Efficient Privacy-Preserving Inference Outsourcing for Convolutional Neural Networks

Xuanang Yang, Jing Chen, *Senior Member, IEEE*, Kun He, *Associate Member, IEEE*, Hao Bai, Cong Wu, and Ruiying Du

***Abstract*— Inference outsourcing enables model owners to deploy their machine learning models on cloud servers to serve users. In this paradigm, the privacy of model owners and users should be considered. Existing solutions focus on Convolutional Neural Networks (CNNs) but their efficiency is much lower than GALA, which is a solution that only protects user privacy. Furthermore, these solutions adopt approximations that reduce the model accuracy and thus require model owners to retrain the models. In this paper, we present an efficient CNN inference outsourcing solution that protects the privacy of both model owners and users. Specifically, we design secure two-party computation protocols based on two non-colluding cloud servers, which calculate with additive secret shares of the model and the user's input. Our protocols avoid the expensive permutation operations in linear calculations and approximations in non-linear calculations. We implement our solution on realistic CNNs and experimental results show that our solution is even 2–4 times faster than GALA.**

***Index Terms*— Privacy-preserving inference, convolutional neural networks.**

I. Introduction

OUTSOURCED Prediction as a Service (PaaS) has received widespread attention. In this paradigm, model owners provide services to users by hosting their machine learning models on cloud servers to avoid the cost of maintaining online services [1]. Many cloud service providers, such as Amazon [2], Alibaba [3], and Tencent [4], already offer outsourced PaaS solutions. For instance, by deploying the model on the Amazon cloud server, Sygic provides street view maps to more than 200 million users [5].

Since users' input and prediction results may involve sensitive information, they should not be disclosed to cloud servers or model owners [6], [7]. For example, physiological images and diagnosis results in disease prediction services are sensitive personal data according to the General Data Protection Regulation (GDPR) [8]. To this end, researchers propose privacy-preserving inference based on cryptographic tools to alleviate users' privacy concerns [9], [10]. Most solutions focus on *Convolutional Neural Networks* (CNNs), which have demonstrated their exceptional performance in computer vision [11], [12]. One type of solution [13], [14] needs to modify the CNN model, which leads to a decline in the model accuracy and an extra retraining process to reduce the decline. Among existing solutions that do not modify the model, GALA [15] is the most efficient one.

Unfortunately, most privacy-preserving inference solutions do not consider model privacy and thus are not suitable for the outsourced PaaS paradigm. Specifically, a CNN model is the property of the model owner who spends significant resources to obtain the dataset and train the model. Therefore, the model owner hopes to hide the model parameters from cloud servers and users to keep commercial interests [16], [17]. Some solutions consider both users' and model owners' privacy [18], [19], [20]. However, these solutions are less efficient than GALA. Moreover, they require the model owner to modify the model and retrain it before deploying.

A. Technical Overview

In this paper, we present an efficient privacy-preserving inference solution for the outsourced PaaS paradigm, called Pio. From a technical perspective, our solution consists of two parts. First, we securely deploy a CNN model on the semi-honest server(s). Second, we securely and efficiently calculate with secret inputs in our deployment.

To securely deploy a CNN model, researchers propose two technical routes: *Homomorphic Encryption* (HE) [18], [19] and *Secret Sharing* (SS) [20]. We can deploy an encrypted model on a single cloud server using HE. However, the decryption of users' prediction results needs the participation of the model owner, which places much computational burden on the model owner and deviates from the original intention of outsourcing. Therefore, we employ SS to deploy the model on two non-colluding cloud servers. The non-colluding assumption can be achieved by renting cloud servers from different providers [16], [28]. In addition, the operations in SS are more computationally efficient than those in HE, which makes our solution suitable for resource-constrained clients such as smartphones and Internet-of-Things devices.

Manuscript received 2 November 2022; revised 1 May 2023; accepted 4 June 2023. Date of publication 16 June 2023; date of current version 9 August 2023. This work was supported in part by the National Key Research and Development Program of China under Grant 2022YFB3102100, in part by the Fundamental Research Funds for the Central Universities under Grant 2042022kf1195 and Grant 2042022kf0046, in part by the National Natural Science Foundation of China under Grant 62076187 and Grant 62172303, in part by the Key Research and Development Program of Hubei Province under Grant 2022BAA039, and in part by the Key Research and Development Program of Shandong Province under Grant 2022CXPT055. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Debdeep Mukhopadhyay. *(Corresponding author: Jing Chen.)*

The authors are with the Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education, School of Cyber Science and Engineering, Wuhan University, Wuhan 430072, China (e-mail: whuyxa@whu.edu.cn; chenjing@whu.edu.cn; hekun@whu.edu.cn; bh0036@whu.edu.cn; cnacwu@whu.edu.cn; duraying@whu.edu.cn).

