Neurocomputing 483 (2022) 221–234

Contents lists available at ScienceDirect

Neurocomputing

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

Privacy-preserving and verifiable deep learning inference based on secret sharing

Jia Duan [a], Jiantao Zhou [a,*], Yuanman Li [b], Caishi Huang [a]

[a] State Key Laboratory of Internet of Things for Smart City, Department of Computer and Information Science, Faculty of Science and Technology, University of Macau, China
[b] College of Electronics and Information Engineering, Shenzhen University, China

ARTICLE INFO

Article history:
Received 21 February 2021
Revised 22 November 2021
Accepted 15 January 2022
Available online 21 January 2022
Communicated by Xiwei Wang

Keywords:
Deep neural network inference
Deep learning prediction
Secure multi-party computation
Privacy-preserving
Verifiable computation

ABSTRACT

Deep learning inference, providing the model utilization of deep learning, is usually deployed as a cloud-based framework for the resource-constrained client. However, the existing cloud-based frameworks suffer from severe information leakage or lead to significant increase of communication cost. In this work, we address the problem of privacy-preserving deep learning inference in a way that both the privacy of the input data and the model parameters can be protected with low communication and computational costs. Additionally, the user can verify the correctness of results with small overhead, which is very important for critical application. Specifically, by designing secure sub-protocols, we introduce a new layer to collaboratively perform the secure computations involved in the inference. With the cooperation of the secret sharing, we inject the verifiable data into the input, enabling us to check the correctness of the returned inference results. Theoretical analyses and extensive experimental results over MNIST and CIFAR10 datasets are provided to validate the superiority of our proposed privacy-preserving and verifiable deep learning inference (PVDLI) framework.

1. Introduction

In past years, deep neural network (DNN) has achieved remarkable progresses in various fields, such as object detection [39,26], object classification [18,10], machine translation [6,7] and face recognition [40,33]. As the size of datasets increases, the computational intensity of deep learning grows proportionally. Recently, significant advances have been made in GPU hardware, network architectures and algorithms; but for a resource-constrained client, the large-scale DNN training and inference still take an impractically long time and massive computational power.

Fortunately, many DNN frameworks have been suggested to leverage the cloud service to perform the extensive computations. However, these cloud-based frameworks also bring new security and privacy challenges. The privacy disclosure would occur at both the training and the inference phases. Many privacy-preserving frameworks have been proposed to deal with the privacy issues at the training phase [27,8,28,37,23,4,13,19], and the inference phase [46,12,16,32,42]. In the following, we mainly focus on the privacy issues at the inference phase, which are most relevant to our work.

In the deep learning inference phase, the privacy leakage mainly comes from the input data and the deep model itself. Specifically, the input data usually contain lots of sensitive information that should be kept private against the cloud server, which cannot be fully trusted. In addition, the parameters of the well-trained model also should not be revealed to the unauthorized party. Because the well-trained model is usually treated as an important asset due to its extensive training cost. In other words, protecting the privacy of model parameters has the same significance as keeping the input data private.

To address these issues, Ocia et al. [24,25] and Chi et al. [5] proposed to split the neural network into two parts: one is conducted locally, and the other is performed by the cloud. Xu et al. [43] and Shen et al. [36] suggested to utilize obfuscation neural network [43] or data morphing [36] to protect the privacy of input data locally. Additionally, to provide higher security, Zhang et al. [46] and Tian et al. [41] incorporated with homomorphic encryption (HE) [2] to encrypt parts of computations in the inference. They categorized all the DNN computations into two groups: linear and non-linear computations. Only the linear computations are performed on the cloud server in the encrypted domain.

* Corresponding author.
E-mail addresses: xuelandj@gmail.com (J. Duan), jtzhou@um.edu.mo (J. Zhou), yuanmanli@szu.edu.cn (Y. Li), cshuang@um.edu.mo (C. Huang).

https://doi.org/10.1016/j.neucom.2022.01.061
0925-2312/

Although the above frameworks preserved the input privacy, the privacy of model parameters was not considered. Recently, several privacy-preserving frameworks that focus on protecting the privacy of both the input and model parameters were investigated. The works [12,9,15] employed HE to encrypt the whole network and conducted the inference in the encrypted domain. Juvekar et al. [16] proposed a framework that utilizes garbled circuits (GC) [44] and HE to provide the input privacy and model privacy, similar to frameworks[32,29,31,30].

Also, many works resorted to secure multi-party computation to achieve the protection of both input and model parameters. Ma et al. [22] introduced fully non-interactive privacy-preserving inference framework under two non-colluding servers. Furthermore, Liu et al. [21] presented MiniONN for transforming the well-trained neural network to an oblivious neural network with reasonable efficiency. Shamsabadi et al. [35] investigated a private DNN training and inference framework for classification, where the private inference based on secret sharing scheme [34] was conducted among two non-colluding servers.

Additionally, the cloud server may return invalid results for financial incentives. In many deep learning based applications, such as autonomous driving and financial risk assessment, invalid results may cause catastrophic events. Hence, it is very important that the user has a mechanism to verify the correctness of the returned results from the cloud server. Certainly, the overhead induced in performing the verification should be minimized.

In this work, we propose a privacy-preserving and verifiable deep learning inference (PVDLI) framework that the privacy of both the input and model parameters can be protected with low communication and computational costs. Additionally, the user can also verify the correctness of inference results with small overhead. Inspired by the proxy pattern of software design pattern [11], we introduce $N+1$ nodes, called proxy layer, to collaboratively perform the secure computations involved in the inference. To securely perform the activation function (non-linear operation), we approximate the activation function with polynomials during the fine-tuning phase [3]. By injecting labeled verifiable data into the input, we can effectively verify the correctness of the inference results with a high probability. Thanks to the secret sharing, the injected verification data are indistinguishable from the normal ones, and hence, preventing the potential attacks which only return correct results for verification data. It is shown theoretically that the proposed framework can provide verifiability and privacy of both input and model parameters against honest-but-curious participants, even under the challenging case that there exists collusion among the participants. Extensive experimental results are also provided to validate the superiority of our proposed PVDLI framework.

The rest of this paper is organized as follows. Section 2 introduces the preliminary. The system model and design goals are given in Section 3. Section 4 presents the proposed subprotocols. The details of the proposed PVDLI framework are described in Section 5. In Section 6, we provide verifiability analysis and security analysis for evaluating our proposed framework, under the designed security experiments. Section 7 offers the experimental results and finally we conclude in Section 8.

2. Preliminary

Deep learning aims to extract complex features from high-dimensional data and use them to build a model which can relate inputs to outputs. Usually, deep learning architectures are constructed as multi-layer networks so that more abstract features are computed as non-linear functions of lower-level ones. Each layer is connected with the output of the previous layer. In Fig. 1, we show an example of deep neural network inference. By performing a series of computations, the DNN outputs the logit value. Then it produces the class prediction probability of the input by conducting a softmax function with the logit value. Without loss of generality, we detailedly describe a typical block in VGG16 model [38], which is composed of linear computation, batch normalization, activation and pooling.

Linear Computation. Assume that an input $\mathbf{x} \in \mathbb{R}^n$ is fed in this block. Then it performs linear computation as

$$\mathbf{z} = \mathbf{W} \times \mathbf{x} + \mathbf{b} \tag{1}$$

where the matrix $\mathbf{W}$ and vector $\mathbf{b}$ denote the weights and bias. Since the convolutional operation can be transformed to matrix multiplication efficiently, we only consider (1) in the linear computation.

Batch Normalization. After the linear computation, batch normalization is applied to solve internal covariate shift. At the inference phase, the parameters of batch normalization are fixed. The batch normalization can be considered as a linear operation expressed as

$$\mathbf{u} = \gamma z + \beta \tag{2}$$

where γ and β are well-trained and fixed.

Activation Function. Upon receiving the output of batch normalization, activation function is adopted to selectively activate neurons. Here, the widely-used ReLU is adopted, which can be expressed as

$$\mathbf{t} = \mathrm{ReLU}(\mathbf{u}) \tag{3}$$

Pooling. A pooling layer can be considered as a down-sampling operation. Here, an average pooling is employed to divide the input into rectangular pooling regions and computing the average values of each region. Letting Avg denote the average operation, the procedure can be expressed as

$$\mathbf{y} = \mathrm{Avg}(\mathbf{t}) \tag{4}$$

Then, the output of the pooling layer is fed into the next layer for the subsequent computations. Finally, the DNN network outputs a logit value and produces the class prediction probability with a softmax function.

3. System model and design goals

In this section, we provide the details of our proposed framework, starting with the descriptions on design goals and the system model.

3.1. Design goals

We consider a privacy-preserving and verifiable deep learning inference framework with three entities: one model owner $\mathcal{O}$, one user $\mathcal{U}$ and $N+1$ nodes, called proxy layer placed in between the user and the model owner. In general, the security threats faced by the proposed PVDLI come from the behavior of the proxy layer. Assume that the framework is operated under the honest-but-curious setting. It implies that the adversary may follow the framework specification, but are curious about the input data and model parameters. Additionally, from the aspect of verifiability, some nodes in the proxy layer could be lazy. It indicates that the proxy layer may generate the random/false results to save the computational power. To provide the privacy protection and

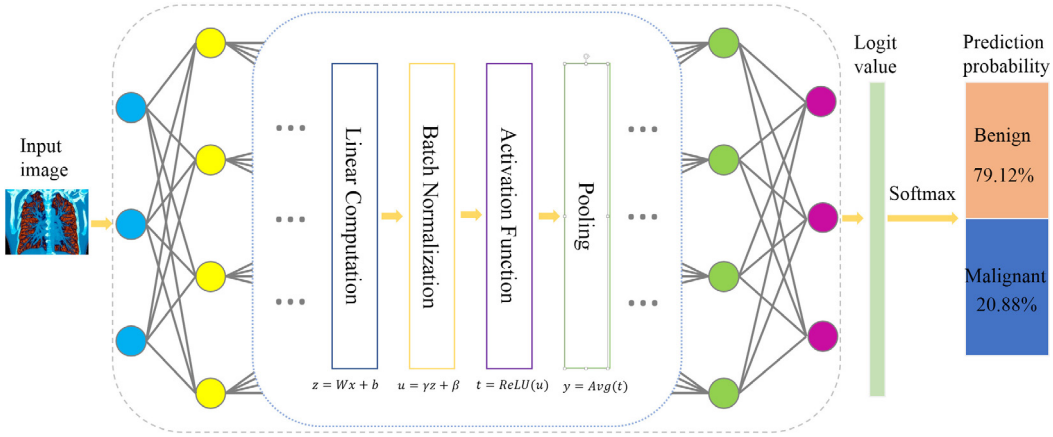

Fig. 1. The architecture of deep learning inference.

verifiability of the proposed framework, we identify the following four design goals.

- **Privacy**: The privacy includes *the privacy of input data* and *the privacy of model parameters*. Specifically, the input data of the user should be kept secret against the proxy layer. Furthermore, the model parameters should not be revealed to the proxy layer.
- **Efficiency**: The computational complexity of the user should be substantially less than the original inference computation on its own. Also, the communication cost among the participants should be minimized.
- **Verifiability**: The correct results must be verified successfully by the user. No false results from a cheating proxy layer can pass the verification with non-negligible probability.
- **Accuracy**: The inference accuracy of the model should be close to the one if the deep learning inference is performed on a single server.

3.2. System model

The proposed privacy-preserving and verifiable deep learning inference framework (PVDLI) is illustrated in Fig. 2. As can be seen, we introduce $N+1$ nodes $\mathscr{E} = \{\mathscr{E}_0, \mathscr{E}_1, \cdots, \mathscr{E}_N\}$ as a proxy layer, which is inspired by the proxy pattern [11]. In software design pattern, a proxy [11] is an agent object that links the client and the real serving object behind the scenes. Similarly, our proposed proxy layer also provides the access of the logical operation, but hides the information of input data and model parameters.

The pseudocode of the whole framework is shown in Algorithm 1. More specifically, the model owner possesses a well-trained deep learning model $\mathscr{F}$ composed of the model parameters $\mathbf{\Theta} = \{\theta_1, \theta_2, \cdots, \theta_M\}$, where M is the total number of the parameters in the network. The user $\mathscr{U}$ requests to feed a private input $\mathbf{x}$ into the model $\mathscr{F}$ for the inference. Firstly, the model owner $\mathscr{O}$ negotiates with the user $\mathscr{U}$ to choose a proxy layer $\mathscr{E}$ composed of one auxiliary node $\mathscr{E}_0$ and N computing nodes $\{\mathscr{E}_1, \cdots, \mathscr{E}_N\}$. Here, the computing nodes $\{\mathscr{E}_1, \cdots, \mathscr{E}_N\}$ are responsible for the computation of DNN inference. The auxiliary node $\mathscr{E}_0$ mainly communicates with each computing node to collect and distribute the temporary results involved in the inference.

Algorithm 1.

