from user, and $\langle t_j \rangle_k = u_j^{\langle r \rangle_k}$. In addition, it is computationally infeasible for user to obtain $\langle r \rangle_k$ based on $\langle t_j \rangle_k$ $(k \in [1,3], j \in [1,s])$. This is because DL problem in G_1 is hard. Therefore, it is difficult for user to get the real aggregated vectorized parameter $\langle \mu_j^{(l)\prime} \rangle_k$ $(k \in [1,3], l \in [1,L], j \in [1,s])$ based on the blinded aggregated vectorized parameter $\langle \mu_j^{(l)} \rangle_k$.

On the other hand, based on the following equation:

$$\prod_{k=1}^{3}\langle\sigma^{(l)}\rangle_k$$
$$=\prod_{k=1}^{3}\prod_{i\in I}(\langle\sigma_i^{(l)}\rangle_k)^{v_i}$$
$$=\prod_{k=1}^{3}\prod_{i\in I}(H_1(i\parallel V)\cdot\prod_{j=1}^{s}u_j^{H_2(\llbracket w_{i,j}^{(l)}\rrbracket_k)})^{sk\cdot v_i}$$
$$=\prod_{i\in I}(H_1(i\parallel V))^{sk\cdot v_i}\cdot\prod_{k=1}^{3}(\prod_{j=1}^{s}u_j^{\sum_{i\in I}v_i\cdot H_2(\llbracket w_{i,j}^{(l)}\rrbracket_k)})^{sk}$$
$$=\prod_{i\in I}(H_1(i\parallel V))^{sk\cdot v_i}\cdot\prod_{k=1}^{3}(\prod_{j=1}^{s}u_j^{\langle\mu_j^{(l)}\prime\rangle_k})^{sk}.$$

It is obvious that $\prod_{k=1}^{3}(\prod_{j=1}^{s}u_j^{\langle\mu_j^{(l)}\prime\rangle_k})^{sk}$ is masked by $\prod_{i\in I}(H_1(i\parallel V))^{sk\cdot v_i}$. It is difficult to calculate $\prod_{i\in I}(H_1(i\parallel V))^{sk\cdot v_i}$ based on $\prod_{i\in I}(H_1(i\parallel V))^{v_i}$ and g^{sk} because CDH problem in G_1 is hard. This implies that user is not capable of inferring $\prod_{k=1}^{3}(\prod_{j=1}^{s}u_j^{\langle\mu_j^{(l)}\prime\rangle_k})^{sk}$ based on $\prod_{k=1}^{3}\langle\sigma^{(l)}\rangle_k$ let along $\langle\mu_j^{(l)}\prime\rangle_k$ ($k\in[1,3], l\in[1,L], j\in[1,s]$). Therefore, model privacy can be ensured in the model integrity verification process since the real model is not revealed to user.

Theorem 3 (*Detectability*). *Suppose that a model includes L linear layers and each linear layer includes n vectorized parameters. Thus, a model includes $L\times n$ vectorized parameters. Each vectorized parameter is divided into three shares. These shares are respectively stored on three CSs, with each cloud server storing $L\times n$ vectorized parameter shares. Assuming that t vectorized parameters are corrupted and the number of challenged vectorized parameters is m. The probability of detecting the corrupted vectorized parameters is at least* $1-(\frac{L\times n\times 3-t}{L\times n\times 3})^{m\times 3}$.

Proof. There are three CSs and each cloud server stores $L\times n$ vectorized parameter shares. m vectorized parameter shares are challenged in each cloud server. Thus, the total number of vectorized parameter shares stored in the CSs is $L\times n\times 3$ and the number of challenged vectorized parameter shares is $m\times 3$. Let X denotes the number of corrupted vectorized parameters being challenged. The probability P_X of detecting the corrupted vectorized parameters can be calculated:

$$\begin{aligned}P_X &= P\{X\geq 1\}\\ &= 1-P\{X=0\}\\ &= 1-\frac{L\times n\times 3-t}{L\times n\times 3}\cdot\frac{L\times n\times 3-1-t}{L\times n\times 3-1}\cdot\ldots\\ &\quad\cdot\frac{L\times n\times 3-m\times 3+1-t}{L\times n\times 3-m\times 3+1}\\ &\geq 1-(\frac{L\times n\times 3-t}{L\times n\times 3})^{m\times 3}.\end{aligned}$$

As shown in Fig. 4, when the number t of corrupted vectorized parameters is 32, the probability of detecting the corrupted vectorized parameters is at least 95% when the number m of challenged vectorized parameters is set to 60, and the probability of detecting the corrupted vectorized parameters is at least 99% when the number m of challenged vectorized parameters is set to 90. As a result, our scheme can detect the corrupted vectorized parameters with high probability.