Digital Object Identifier 10.1109/TIFS.2023.3287072

TABLE I

A High-Level Comparison of Privacy-Preserving Inference Solutions. "Model Privacy" Means That the Solution Protects the Model Parameters; "Native Model" Means That the Solution Does Not Modify the CNN Model; "Lightweight Client" Means That the User Does Not Perform Expensive HE Operations; "Linear" and "Non-Linear" Mean the Techniques Utilized to Implement Privacy-Preserving Linear and Non-Linear Calculations Respectively. The Involved Cryptographic Techniques Include Homomorphic Encryption (HE), Ciphertext Packing Technique (CPT), Secret Sharing (SS), Beaver's Multiplication Triplet (BMT), Garbled Circuits (GC), and Oblivious Transfer (OT)

Solution	Model Privacy	Native Model	Lightweight Client	Linear	Non-linear
CryptoNets [21]	✗	✗	✓	HE,CPT	Approximation
MiniONN [22]	✗	✓	✗	SS, BMT	GC
DeepSecure [23]	✗	✓	✓	GC	GC
Gazelle [24]	✗	✓	✗	SS, HE, CPT	GC
E2DM [18]	✓	✗	✓	HE, CPT	Approximation
LoLa [13]	✗	✗	✓	HE, CPT	Approximation
CDKS [19]	✓	✗	✓	HE, CPT	Approximation
XONN [25]	✗	✗	✓	GC, OT	GC
OPPOCNNP [20]	✓	✗	✓	SS, BMT, GC	Approximation, GC
Delphi [26]	✗	✗	✗	SS, HE, CPT	Approximation, GC
CrypTFlow2 [27]	✗	✓	✗	SS, HE, CPT, OT	OT
GALA [15]	✗	✓	✗	SS, HE, CPT	GC
Pio	**✓**	**✓**	**✓**	SS, HE, CPT	GC

To implement secure and efficient calculations in our deployment, we elaborate secure two-party computation protocols to enable two cloud servers to complete CNN inference with the secret share of the model and user's input. Specifically, the input of each layer is secretly shared between the two servers who then secretly share the output of that layer.

The calculations of CNN inference can be categorized into *non-linear* such as Rectified Linear Unit (ReLU) and max-pooling, and *linear* such as matrix-vector product and convolution. Some solutions [18], [19], [20] adopt linear approximation for non-linear calculations to improve their efficiency. For example, they use mean-pooling instead of max-pooling. As a result, those solutions change the original characteristic of the model and thus reduce the model accuracy. To retain the original characteristic, we employ *Garbled Circuits* (GC) to implement privacy-preserving non-linear calculations.

Although those solutions use linear approximations, they are still less efficient than GALA [15]. That is because linear calculations account for the dominant computational overhead. As shown in GALA, linear calculations take up 85%–95.1% of the running time on state-of-the-art CNN models such as AlexNet, VGG, and ResNet. In GALA, privacy-preserving linear calculations are implemented by three basic operations: *homomorphic addition*, *homomorphic multiplication*, and *permutation*. GALA proves that permutation is 34 times slower than multiplication and 56 times slower than addition and dominates the computational cost of linear calculations. To improve the efficiency in privacy-preserving linear calculations, we design protocols that avoid the expensive permutation operation. Moreover, our noise management in linear calculations is better than GALA and thus reduces the probability of failure in the privacy-preserving inference.

B. Contributions

Our main contributions are summarized as follows. A high-level comparison between Pio and related work is shown in Table I.

- We present an efficient privacy-preserving CNN inference solution for the outsourced PaaS paradigm, called Pio. In our solution, both model parameters and users' input and prediction results are protected.
- We propose calculation methods for matrix-vector product and convolution in a homomorphic manner which minimize the number of communication rounds and avoid the expensive permutation operations.
- We evaluate our solution and the experimental results show that our methods for matrix-vector product and convolution are $2.9-3.4$ and $2.9-4$ times faster than the state-of-the-art solutions respectively. On realistic neural networks including AlexNet, VGG, and ResNet (152 layers), Pio is at least 2.3 times faster.

II. Related Work

According to whether model parameters are kept secret from the inference server, we divide privacy-preserving inference solutions based on cryptographic techniques into two categories: *plaintext model* and *non-plaintext model*.

A. Inference With Plaintext Model

In this category, the model is deployed in the inference server in plaintext. Existing solutions protect users' data and prediction results from leaking to the server during inference and can be seen as a secure two-party computation protocol between the server and the client [22]. To improve the efficiency of non-linear calculations, Delphi [26] replaces ReLU with a square function and CrypTFlow2 [27] proposes an OT based secure comparison protocol to calculate ReLU and max-pooling. To improve the efficiency of linear calculations, there is a line of works (including CryptoNets [21], Gazelle [24], LoLa [13], and GALA [15]) that makes efforts to elaborate packing techniques to reduce HE operations. We also focus on the design of packing techniques. Compared to them, Pio avoids the expensive permutation operations, makes better use of the slots of ciphertext, and reduces the number of communication rounds. Moreover, Pio protects model parameters and users' data at the same time and thus can be applied to the outsourced inference scenario. Furthermore, we propose a universal framework that can smoothly extend secure two-party inference solutions with plaintext models (e.g. GALA) to the outsourced scenario (see Section VII-C).

B. Inference With Non-Plaintext Model

In this category, the inference server only knows the models structures. This class of solutions is designed for inference outsourcing, where cloud servers provide inference services on behalf of the model owner without revealing model parameters. Jiang et al. [18] and Chen et al. [19] encrypt CNN models and clients' inputs with HE. Then, the cloud server can conduct inference in the ciphertext form. However, these solutions have three disadvantages. First, the model owner needs to be online to help clients decrypt and obtain prediction results. Second, they replace non-linear calculations with linear ones and thus reduce the accuracy. Specifically, they replace ReLU with a square function in the activation layer and only support mean-pooling in the pooling layer. Third, they are much less efficient than Gazelle since they employ time-consuming bootstrapping in the inference phase to reduce the ciphertext's noise. Pio avoids those disadvantages by designing secure two-party computation protocols between two cloud servers.