Algorithm 1 The procedure of performing PVDLI

Input: Security parameter λ, input data $\mathbf{x} \in \mathbb{Z}_p^n$, deep learning model $\mathscr{F}$, the number of computing nodes N.
Output: Inference result $\mathbf{y}$ or error $\perp$.
1: The model owner $\mathscr{O}$ and the user $\mathscr{U}$ select a proxy layer $\mathscr{E} = \{\mathscr{E}_0, \mathscr{E}_1, \cdots, \mathscr{E}_N\}$ with $N+1$ nodes.
2: The user $\mathscr{U}$ generates a verifiable dataset $\mathbf{D}$.
3: The model owner $\mathscr{O}$ obtains the approximation model $\overline{\mathscr{F}}$ by polynomial approximations. Then it splits the model parameters $\mathbf{\Theta}$ into N shares:
$\left\{\mathbf{\Theta}^{(1)}, \cdots, \mathbf{\Theta}^{(N)}\right\} \leftarrow Split(\mathbf{\Theta}, \lambda, N)$.
4: The user $\mathscr{U}$ inserts the verifiable data
$\{\mathbf{X}, \mathbf{V}, r_v\} \leftarrow Insert(\mathbf{x}, \mathbf{D})$.
5: The user splits the mixed data $\mathbf{X}$ into N shares:
$\left\{\mathbf{X}^{(1)}, \cdots, \mathbf{X}^{(N)}\right\} \leftarrow Split(\mathbf{X}, \lambda, N)$.
6: The proxy layer ϵ conducts a series of deep learning inference computations to obtain logits:
$\left\{\mathbf{L}^{(1)}, \cdots, \mathbf{L}^{(N)}\right\} \leftarrow \left\{f\left(\mathbf{X}^{(1)}; \Theta^{(1)}\right), \cdots, f\left(\mathbf{X}^{(N)}; \Theta^{(N)}\right)\right\}$.
7: The user $\mathscr{U}$ aggregates the logits:
$\mathbf{R} \leftarrow \text{Aggregation}\left(\mathbf{L}^{(1)}, \cdots, \mathbf{L}^{(N)}\right)$.
8: The user $\mathscr{U}$ extracts the result $\mathbf{y} = \mathbf{R}[r_v]$, and the verifiable results $\overline{\mathbf{R}} = \{\mathbf{R}[i] | i \in [1, N], i \neq r_v\}$.
9: if verifiable results $\overline{\mathbf{R}} == \mathbf{V}$
10: **Return** $\mathbf{y}$ as the inference result of input data $\mathbf{x}$.
11: else
12: **Return** an error $\perp$.
13: end if

Secondly, the model owner $\mathscr{O}$ adopts the polynomials to approximate the activation functions of the well-trained model $\mathscr{F}$. To protect the model parameters, the model owner splits the model parameters $\mathbf{\Theta}$ into N shares and distributes the shares to the proxy layer. Here, the shares of the model parameters are generated through a simple yet effective secret sharing scheme [34]. To pro-

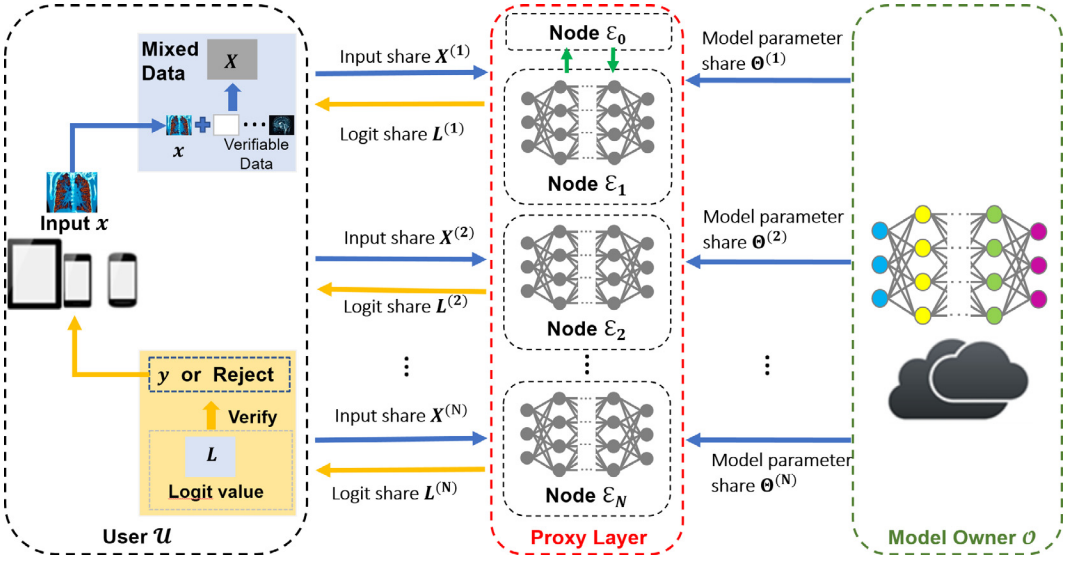

Fig. 2. The system model of the PVDLI framework.

vide verifiability, the user $\mathscr{U}$ mixes several labeled verifiable data with the input data $\mathbf{x}$ to generate the mixed data $\mathbf{X}$. Similarly, the user $\mathscr{U}$ also splits the mixed data $\mathbf{X}$ into N shares and distributes them to proxy layer to preserve the input privacy.

Upon receiving the shares of model parameters $\mathbf{\Theta}^{(i)}$ and the shares of input $\mathbf{X}^{(i)}$, each computing node $\mathscr{E}_i$ in the proxy layer makes secure matrix computation collaboratively. The details will be presented in the subsections below. Eventually, each computing node in the proxy layer outputs and returns its logit value $\mathbf{L}^{(i)}$ to the user $\mathscr{U}$. The user $\mathscr{U}$ aggregates them to obtain the results $\mathbf{R}$. At last, the user $\mathscr{U}$ verifies the correctness of labeled verifiable data. If it passes the verification, the user $\mathscr{U}$ outputs the predicted result $\mathbf{y}$ of the input $\mathbf{x}$. Otherwise, it rejects and outputs an error.

The proposed PVDLI framework consists of the six key modules:

- $\mathsf{Init}(N) \rightarrow (\mathscr{E}, \mathbf{D})$: The model owner $\mathscr{O}$ negotiates with the user $\mathscr{U}$ to chooses a proxy layer with $N+1$ nodes $\mathscr{E} = \{\mathscr{E}_0, \mathscr{E}_1, \cdots, \mathscr{E}_N\}$ for the subsequent collaborative privacy-preserving computations. The user $\mathscr{U}$ generates a verifiable dataset $\mathbf{D}$ for the subsequent verification.
- $\mathsf{DeployModel}(\mathscr{F}, \lambda, N) \rightarrow (\mathbf{\Theta}^{(1)}, \cdots, \mathbf{\Theta}^{(N)})$: The model owner $\mathscr{O}$ approximates the activation functions of the model $\mathscr{F}$ with the polynomials. Then, on input the security parameter λ, the model owner $\mathscr{O}$ splits its model parameters of the approximated model into N shares and distributes them to the computing nodes in the proxy layer. The details of approximation and distributing shares will be provided in the subsequent section.
- $\mathsf{GenSplitData}(\mathbf{x}, \mathbf{D}, \lambda, N) \rightarrow \left(\mathbf{X}^{(1)}, \cdots, \mathbf{X}^{(N)}, \mathbf{V}, r_v\right)$: On input the security parameter λ, the user $\mathscr{U}$ randomly selects several verifiable data from the verifiable dataset $\mathbf{D}$. The matrix $\mathbf{V}$ represents the labels of the verifiable data. Then the user $\mathscr{U}$ picks up a random key r_v and utilizes the key to combine verifiable data with the input $\mathbf{x}$ to generate the mixed data $\mathbf{X}$. At last, the user $\mathscr{U}$ splits the mixed data $\mathbf{X}$ into N shares and distributes the share $\mathbf{X}^{(i)}$ to the computing node $\mathscr{E}_i$ in the proxy layer.
- $\mathsf{Compute}\left(\mathbf{\Theta}^{(1)}, \cdots, \mathbf{\Theta}^{(N)}, \mathbf{X}^{(1)}, \cdots, \mathbf{X}^{(N)}\right) \rightarrow \left(\mathbf{L}^{(1)}, \cdots, \mathbf{L}^{(N)}\right)$: Upon receiving the input share $\mathbf{X}^{(i)}$ and model parameter share $\mathbf{\Theta}^{(i)}$, each computing node $\mathscr{E}_i$ in the proxy layer performs the specific linear operations of the deep learning model, where $1 \leqslant i \leqslant N$. Finally, the computing node $\mathscr{E}_i$ outputs a logit value $\mathbf{L}^{(i)}$ and returns it to the user $\mathscr{U}$, where $1 \leqslant i \leqslant N$.
- $\mathsf{Aggregation}\left(\mathbf{L}^{(1)}, \cdots, \mathbf{L}^{(N)}\right) \rightarrow \mathbf{R}$: Upon receiving all the logit value $\left\{\mathbf{L}^{(1)}, \cdots, \mathbf{L}^{(N)}\right\}$ from the proxy layer, the user $\mathscr{U}$ aggregates them to obtain the result $\mathbf{R}$ of the mixed data $\mathbf{X}$ over the deep learning model $\mathscr{F}$.
- $\mathsf{Verify}(\mathbf{R}, \mathbf{V}, r_v) \rightarrow \mathbf{y} \cup \perp$: On input the predicted results $\mathbf{R}$ and the key r_v, the user $\mathscr{U}$ checks the predicted results of verifiable data with the labels $\mathbf{V}$. If the verification is successful, it extracts and outputs the predicted result $\mathbf{y}$ of the input $\mathbf{x}$. Otherwise, it rejects the results and produces an error $\perp$.

4. Secret sharing based secure sub-protocols

Essentially, deep learning inference is completed by a series of mathematical operations. In our proposed framework, to avoid the leakage of input and model parameters, all the operations have to be conducted in a privacy-preserving manner. Based on secret sharing scheme, we design and implement three secure computation sub-protocols: secure addition, secure division and secure multiplication.

- **Secure Addition Protocol**(SecAdd): Given the random shares $\left(\mathbf{A}^{(1)}, \mathbf{B}^{(1)}\right), \left(\mathbf{A}^{(2)}, \mathbf{B}^{(2)}\right), \cdots, \left(\mathbf{A}^{(N)}, \mathbf{B}^{(N)}\right)$ of two input matrices $\mathbf{A} \in \mathbb{Z}_p^{u \times n}$ and $\mathbf{B} \in \mathbb{Z}_p^{n \times v}$, the proxy layer perform secure addition and output shares $\left(\mathbf{C}^{(1)}, \mathbf{C}^{(2)}, \cdots, \mathbf{C}^{(N)}\right)$, where $\mathbf{C}^{(1)} + \mathbf{C}^{(2)} + \cdots + \mathbf{C}^{(N)} = \mathbf{A} + \mathbf{B}$. During the execution, no intermediate values are exchanged.

Specifically, due to the additive property of secret sharing, it can naturally perform the additive operation in a secure manner. Each computing node $\mathscr{E}_i$ in the proxy layer possesses a pair of share $\left(\mathbf{A}^{(i)}, \mathbf{B}^{(i)}\right)$ of two input matrices $\mathbf{A}$ and $\mathbf{B}$, where $1 \leqslant i \leqslant N$. Then $\mathscr{E}_i$ just adds the shares and outputs the result $\mathbf{C}^{(i)}$. Here, it should be noted that the auxiliary node $\mathscr{E}_0$ does not involved in the computation.

- **Secure Division Protocol**(SecDiv): Given a division factor $k \in \mathbb{Z}_p$, and the random shares $\left(\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \cdots, \mathbf{A}^{(N)}\right)$ of input

$\mathbf{A} \in \mathbb{Z}_p^{u\times n}$, the proxy layer performs secure division and output shares $\left(\mathbf{C}^{(1)}, \mathbf{C}^{(2)}, \cdots, \mathbf{C}^{(N)}\right)$, where $\mathbf{C}^{(1)} + \mathbf{C}^{(2)} + \cdots + \mathbf{C}^{(N)} = \mathbf{A}/k$. Here, the division denotes the element-wised division.

Specifically, Each computing node $\mathscr{E}_i$ possesses a share $\mathbf{A}^{(i)}$ of input matrix $\mathbf{A} \in \mathbb{Z}_p^{u\times n}$, where $1 \leqslant i \leqslant N$. Then the computing node $\mathscr{E}_i$ just divides the share with the factor k and outputs the result $\mathbf{C}^{(i)}$.

- **Secure Matrix Multiplication Protocol** (SecMul): Given the random shares $\left(\mathbf{A}^{(1)}, \mathbf{B}^{(1)}\right), \left(\mathbf{A}^{(2)}, \mathbf{B}^{(2)}\right), \cdots, \left(\mathbf{A}^{(N)}, \mathbf{B}^{(N)}\right)$ of two inputs $\mathbf{A} \in \mathbb{Z}_p^{u\times n}$ and $\mathbf{B} \in \mathbb{Z}_p^{n\times v}$, the proxy layer performs secure multiplication and outputs shares $\left(\mathbf{C}^{(1)}, \mathbf{C}^{(2)}, \cdots, \mathbf{C}^{(N)}\right)$, where $\mathbf{C}^{(1)} + \mathbf{C}^{(2)} + \cdots + \mathbf{C}^{(N)} = \mathbf{A} \times \mathbf{B}$. It should be noted that any share of input and the model parameters should not be sent to the auxiliary node $\mathscr{E}_0$. The auxiliary node $\mathscr{E}_0$ merely communicates with the computing nodes to collect and distribute the temporary result during the multiplication.