The correctness of secure inference and training

The correctness of secure inference and training is analyzed. In the proposed scheme, the RSS technique is employed to divide the inference input α and model parameters β into three shares, denoted as $\alpha=\alpha_1+\alpha_2+\alpha_3$ and $\beta=\beta_1+\beta_2+\beta_3$, respectively. Intuitively, the outputs of secure inference and training protocols may differ from the outputs produced by the initial algorithm. However, the exactness

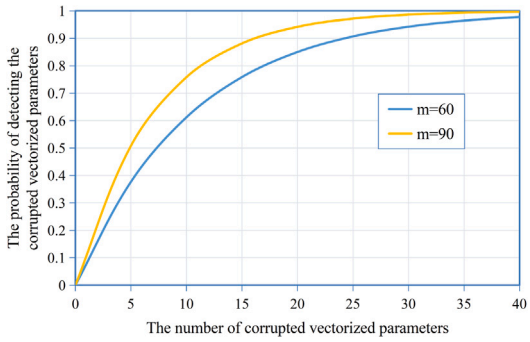

Fig. 4. The probability of detecting the corrupted vectorized parameters.

of the outputs obtained from secure inference and training protocols can be demonstrated through the following derivation. First, in secure inference and training phases, the SecConv and SecFC layers primarily perform the dot product operation. For the input $x=x_1+x_2+x_3$, the output $y=y_1+y_2+y_3$ is obtained. Second, for the SecBN layer, only linear operations are performed in the secure inference phase since the parameters are pre-learned and fixed. During the secure training phase, an iterative method is utilized to convert square root and inverse operations to polynomial operations. Thus, for the input $x=x_1+x_2+x_3$, the equation $y_1+y_2+y_3=ax_1+ax_2+ax_3=ax=y$ still holds. Third, for the SecReLU layer, during the secure inference phase, if the input x is greater than 0, the CSs take the input as the output, *i.e.* $y_1+y_2+y_3=x_1+x_2+x_3=y$. If the input x is less than 0, then the output is 0, *i.e.* $y_1+y_2+y_3=y=0$. During the secure training phase, if the input x is greater than 0, the CSs output 1, *i.e.* $y_1+y_2+y_3=y=1$. If the input x is less than 0, the CSs output 0, *i.e.* $y_1+y_2+y_3=y=0$. Finally, for the SecMP layer, during the secure inference phase, the CSs collaboratively compute the maximum value x_{max} and the index max. The equation $y_1+y_2+y_3=x_{max_1}+x_{max_2}+x_{max_3}=y$ holds. During the secure training phase, the CSs compute the index max of the maximum value and propagate the gradients through the position where the maximum value is located.

The security of secure inference and training

The security proof for the protocols employed in the secure inference and training phases is provided within the universal composability framework [58]. In the semi-honest model, the adversary is defined as having the capability to compromise a maximum of one of three cloud servers CS_k ($k\in[1,3]$). To establish the protocol's security, it is adequate to illustrate that the compromised party's view can be effectively simulated based on the provided inputs and outputs [59]. In the following, the security of the protocols for the secure inference and training phases is analyzed.

Definition 1 ([59]). A protocol is considered secure when a probabilistic polynomial-time simulator is capable of producing an ideal world, in which the adversary's perspective is computationally indistinguishable from their view in real world.

Lemma 1 ([59]). *A protocol achieves perfect simulatability when each of its sub-protocols also achieves perfect simulatability.*

Lemma 2 ([59]). *The element $x\pm r$ is uniformly and independent from the random variable $x\in Z_{2^\lambda}$ when the random element $r\in Z_{2^\lambda}$ is uniformly and independent from x.*

Lemma 3 ([59]). *The protocols SecAdd, SecMul, MSB and the linear combination of SecAdd, SecMul and MSB are secure in semi-honest model.*

Based on Lemma 1, the protocols of secure inference and training achieve security when all sub-protocols are demonstrated to be secure or simulatable. If the protocols are performed in a localized manner, they are amenable to perfect simulation. Consequently, our primary focus centers on proving the security of protocols that require collaborative execution among the three cloud servers CS_k ($k \in [1,3]$), as delineated below.

Theorem 4. *In the proposed scheme, the interactive protocols for secure inference are secure in semi-honest model.*

Proof. During secure inference phase, the operation of SecConv layer, SecFC layer, SecBN layer, SecReLU layer and SecMP layer are based on SecAdd, SecMul, and MSB protocols. SecAdd, SecMul, and MSB protocols are proved secure in Lemma 3. Derived from Lemma 2, it is reasonable to consider all inputs as uniformly random and susceptible to simulation by the simulator. Consequently, simulator is capable of simulating all secure protocols of secure inference. This leads us to conclude that, in compliance with Lemma 1, the interactive protocols for secure inference are secure in semi-honest model.

Theorem 5. *In the proposed scheme, the interactive protocols for secure training are secure in semi-honest model.*

Proof. Similar to Theorem 4, the protocols for secure training rely on simulatable SecAdd, SecMul, and MSB. The detailed proof is omitted here. As a result, the interactive protocols for secure training maintain security in semi-honest model.

7. Performance evaluation

This section presents experimental evaluation of the proposed scheme, focusing on its performance in terms of model integrity verification as well as privacy-preserving inference and training. All experiments are performed on the Ubuntu 20.04 platform with an Intel(R) Core (TM) i7-11700T CPU @3.6 GHz and 16 GB of RAM. We employ GNU Multiple Precision Arithmetic Library [60] and Pairing-Based Cryptography Library [61] implemented in C language, as well as the MP-SPDZ Library [62] implemented in Python language. The basic field size is set to 512 bits, and the element size in Z_p^* is set to $|p|$=160 bits.

A CNN model with the MNIST dataset is selected for experiments. Table 2 illustrates the detailed architecture of our CNN model. To generate the authenticators for the shares of vectorized parameters, the parameters of each SecLin layer (SecConv layer and SecFC layer) are reshaped into a 2D matrix, denoted as $W^{(l)}$ ($l \in [1,5]$), where the second dimension denotes the channel for the convolutional operation and the number of neurons in the output layer for fully connected operation.