III. Preliminaries

We describe an abstraction of Convolutional Neural Networks (CNNs) and review some cryptographic primitives used in our solution.

A. Convolutional Neural Networks

A CNN is composed of a pipeline of layers. The first layer receives inputs to the CNN and is called the input layer. The input of each subsequent layer is the output of the previous layer. The last layer is called the output layer, whose outputs serve as predictions of the CNN. According to the calculation involved, there are the following four types of layers [29].

1) Fully Connected (FC) Layer: This layer takes as input an $n \times 1$ matrix X and outputs an $m \times 1$ matrix Y, which is the product of an $m \times n$ parameter matrix W and the input X. That is, $Y = W \cdot X$.

2) Convolutional Layer: In this layer, the input is an $n \times u_w \times u_h$ matrix X and the output has m channel(s), denoted as $Y = (Y_0, Y_1, \ldots, Y_{m-1})$, where $m \geq 1$. Each channel Y_i is a matrix calculated by the convolution of an $n \times w \times h$ filter matrix F_i and the input X, i.e., $Y_i = F_i \odot X$. In convolution, each element of Y_i is the dot-product of F_i and a subarea of X, where the subarea is slid by stride. We denote the filters in this layer as $F = (F_0, F_1, \ldots, F_{m-1})$.

3) Activation Layer: This layer is usually placed after the convolutional or FC layer to model the non-linear relationship between the input and the output. In this layer, the input matrix and the output matrix have the same dimension. Each element of the output is calculated from the corresponding element of the input through an activation function. Rectified Linear Unit (ReLU) is the most common activation function [30], [31], that outputs the maximum value between 0 and the input.

4) Pooling Layer: This layer is used to reduce data size and manifest main features. In this layer, the input matrix is divided into non-overlapping subareas and the size of subareas is called window size. Each element of the output matrix is the maximum or average value of these subareas, called max-pooling and mean-pooling, respectively. The calculation of max-pooling is non-linear while the calculation of mean-pooling is linear.

B. Cryptographic Primitives

1) Additive Secret Sharing (ASS): In 2-out-of-2 ASS, a pair of secrets $(\langle x\rangle_0, \langle x\rangle_1)$ is generated for a value x and shared between two parties, where $x = \langle x\rangle_0 + \langle x\rangle_1$. ASS guarantees that the party with one secret cannot learn any information about x. In our solution, the model parameters, the input, the prediction, and the intermediate results of the CNN model are shared additively between two non-colluding servers.

2) Garbled Circuits (GC): GC is a kind of generic secure two-party computation protocol, in which a function to be evaluated is converted into a Boolean circuit with input gates for two parties. Our solution employs GC to calculate ReLU and max-pooling since they can be represented by simple circuits and be calculated with small overheads.

3) Homomorphic Encryption (HE): With HE, the ciphertext of $f(x)$ can be calculated from the ciphertext of x without decryption [32], [33]. Since we only need addition and scalar multiplication, Additive Homomorphic Encryption (AHE) [34] is sufficient, which supports calculating $[x + y]$ from $[x]$ and $[y]$, where $[\cdot]$ denotes the ciphertext of the input under the corresponding encryption algorithm.

4) Ciphertext Packing Technique (CPT): This technique allows to pack multiple elements into a single ciphertext and perform component-wise HE operations parallelly in a Single Instruction Multiple Data (SIMD) manner [35]. The ciphertext has N slots called the capacity of ciphertext. In this way, CPT improves the space utilization and computation efficiency of HE. Using CPT, we encrypt a plaintext vector $(x_0, \ldots, x_{n-1})$ into a ciphertext denoted as $[(x_0, \ldots, x_{n-1})]$. Then, we can perform HE operations on this ciphertext and obtain $[f(x_0, \ldots, x_{n-1})]$. Note that noise is introduced into the ciphertext to hide the original message and the noise accumulates with HE operations. The decryption will fail if the noise in the ciphertext grows beyond a certain level.

Our solution utilizes an AHE that leverages CPT [36], called Packed AHE (PAHE), in the FC and convolutional layers.

We briefly review the algorithms in PAHE as follows, where η_i represents the noise in ciphertext C_i.

- `KeyGen`. This key generation algorithm takes as input a security parameter λ and outputs a pair of public-private keys (pk, sk), denoted as $(pk, sk) \leftarrow \texttt{KeyGen}(1^\lambda)$.
- `Enc`. With the input of a public key pk and a plaintext vector $X = (x_0, \ldots, x_{n-1})$, this encryption algorithm outputs a ciphertext $C = [(x_0, \ldots, x_{n-1})]$ and is denoted as $C \leftarrow \texttt{Enc}(pk, X)$.
- `Dec`. With the input of a private key sk and a ciphertext $C = [(x_0, \ldots, x_{n-1})]$, this decryption algorithm outputs the plaintext vector $X = (x_0, \ldots, x_{n-1})$ and is denoted as $X \leftarrow \texttt{Dec}(sk, C)$.
- `CPTAdd`. Given two ciphertexts $C_0 = [(x_0, \ldots, x_{n-1})]$ and $C_1 = [(y_0, \ldots, y_{n-1})]$, this addition operation returns $C_2 = [(x_0 + y_0, \ldots, x_{n-1} + y_{n-1})]$ and is denoted as $C_2 \leftarrow \texttt{CPTAdd}(C_0, C_1)$. The noise is $\eta_2 = \eta_0 + \eta_1$.
- `CPTScMult`. Given a ciphertext vector $C_0 = [(x_0, \ldots, x_{n-1})]$ and a plaintext $V = (v_0, \ldots, v_{n-1})$, this scalar multiplication operation returns $C_3 = [(v_0 \cdot x_0, \ldots, v_{n-1} \cdot x_{n-1})]$ and is denoted as $C_3 \leftarrow \texttt{CPTScMult}(C_0, V)$. The noise is $\eta_3 = \eta_0 \cdot \eta_{mult}$, where η_{mult} is the noise growth of the multiplication operation in the SIMD manner.
- `CPTPerm`. This permutation operation permutes elements in a ciphertext. For example, it can turn $C_0 = [(x_0, x_1, x_2, x_3)]$ into $C_4 = [(x_1, x_2, x_3, x_0)]$. The noise is $\eta_4 = \eta_0 + \eta_{rot}$, where η_{rot} is the noise growth of the permutation operation in the SIMD manner.