Specifically, each computing node $\mathscr{E}_i$ in the proxy layer possesses shares $\left(\mathbf{A}^{(i)}, \mathbf{B}^{(i)}\right)$ of two input matrices $\mathbf{A} \in \mathbb{Z}_p^{u\times n}$ and $\mathbf{B} \in \mathbb{Z}_p^{n\times v}$, where $1 \leqslant i \leqslant N$. Since the multiplication is not supported by the secret sharing scheme, the proxy layer requires extra auxiliary random matrices to obtain the production of $\mathbf{A}$ and $\mathbf{B}$ in a secure manner. Assume that there exists random matrices $\mathbf{P} \in \mathbb{Z}_p^{u\times n}, \mathbf{Q} \in \mathbb{Z}_p^{n\times v}$ and $\mathbf{O} \in \mathbb{Z}_p^{u\times v}$, where $\mathbf{O} = \mathbf{P} \times \mathbf{Q}$. Each computing node $\mathscr{E}_i$ in the proxy layer can only access the share $\left(\mathbf{P}^{(i)}, \mathbf{Q}^{(i)}, \mathbf{O}^{(i)}\right)$ of the matrices $\mathbf{P}, \mathbf{Q}$ and $\mathbf{O}$, respectively, where $1 \leqslant i \leqslant N$. But the matrices $\mathbf{P}, \mathbf{Q}$ and $\mathbf{O}$ are kept secret against all the nodes in the proxy layer (including the auxiliary node $\mathscr{E}_0$). Here, the shares of the matrices $\mathbf{P}, \mathbf{Q}$ and $\mathbf{O}$ can be easily generated by the generating shares algorithm [8]. It should be noted that the auxiliary random matrices can be prepared efficiently in an offline manner, implying that the communication and computational cost do not increase. The procedure of the proposed secure matrix multiplication, illustrated in Fig. 3, consists of the following four parts:

- $\mathsf{GenAux}\left(1^{\lambda}\right) \rightarrow \left\{\mathbf{P}^{(i)}, \mathbf{Q}^{(i)}, \mathbf{O}^{(i)}\right\}_{i=1}^{N}$: Given the security parameter λ, the proxy layer negotiates to choose the auxiliary random matrices shares $\left\{\mathbf{P}^{(i)}, \mathbf{Q}^{(i)}, \mathbf{O}^{(i)}\right\}_{i=1}^{N}$, where $\mathbf{P}^{(i)}, \mathbf{Q}^{(i)}, \mathbf{O}^{(i)}$ present the i-th share of matrices $\mathbf{P} \in \mathbb{Z}_p^{u\times n}, \mathbf{Q} \in \mathbb{Z}_p^{n\times v}, \mathbf{O} \in \mathbb{Z}_p^{u\times v}$, and $\mathbf{O} = \mathbf{P} \times \mathbf{Q}$. The computing node $\mathscr{E}_i$ in the proxy layer can only possess the corresponding share $\left(\mathbf{P}^{(i)}, \mathbf{Q}^{(i)}, \mathbf{O}^{(i)}\right)$, where $1 \leqslant i \leqslant N$.
- $\mathsf{Mask}\left(\left\{\mathbf{A}^{(i)}, \mathbf{B}^{(i)}, \mathbf{P}^{(i)}, \mathbf{Q}^{(i)}\right\}_{i=1}^{N}\right) \rightarrow \left\{\mathbf{E}^{(i)}, \mathbf{F}^{(i)}\right\}_{i=1}^{N}$: The computing node $\mathscr{E}_i$ in the proxy layer, where $1 \leqslant i \leqslant N$, performs the following operations to mask the input matrices shares:

$$\mathbf{E}^{(i)} = \mathbf{A}^{(i)} - \mathbf{P}^{(i)}, \qquad \mathbf{F}^{(i)} = \mathbf{B}^{(i)} - \mathbf{Q}^{(i)} \tag{5}$$

Then, the computing node $\mathscr{E}_i$ sends the share $\left(\mathbf{E}^{(i)}, \mathbf{F}^{(i)}\right)$ to the auxiliary node $\mathscr{E}_0$, where $1 \leqslant i \leqslant N$.

- $\mathsf{Split}\left(\left\{\mathbf{E}^{(i)}, \mathbf{F}^{(i)}\right\}_{i=1}^{N}\right) \rightarrow \left(\mathbf{E}, \mathbf{F}, \left\{\mathbf{G}^{(i)}\right\}_{i=1}^{N}\right)$: Upon receiving all the shares $\left\{\mathbf{E}^{(i)}, \mathbf{F}^{(i)}\right\}_{i=1}^{N}$ from N computing nodes in the proxy layer, the auxiliary node $\mathscr{E}_0$ first aggregates the shares to reconstruct the matrices $\mathbf{E}$ and $\mathbf{F}$, and then computes the production of them.

$$\mathbf{E} = \sum_{i=1}^{N} \mathbf{E}^{(i)}, \qquad \mathbf{F} = \sum_{i=1}^{N} \mathbf{F}^{(i)}, \qquad \mathbf{G} = \mathbf{E} \times \mathbf{F} \tag{6}$$

Finally, the auxiliary node $\mathscr{E}_0$ utilizes additive secret sharing scheme to split the production $\mathbf{G}$ into shares $\left\{\mathbf{G}^{(1)}, \cdots, \mathbf{G}^{(N)}\right\}$. The algorithm details of the generating shares will be described in the subsequent section. Finally, the auxiliary node $\mathscr{E}_0$ returns $\left\{\mathbf{E}, \mathbf{F}, \mathbf{G}^{(i)}\right\}$ to the computing node $\mathscr{E}_i$ in the proxy layer, respectively, where $1 \leqslant i \leqslant N$.

- $\mathsf{Cal}\left(\mathbf{E}, \mathbf{F}, \left\{\mathbf{G}^{(i)}, \mathbf{P}^{(i)}, \mathbf{Q}^{(i)}, \mathbf{O}^{(i)}\right\}_{i=1}^{N}\right) \rightarrow \left\{\mathbf{C}^{(i)}\right\}_{i=1}^{N}$: Upon receiving $\left\{\mathbf{E}, \mathbf{F}, \mathbf{G}^{(i)}\right\}$ from the auxiliary node $\mathscr{E}_0$, each computing node $\mathscr{E}_i$ in the proxy layer, where $1 \leqslant i \leqslant N$, computes:

$$\mathbf{S}^{(i)} = \mathbf{E} \times \mathbf{Q}^{(i)}, \qquad \mathbf{T}^{(i)} = \mathbf{P}^{(i)} \times \mathbf{F} \tag{7}$$

Finally, the computing node $\mathscr{E}_i$ in the proxy layer, where $1 \leqslant i \leqslant N$, outputs a matrix share

$$\mathbf{C}^{(i)} = \mathbf{G}^{(i)} + \mathbf{S}^{(i)} + \mathbf{T}^{(i)} + \mathbf{O}^{(i)} \tag{8}$$

All the shares satisfy the following equation:

$$\mathbf{C} = \mathbf{C}^{(1)} + \cdots + \mathbf{C}^{(N)} = \mathbf{A} \times \mathbf{B} \tag{9}$$

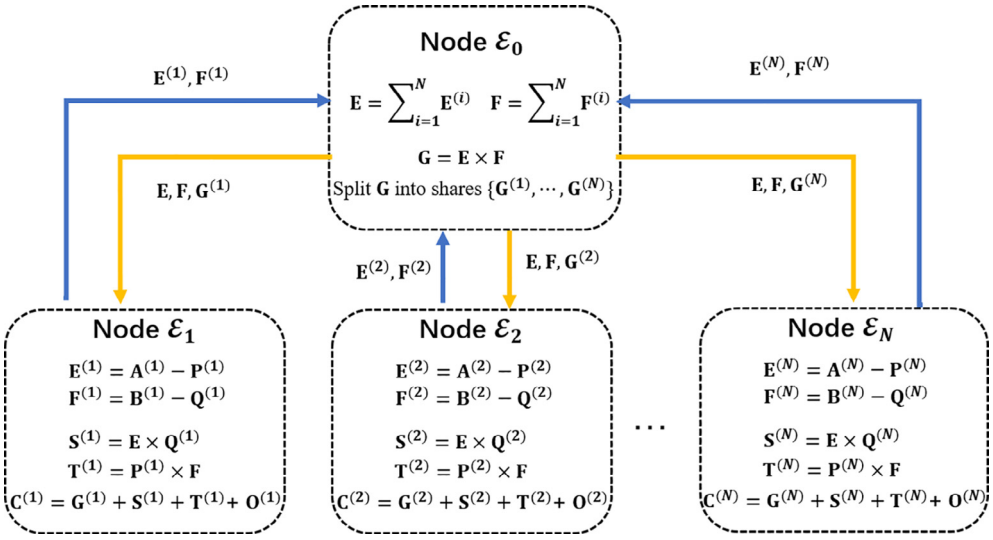

Fig. 3. The architecture of secure matrix multiplication.

The correctness of the secure matrix multiplication can be demonstrated by the following equations:

$$\begin{aligned}\mathbf{C} &= \sum_{i=1}^{N}\mathbf{C}^{(i)}\\ &= \sum_{i=1}^{N}\left(\mathbf{G}^{(i)}+\mathbf{S}^{(i)}+\mathbf{T}^{(i)}+\mathbf{O}^{(i)}\right)\\ &= \mathbf{G}+\mathbf{S}+\mathbf{T}+\mathbf{O}\\ &= \mathbf{E}\times\mathbf{F}+\mathbf{E}\times\mathbf{Q}+\mathbf{P}\times\mathbf{F}+\mathbf{P}\times\mathbf{Q}\\ &= (\mathbf{A}-\mathbf{P})\times(\mathbf{B}-\mathbf{Q})+(\mathbf{A}-\mathbf{P})\times\mathbf{Q}\\ &\quad +\mathbf{P}\times(\mathbf{B}-\mathbf{Q})+\mathbf{P}\times\mathbf{Q}\\ &= \mathbf{A}\times\mathbf{B}\end{aligned} \tag{10}$$

Compared with the existing secure matrix multiplication protocols [35,16,22], our proposed sub-protocols can securely support two matrix multiplication with more than 2 parties, even under the colluding settings. Specifically, since the $N-out-of-N$ secret sharing scheme is adopted, even though some parties collude with each other, our proposed protocols can still effectively protect the privacy of the input and output. Additionally, our proposed protocol can support secure matrix multiplication with small communication cost. Since the random masks matrices $\{\mathbf{P},\mathbf{Q},\mathbf{O}\}$ can be prepared offline and reused, it reduces the communication cost among all the computing parties.

5. The proposed framework

In this section, we provide the details concerning the six parts of our proposed PVDLI framework.

5.1. $\mathsf{Init}(N)\rightarrow(\mathscr{E},\mathbf{D})$

Firstly, the model owner $\mathscr{O}$ negotiates with the user $\mathscr{U}$ to select a proxy layer with $N+1$ nodes including one auxiliary node $\mathscr{E}_0$ and N computing nodes $\{\mathscr{E}_1,\cdots,\mathscr{E}_N\}$ for the subsequent collaborative computations. Here, it is reasonable to assume that $N\geqslant 2$, which is a requirement raised from the secret sharing scheme to be adopted.

Here, it should be noted that the number of nodes will not affect the accuracy of the inference since all the nodes only involve the secure sub-protocols, which are proved to be equivalent to the original computations. But the number of nodes does have impact on the privacy guarantees. Specifically, the larger number of nodes indicates higher privacy-preserving level when collusion occurs among the nodes. More security analysis will be provided in the Section 6.

Then, to provide the verifiability, the user $\mathscr{U}$ generates a verifiable dataset $\mathbf{D}=\{\mathbf{d}_1,\cdots,\mathbf{d}_L\}$. Here, the verifiable dataset can be generated offline and reusable. The requirement of each $\mathbf{d}_i$ is that its inference result (the label) can be easily estimated or obtained. Since the privacy of input data can be well-preserved via secret sharing scheme, in the view of the proxy layer, all the input data are random values. Therefore, the proxy layer cannot distinguish the verifiable data $\mathbf{d}_i$ and the input $\mathbf{x}$. There are many options when choosing the verifiable data $\mathbf{d}_i$; it could be labeled testing data or even dummy data (noise or meaningless data) that can be easily verified.

5.2. $\mathsf{DeployModel}(\mathscr{F},\lambda,N)\rightarrow\left(\mathbf{\Theta}^{(1)},\cdots,\mathbf{\Theta}^{(M)}\right)$

With the well-trained model $\mathscr{F}$, security parameter λ and the number of computing nodes N, this module approximates the activation function of the model $\mathscr{F}$ with polynomials, and splits the model parameters into N shares.

Since the activation functions are usually non-linear operations, which cannot be securely performed with secret sharing scheme. Although many works focused on investigating secure computations of non-linear operations, e.g. secure division, secure comparison and secure exponentiation, high communication cost and inefficiency make these methods impractical in real applications. To solve these challenges, we replace the activation functions with polynomials to facilitate the subsequent secure computations. Specifically, the model owner $\mathscr{O}$ adopts polynomials with degree d to approximate all the activation functions of the well-trained model $\mathscr{F}$. The criterion of choosing the degree d will be discussed in Section 7. The polynomial $f(x)$ can be formulated as

$$f(x)=a_0+a_1x+\cdots+a_dx^d$$

Here, all the coefficients $\{a_0,a_1,\cdots,a_d\}$ in the polynomials are *trainable*. After the replacement, the model owner $\mathscr{O}$ continues to fine-tune the model $\mathscr{F}$ with a few iteration to obtain the new approximation model $\tilde{\mathscr{F}}$.

Let $\mathbf{\Theta}=\{\theta_1,\cdots,\theta_M\}$ denote the flatten vectors of all the weight variables of the deep learning model $\tilde{\mathscr{F}}$, where M denotes the number of parameters in the model. To protect the privacy of the model parameters $\mathbf{\Theta}$, we propose to use a simple yet effective N-out-of-N additive secret sharing scheme [34]. Specifically, the model owner $\mathscr{O}$ splits the model parameters $\mathbf{\Theta}$ into N shares, and distributes the share $\mathbf{\Theta}^{(i)}$ to the computing node $\mathscr{E}_i$ in the proxy layer, where $1\leqslant i\leqslant N$. In this way, each computing node in the proxy layer only possesses a share of the model parameters $\mathbf{\Theta}$.