7.1. Authenticator generation phase

We evaluate computational time for authenticator generation of each layer. As illustrated in Table 3, the computational time for authenticator generation is influenced by the size of the parameter matrix. Compared with other layers, the computational time for authenticator generation of $W^{(3)}$ is the longest since it has the largest parameter matrix.

Table 2
The architecture of our model.

Layer	Stride	Parameter matrix size	Note
SecConv (input size: 1 × 28 × 28, kernel size: 1 × 6 × 5 × 5, output size: 6 × 24 × 24) – SecBN	1	25 × 6	$W^{(1)}$
SecMP (input size: 6 × 24 × 24, window: 6 × 2 × 2, output size: 6 × 12 × 12) – SecReLU	2	–	–
SecConv (input size: 6 × 12 × 12, kernel size: 6 × 16 × 5 × 5, output size: 16 × 8 × 8) – SecBN	1	150 × 16	$W^{(2)}$
SecMP (input size: 16 × 8 × 8, window: 16 × 2 × 2, output size: 16 × 4 × 4) – SecReLU	2	–	–
SecFC (input size: 256, output size: 120) – SecReLU	NA	256 × 120	$W^{(3)}$
SecFC (input size: 120, output size: 84) – SecReLU	NA	120 × 84	$W^{(4)}$
SecFC (input size: 84, output size: 10)	NA	84 × 10	$W^{(5)}$

Table 3
The computational time for authenticator generation of each layer.

Layer	$W^{(1)}$	$W^{(2)}$	$W^{(3)}$	$W^{(4)}$	$W^{(5)}$
Computational time (s)	0.38	5.30	62.03	20.53	1.97

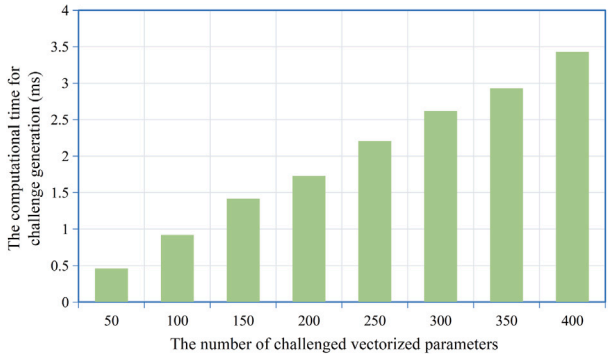

Fig. 5. The computational time for challenge generation of different numbers of challenged vectorized parameters.

7.2. Model integrity verification phase

We evaluate the computational time for challenge generation. The challenged vectorized parameters are selected from 0 to 400 with an interval of 50. Fig. 5 shows that computational time for challenge generation linearly increases with the number of challenged vectorized parameters. When the number of challenged vectorized parameters is set to 400, the computational time for challenge generation is 3.43 ms.

We evaluate the computational time required for proof generation and proof verification. As described in Section 5, if a model parameter is corrupted, its corresponding vectorized parameter's authenticator cannot pass the verification. Considering that the number of the vectorized parameters in each layer is different, we choose a different number of challenged vectorized parameters to ensure that the proportion of challenged vectorized parameters in each layer to the total vectorized parameters in each layer ranges from 5% to 40%. Fig. 6 illustrates the computational time for proof generation of each layer, which exhibits an increase as the proportion of challenged vectorized parameters per layer relative to the total vectorized parameters per layer increases. For five layers $W^{(l)}$ ($l \in [1,5]$), the layer $W^{(3)}$ with the largest parameter matrix size requires the longest computational time. Furthermore, the

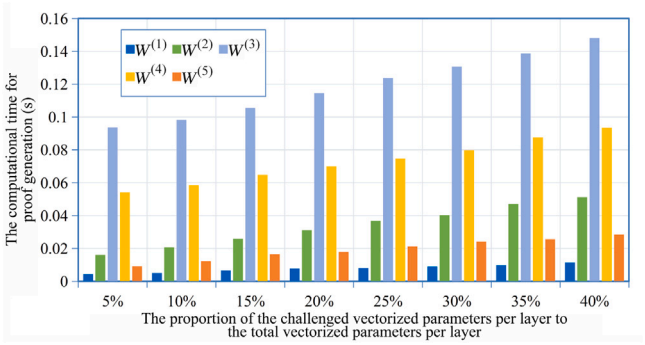

Fig. 6. The computational time for proof generation of each layer.

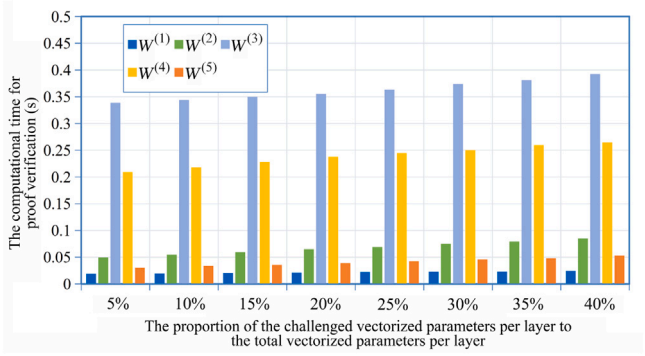

Fig. 7. The computational time for proof verification of each layer.