The experimental results in GALA show that the computational cost of `CPTPerm` is 56 times of `CPTAdd` and 34 times of `CPTScMult` [15]. In addtion, the noise in the above operations satisfies $\eta_{rot} > \eta_{mult} \gg \eta_i \gg 1$. These facts motivate us to remove `CPTPerm` in the construction of the function f. Therefore, our solution has better efficiency and higher success probability in the privacy-preserving linear calculations than other HE-based solutions, e.g., GALA.

IV. System Model

Our solution is designed for the outsourced PaaS paradigm, as shown in Figure 1. There are the following three types of entities in the system.

- *Model Owners.* A model owner has a CNN model and deploys the model on two cloud servers.
- *Cloud Servers.* Two cloud servers provide prediction services to clients through interactions.
- *Clients.* A client sends its input to the two cloud servers and obtains the prediction results from them.

A. Threat Model

Similar to previous research [22], [26], we consider semi-honest adversaries, where all entities honestly adhere to our protocol with an attempt to learn additional information of other entities. Specifically, the model owner attempts to learn the input and prediction results of clients; the cloud server is curious about the model parameters and the input and prediction results of clients; and the client attempts to obtain the model parameters. In addition, we require that the two cloud servers do not collude and this non-colluding requirement can be achieved by employing servers from two cloud service providers in practice [37], [38].

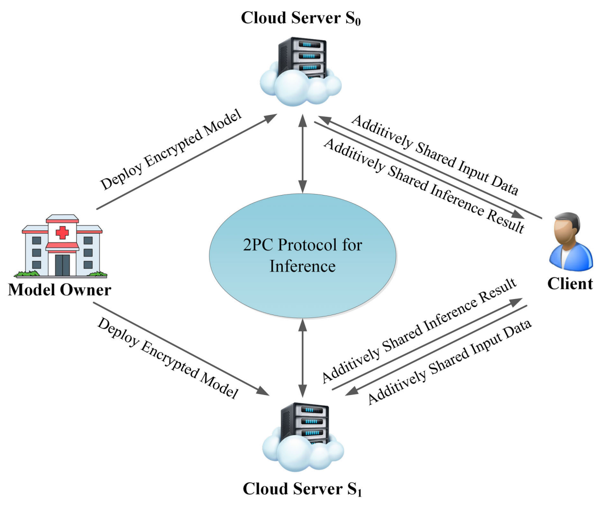

Cloud Server S₀
Deploy Encrypted Model
Additively Shared Input Data
Additively Shared Inference Result
2PC Protocol for Inference
Model Owner
Client
Deploy Encrypted Model
Additively Shared Inference Result
Additively Shared Input Data
Cloud Server S₁

Fig. 1. System model.

B. Security Goals

Our security goals are two-fold. On one hand, the input and prediction results of a client should be kept secret from model owners and cloud servers. On the other hand, the model parameters should be kept secret from clients and cloud servers.

Our security definition follows the standard ideal/real world paradigm, which requires that an adversary's view in the real world is indistinguishable from that in the ideal world. Note that the ideal function does not hide model structures [26], [27], since this can be done with non-cryptographic techniques such as adding fake layers [24]. A major concern for the protection of model parameters comes from model stealing attacks [39], in which an adversary may recover model parameters from predictions. However, this kind of attacks cannot be prevented through cryptographic techniques [15]. Prior privacy-preserving inference solutions do not focus on these attacks.

V. Our Solution

Our privacy-preserving inference outsourcing solution consists of three phases: *model deployment*, *data upload*, and *inference*. In the rest of this paper, we denote the cloud server as S_b, where $b \in \{0, 1\}$. We use $out \leftarrow \texttt{Algo}(in)$ to denote an algorithm `Algo` that takes as input in and outputs out. We use $\{out_0; out_1\} \leftarrow \texttt{Protocol}\{in_0; in_1\}$ to denote a two-party protocol `Protocol` carried out by the two cloud servers, in which S_b possesses in_b as the input and obtains out_b.

A. Model Deployment

In this phase, the model owner deploys his/her CNN model on two non-colluding cloud servers in a privacy-preserving manner. At the beginning of the deployment, S_b generates public-private keys $(pk_b, sk_b) \leftarrow \texttt{KeyGen}(1^\lambda)$ and sends pk_b to S_{b-1}. The CNN model is deployed layer by layer and each layer is deployed according to its type. To carry out correct calculations, the cloud servers know the type of each layer. However, ASS and HE guarantee that parameter matrices and filters of the model are not disclosed to the cloud servers.