Specifically, the procedure of generating the parameter shares consists of the following two parts:

- $\mathsf{Setup}\left(1^{\lambda}\right)\rightarrow K^o$: Given the security parameter λ, model owner $\mathscr{O}$ generates a secret key K^o from a specified key space $\{0,1\}^{\lambda}$. Both the resource requirements of the cryptographic algorithm and the adversary's probability of breaking the security can be expressed in terms of the security parameter λ.
- $\mathsf{GenShares}(K^o,\mathbf{\Theta},N)\rightarrow\left(\mathbf{\Theta}^{(1)},\cdots,\mathbf{\Theta}^{(N)}\right)$: The model owner $\mathscr{O}$ utilizes the secret key K^o to generate pseudorandom number sequence so as to split the model parameters $\mathbf{\Theta}$ into N shares. Specifically, on input a seed K^o, a ϵ-forward secure pseudorandom number generator (PRNG) [1] is adopted to produce a pseudorandom sequence $\mathbf{r}^o=\{r_i\}_{i=1}^{t}$, where $t=M\times(N-1)$ and $r_i\in\{0,1\}^{l_1}$. Then the model owner employs this sequence $\mathbf{r}^o$ to generate shares of the model parameter $\mathbf{\Theta}$ by employing the generating algorithm [8], denoted as **Alg**. Then the procedure can be expressed as

$$\mathbf{Alg}(\mathbf{\Theta},\mathbf{r}^o,N)\rightarrow\left\{\mathbf{\Theta}^{(1)},\cdots,\mathbf{\Theta}^{(N)}\right\}\in\mathbb{Z}_p^{M\times N} \tag{12}$$

  Finally, the model owner $\mathscr{O}$ distributes the share $\mathbf{\Theta}^{(i)}$ to the computing node $\mathscr{E}_i$ in the proxy layer, respectively, where $1\leqslant i\leqslant N$.

5.3. $\mathsf{GenSplitData}(\mathbf{x},\mathbf{D},\lambda,N)\rightarrow\left(\mathbf{X}^{(1)},\cdots,\mathbf{X}^{(N)},\mathbf{V},r_v\right)$

With the security parameter λ and the verifiable dataset $\mathbf{D}$, the user $\mathscr{U}$ mixes the verifiable data and input data to generate mixed data $\mathbf{X}$. Then, the user $\mathscr{U}$ splits the mixed data $\mathbf{X}$ into N shares and distributes them to the proxy layer. Specifically, to provide verifiability, the user $\mathscr{U}$ randomly chooses J verifiable data $\{\mathbf{I}_1,\cdots,\mathbf{I}_J\}\in\mathbb{R}^{n\times J}$ from dataset $\mathbf{D}$. Here, let $\mathbf{V}=\{V_1,\cdots,V_J\}$ denote the labels of J verifiable data. Then, the user $\mathscr{U}$ randomly picks up an index r_v, where $1\leqslant r_v\leqslant J$, and mixes the input data $\mathbf{x}$ and the verifiable data $\mathbf{I}$ to construct the mixed data $\mathbf{X}=\{\mathbf{I}_1,\cdots,\mathbf{I}_{r_v-1},\mathbf{x},\mathbf{I}_{r_v},\cdots,\mathbf{I}_J\}\in\mathbb{R}^{n\times(J+1)}$.

Similarly, a secret key K^u is randomly selected from all bit-strings of length λ, then the user $\mathscr{U}$ splits the mixed data $\mathbf{X}$ into shares $\left\{\mathbf{X}^{(1)}, \cdots, \mathbf{X}^{(N)}\right\}$ by the shares generating algorithm in [8]. Additionally, the user $\mathscr{U}$ distributes the share $\mathbf{X}^{(i)}$ to the computing node $\mathscr{E}_i$ in the proxy layer, respectively, where $1 \leqslant i \leqslant N$. In this situation, each computing node in the proxy layer only accesses a share of the mixed data $\mathbf{X}$, indicating that the input privacy can be provided. Since splitting the mixed data into shares, the verifiable data can be perfectly mixed with the input data, implying that the adversary cannot separate them apart.

5.4. $\mathsf{Compute}\left(\mathbf{\Theta}^{(1)}, \cdots, \mathbf{\Theta}^{(N)}, \mathbf{X}^{(1)}, \cdots, \mathbf{X}^{(N)}\right) \rightarrow \left(\mathbf{L}^{(1)}, \cdots, \mathbf{L}^{(N)}\right)$

In this module, the proxy layer securely performs the deep learning inference with the shares of the input and model parameters. Based on secret sharing scheme, we propose three secure sub-protocols: SecAdd, SecDiv and SecMul, whose details have been provided in Section 4. As described in Section 2, the operations in the deep learning inference consist of linear computation, batch normalization, activation and pooling. All these computations can be securely conducted by utilizing proposed sub-protocols. Without loss of generality, we consider a typical block in VGG16 [38] which is composed of linear computation, batch normalization, activation and average pooling, illustrated in Fig. 4. Here, to provide the verifiability, we feed a batch of data $\mathbf{X}$ (consisting of the original input $\mathbf{x}$ and verifiable data $\mathbf{I}$) into the network for inference. The computation of this block can be expressed as:

$$\mathbf{Z} = \mathbf{W} \times \mathbf{X} + \mathbf{B} \tag{13}$$

$$\mathbf{U} = \gamma Z + \beta \tag{14}$$

$$\mathbf{T} = a_1\mathbf{U} + \cdots + a_d\mathbf{U}^d \tag{15}$$

$$\mathbf{Y} = \mathsf{Avg}(\mathbf{T}) \tag{16}$$

where Avg denotes the average operation, $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are well-trained and fixed. Subsequently, we present how to securely perform these computations with the proxy layer in details.

Linear Computation. Recall that each computing node $\mathscr{E}_i$ possesses the model parameters share $\mathbf{\Theta}^{(i)}$ and input share $\mathbf{X}^{(i)}$, consisting of the shares of input $\mathbf{x}$ and verifiable data $\mathbf{I}$. Firstly, the computing node $\mathscr{E}_i$ extracts and reformats $\mathbf{\Theta}^{(i)}$ to obtain the weight matrix $\mathbf{W}^{(i)} \in \mathbb{Z}_p^{u\times n}$ and bias $\mathbf{b}^{(i)} \in \mathbb{Z}_p^u$. Here, due to the change of input data $\mathbf{X}^{(i)} \in \mathbb{Z}_p^{n\times(J+1)}$ (rather than the original input $\mathbf{x} \in \mathbb{Z}_p^n$), the bias $\mathbf{b}^{(i)} \in \mathbf{Z}_p^u$ is required to extend to matrix $\mathbf{B}^{(i)} \in \mathbb{Z}_p^{u\times(J+1)}$ for batch inference, where $\mathbf{B}^{(i)} = \mathbf{b}^{(i)} \times \mathbf{1}$. Then, to perform the linear computation $\mathbf{Z} = \mathbf{W} \times \mathbf{X} + \mathbf{B}$, the proxy layer merely performs secure matrix multiplication and secure addition, expressed as:

$$\mathsf{SecMul}\left(\left\{\mathbf{W}^{(i)}, \mathbf{X}^{(i)}\right\}_{i=1}^N\right) \rightarrow \mathbf{C}^{(1)}, \cdots, \mathbf{C}^{(N)} \tag{17}$$

$$\mathsf{SecAdd}\left(\left\{\mathbf{C}^{(i)}, \mathbf{B}^{(i)}\right\}_{i=1}^N\right) \rightarrow \mathbf{Z}^{(1)}, \cdots, \mathbf{Z}^{(N)} \tag{18}$$

The output $\left\{\mathbf{Z}^{(1)}, \cdots, \mathbf{Z}^{(N)}\right\}$ are the shares of $\mathbf{Z}$, where $\mathbf{Z} = \mathbf{W} \times \mathbf{X} + \mathbf{B}$.

Batch Normalization. After linear computation, batch normalization is applied to solve internal covariate shift. Since the parameters in the batch normalization are fixed at the inference phase, the batch normalization operation can be performed as a linear computation described in (14). Similarly, each computing node $\mathscr{E}_i$ in the proxy layer can extract from $\mathbf{\Theta}^{(i)}$ and reformat to obtain the shares $\boldsymbol{\gamma}^{(i)}$ and $\boldsymbol{\beta}^{(i)}$, where $1 \leqslant i \leqslant N$. Then, the proxy layer performs secure matrix multiplication and secure addition to complete the batch normalization, expressed as

$$\mathsf{SecMul}\left(\left\{\boldsymbol{\gamma}^{(i)}, \mathbf{Z}^{(i)}\right\}_{i=1}^N\right) \rightarrow \mathbf{C}^{(1)}, \cdots, \mathbf{C}^{(N)} \tag{19}$$

$$\mathsf{SecAdd}\left(\left\{\mathbf{C}^{(i)}, \boldsymbol{\beta}^{(i)}\right\}_{i=1}^N\right) \rightarrow \mathbf{U}^{(1)}, \cdots, \mathbf{U}^{(N)} \tag{20}$$

Activation. As described in Section 5.2, we have replaced the activation functions with the approximated polynomials [3]. In this way, the non-linear operation is transformed to linear computations. Since it is similar with the above linear computation, we just provide the major procedures. Specifically, in (16), the coefficients $\{a_1, a_2, \cdots, a_d\}$ are extracted from $\mathbf{\Theta}^{(i)}$. The element-wise exponentiation can be securely implemented by several multiplications. Hence, the computation of activation can be approximated with polynomials by securely performing a series of secure multiplications and additions. Eventually, each computing node $\mathscr{E}_i$ outputs a share $\mathbf{T}^{(i)}$, where $1 \leqslant i \leqslant N$.

Average Pooling. The main procedure of average pooling is to add the values in a rectangular region and perform the average. It consists of addition and division with pool size. Hence, we can also utilize the secure addition and secure division to complete the average pooling.

Finally, all the computations in the deep learning inference can be performed one layer by one layer. Finally, in the last layer, each computing node $\mathscr{E}_i$ outputs a logit value share, expressed as $\mathbf{L}^{(i)}$, where $1 \leqslant i \leqslant N$.

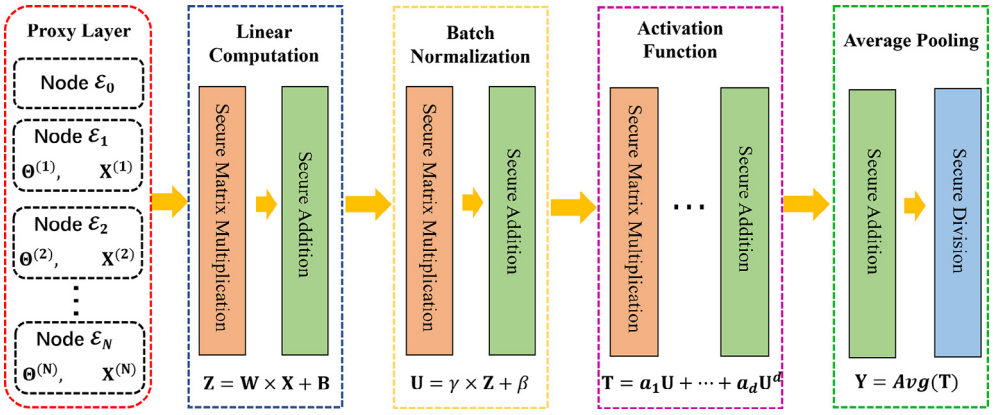

Fig. 4. Performing the computations by secure sub protocols.

5.5. Aggregation$\left(\mathbf{L}^{(1)},\cdots,\mathbf{L}^{(N)}\right)\rightarrow\mathbf{R}$

Upon receiving all the logit value shares $\left\{\mathbf{L}^{(1)},\cdots,\mathbf{L}^{(N)}\right\}$ from the proxy layer, the user $\mathscr{U}$ aggregates them to obtain the logit value of the deep model.

$$\mathbf{L}=\sum_{i=1}^{N}\mathbf{L}^{(i)} \tag{21}$$

The final inference result $\mathbf{R}$ depends on the type of the deep learning task. If the deep learning model is a prediction task, then $\mathbf{R}=\mathbf{L}$. If it is a classification task, then $\mathbf{R}=\mathrm{softmax}(\mathbf{L})$.

5.6. Verify$(\mathbf{R},\mathbf{V},r_v)\rightarrow\mathbf{y}\quad\cup\perp$

Upon aggregating all the shares to obtain the inference results $\mathbf{R}$, the user $\mathscr{U}$ checks the correctness of the inference. Specifically, on input the key r_v, the user $\mathscr{U}$ extracts the r_v-th column vector $\mathbf{y}=\mathbf{R}_{r_v}$. Let $\overline{\mathbf{R}}$ denote the rest of the matrix $\mathbf{R}$. Then the user $\mathscr{U}$ compares the results $\overline{\mathbf{R}}$ with the labels $\mathbf{V}$. If it is the same, the user $\mathscr{U}$ accepts $\mathbf{y}$ as the inference result of the input $\mathbf{x}$. Otherwise, it rejects and outputs an error $\perp$.

It should be noted that, if the input privacy is not provided, meaning that the proxy layer can easily distinguish the verifiable data from input data $\mathbf{x}$, the proxy layer can output the correct result of verifiable data and false result of input data to destroy the verifiability. Hence, the success of verification is coupled with the protection of input privacy.

6. Security experiments, security analysis and verifiability analysis

In this section, we theoretically analyze our proposed framework under the honest-but-curious setting. Similar with the works [45,14], we conduct the security analysis based on the computational indistinguishability with designed experiments. Before giving the analysis, we first describe the security experiments, based on which we then present the security analysis. We use the notation $x\xleftarrow{R}\mathscr{S}$ to denote that x is chosen uniformly at random from the set $\mathscr{S}$. A function $f(n)$ is said to be negligible if for a sufficiently large n, its value is smaller than the inverse of any polynomial $poly(n)$.

6.1. Security experiments

The security requirement of privacy includes *the privacy of input data* and *the privacy of model parameters*, meaning that both the privacy of input data and the model parameters should be protected. Before discussing the security of the whole framework, we first analyze the security experiments of three sub-protocols: secure addition, secure division and secure multiplication.