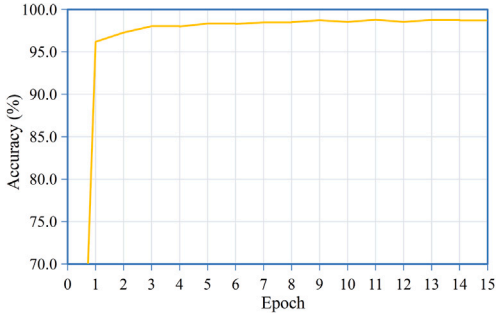

Fig. 8. The secure training accuracy.

proof verification time of each layer is displayed in Fig. 7. Similar to proof generation, as the proportion of challenged vectorized parameters per layer to the total vectorized parameters per layer increases, the proof verification time of each layer also increases.

7.3. Secure inference and training phases

We evaluate the secure training accuracy, the secure forward propagation performance, and the backward propagation performance based on the ring $Z_{2^{64}}$ using the RSS technique. The selected batch size is 256 and the learning rate is 0.01. The training and testing datasets contain 60,000 and 10,000 samples, respectively. We compute the accuracy at every training epoch. Fig. 8 shows that after 15 epochs of secure training, the model achieves an accuracy of 98.85%.

We also evaluate computational time and communication overhead of secure forward and backward propagations under different batch sizes ranging from 0 to 400. As shown in Fig. 9, the computational time and communication overhead for secure forward and backward propagations linearly increase with batch size. Moreover,

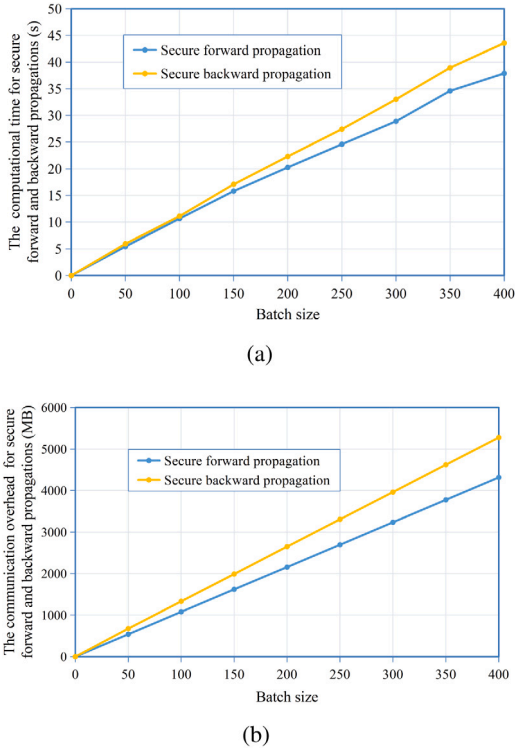

Fig. 9. Performance for secure forward and backward propagations. (a) Computational time of different batch sizes; (b) Communication overhead of different batch sizes.

Table 4
The computational time and communication overhead for secure forward and backward propagations using a single sample.

Phase	Communication time (ms)	Communication overhead (MB)
Secure forward propagation	158.88	11.651
Secure backward propagation	388.98	30.898

the computational time and communication overhead of secure backward propagation are higher than that of secure forward propagation. This is because secure backward propagation requires additional gradient calculations compared to secure forward propagation. Table 4 demonstrates the computational time and communication overhead for secure forward and backward propagations using a single sample (batch size=1). Due to the additional gradient computations involved in backward propagation, the computational time and communication overhead of secure backward propagation are higher than that of secure forward propagation.

Finally, we test the computational time and communication overhead for each layer using a single sample. As depicted in Figs. 10 and 11, the computational time and communication overhead of SecBN layer in secure forward propagation exceed that of other layers. Figs. 12 and 13 show that SecBN layer and SecFC layer incur higher computational time and communication overhead than other layers (SecConv layer, SecMP layer and SecReLU layer) in secure backward propagation. This is because the SecBN layer involves nonlinear operations for computing running mean and running variance, while the SecFC layer includes a significant number of parameters requiring gradient calculation.

7.4. Comparison with existing schemes

We analyze the performance of our scheme in comparison with several related schemes [21,22,24,39], in which CryptoNets [21] and

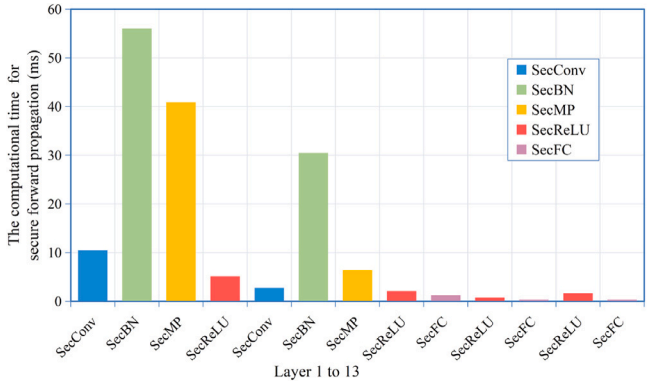

Fig. 10. The computational time for each layer of secure forward propagation using a single sample.

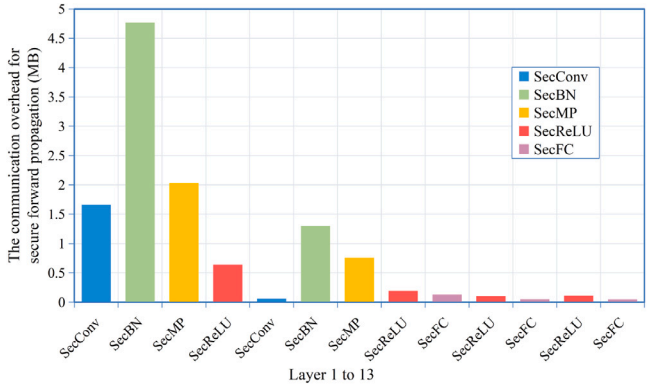

Fig. 11. The communication overhead for each layer of secure forward propagation using a single sample.