1) FC Layer Deployment: To deploy an FC layer whose parameter matrix is W, the model owner chooses a random matrix $\langle W\rangle_0$ and computes $\langle W\rangle_1 \leftarrow W - \langle W\rangle_0$. Then, the model owner sends $\langle W\rangle_b$ to S_b. Finally, S_b computes $[\langle W\rangle_b] \leftarrow \texttt{EncMat}(pk_b, \langle W\rangle_b)$ and sends $[\langle W\rangle_b]$ to S_{1-b}, where $\texttt{EncMat}(\cdot,\cdot)$ is a homomorphic encryption algorithm whose inputs are a public key and a matrix and pk_b is the public key of S_b (see Section VI-A.3).

2) Convolutional Layer Deployment: This process is similar to the FC layer deployment. First, the model owner generates the secret shares $\langle F\rangle_0$ and $\langle F\rangle_1$ that satisfy $F = \langle F\rangle_0 + \langle F\rangle_1$, where F is the filters of this layer. Then, S_b obtains $\langle F\rangle_b$ and sends $[\langle F\rangle_b] \leftarrow \texttt{EncFilter}(pk_b, \langle F\rangle_b)$ to S_{1-b}, where $\texttt{EncFilter}(\cdot,\cdot)$ is the homomorphic encryption algorithm for filters (see Section VI-B.1).

3) Activation Layer Deployment: To deploy an activation layer, the model owner sends the type of activation function (e.g., ReLU) to both servers.

4) Pooling Layer Deployment: To deploy a pooling layer, the model owner sends the type of pooling function (e.g., max-pooling) and the window size to both servers.

B. Data Upload

In this phase, a client uploads his/her data to the cloud servers in a privacy-preserving manner. Specifically, the client, whose data is a matrix X, selects a random matrix $\langle X\rangle_0$ and computes $\langle X\rangle_1 \leftarrow X - \langle X\rangle_0$. Then, the client sends $\langle X\rangle_b$ to S_b. ASS ensures that the cloud servers cannot learn any information about the client's input.

C. Inference

In this phase, the cloud servers feed the input obtained in the data upload phase into the model obtained in the model deployment phase to carry out CNN layer by layer and finally obtain a prediction for the client. Specifically, we devise secure two-party computation protocols for each type of CNN layer, and the cloud servers connect our protocols sequentially to complete CNN inference. Our secure two-party protocols are connectable because they all follow the same paradigm: before running the protocol, the input to the layer is secretly shared between the cloud servers; after running the protocol, the output of the layer is also secretly shared. The input layer's input, which is the client's input data, is secretly shared in the data upload phase, and can just be fed into our protocol. After running our protocols, the servers secretly share the prediction and send the shares to the client who adds them up to obtain the prediction. Our protocols consist of FC layer protocol, convolutional layer protocol, pooling layer protocol, and activation layer protocol, which are detailed in Section VI.

VI. Secure Two-Party Computation Protocols for CNN Inference

We now detail the design of secure two-party computation protocols used in the inference phase of Pio.

A. FC Layer Protocol

Recall that in the FC layer, the cloud servers S_0 and S_1 secretly share the input X and the parameter matrix W, i.e., S_b holds $\langle W\rangle_b$ and $\langle X\rangle_b$, aiming at secretly sharing the output $Y = W \cdot X$. Formally, our secure FC layer computation protocol is denoted as $\{\langle Y\rangle_0; \langle Y\rangle_1\} \leftarrow \texttt{FL}\{\langle W\rangle_0, \langle X\rangle_0; \langle W\rangle_1, \langle X\rangle_1\}$, where $W = \langle W\rangle_0 + \langle W\rangle_1$, $X = \langle X\rangle_0 + \langle X\rangle_1$, and $Y = \langle Y\rangle_0 + \langle Y\rangle_1$.

We can express the computation of FL protocol as follows. $Y = W \cdot X = (\langle W\rangle_0 + \langle W\rangle_1)(\langle X\rangle_0 + \langle X\rangle_1) = \langle W\rangle_0 \cdot \langle X\rangle_0 + \langle W\rangle_0 \cdot \langle X\rangle_1 + \langle W\rangle_1 \cdot \langle X\rangle_0 + \langle W\rangle_1 \cdot \langle X\rangle_1$. Since $\langle W\rangle_0 \cdot \langle X\rangle_0$ and $\langle W\rangle_1 \cdot \langle X\rangle_1$ can be computed by S_0 and S_1 separately, we focus on the calculation method that outputs the additive shares of $\langle W\rangle_b \cdot \langle X\rangle_{1-b}$. Then, S_b outputs $\langle Y\rangle_b$ in the FL protocol as follows.

$$\langle Y\rangle_b = \langle W\rangle_b \cdot \langle X\rangle_b + \langle\langle W\rangle_0 \cdot \langle X\rangle_1\rangle_b + \langle\langle W\rangle_1 \cdot \langle X\rangle_0\rangle_b$$

To better illustrate our calculation method (in Section VI-A.3), we first discuss a naive method in Section VI-A.1 and a state-of-the-art method in Section VI-A.2. In the following of this subsection, we denote an $m \times n$ matrix as W, the i-th row of W as W_i, and the j-th element of W_i as $W_{i,j}$. We also denote the i-th element of vector X as X_i. We use Y to denote the matrix-vector product $W \cdot X$ and Y_i to denote the i-th element of Y.