We first assume that the proposed framework is operated under non-collusion situation, indicating that there is no collusion between the auxiliary node $\mathscr{E}_0$ and computing nodes $\{\mathscr{E}_1,\cdots,\mathscr{E}_N\}$ in the proxy layer. Due to the natural property of secret sharing scheme, secure addition protocol SecAdd and secure division protocol SecDiv are secure against honest-but-curious proxy layer. In the following, we mainly focus on analyzing security experiments of secure multiplication protocol.

Firstly, we analyze the security game of secure matrix multiplication with honest-but-curious nodes in the proxy layer. Since the auxiliary node $\mathscr{E}_0$ and computing nodes $\{\mathscr{E}_1,\cdots,\mathscr{E}_N\}$ have different knowledge of the intermediate data, we consider the auxiliary node $\mathscr{E}_0$ and computing nodes $\{\mathscr{E}_1,\cdots,\mathscr{E}_N\}$ separately. We first analyze the security game of secure matrix multiplication with honest-but-curious computing nodes $\{\mathscr{E}_1,\cdots,\mathscr{E}_N\}$. Under this scenario, all the computing nodes $\{\mathscr{E}_1,\cdots,\mathscr{E}_N\}$ in the proxy layer follow the system specification; but each computing node attempts to obtain or recover the matrix $\mathbf{A}$ and $\mathbf{B}$. Since the secure matrix multiplication protocol performs the same operation with the matrix $\mathbf{A}$ and $\mathbf{B}$, we only need to analyze one, and the result is applicable to the other one. For simplicity, we consider to analyze the privacy of $\mathbf{A}$ as an example, and consider the computing node $\mathscr{E}_0$ as the challenger, who protects the matrix $\mathbf{A}$ from other computing nodes' attack. In the experiment, the adversary $\mathscr{A}$ is allowed to query the protocol on the matrix $\mathbf{A}_j$ of its choice a polynomial times, and obtain the input matrix shares $\overline{\mathbf{A}}=\left\{\mathbf{A}^{(2)},\cdots,\mathbf{A}^{(N)}\right\}$, corresponding results shares $\overline{\mathbf{C}}_j=\left\{\mathbf{C}^{(2)},\cdots,\mathbf{C}^{(N)}\right\}$, the intermediate result $\mathbf{E}_j,\mathbf{F}_j$ and $\overline{\mathbf{G}}_j=\left\{\mathbf{G}_j^{(2)},\cdots,\mathbf{G}_j^{(N)}\right\}$, where j is the index for the query. $\mathscr{A}$ outputs two matrices $\mathbf{A}_{[0]}$ and $\mathbf{A}_{[1]}$ on which it would like to be challenged. A random bit b is drawn. $\mathscr{A}$ is then given $\overline{\mathbf{C}}_{[b]}$, which is the output share of $\mathbf{A}_{[b]}$ and $\mathbf{B}_{[b]}$. Eventually, the adversary $\mathscr{A}$ outputs its guess for b and wins if it could correctly determine the value of b, i.e., distinguishing between $\mathbf{A}_{[0]}$ and $\mathbf{A}_{[1]}$. Since the query is almost the same as that of distinguishing procedure, in the below, we only provide the procedures after q times queries for simplicity.

Experiment $\mathbf{Exp}_{\mathscr{A}}^{\mathsf{CPA-M-I}}(\lambda,N)$

$\left\{\mathbf{A}_{[0]},\mathbf{A}_{[1]}\right\}\leftarrow\mathscr{A}\left(\left\{\overline{\mathbf{A}}_j,\overline{\mathbf{C}}_j,\overline{\mathbf{G}}_j,\mathbf{E}_j,\mathbf{F}_j\right\}_{j=1}^{q}\right)$

$b\xleftarrow{R}\{0,1\},\quad K^o,K^u\leftarrow\mathsf{Setup}\left(1^{\lambda}\right),\quad\mathbf{B}_{[b]}\xleftarrow{R}\mathbb{Z}_p^{n\times v}$

$\left\{\mathbf{P}_{[b]}^{(i)}\right\}_{i=1}^{N}\xleftarrow{R}\mathbb{Z}_p^{u\times n},\quad\left\{\mathbf{Q}_{[b]}^{(i)}\right\}_{i=1}^{N}\xleftarrow{R}\mathbb{Z}_p^{n\times v},\quad\left\{\mathbf{O}_{[b]}^{(i)}\right\}_{i=1}^{N}\xleftarrow{R}\mathbb{Z}_p^{u\times v}$

$\left\{\mathbf{B}_{[b]}^{(1)},\cdots,\mathbf{B}_{[b]}^{(N)}\right\}\leftarrow\mathsf{GenShares}\left(K^u,\mathbf{B}_{[b]},N\right)$

$\left\{\mathbf{A}_{[b]}^{(1)},\cdots,\mathbf{A}_{[b]}^{(N)}\right\}\leftarrow\mathsf{GenShares}\left(K^o,\mathbf{A}_j,N\right)$

$\left\{\mathbf{E}_{[b]}^{(i)},\mathbf{F}_{[b]}^{(i)}\right\}_{i=1}^{N}\leftarrow\mathsf{Mask}\left(\left\{\mathbf{A}_{[b]}^{(i)},\mathbf{B}_{[b]}^{(i)},\mathbf{P}_{[b]}^{(i)},\mathbf{Q}_{[b]}^{(i)}\right\}_{i=1}^{N}\right)$

$\left(\mathbf{E}_{[b]},\mathbf{F}_{[b]},\left\{\mathbf{G}_{[b]}^{(i)}\right\}_{i=1}^{N}\right)\leftarrow\mathsf{Split}\left(\left\{\mathbf{E}_{[b]}^{(i)},\mathbf{F}_{[b]}^{(i)}\right\}_{i=1}^{N}\right)$

$\left\{\mathbf{C}_{[b]}^{(i)}\right\}_{i=1}^{N}\leftarrow\mathsf{Cal}\left(\mathbf{E}_{[b]},\mathbf{F}_{[b]},\left\{\mathbf{G}_{[b]}^{(i)},\mathbf{P}_{[b]}^{(i)},\mathbf{Q}_{[b]}^{(i)},\mathbf{O}_{[b]}^{(i)}\right\}_{i=1}^{N}\right)$

$\overline{\mathbf{C}}_{[b]}\triangleq\left\{\mathbf{C}_{[b]}^{(i)}\right\}_{i=2}^{N},\quad\overline{\mathbf{A}}_{[b]}\triangleq\left\{\mathbf{A}_{[b]}^{(i)}\right\}_{i=2}^{N},\quad\overline{\mathbf{G}}_{[b]}\triangleq\left\{\mathbf{G}_{[b]}^{(i)}\right\}_{i=2}^{N}$

$\hat{b}\leftarrow\mathscr{A}\left(\left\{\overline{\mathbf{A}}_j,\overline{\mathbf{C}}_j,\overline{\mathbf{G}}_j,\mathbf{E}_j,\mathbf{F}_j\right\}_{j=1}^{q},\overline{\mathbf{A}}_{[b]},\overline{\mathbf{C}}_{[b]},\overline{\mathbf{G}}_{[b]},\mathbf{E}_{[b]},\mathbf{F}_{[b]}\right)$

if $\hat{b}=b$ return 1, else return0

For any $\lambda\in\mathbb{N}$, we define the advantage of an adversary $\mathscr{A}$ making at most $q=poly(\lambda)$ queries in the above security game as

$$\mathbf{Adv}_{\mathscr{A}}^{\mathsf{CPA-M-I}}(\lambda,N,q)=\left|\Pr\left[\mathbf{Exp}_{\mathscr{A}}^{\mathsf{CPA-M-I}}(\lambda,N)=1\right]-1/2\right| \tag{22}$$

Definition 1. We say that a secure matrix multiplication protocol ensures the privacy of the matrix $\mathbf{A}$ against any honest-but-curious computing nodes $\{\mathscr{E}_1,\cdots,\mathscr{E}_N\}$ in the proxy layer if for any adversary $\mathscr{A}$ and λ, it holds that the advantage $\mathbf{Adv}_{\mathscr{A}}^{\mathsf{CPA-M-I}}(\lambda,N,q)$ is negligible.

Secondly, we analyze the security game of secure matrix multiplication with honest-but-curious auxiliary node $\mathscr{E}_0$ in the proxy layer. Under this scenario, all the nodes in the proxy layer follow the system specification; but the auxiliary node $\mathscr{E}_0$ attempts to obtain or recover the matrix $\mathbf{A}$ and $\mathbf{B}$. For simplicity, we consider to analyze the privacy of $\mathbf{A}$ as an example. In the experiment, the

adversary $\mathscr{A}$ is allowed to query the protocol on the matrix $\mathbf{A}_j$ of its choice a polynomial times, and obtain the corresponding result shares $\widehat{\mathbf{C}}_j = \left\{\mathbf{C}^{(1)}, \cdots, \mathbf{C}^{(N)}\right\}$ the intermediate result $\mathbf{E}_j, \mathbf{F}_j, \mathbf{G}_j$ and $\widehat{\mathbf{G}}_j = \left\{\mathbf{G}_j^{(1)}, \cdots, \mathbf{G}_j^{(N)}\right\}$, where j is the index for the query. $\mathscr{A}$ outputs two matrices $\mathbf{A}_{[0]}$ and $\mathbf{A}_{[1]}$ on which it would like to be challenged. A random bit b is drawn. $\mathscr{A}$ is then given $\overline{\mathbf{C}}_{[b]}$, which is the output share of $\mathbf{A}_{[b]}$ and $\mathbf{B}_{[b]}$. Eventually, the adversary $\mathscr{A}$ outputs its guess for b and wins if it could correctly determine the value of b, i.e., distinguishing between $\mathbf{A}_{[0]}$ and $\mathbf{A}_{[1]}$. Since the query is almost the same as that of distinguishing procedure, in the below, we only provide the procedures after q times queries for simplicity.

For any $\lambda \in \mathbb{N}$, we define the advantage of an adversary $\mathscr{A}$ making at most $q = poly(\lambda)$ queries in the above security game as

$$\mathbf{Adv}_{\mathscr{A}}^{\mathsf{CPA-M-II}}(\lambda, N, q) = \left|\Pr\left[\mathbf{Exp}_{\mathscr{A}}^{\mathsf{CPA-M-II}}(\lambda, N) = 1\right] - 1/2\right| \tag{23}$$

Definition 2. We say that a secure matrix multiplication protocol ensures the privacy of the matrix $\mathbf{A}$ against any honest-but-curious auxiliary node $\mathscr{E}_0$ in the proxy layer if for any adversary $\mathscr{A}$ and λ, it holds that the advantage $\mathbf{Adv}_{\mathscr{A}}^{\mathsf{CPA-M-II}}(\lambda, N, q)$ is negligible.

Experiment $\mathbf{Exp}_{\mathscr{A}}^{\mathsf{CPA-M-II}}(\lambda, N)$

$$\left(\{\mathbf{A}_{[0]}, \mathbf{A}_{[1]}\} \leftarrow \mathscr{A}\left(\left\{\widehat{\mathbf{C}}_j, \widehat{\mathbf{G}}_j, \mathbf{E}_j, \mathbf{F}_j, \mathbf{G}_j\right\}_{j=1}^{q}\right)\right.$$

$$b \xleftarrow{R} \{0,1\}, \quad K^o, K^u \leftarrow \mathsf{Setup}\left(1^{\lambda}\right), \mathbf{B}_{[b]} \xleftarrow{R} \mathbb{Z}_p^{n\times v}$$

$$\left\{\mathbf{P}_{[b]}^{(i)}\right\}_{i=1}^{N} \xleftarrow{R} \mathbb{Z}_p^{u\times n}, \left\{\mathbf{Q}_{[b]}^{(i)}\right\}_{i=1}^{N} \xleftarrow{R} \mathbb{Z}_p^{n\times v}, \left\{\mathbf{O}_{[b]}^{(i)}\right\}_{i=1}^{N} \xleftarrow{R} \mathbb{Z}_p^{u\times v}$$

$$\left\{\mathbf{B}_{[b]}^{(1)}, \cdots, \mathbf{B}_{[b]}^{(N)}\right\} \leftarrow \mathsf{GenShares}\left(K^u, \mathbf{B}_{[b]}, N\right)$$

$$\left\{\mathbf{A}_{[b]}^{(1)}, \cdots, \mathbf{A}_{[b]}^{(N)}\right\} \leftarrow \mathsf{GenShares}\left(K^o, \mathbf{A}_j, N\right)$$

$$\left\{\mathbf{E}_{[b]}^{(i)}, \mathbf{F}_{[b]}^{(i)}\right\}_{i=1}^{N} \leftarrow \mathsf{Mask}\left(\left\{\mathbf{A}_{[b]}^{(i)}, \mathbf{B}_{[b]}^{(i)}, \mathbf{P}_{[b]}^{(i)}, \mathbf{Q}_{[b]}^{(i)}\right\}_{i=1}^{N}\right)$$

$$\left(\mathbf{E}_{[b]}, \mathbf{F}_{[b]}, \left\{\mathbf{G}_{[b]}^{(i)}\right\}_{i=1}^{N}\right) \leftarrow \mathsf{Split}\left(\left\{\mathbf{E}_{[b]}^{(i)}, \mathbf{F}_{[b]}^{(i)}\right\}_{i=1}^{N}\right)$$

$$\left\{\mathbf{C}_{[b]}^{(i)}\right\}_{i=1}^{N} \leftarrow \mathsf{Cal}\left(\mathbf{E}_{[b]}, \mathbf{F}_{[b]}, \left\{\mathbf{G}_{[b]}^{(i)}, \mathbf{P}_{[b]}^{(i)}, \mathbf{Q}_{[b]}^{(i)}, \mathbf{O}_{[b]}^{(i)}\right\}_{i=1}^{N}\right)$$

$$\overline{\mathbf{C}}_{[b]} \triangleq \left\{\mathbf{C}_{[b]}^{(i)}\right\}_{i=2}^{N}, \quad \overline{\mathbf{G}}_{[b]} \triangleq \left\{\mathbf{G}_{[b]}^{(i)}\right\}_{i=2}^{N}$$

$$\hat{b} \leftarrow \mathscr{A}\left(\left\{\widehat{\mathbf{C}}_j, \widehat{\mathbf{G}}_j, \mathbf{E}_j, \mathbf{F}_j, \mathbf{G}_j\right\}_{j=1}^{q}, \widehat{\mathbf{C}}_{[b]}, \widehat{\mathbf{G}}_{[b]}, \mathbf{E}_{[b]}, \mathbf{F}_{[b]}, \mathbf{G}_{[b]}\right)$$

if $\hat{b} = b$ return 1, else return 0

6.2. Security analysis

To formulate the data secrecy in our proposed framework, we proceed with security games and analyze the adversary $\mathscr{A}$'s advantage in winning the experiment.