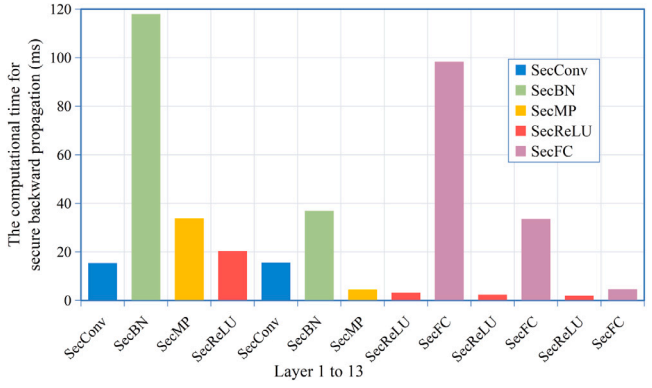

Fig. 12. The computational time for each layer of secure backward propagation using a single sample.

Chameleon [39] support privacy protection, and SecureDL [22] and PVDLI [24] support privacy protection and verifiability. For fair comparison, in this experiment, we adopt the identical CNN neural network architecture and MNIST dataset as utilized in Chameleon [39]. Table 5 shows the detailed model architecture, which consists of one SecConv layer and two SecFC layers. We set the batch size to 5. Table 6 shows the computational time and communication overhead in each round of inference in schemes [21,22,24,39]. As depicted in Table 6, our scheme

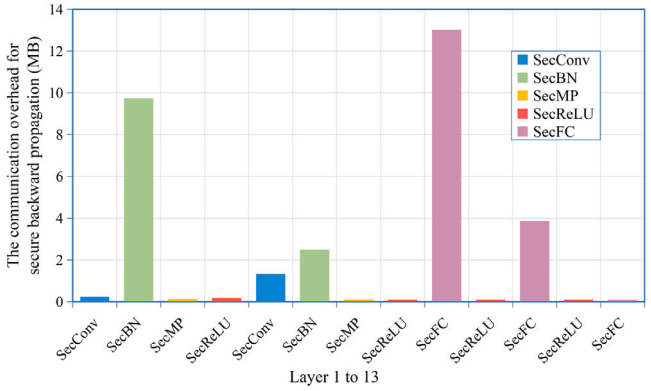

Fig. 13. The communication overhead for each layer of secure backward propagation using a single sample.

Table 5
The CNN architecture.

Layer	Stride	Parameter matrix size
SecConv (input size: $1 \times 28 \times 28$, kernel size: $1 \times 5 \times 5 \times 5$, output size: $5 \times 14 \times 14$) – SecReLU	2	70×14
SecFC (input size: 980, output size: 100) – SecReLU	NA	980×100
SecFC (input size: 100, output size: 10)	NA	100×10

Table 6
Comparison of computational time and communication overhead with schemes [21], [22], [39], [24] during inference phase.

Scheme	Computational time (s)	Communication overhead (MB)
CryptoNets [21]	296.23	372.2
SecureDL [22]	319.00	330.7
Chameleon [39]	2.65	12.90
PVDLI [24]	2.21	12.02
Ours	0.098	6.80

Table 7
Comparison of detection probability with schemes [22] and [24].

Scheme	Detection probability
SecureDL [22]	97.7%
PVDLI [24]	80.0%
Ours	99.9%

exhibits excellent performance during the inference phase. Compared to CryptoNets [21], SecureDL [22], Chameleon [39] and PVDLI [24], our scheme requires the shortest computational time and the lowest communication overhead. The reason is that CryptoNets [21] and SecureDL [22] use HE technique to protect privacy, which involves complex encryption and decryption operations, thereby increasing computational time. Chameleon [39] and PVDLI [24] employ SS technique, which also incur significant communication overhead in dot product operations. Our scheme utilizes RSS technique, which can optimize dot product operations in neural networks, resulting in enhanced efficiency.

During the model integrity verification phase, we set the base field size to 512 bits and the element size in Z_p^* to 160 bits. Assuming that the proportion of corrupted vectorized parameters relative to total vectorized parameters is 2%. As shown in Table 7, when the challenged vectorized parameters are set to 30% of the total vectorized parameters, the probability of detecting the corrupted vectorized parameters in our scheme is at least 99.9%, which is higher than that of SecureDL [22] and PVDLI [24]. This experimental result is consistent with Theorem 3.

8. Discussion

From a managerial perspective, our proposed scheme provides cloud service providers with a more efficient and secure method for performing CNN inference and training. Additionally, it offers users and MO a secure integrity verification method, enhancing service quality and fostering user trust. From an academic perspective, our proposed scheme demonstrates a novel application of privacy-preserving neural network. By leveraging RSS technique and the designed authenticators, the proposed scheme achieves efficient and secure inference and training, while also providing a high detection probability of model integrity verification. This contribution provides new insight for future research in privacy-preserving and model integrity verification.