1) Naive Method: The key idea of the naive method is to mask the product of the plaintext matrix W and the ciphertext vector $[X]$ with a random plaintext vector R, as shown in Figure 3a. Specifically, S_0 first encrypts the input X and sends $[X]$ to S_1. Then, S_1 calculates $[W \cdot X - R]$ and sends it to S_0, where W is a fixed matrix and R is a randomly chosen vector. Finally, S_1 outputs R as $\langle W \cdot X\rangle_1$ and S_0 decrypts $[W \cdot X - R]$ as $\langle W \cdot X\rangle_0$.

Since masking R in the ciphertext form can be done by CPTAdd, we only consider the calculation of matrix-vector multiplication $[W \cdot X]$ with the three operations in PAHE. Recall that S_1 possesses W and $[X]$. It first computes $C_i \leftarrow \texttt{CPTScMult}([X], W_i)$ for each row W_i. Note that $C_i = [(W_{i,0} \cdot X_0, \ldots, W_{i,n-1} \cdot X_{n-1})]$ and our aim is to compute the inner-product $Y_i = \sum_{j=0}^{n-1}(W_{i,j} \cdot X_j)$. To this end, it performs rotate-and-sum calculation ($\log(n)$ times) [24] with CPTPerm and CPTAdd to obtain $C'_i = [(Y_i, \ldots, Y_i)]$. Then, it calculates $C^*_i \leftarrow \texttt{CPTScMult}(C_i, I_i)$, where I_i is the i-th row of an $n \times n$ identity matrix. Finally, it invokes CPTAdd on $(C^*_0, \ldots, C^*_{n-1})$ to obtain $[(Y_0, \ldots, Y_{n-1})]$. Figure 2 depicts an example of multiplying a 4×4 matrix by a vector.

We now analyze the noise in the matrix-vector multiplication. We denote the noise of $[X]$ as η_0. The CPTScMult makes the noise of C_i grow to $\eta_0 \cdot \eta_{mult}$. The $\log(n)$ times rotate-and-sum leads the noise of C^*_i to $n \cdot \eta_0 \cdot \eta_{mult} + (n-1)\eta_{rot}$. The following CPTScMult operation on C^*_i increases the noise to $n \cdot \eta_0 \cdot \eta^2_{mult} + (n-1)\eta_{mult} \cdot \eta_{rot}$.

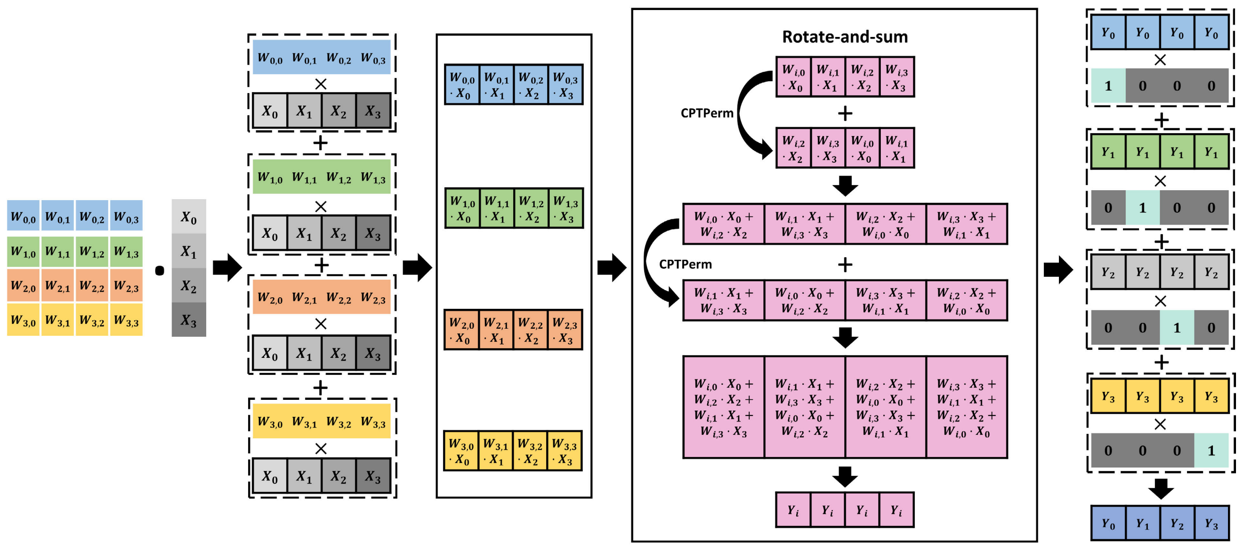


Fig. 2. Vector encryption and matrix-vector multiplication in the naive method, where + and × denote `CPTAdd` and `CPTScMult`.

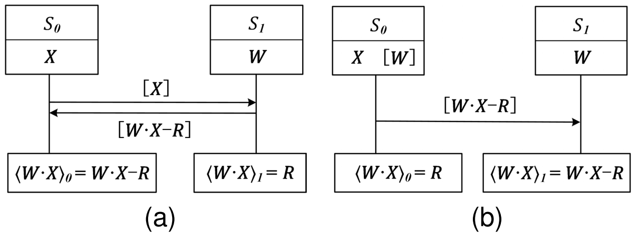

S_0
X
S_1
W
$[X]$
$[W{\cdot}X{-}R]$
$\langle W{\cdot}X\rangle_0 = W{\cdot}X{-}R$
$\langle W{\cdot}X\rangle_1 = R$
(a)
S_0
$X\ \ [W]$
S_1
W
$[W{\cdot}X{-}R]$
$\langle W{\cdot}X\rangle_0 = R$
$\langle W{\cdot}X\rangle_1 = W{\cdot}X{-}R$
(b)

Fig. 3. The flow of calculating the matrix-vector product. (a) The naive method. (b) Our method.