Theorem 1. *The proposed secure matrix multiplication protocol holds indistinguishability with honest-but-curious computing nodes* $\{\mathscr{E}_1, \cdots, \mathscr{E}_N\}$ *in the proxy layer according to the Definition 1.*

Proof. The proof is based on the analysis of $\mathscr{A}$'s advantage $\mathbf{Adv}_{\mathscr{A}}^{\mathsf{CPA-M-I}}(\lambda, N, q)$. Similar to the proof of above theorems, we can also define security games to analyze the adversary's advantage.

Game $\mathbf{Game_0}$. Define $\mathbf{Game_0}$ to be the same as $\mathbf{Exp}_{\mathscr{A}}^{\mathsf{CPA-M-I}}(\lambda, N)$. Specifically, $\mathscr{A}$ wins the game if the experiment returns 1, indicating that the adversary $\mathscr{A}$ can distinguish $\mathbf{A}_{[0]}$ from $\mathbf{A}_{[1]}$. When $\mathbf{A}_{[b]}$ is given, where $b \in \{0,1\}$ is a random bit, the computing nodes output the production result shares $\mathbf{C}_{[b]}^{(2)}, \cdots, \mathbf{C}_{[b]}^{(N)}$. Here, the generation procedure of $\mathbf{A}_{[b]}$ is driven by a ϵ-forward secure PRNG. We now construct a detailed version of the security game.

Game $\mathbf{Game_1}$. Define $\mathbf{Game_1}$ in the pseudocode below. Let T_1 be the event that $\hat{b} = b$ in the game $\mathbf{Game_1}$. It can be easily seen that the probability of distinguishing $\mathbf{A}_{[0]}$ from $\mathbf{A}_{[1]}$ is equal to $\Pr[T_1]$.

Game $\mathbf{Game_2}$. Define $\mathbf{Game_2}$ to be the same as $\mathbf{Game_1}$ except that the sequence $\mathbf{r}^{(1)} = \left\{r_1^{(1)}, \cdots, r_t^{(1)}\right\}$ is replaced by a truly random one. Let T_2 be the event that $\hat{b} = b$ in the game $\mathbf{Game_2}$. Due to the employment of the truly random sequence, the adversary's output $\hat{b}$ is independent of the hidden bit b. Hence, the adversary $\mathscr{A}$ can correctly guess the bit b with probability $1/2$, i.e.,

$$\Pr[T_2] = \frac{1}{2} \tag{24}$$

Experiment $\mathbf{Exp}_{\mathscr{A}}^{\mathsf{CPA-M-I}}(\lambda, N)$

$$\{\mathbf{A}_{[0]}, \mathbf{A}_{[1]}\} \leftarrow \mathscr{A}\left(\left\{\overline{\mathbf{A}}_j, \overline{\mathbf{C}}_j, \overline{\mathbf{G}}_j, \mathbf{E}_j, \mathbf{F}_j\right\}_{j=1}^{q}\right)$$

$$b \xleftarrow{R} \{0,1\}, \quad K^o, K^u \leftarrow \mathsf{Setup}\left(1^{\lambda}\right), \mathbf{B}_{[b]} \xleftarrow{R} \mathbb{Z}_p^{n\times v} \left\{\mathbf{P}_{[b]}^{(i)}\right\}_{i=1}^{N} \xleftarrow{R} \mathbb{Z}_p^{u\times n},$$

$$\left\{\mathbf{Q}_{[b]}^{(i)}\right\}_{i=1}^{N} \xleftarrow{R} \mathbb{Z}_p^{n\times v}, \left\{\mathbf{O}_{[b]}^{(i)}\right\}_{i=1}^{N} \xleftarrow{R} \mathbb{Z}_p^{u\times v}$$

$$\left\{\mathbf{B}_{[b]}^{(1)}, \cdots, \mathbf{B}_{[b]}^{(N)}\right\} \leftarrow \mathsf{GenShares}\left(K^u, \mathbf{B}_{[b]}, N\right)$$

$$\mathbf{r}_{[b]} \triangleq \{r_1, \cdots, r_t\} \leftarrow G_{fs}^t(K^o), \quad \left\{\mathbf{A}_{[b]}^{(1)}, \cdots, \mathbf{A}_{[b]}^{(N)}\right\} \leftarrow \mathsf{Alg}\left(\mathbf{A}_{[b]}, \mathbf{r}_{[b]}, N\right)$$

$$\left\{\mathbf{E}_{[b]}^{(i)}, \mathbf{F}_{[b]}^{(i)}\right\}_{i=1}^{N} \leftarrow \mathsf{Mask}\left(\left\{\mathbf{A}_{[b]}^{(i)}, \mathbf{B}_{[b]}^{(i)}, \mathbf{P}_{[b]}^{(i)}, \mathbf{Q}_{[b]}^{(i)}\right\}_{i=1}^{N}\right)$$

$$\left(\mathbf{E}_{[b]}, \mathbf{F}_{[b]}, \left\{\mathbf{G}_{[b]}^{(i)}\right\}_{i=1}^{N}\right) \leftarrow \mathsf{Split}\left(\left\{\mathbf{E}_{[b]}^{(i)}, \mathbf{F}_{[b]}^{(i)}\right\}_{i=1}^{N}\right)$$

$$\left\{\mathbf{C}_{[b]}^{(i)}\right\}_{i=1}^{N} \leftarrow \mathsf{Cal}\left(\mathbf{E}_{[b]}, \mathbf{F}_{[b]}, \left\{\mathbf{G}_{[b]}^{(i)}, \mathbf{P}_{[b]}^{(i)}, \mathbf{Q}_{[b]}^{(i)}, \mathbf{O}_{[b]}^{(i)}\right\}_{i=1}^{N}\right),$$

$$\overline{\mathbf{C}}_{[b]} \triangleq \left\{\mathbf{C}_{[b]}^{(i)}\right\}_{i=1}^{N} \hat{b} \leftarrow \mathscr{A}\left(\left\{\overline{\mathbf{A}}_j, \overline{\mathbf{C}}_j, \overline{\mathbf{G}}_j, \mathbf{E}_j, \mathbf{F}_j\right\}_{j=1}^{q}, \overline{\mathbf{A}}_{[b]}, \overline{\mathbf{C}}_{[b]}, \overline{\mathbf{G}}_{[b]}, \mathbf{E}_{[b]}, \mathbf{F}_{[b]}\right)$$

if $\hat{b} = b$ return 1, else return 0

Here, Alg denotes the generating algorithm in [8], and G_{fs}^t represents a ϵ-forward secure PRNG [1].

Then, we can infer that $|\Pr[T_1] - \Pr[T_2]|$ is equivalent to the adversary's advantage for ϵ-forward secure PRNG, i.e.,

$$|\Pr[T_1] - \Pr[T_2]| = \left|\Pr\left[\mathscr{D}\left(G_{fs}^t(K^o)\right) = 1\right] - \Pr\left[\mathscr{D}(r_1', \cdots, r_t') = 1\right]\right| \leqslant \epsilon \tag{25}$$

where the last inequality holds from the definition of ϵ-forward secure PRNG [8]. Then, combining (24) and (25), we have

$$|\Pr[T_1] - 1/2| \leqslant \epsilon \tag{26}$$

where ϵ is considered as negligible. It implies that the probability of distinguishing $\mathbf{A}_{[0]}$ from $\mathbf{A}_{[1]}$ is arbitrarily close to $1/2$. Alternatively, the adversary's advantage can be formulated as

$$\mathbf{Adv}_{\mathscr{A}}^{\mathsf{CPA-M-I}}(\lambda, N, q) = |\Pr[T_1] - 1/2| \leqslant \epsilon \tag{27}$$

Therefore, we conclude that the adversary's advantage is negligible in wining the game $\mathbf{Game_0}$. This completes the proof. □

Theorem 2. *The proposed secure matrix multiplication protocol holds indistinguishability with honest-but-curious auxiliary node $\mathscr{E}_0$ in the proxy layer according to the Definition 2.*

Proof. The proof can be also based on the analysis of the adversary's advantage in the security experiment $\mathbf{Exp}_{\mathscr{A}}^{\mathrm{CPA-M-II}}(\lambda, N)$. Similar to the proof of Theorem 1, we can also define security games to analyze the adversary's advantage. For simplicity, we only provide a proof sketch. In the secure matrix multiplication protocol, the auxiliary node $\mathscr{E}_0$ only has the knowledge of $\mathbf{E}, \mathbf{F}$ (masked version of input matrix $\mathbf{A}, \mathbf{B}$), their production matrix $\mathbf{G}$ and the shares $\left\{\mathbf{G}^{(1)}, \cdots, \mathbf{G}^{(N)}\right\}$. As the input matrix is masked by two random matrices $\mathbf{P}$ and $\mathbf{Q}$, which are generated by a ϵ-forward secure PRNG, the auxiliary node $\mathscr{E}_0$ cannot distinguish the input matrix from a random matrix. Therefore, secure matrix multiplication protocol holds the indistinguishability against the honest-but-curious auxiliary node $\mathscr{E}_0$. This completes the proof. □

Theorem 3. *The proposed secure matrix multiplication protocol can protect both the matrix $\mathbf{A}$ and $\mathbf{B}$ (the input and the model parameters) against honest-but-curious auxiliary node $\mathscr{E}_0$ and computing nodes $\{\mathscr{E}_1, \cdots, \mathscr{E}_N\}$ in the proxy layer under the no-collusion situation.*

Proof. Since secure matrix multiplication protocol performs the same operation with the matrix $\mathbf{A}$ and $\mathbf{B}$, we can similarly prove that secure matrix multiplication protocol can protect the privacy of the matrix $\mathbf{B}$ against the honest-but-curious the auxiliary node $\mathscr{E}_0$ and computing nodes $\{\mathscr{E}_1, \cdots, \mathscr{E}_N\}$ according to Theorem 1 and Theorem 2. According to the above analysis, it can be inferred that our proposed sub-protocols can protect the input privacy against the proxy layer under the no-collusion situation, implying that the input data and model parameters are both protected. Since our proposed framework is essentially composed of a series of secure sub-protocols, it implies that our proposed framework can also preserve the privacy of input and model parameters against the proxy layer under the no-collusion situation. □

At last, we consider the challenging case, when the collusion exists in the proxy layer. It implies that some computing nodes collude with the auxiliary node to obtain the input or model parameters.

Theorem 4. *The proposed PVDLI framework protects the privacy of the model parameters and the input against the honest-but-curious proxy layer under collusion situation if and only if there exists at least one trustful node among the computing nodes $\{\mathscr{E}_1, \cdots, \mathscr{E}_N\}$.*

Proof. Firstly, we analyze the case when the number of the trustful computing node $|\overline{\mathscr{E}}| \geqslant 1$, and then consider the extreme case when $|\overline{\mathscr{E}}| = 1$. Without loss of generality, we assume that the trustful node is $\mathscr{E}_1$, and the rest of participants $\{\mathscr{E}_2, \cdots, \mathscr{E}_N\}$ collude with the auxiliary node $\mathscr{E}_0$.

The adversary has the knowledge of input shares $\left\{\mathbf{X}^{(2)}, \cdots, \mathbf{X}^{(N)}\right\}$, model parameter shares $\left\{\mathbf{\Theta}^{(2)}, \cdots, \mathbf{\Theta}^{(N)}\right\}$ and the intermediate values $\left\{\mathbf{G}^{(2)}, \cdots, \mathbf{G}^{(N)}\right\}, \mathbf{E}$ and $\mathbf{F}$. Since the protection strategy of input is the same as that of model parameters, we consider the privacy of input as an example. The adversary tries to aggregate the known input shares to recover the input $\mathbf{X}$, expressed as

$$\overline{\mathbf{X}} = \sum_{i=2}^{N} \mathbf{X}^{(i)} \tag{28}$$

According to the generating algorithm in [8], it can be inferred that the difference between the input $\mathbf{X}$ and the recovered input $\overline{\mathbf{X}}$ is the share $\mathbf{X}^{(1)}$. Since the share is split by a ϵ-forward secure PRNG, $\mathbf{X}^{(1)}$ is random drawn from $\mathbb{Z}_p^{n\times(J+1)}$. It implies that the adversary cannot distinguish it from the truely random values when $|\overline{\mathscr{E}}| = 1$.

Now we analyze the situation when the number of trustful computing node $|\overline{\mathscr{E}}| < 1$. It implies that all the computing nodes collude with the auxiliary node $\mathscr{E}_0$. Assume that each computing node has the probability p of colluding with the other nodes, then the collusion probability can be easily computed as p^N. The adversary has the knowledge of input shares $\left\{\mathbf{X}^{(1)}, \cdots, \mathbf{X}^{(N)}\right\}$, model parameter shares $\mathbf{\Theta}^{(1)}, \cdots, \mathbf{\Theta}^{(N)}$ and the intermediate values $\mathbf{E}, \mathbf{F}$ and $\left\{\mathbf{G}^{(1)}, \cdots, \mathbf{G}^{(N)}\right\}$. Similar to the analyses above, we also consider the privacy of input as an example. The input $\mathbf{X}$ can be easily recovered by

$$\overline{\mathbf{X}} = \sum_{i=1}^{N} \mathbf{X}^{(i)} \tag{29}$$

It implies that, in this case, the input and the model parameters are leaked. This completes the proof. □

6.3. Verifiability analysis

.