9. Conclusion

In this paper, we present a novel verifiable and privacy-preserving scheme for CNN inference and training in cloud computing. In the proposed scheme, we enable the verification of the integrity of the model stored in CSs while preserving the privacy of both the data and the model. Compared to existing schemes, our scheme achieves a higher detection probability for verifying model integrity and requires lower computational time and communication overhead during the secure inference phase. We demonstrate the security and the efficiency of the proposed scheme. However, the current design is specifically tailored for CNN. Future work will focus on expanding this work to other types of neural networks.

CRediT authorship contribution statement

Wei Cao: Writing – original draft. **Wenting Shen:** Writing – review & editing, Supervision, Funding acquisition. **Jing Qin:** Validation. **Hao Lin:** Validation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research is supported by National Natural Science Foundation of China (62102211, 62072276), Shandong Provincial Natural Science Foundation, China (ZR2021QF018), and Shandong Province Higher Education Institutions Youth Innovation and Technology Support Program, China (2023KJ365).

Data availability

No data was used for the research described in the article.

References

[1] Kai Huang, Ximeng Liu, Shaojing Fu, Deke Guo, Ming Xu, A lightweight privacy-preserving CNN feature extraction framework for mobile sensing, IEEE Trans. Dependable Secure Comput. 18 (3) (2019) 1441–1455.

[2] Zhuo Ma, Yang Liu, Ximeng Liu, Jianfeng Ma, Feifei Li, Privacy-preserving outsourced speech recognition for smart IoT devices, IEEE Internet Things J. 6 (5) (2019) 8406–8420.

[3] Qian Lu, Zaikai Yang, Hanlin Zhang, Fei Chen, Hequn Xian, MRFE: A deep learning based multidimensional radio frequency fingerprinting enhancement approach for IoT device identification, IEEE Internet Things J. (2024).

[4] Zhihao Zhou, Kyle Chen, Xiaoshi Li, Songlin Zhang, Yufen Wu, Yihao Zhou, Keyu Meng, Chenchen Sun, Qiang He, Wenjing Fan, et al., Sign-to-speech translation using machine-learning-assisted stretchable sensor arrays, Nat. Electron. 3 (9) (2020) 571–578.

[5] Shahnawaz Ahmad, Iman Shakeel, Shabana Mehfuz, Javed Ahmad, Deep learning models for cloud, edge, fog, and IoT computing paradigms: Survey, recent advances, and future directions, Comp. Sci. Rev. 49 (2023) 100568.

[6] Chengliang Tian, Jia Yu, Hanlin Zhang, Haiyang Xue, Cong Wang, Kui Ren, Novel secure outsourcing of modular inversion for arbitrary and variable modulus, IEEE Trans. Serv. Comput. 15 (1) (2022) 241–253, http://dx.doi.org/10.1109/TSC.2019.2937486.

[7] Hanlin Zhang, Peng Gao, Jia Yu, Jie Lin, Neal N Xiong, Machine learning on cloud with blockchain: a secure, verifiable and fair approach to outsource the linear regression, IEEE Trans. Netw. Sci. Eng. 9 (6) (2021) 3956–3967.

[8] Shahnawaz Ahmad, Shabana Mehfuz, Fateh Mebarek-Oudina, Javed Beg, RSM analysis based cloud access security broker: a systematic literature review, Cluster Comput. 25 (5) (2022) 3733–3763.

[9] Zhixiang Zhang, Hanlin Zhang, Xiangfu Song, Jie Lin, Fanyu Kong, Secure outsourcing evaluation for sparse decision trees, IEEE Trans. Dependable Secure Comput. (2024).

[10] Wenjing Gao, Jia Yu, Rong Hao, Fanyu Kong, Xiaodong Liu, Privacy-preserving face recognition with multi-edge assistance for intelligent security systems, IEEE Internet Things J. 10 (12) (2023) 10948–10958.

[11] Jiewang Cai, Wenting Shen, Jing Qin, ESVFL: Efficient and secure verifiable federated learning with privacy-preserving, Inf. Fusion 109 (2024) 102420.

[12] Wenting Shen, Chao Gai, Jia Yu, Ye Su, Keyword-based remote data integrity auditing supporting full data dynamics, IEEE Trans. Serv. Comput. (2023).

[13] Wenting Shen, Jia Yu, Ming Yang, Jiankun Hu, Efficient identity-based data integrity auditing with key-exposure resistance for cloud storage, IEEE Trans. Dependable Secure Comput. (2022).

[14] Wenting Zheng Srinivasan, PM.R.L. Akshayaram, Popa Raluca Ada, DELPHI: A cryptographic inference service for neural networks, in: Proc. 29th USENIX Secur. Symp, 2019, pp. 2505–2522.

[15] Andre Esteva, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau, Sebastian Thrun, Dermatologist-level classification of skin cancer with deep neural networks, Nature 542 (7639) (2017) 115–118.

[16] Eliana Angelini, Giacomo Di Tollo, Andrea Roli, A neural network approach for credit risk evaluation, Q. Rev. Econ. Fin. 48 (4) (2008) 733–755.

[17] Shahnawaz Ahmad, Shabana Mehfuz, Javed Beg, Hybrid cryptographic approach to enhance the mode of key management system in cloud environment, J. Supercomput. 79 (7) (2023) 7377–7413.

[18] Shahnawaz Ahmad, Shabana Mehfuz, Javed Beg, An efficient and secure key management with the extended convolutional neural network for intrusion detection in cloud storage, Concurr. Comput.: Pract. Exper. 35 (23) (2023) e7806.