Finally, invoking `CPTAdd` on $(C_0^*, \ldots, C_{n-1}^*)$ makes the noise of the output ciphertext grow to $m \cdot n \cdot \eta_0 \cdot \eta_{mult}^2 + m \cdot (n-1)\eta_{mult} \cdot \eta_{rot}$.

2) Diagonal Method in GALA: GALA proposes the state-of-the-art calculation method which is a variant of diagonal methods. This method is similar with the naive one but adopts a different strategy to calculate $[W \cdot X]$. The key idea of their strategy is to encode the matrix W and the vector X in an ingenious way to reduce HE operations.

S_0 first fills X with as few zeros as possible to make its length n' divide the capacity of ciphertext N. Then, it encodes N/n' copies of X into a vector X^* and outputs $[X^*]$. S_1 first fills W with as few zeros as possible to make its height m' and width n' both divide N. Then, it encodes W into $s = m' \cdot n'/N$ vectors and the j-th vector V_j is as follows.

$$V_j = \begin{pmatrix} W_{j,0}, & \ldots, & W_{j+n-1 \bmod s, n-1} \\ W_{j+s,0}, & \ldots, & W_{j+n-1 \bmod s+s, n-1} \\ & \vdots & \\ W_{j+n-s,0}, & \ldots, & W_{j+n-1 \bmod s+n-s, n-1} \end{pmatrix}$$

To calculate the matrix-vector multiplication $[W \cdot X]$, S_1 computes $C_j \leftarrow$ `CPTScMult`$([X^*], V_j)$ and performs `CPTPerm` on C_j to obtain C_j^* as follows, in which we omit mod n in X's subscripts and the second element of W's subscripts.

$$\begin{pmatrix} W_{0,j} \cdot X_j, & \ldots, & W_{n-1 \bmod s, j+n-1} \cdot X_{j+n-1} \\ W_{s,j} \cdot X_j, & \ldots, & W_{n-1 \bmod s+s, j+n-1} \cdot X_{j+n-1} \\ & \vdots & \\ W_{n-s,j} \cdot X_j, & \ldots, & W_{n-1 \bmod s+n-s, j+n-1} \cdot X_{j+n-1} \end{pmatrix}$$

Finally, S_1 invokes `CPTAdd` on all C_j^* to obtain a ciphertext C as the output. Figure 4 depicts an example of GALA on a 3×4 matrix, where the capacity of ciphertext $N = 8$.

The noise in the matrix-vector multiplication is as follows. We denote the noise of $[X^*]$ as η_0. The `CPTScMult` makes the noise of C_j grow to $\eta_0 \cdot \eta_{mult}$. The `CPTPerm` leads the noise of $C_j^* (j \neq 0)$ to $\eta_0 \cdot \eta_{mult} + \eta_{rot}$. Finally, invoking `CPTAdd` on $(C_0^*, \ldots, C_{s-1}^*)$ increases the noise to $s \cdot \eta_0 \cdot \eta_{mult} + (s-1)\eta_{rot}$.

3) Our Method: Although GALA reduces HE operations significantly, it still involves the expensive `CPTPerm` which dominates its computational cost. Moreover, to calculate the matrix-vector product, the naive method and the diagonal method in GALA both require the two participants to interact for two rounds. This motivates us to design a method that eliminates the expensive `CPTPerm` and minimizes the number of communication rounds. To this end, we elaborate a Homomorphic Matrix Encryption (HME) scheme that supports calculating the matrix-vector product through a ciphertext matrix and a plaintext vector (instead of a plaintext matrix and a ciphertext vector).

In our method, S_0 calculates $[W \cdot X - R]$ and sends it to S_1, where $[W]$ is obtained in the model deployment phase and R is a randomly chosen vector. Then, S_0 outputs R as $\langle W \cdot X \rangle_0$ and S_1 decrypts $[W \cdot X - R]$ as $\langle W \cdot X \rangle_1$. Figure 3b depicts the flow of our method.

Table II summarizes these matrix-vector product calculation methods. Compared with GALA, Pio avoids the most expensive `CPTPerm` operations. This is a significant improvement in efficiency since $\frac{m' \cdot n'}{N}$ expands with the size of the parameter matrix and can be 1024 in VGG. Pio also reduces `CPTAdd`

TABLE II
COMPARISON OF MATRIX-VECTOR PRODUCT CALCULATION METHODS

Method	CPTAdd	CPTScMult	CPTPerm	Rounds	Noise Growth
Naive	$m \cdot log(n) + m - 1$	$2 \cdot m$	$m \cdot log(n)$	2	$m \cdot n \cdot \eta_0 \cdot \eta_{mult}^2 + m \cdot (n-1)\eta_{mult} \cdot \eta_{rot}$
GALA	$\frac{m' \cdot n'}{N} - 1$	$\frac{m' \cdot n'}{N}$	$\frac{m' \cdot n'}{N} - 1$	2	$\frac{m' \cdot n'}{N} \cdot \eta_0 \cdot \eta_{mult} + (\frac{m' \cdot n'}{N} - 1)\eta_{rot}$
Pio	$\frac{n}{\lfloor N/m \rfloor} - 1$	$\frac{n}{\lfloor N/m \rfloor}$	0	1	$\frac{n}{\lfloor N/m \rfloor} \cdot \eta_0 \cdot \eta_{mult}$