Theorem 5. *The user $\mathscr{U}$ can verify the correctness of the inference result with the probability of $\frac{J}{J+1}$, where J denotes the number of verifiable data.*

Proof. As described in Section 5.6, the proxy layer returns $J+1$ results. Our framework checks the correctness of J verifiable data to verify the inference result $\mathbf{y}$ of original input $\mathbf{x}$. If the inference results of J verifiable data are correct, then the user $\mathscr{U}$ accepts the inference result $\mathbf{y}$. Now we analyze the probability that the proxy layer returns a false result but the user $\mathscr{U}$ still accepts it. It implies that the proxy layer must perform correct inference of J verifiable data and return a false result of the original input $\mathbf{x}$. Since the indistinguishability between the input and random values, the proxy layer cannot distinguish the original input and verifiable data. Let T_v denote this event, then the probability can be calculated as

$$Pr[T_v] = \frac{1}{J+1} \tag{30}$$

Hence, we can demonstrate that the event T_s that the user verifies the correctness of the inference result $\mathbf{y}$ successfully with the probability:

$$Pr[T_s] = 1 - Pr[T_v] = \frac{J}{J+1} \tag{31}$$

This completes the proof. □

7. Experimental results

In this section, we conduct the experiments to evaluate the performance of our proposed PVDLI framework. We implement our framework on Ubuntu 18.04 operation system by using Pytorch framework. All the following experiments are performed on a computer with a Intel Core i7 CPU, a GTX 2080 Super GPU and 32 GB RAM.

Table 1
The accuracy comparison between our proposed framework and traditional insecure counterpart.

	Traditional	Proposed
MLP-MNIST	99.01%	98.14%
VGG16-CIFAR10	91.40%	89.35%

Table 2
The results of adversary guessing the random choice b.

Size		Repeat	Correct	Wrong	Advantage
u	n	Q	$\hat{b}=b$	$\hat{b}\neq b$	$\mathbf{Adv}_{\mathscr{A}}^{\mathrm{CPA-M-I}}(\lambda,N,q)$
10	8	200	99	101	0.005
10	8	500	251	249	0.002
50	40	200	100	100	0
50	40	500	248	252	0.004
100	80	200	101	99	0.005
100	80	500	249	251	0.002

We evaluate our proposed framework on two widely-used datasets: MNIST [20] and CIFAR-10 [17]. MNIST dataset[1] consists of 70000 handwritten digits images of size 28×28, among which 60000 images are used for training and the remaining 10000 for testing. CIFAR-10 dataset[2] contains 60000 color images of size 32×32 for 10 classes. There are 50000 training images and 10000 test images. To evaluate on different models, we adopt multi-layer perceptron (MLP) for MNIST dataset and a typical convolutional neural network VGG16 [38] for CIFAR-10. Here, average pooling and batch normalization are employed in the settings of VGG16.

Firstly, we evaluate the performance of our framework on two datasets, and compare it with the traditional deep learning inference scheme, where the server possesses the model parameters and user's input. In this scenario, the server performs the inference and outputs a result to the user, where the privacy and verifiability are compromised. As can be seen from Table 1, compared with traditional insecure scheme, our proposed PVDLI framework sacrifices 1%-2% accuracy. This accuracy degradation mainly comes from the approximation of the activation function. In real applications, to achieve privacy and verifiability, this accuracy degradation can be considered as acceptable.

Secondly, to validate the security analysis above, we conduct an experiment to evaluate the advantage of the adversary derived in Section 6.1. As the capabilities of the adversary are difficult to be measured, many works [45,14] resorted to analyze the advantage of adversary theoretically. Here, we simulate the adversary with a deep neural network to execute the security experiment $\mathbf{Exp}_{\mathscr{A}}^{\mathrm{CPA-M-I}}(\lambda,N)$. Specifically, we perform the secure matrix multiplication SecMul q times. As described in Section 6.1, the adversary has the knowledge of the input matrix choices $\mathbf{A}_{[0],j},\mathbf{A}_{[1],j}$, the input matrix shares $\overline{\mathbf{A}}_j$, output matrix shares $\overline{\mathbf{C}}_j$, and intermediate results $\mathbf{E}_j,\mathbf{F}_j,\overline{\mathbf{G}}_j$, where j is the index of query. The target of adversary is to guess/predict the random choice b (0 or 1). Here, we utilize q times queries as training data and q ground truth choices $\{b_j\}_{j=1}^{q}$ as labels to train a deep neural network. Then we utilize the well-trained network as the adversary to repeat $\mathbf{Exp}_{\mathscr{A}}^{\mathrm{CPA-M-I}}(\lambda,N)$ Q times. Here, we set the number of the computing nodes $N=5$, and query times $q=500$. We vary the size (u,v) of the input matrix $\mathbf{A}$. The value of Q is in the range of $\{200,500\}$. As can be seen from Table 2, the probability that the adversary can correctly determine the value b is very close to $1/2$. Alternatively, the advantage of the adversary in wining the security game is in the range of $[0,0.005]$, which can be considered as negligible in practice. This validates our security analysis in Section 6.2.

Then, to demonstrate the privacy of our proposed PVDLI, we also investigate the visual comparison of the mixed data (input and verifiable data) together with their encrypted versions. To assist the illustration, we employ the images from CIFAR-10 dataset. As described in Section 5.3, the input and verifiable data are split into shares and sent to the computing nodes $\{\mathscr{E}_1,\mathscr{E}_2,\cdots,\mathscr{E}_N\}$ in the proxy layer. Here, we set the number of computing nodes $N=5$ and the polynomial degree $d=2$. The original input, verifiable data and the shares received by the computing nodes are shown in Fig. 5. As can be seen, in the view of the computing nodes, both the input and verifiable data are meaningless images, making it impossible to distinguish. In a word, our proposed framework is effective in destroying the semantic meaning of the input images and verifiable data, implying that the PVDLI can well preserve the privacy of the input. Additionally, due to the well-preserved privacy, the verifiability can be also provided.

Thirdly, to prove the verifiability of our proposed PVDLI, we also conduct the verifiable experiments. As described in Section 5.3, we mix J verifiable data with the original input $\mathbf{x}$, where the number of verifiable data J is in the range of $\{4,6,8,10,12\}$. Assume that the proxy layer returns e false inference results and $J+1-e$ correct results to save the computational power, where $1\leqslant e\leqslant J+1$. Since the proxy layer cannot distinguish the input data and verifiable data, the false results are randomly selected and generated. We fix the number of computing nodes $N=5$ and the polynomial degree $d=2$. For each J in the range of $\{4,6,8,10,12\}$, we test 100 times, and record the rate that our proposed framework successfully detects the false inference result. The detection results are shown in Fig. 6. According to Theorem 1, we can easily obtain the detection probability under different choices of J. The experiments also demonstrate the correctness of Theorem 1. As can be seen, when the number of false result $e>1$, our proposed verification method can perfectly detect the false result. When the proxy layer only return 1 false result and correctly perform J inference, the detection rate can reach 90% if the number of verifiable data $J\geqslant 10$.

We would also like to investigate the impact of the polynomial degree on the model approximation accuracy. Here, we utilize different polynomials of degree in the range $\{2,3,4,5,6\}$ to approximate the well trained MLP and VGG16 model. The model accuracies are given in Fig. 7. Since the errors spread and propagate from one layer to the next layer, higher degree polynomials, which also have more coefficients to learn, may cause bad approximation performance. As can be noticed, the accuracy decreases with the increasing of the polynomial degrees.

[1] http://yann.lecun.com/exdb/mnist/
[2] http://www.cs.toronto.edu/ kriz/cifar.html

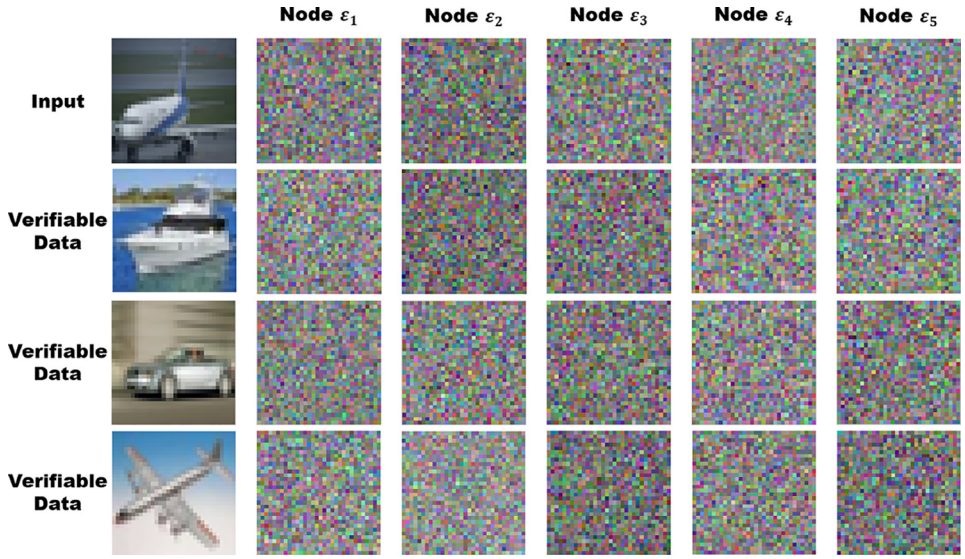

Fig. 5. The input images and their shares located in different computing nodes.

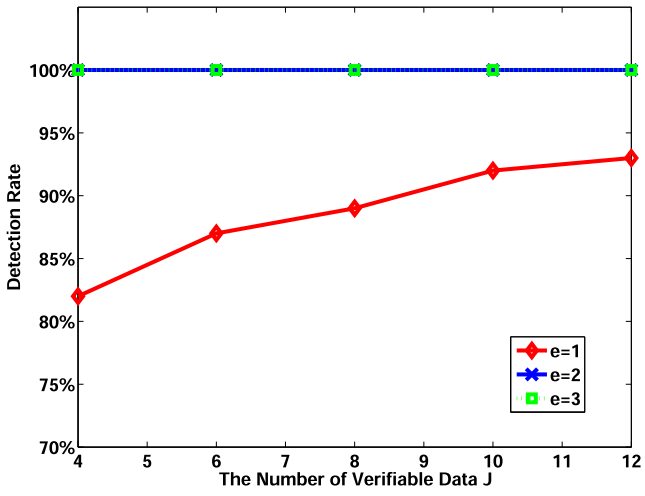

Fig. 6. Detection rate on different number of verifiable data.

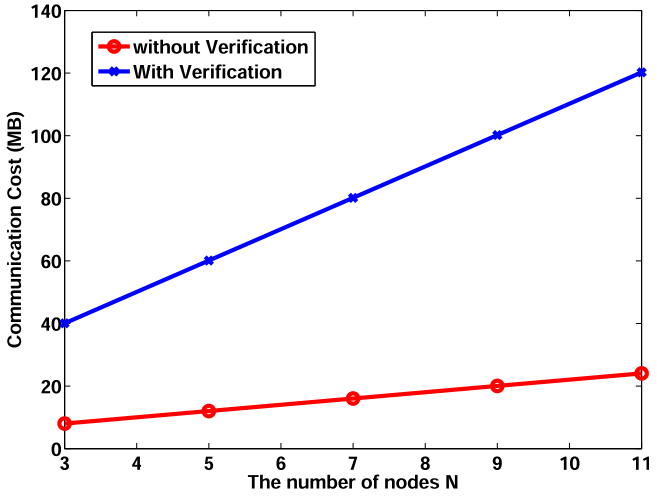

Fig. 8. Communication cost under different number of computing nodes *N*.

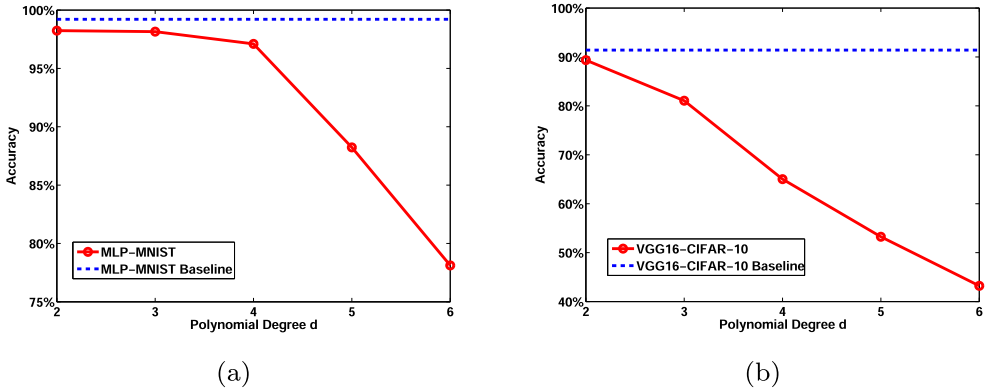

Fig. 7. (a) Accuracy on MNIST dataset with different polynomial degrees. (b) Accuracy on CIFAR-10 dataset with different polynomial degrees.

Table 3
Comparison of communication cost and time cost.