[19] Wei Cao, Wenting Shen, Zhixiang Zhang, Jing Qin, Privacy-preserving healthcare monitoring for IoT devices under edge computing, Comput. Secur. (2023) 103464.

[20] Shahnawaz Ahmad, Shabana Mehfuz, Javed Beg, Cloud security framework and key management services collectively for implementing DLP and IRM, Mater. Today: Proceedings 62 (2022) 4828–4836.

[21] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, John Wernsing, Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy, in: International Conference on Machine Learning, PMLR, 2016, pp. 201–210.

[22] Guowen Xu, Hongwei Li, Hao Ren, Jianfei Sun, Shengmin Xu, Jianting Ning, Haomiao Yang, Kan Yang, Robert H Deng, Secure and verifiable inference in deep neural networks, in: Annual Computer Security Applications Conference, 2020, pp. 784–797.

[23] Xiaoning Liu, Yifeng Zheng, Xingliang Yuan, Xun Yi, Securely outsourcing neural network inference to the cloud with lightweight techniques, IEEE Trans. Dependable Secure Comput. (2022).

[24] Jia Duan, Jiantao Zhou, Yuanman Li, Caishi Huang, Privacy-preserving and verifiable deep learning inference based on secret sharing, Neurocomputing 483 (2022) 221–234.

[25] Naman Agarwal, Ananda Theertha Suresh, Felix Xinnan X Yu, Sanjiv Kumar, Brendan McMahan, cpSGD: Communication-efficient and differentially-private distributed SGD, Adv. Neural Inf. Process. Syst. 31 (2018).

[26] Zhikun Zhang, Tianhao Wang, Ninghui Li, Shibo He, Jiming Chen, Calm: Consistent adaptive local marginal for marginal release under local differential privacy, in: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, 2018, pp. 212–229.

[27] Danni Yuan, Xiaoyan Zhu, Mingkui Wei, Jianfeng Ma, Collaborative deep learning for medical image analysis with differential privacy, in: 2019 IEEE Global Communications Conference, GLOBECOM, 2019, pp. 1–6, http://dx.doi.org/10.1109/GLOBECOM38437.2019.9014259.

[28] Chuan Ma, Jun Li, Ming Ding, Bo Liu, Kang Wei, Jian Weng, H. Vincent Poor, RDP-GAN: A rényi-differential privacy based generative adversarial network, IEEE Trans. Dependable Secure Comput. 20 (6) (2023) 4838–4852, http://dx.doi.org/10.1109/TDSC.2022.3233580.

[29] Ehsan Hesamifard, Hassan Takabi, Mehdi Ghasemi, Rebecca N Wright, Privacy-preserving machine learning as a service., Proc. Priv. Enhancing Technol. 2018 (3) (2018) 123–142.

[30] Dongwoo Kim, Cyril Guyot, Optimized privacy-preserving cnn inference with fully homomorphic encryption, IEEE Trans. Inf. Forensics Secur. 18 (2023) 2175–2187.