Framework	Communication Cost (MB)	Time Cost (seconds)		
		Deploy	Inference	Total
CryptoNets [12]	372.2	297.5	0.15	297.65
Chameleon [30]	12.9	1.34	1.36	2.7
MiniONN [21]	47.6	0.88	0.4	1.28
DeepSecure [32]	791	1.98	9.67	11.65
Our proposed w/o verification	12.02	2.13	0.081	2.211
Our proposed w/ verification	60.1	2.13	0.412	2.542

Then, to measure the influence of different number of computing nodes on the model communication cost, we employ MLP network on dataset MNIST and vary N from 3 to 11. Here, we consider two scenarios: one is inference with verification; the other is without verification. Regarding the verification scenario, the number of verifiable data J is fixed to 4. It implies that we choose 4 extra images to verify the correctness of the returned result with the confidence of 80%. As can be seen in Fig. 8, the communication cost increases almost linearly with respect to the number of computing nodes N.

Prior to concluding this section, let us give some comparisons with the existing privacy-preserving framework. To be consistent with [30], we utilize MLP model on dataset MNIST for comparison. Here, the number of computing nodes N is fixed to 5. Under the verification scenario, the number of verifiable data $J = 4$. As can be seen from Table 3, our proposed framework takes less communication cost while performing the inference task around 3 s. Although it takes 60.1 MB communication cost with the verification, it provides verifiability for the user to check the correctness with the probability of 80%.

8. Conclusion

We have designed and implemented a PVDLI framework that enables the user can keep both the input and the model parameters private with low communication and computational cost. In this way, the user can also verify the correctness of the inference with high probability. It has been proved theoretically that the input and the model parameters can be protected in a satisfactory manner against the honest-but-curious participants. Finally, experimental results have been demonstrated to show the superior performance of our proposed PVDLI framework.

CRediT authorship contribution statement

Jia Duan: Writing-original-draft, Software, Conceptualization, Methodology. **Jiantao Zhou:** Writing-review-editing, Supervision, Methodology, Formal-analysis. **Yuanman Li:** Validation, Data-curation, Visualization, Investigation. **Caishi Huang:** 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.

Acknowledgment

This work was supported by Macau Science and Technology Development Fund under SKL-IOTSC-2021–2023, 0072/2020/AMJ, 077/2018/A2 and 0060/2019/A1, by Research Committee at University of Macau under MYRG2018-00029-FST and MYRG2019-00023-FST, by Natural Science Foundation of China under 61971476.

References

[1] M. Bellare, B. Yee, Forward-security in private-key cryptography, Proc. RSA Conf. on Cryptographers' Track (2003) 1–18.
[2] Z. Brakerski, C. Gentry, V. Vaikuntanathan, (leveled) fully homomorphic encryption without bootstrapping, ACM Trans. Computat. Theory 6 (3) (2014) 1–36.
[3] H. Chabanne, A. de Wargny, J. Milgram, C. Morel, E. Prouff, Privacy-preserving classification on deep neural network, IACR Cryptology ePrint Archive 2017 (2017) 35–53.
[4] M. Chase, R. Gilad-Bachrach, K. Laine, K.E. Lauter, P. Rindal, Private collaborative neural network learning, IACR Cryptology ePrint Archive 2017 (2017) 762–779.
[5] J. Chi, E. Owusu, X. Yin, T. Yu, W. Chan, P. Tague, Y. Tian. Privacy partitioning: Protecting user data during the deep learning inference phase. arXiv preprint arXiv:181202863 (2018).
[6] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using rnn encoder–decoder for statistical machine translation, Proc. Empirical Methods Natural Language Process. (2014) 1724–1734.
[7] J. Devlin, R. Zbib, Z. Huang, T. Lamar, R. Schwartz, J. Makhoul. Fast and robust neural network joint models for statistical machine translation. In: Proc. Conf. Association for Computational Linguistics (2014) p. 1370–1380..
[8] J. Duan, J. Zhou, Y. Li, Privacy-preserving distributed deep learning based on secret sharing, Inf. Sci. 1 (1) (2020) 1.
[9] T. van Elsloo, G. Patrini, H. Ivey-Law. SEALion: a framework for neural network inference on encrypted data. arXiv preprint arXiv:190412840 (2019)..
[10] A. Esteva, B. Kuprel, R.A. Novoa, J. Ko, S.M. Swetter, H.M. Blau, S. Thrun, Dermatologist-level classification of skin cancer with deep neural networks, Nature 542 (7639) (2017) 115–118.
[11] E. Gamma, Design patterns: elements of reusable object-oriented software, Pearson Education India (1995).
[12] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, J. Wernsing, Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy, Proc. Int. Conf. Machine Learn. (2016) 201–210.
[13] B. Jayaraman, L. Wang, D. Evans, Q. Gu, Distributed learning without distress: Privacy-preserving empirical risk minimization, Proc. Adv. Neural Inf. Process. Syst. (2018) 6346–6357.
[14] B. Jayaraman, L. Wang, K. Knipmeyer, Q. Gu, D. Evans, Revisiting membership inference under realistic assumptions, Proc. Privacy Enhancing Technologies (2020) 348–368.
[15] M. Joye, F.A. Petitcolas. PINFER: Privacy-preserving inference for machine learning. arXiv preprint arXiv:191001865 (2019)..
[16] C. Juvekar, V. Vaikuntanathan, A. Chandrakasan, GAZELLE, A low latency framework for secure neural network inference, in: Proc. USENIX Security Symposium (USENIX Security), 2018, pp. 1651–1669.
[17] A. Krizhevsky, G. Hinton, et al., Learning multiple layers of features from tiny images, Technical Report (2009).
[18] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, Proc. Adv. Neural Inf. Process. Syst. (2012) 1097–1105.
[19] O.-A. Kwabena, Z. Qin, T. Zhuang, Z. Qin, Mscryptonet: Multi-scheme privacy-preserving deep learning in cloud computing, IEEE Access 7 (2019) 29344–29354.
[20] F. Lauer, C.Y. Suen, G. Bloch, A trainable feature extractor for handwritten digit recognition, Pattern Recogn. 40 (6) (2007) 1816–1824.
[21] J. Liu, M. Juuti, Y. Lu, N. Asokan. Oblivious neural network predictions via minionn transformations. In: Proc. ACM Conf. Computer Commun. Security (2017) p. 619–631..
[22] X. Ma, X. Chen, X. Zhang, Non-interactive privacy-preserving neural network prediction, Inf. Sci. 481 (2019) 507–519.
[23] P. Mohassel, Y. Zhang, Secureml: A system for scalable privacy-preserving machine learning, in: Proc. IEEE Symposium on Security and Privacy (SP), 2017, pp. 19–38.
[24] S.A. Osia, A.S. Shamsabadi, S. Sajadmanesh, A. Taheri, K. Katevas, H.R. Rabiee, N. D. Lane, H. Haddadi, A hybrid deep learning architecture for privacy-preserving mobile analytics, IEEE Internet Things J. 7 (5) (2020) 4505–4518.
[25] S.A. Osia, A.S. Shamsabadi, A. Taheri, K. Katevas, H.R. Rabiee, N.D. Lane, H. Haddadi. Privacy-preserving deep inference for rich user data on the cloud. arXiv preprint arXiv:171001727 (2017)..
[26] W. Ouyang, X. Wang, X. Zeng, S. Qiu, P. Luo, Y. Tian, H. Li, S. Yang, Z. Wang, C.-C. Loy, et al. Deepid-net: Deformable deep convolutional neural networks for object detection. In: Proc. IEEE Conf. Computer Vision Pattern Recognit. (2015) p. 2403–2412..
[27] L.T. Phong, Y. Aono, T. Hayashi, L. Wang, S. Moriai, Privacy-preserving deep learning via additively homomorphic encryption, IEEE Trans. Inf. Forensics Secur. 13 (5) (2018) 1333–1345.
[28] L.T. Phong, T.T. Phuong, Privacy-preserving deep learning via weight transmission, IEEE Trans. Inf. Forensics Secur. 14 (11) (2019) 3003–3015.
[29] M.S. Riazi, M. Samragh, H. Chen, K. Laine, K. Lauter, F. Koushanfar, XONN: Xnor-based oblivious deep neural network inference, in: Proc. USENIX Security Symposium (USENIX Security), 2019, pp. 1501–1518.

[30] M.S. Riazi, C. Weinert, O. Tkachenko, E.M. Songhori, T. Schneider, F. Koushanfar. Chameleon: A hybrid secure computation framework for machine learning applications. In: Proc. Asia Conf. on Computer and Commun. Security (2018) p. 707–721..
[31] B.D. Rouhani, S.U. Hussain, K. Lauter, F. Koushanfar, Redcrypt: Real-time privacy-preserving deep learning inference in clouds using fpgas, ACM Trans. Reconfigurable Technol. Syst. (TRETS) 11 (3) (2018) 1–21.
[32] B.D. Rouhani, M.S. Riazi, F. Koushanfar, Deepsecure: Scalable provably-secure deep learning, Proc. Ann. Design Automation Conf. (2018) 1–6.
[33] F. Schroff, D. Kalenichenko, J. Philbin. Facenet: A unified embedding for face recognition and clustering. In: Proc. IEEE Conf. Computer Vision Pattern Recognit. (2015) p. 815–823..
[34] A. Shamir, How to share a secret, Commun. ACM 22 (11) (1979) 612–613.
[35] A.S. Shamsabadi, A. Gascón, H. Haddadi, A. Cavallaro, PrivEdge: From local to distributed private training and prediction, IEEE Trans. Inf. Forensics Secur. 1 (1) (2020) 1.
[36] J. Shen, J. Liu, Y. Chen, H. Li. Towards efficient and secure delivery of data for training and inference with privacy-preserving. arXiv preprint arXiv:180909968 (2018)..
[37] R. Shokri, V. Shmatikov. Privacy-preserving deep learning. In: Proc. ACM Conf. Computer Commun. Security (2015) p. 1310–1321..
[38] K. Simonyan, A. Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:14091556 (2014)..
[39] C. Szegedy, A. Toshev, D. Erhan, Deep neural networks for object detection, Proc. Adv. Neural Inf. Process. Syst. (2013) 2553–2561.
[40] Y. Taigman, M. Yang, M. Ranzato, L. Wolf. Deepface: Closing the gap to human-level performance in face verification. In: Proc. IEEE Conf. Computer Vision Pattern Recognit. (2014) p. 1701–1708..
[41] Y. Tian, J. Yuan, S. Yu, Y. Hou. LEP-CNN: A lightweight edge device assisted privacy-preserving cnn inference solution for iot. arXiv preprint arXiv:190104100 (2019)..
[42] K. Wang, Z. Hu, Q. Ai, Q. Liu, M. Chen, K. Liu, Y. Cong, Membership inference attack with multi-grade service models in edge intelligence, IEEE Network 35 (1) (2021) 184–189.
[43] D. Xu, M. Zheng, L. Jiang, C. Gu, R. Tan, P. Cheng, Lightweight and unobtrusive data obfuscation at iot edge for remote inference, IEEE Internet Things J. 1 (1) (2020) 1.
[44] A.C. Yao, Protocols for secure computations, in: Proc. Ann. Symposium on Foundations of Computer Science, 1982, pp. 160–164.
[45] S. Yeom, I. Giacomelli, M. Fredrikson, S. Jha, Privacy risk in machine learning: Analyzing the connection to overfitting, in: Proc. IEEE 31st Computer Security Foundations Symposium (CSF), 2018, pp. 268–282.
[46] Q. Zhang, C. Wang, H. Wu, C. Xin, T.V. Phuong, GELU-Net: A globally encrypted, locally unencrypted deep neural network for privacy-preserved learning, Proc. Int. Joint Conf. Artificial Intelligence (IJCAI) (2018) 3933–3939.

Jia Duan (Member, IEEE) received the B.S. in software engineering from Chongqing University, Chongqing, China, in 2011, and the Ph.D. degree in computer science from University of Macau, Macau, China, in 2020. He is currently a Researcher with Business Growth BU, JD.COM, Beijing, China. His current research interests include privacy-preserving deep learning, federated learning and cloud computing.

Jiantao Zhou (Senior Member, IEEE) received the B.Eng. degree from the Department of Electronic Engineering, Dalian University of Technology, in 2002, the M.Phil. degree from the Department of Radio Engineering, Southeast University, in 2005, and the Ph.D. degree from the Department of Electronic and Computer Engineering, The Hong Kong University of Science and Technology, in 2009. He held various research positions with the University of Illinois at Urbana-Champaign, The Hong Kong University of Science and Technology, and the McMaster University. He is currently an Associate Professor with the Department of Computer and Information Science, Faculty of Science and Technology, University of Macau. His research interests include multimedia security and forensics, multimedia signal processing, artificial intelligence and big data. He holds four granted U.S. patents and two granted Chinese patents. He has coauthored two articles that received the Best Paper Award at the IEEE Pacific-Rim Conference on Multimedia in 2007 and the Best Student Paper Award at the IEEE International Conference on Multimedia and Expo in 2016. He is an Associate Editor of the IEEE TRANSACTIONS ON IMAGE PROCESSING.

Yuanman Li (Member, IEEE) received the B.Eng. degree in software engineering from Chongqing University, Chongqing, China, in 2012, and the Ph.D. degree in computer science from University of Macau, Macau, 2018. From 2018 to 2019, he was a Post-doctoral Fellow with the State Key Laboratory of Internet of Things for Smart City, University of Macau. He is currently an Assistant Professor with the College of Electronics and Information Engineering, Shenzhen University, Shenzhen, China. His current research interests include data representation, multimedia security and forensics, computer vision and machine learning.

Caishi Huang (Member, IEEE) received the B.S. degree in information engineering from Beijing University of Posts and Telecommunications, Beijing, China, in 2005, and the M.Phil. and Ph.D. degree in Electronic and Computer Engineering from Hong Kong University of Science and Technology, in 2007 and 2012 respectively. He is the Associate Master in Cheong Kun Lun College and a member in Faculty of Science and Technology, University of Macau. His general research interests include distributed networks, cellular and computer networks, multimedia, artificial intelligence and big data.