[31] Qifan Wang, Lei Zhou, Jianli Bai, Yun Sing Koh, Shujie Cui, Giovanni Russello, HT2ml: An efficient hybrid framework for privacy-preserving machine learning using HE and TEE, Comput. Secur. 135 (2023) 103509.
[32] Donald Beaver, Precomputing oblivious transfer, in: Advances in Cryptology—CRYPT0'95: 15th Annual International Cryptology Conference Santa Barbara, California, USA, August 27–31, 1995 Proceedings, Springer, 2001, pp. 97–109.
[33] Bita Darvish Rouhani, M. Sadegh Riazi, Farinaz Koushanfar, Deepsecure: Scalable provably-secure deep learning, in: Proceedings of the 55th Annual Design Automation Conference, 2018, pp. 1–6.
[34] Andrew Chi-Chih Yao, How to generate and exchange secrets, in: 27th Annual Symposium on Foundations of Computer Science (Sfcs 1986), IEEE, 1986, pp. 162–167.
[35] Sameer Wagh, Divya Gupta, Nishanth Chandran, Securenn: 3-party secure computation for neural network training., Proc. Priv. Enhancing Technol. 2019 (3) (2019) 26–49.
[36] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, Tal Rabin, F: Honest-majority maliciously secure framework for private deep learning, Proc. Priv. Enhanc. Technol. 2021 (1) (2021) 188–208.
[37] Qi Feng, Debiao He, Zhe Liu, Huaqun Wang, Kim-Kwang Raymond Choo, Securenlp: A system for multi-party privacy-preserving natural language processing, IEEE Trans. Inf. Forensics Secur. 15 (2020) 3709–3721.
[38] Yang Yang, Ke Mu, Robert H. Deng, Lightweight privacy-preserving GAN framework for model training and image synthesis, IEEE Trans. Inf. Forensics Secur. 17 (2022) 1083–1098.
[39] M Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M Songhori, Thomas Schneider, Farinaz Koushanfar, Chameleon: A hybrid secure computation framework for machine learning applications, in: Proceedings of the 2018 on Asia Conference on Computer and Communications Security, 2018, pp. 707–721.
[40] Jian Liu, Mika Juuti, Yao Lu, Nadarajah Asokan, Oblivious neural network predictions via minionn transformations, in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 2017, pp. 619–631.
[41] Chiraag Juvekar, Vinod Vaikuntanathan, Anantha Chandrakasan, {GAZELLE}: A low latency framework for secure neural network inference, in: 27th USENIX Security Symposium (USENIX Security 18), 2018, pp. 1651–1669.
[42] Payman Mohassel, Peter Rindal, ABY3: A mixed protocol framework for machine learning, in: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, 2018, pp. 35–52.
[43] Zahra Ghodsi, Tianyu Gu, Siddharth Garg, Safetynets: Verifiable execution of deep neural networks on an untrusted cloud, Adv. Neural Inf. Process. Syst. 30 (2017).
[44] Jiasi Weng, Jian Weng, Gui Tang, Anjia Yang, Ming Li, Jia-Nan Liu, Pvcnn: Privacy-preserving and verifiable convolutional neural network testing, IEEE Trans. Inf. Forensics Secur. 18 (2023) 2218–2233.
[45] Zecheng He, Tianwei Zhang, Ruby Lee, Sensitive-sample fingerprinting of deep neural networks, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 4729–4737.
[46] Deepthi Praveenlal Kuttichira, Sunil Gupta, Dang Nguyen, Santu Rana, Svetha Venkatesh, Verification of integrity of deployed deep learning models using Bayesian optimization, Knowl.-Based Syst. 241 (2022) 108238.
[47] Heng Yin, Zhaoxia Yin, Zhenzhe Gao, Hang Su, Xinpeng Zhang, Bin Luo, FTG: Score-based black-box watermarking by fragile trigger generation for deep model integrity verification, J. Inf. Int. 2 (1) (2024) 28–41.
[48] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, Kazuma Ohara, High-throughput semi-honest secure three-party computation with an honest majority, in: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, 2016, pp. 805–817.
[49] Alessandro Baccarini, Marina Blanton, Chen Yuan, Multi-party replicated secret sharing over a ring with applications to privacy-preserving machine learning, Proc. Priv. Enhanc. Technol. (2023).
[50] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, Tal Rabin, Falcon: Honest-majority maliciously secure framework for private deep learning, 2020, arXiv preprint arXiv:2004.02229.
[51] Alberto Ibarrondo, Hervé Chabanne, Melek Önen, Banners: Binarized neural networks with replicated secret sharing, in: Proceedings of the 2021 ACM Workshop on Information Hiding and Multimedia Security, 2021, pp. 63–74.
[52] Fei Zheng, Chaochao Chen, Xiaolin Zheng, Mingjie Zhu, Towards secure and practical machine learning via secret sharing and random permutation, Knowl.-Based Syst. 245 (2022) 108609.
[53] Wenxing Zhu, Mengqi Wei, Xiangxue Li, Qiang Li, Securebinn: 3-party secure computation for binarized neural network inference, in: European Symposium on Research in Computer Security, Springer, 2022, pp. 275–294.
[54] Yifeng Zheng, Huayi Duan, Cong Wang, Learning the truth privately and confidently: Encrypted confidence-aware truth discovery in mobile crowdsensing, IEEE Trans. Inf. Forensics Secur. 13 (10) (2018) 2475–2489.
[55] Sergey Ioffe, Christian Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, in: International Conference on Machine Learning, pmlr, 2015, pp. 448–456.
[56] Hvass-Labs, Execution order of ReLU and max-pooling, 2016, https://github.com/tensorflow/tensorflow/issues/3180.
[57] Marcel Keller, Ke Sun, Secure quantized training for deep learning, in: International Conference on Machine Learning, PMLR, 2022, pp. 10912–10938.
[58] Ran Canetti, Universally composable security: A new paradigm for cryptographic protocols, in: Proceedings 42nd IEEE Symposium on Foundations of Computer Science, IEEE, 2001, pp. 136–145.
[59] Dan Bogdanov, Sven Laur, Jan Willemson, Sharemind: A framework for fast privacy-preserving computations, in: European Symposium on Research in Computer Security, Springer, 2008, pp. 192–206.
[60] Torbjörn Granlund, The GNU multiple precision arithmetic library, 2010, http://gmplib.org/.
[61] Ben Lynn, The pairing-based cryptographic library, 2015, https://crypto.stanford.edu/pbc/.
[62] Marcel Keller, MP-SPDZ: A versatile framework for multi-party computation, in: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, 2020, pp. 1575–1590.

Wei Cao received the B.S. degrees from the School of Data Science and Software Engineering, Qingdao University, China, in 2019. He is currently pursuing the M.S. degree in cyberspace security of the College of Computer Science and Technology at Qingdao University. His research interests include privacy protection and machine learning.

Wenting Shen received Ph. D. degree in School of Mathematics from Shandong University, in 2020. She is currently an associate professor of the College of Computer Science and Technology at Qingdao University. She has published several research papers in refereed international journals including IEEE Transactions on Information Forensics and Security and IEEE Transactions on Dependable and Secure Computing. Her research interests include cloud computing security, privacy computing and big data security.

Jing Qin is a professor in School of Mathematics, Shandong University P. R. China. She received her B.S. from Information Engineering University, Zhengzhou, P. R. China in 1982 and Ph.D. degree from School of Mathematics in Shandong University, P. R. China in 2004. Her research interests include computational number theory, information security, design and analysis of security about cryptologic protocols. She has co-authored 2 books and has published about 30 professional research papers. Prof. Qin is a senior member of Chinese Association for Cryptologic Research (CACR) as well as China Computer Federation (CCF).

Hao Lin received the Ph.D. degree s in cyberspace security from the School of Mathematics, Shandong University, Jinan, China in 2022. He is currently a postdoctoral researcher with the Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, Delft, Netherlands. His research interest is cryptography, especially in post-quantum cryptography.