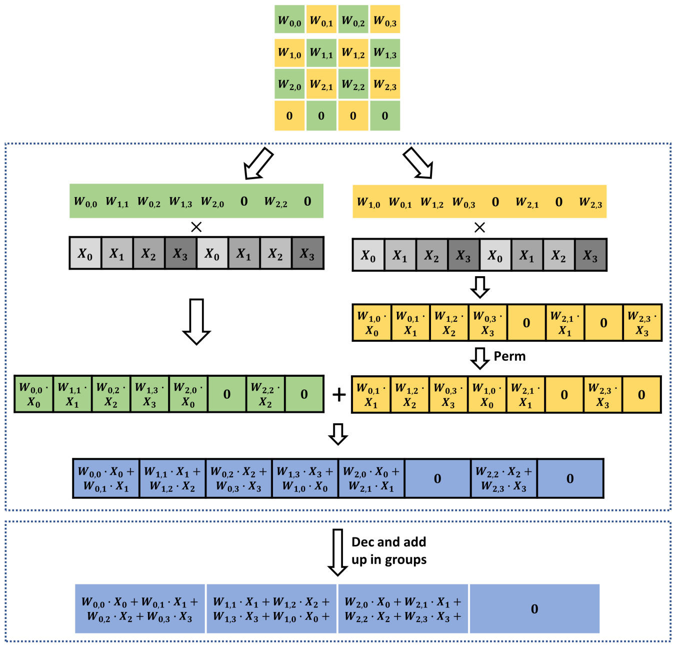


Fig. 4. Vector encryption, decryption, and matrix-vector multiplication in GALA, where $+$ and $\times$ denote CPTAdd and CPTScMult.

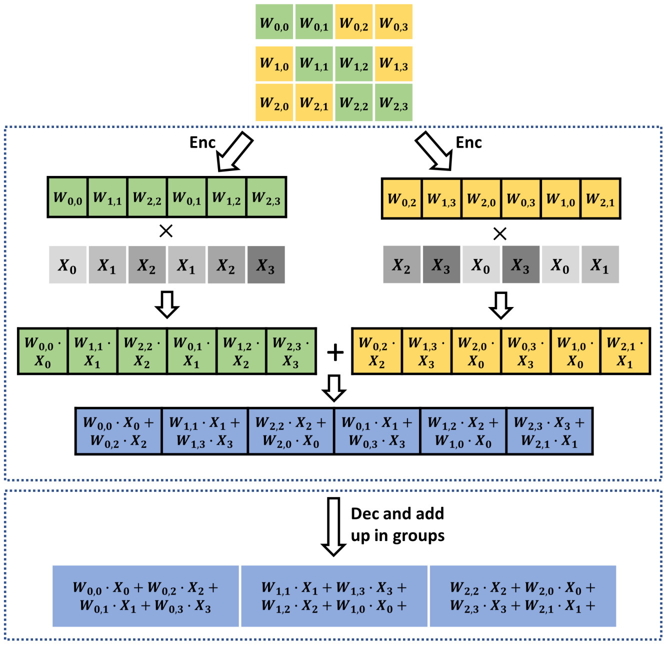


Fig. 5. Matrix encryption, vector decryption, and matrix-vector multiplication in our method, where $+$ and $\times$ denote CPTAdd and CPTScMult.

and CPTScMult operations as $\frac{n}{\lfloor N/m \rfloor}$ is often less than $\frac{m' \cdot n'}{N}$. When multiplying a 128×528 matrix with a vector ($N = 2048$), we have $\frac{n}{\lfloor N/m \rfloor} = 33$ and $\frac{m' \cdot n'}{N} = 64$. Moreover, Pio has much better noise control than GALA, which provides a better guarantee of the success of the FC layers, especially for the neural network with generous neurons.

a) Homomorphic matrix encryption: We first formally introduce HME, which consists of five algorithms: Gen, EncMat, DecVec, Add, and Mul.

- The key generation algorithm Gen takes as input a security parameter λ and outputs a pair of public-private keys (pk, sk).
- The matrix encryption algorithm EncMat takes as input a public key pk and a plaintext matrix W and outputs a ciphertext matrix $[W]$.
- The vector decryption algorithm DecVec takes as input a private key sk and a ciphertext vector $[V]$ and outputs a plaintext vector V.
- The vector addition algorithm Add takes as input a public key pk, a ciphertext vector $[V]$, and a plaintext vector R and outputs a ciphertext vector $[V + R]$.
- The matrix-vector multiplication algorithm Mul takes as input a public key pk, a ciphertext matrix $[W]$, and a plaintext vector X and outputs a ciphertext $[W \cdot X]$.

b) HME construction: We utilize PAHE (recall that PAHE consists of KeyGen, Enc, Dec, CPTAdd, CPTScMult, and CPTPerm) to construct our homomorphic matrix encryption scheme. Figure 5 depicts an example of EncMat, DecVec, and Mul on a 3×4 matrix, where $N = 6$.

Gen(1^λ). The key generation algorithm simply outputs $(pk, sk) \leftarrow$ KeyGen(1^λ).

EncMat(pk, W). Recall that N is the capacity of ciphertext, i.e., the number of slots. Let $t = \lfloor N/m \rfloor$ and $z = n/t$. The matrix encryption algorithm encodes t diagonals into a single vector D_j as follows, in which we omit mod n in the second element of subscripts.

$$D_j = \begin{pmatrix} W_{0,j\cdot t}, & \ldots, & W_{m-1,m+j\cdot t-1}, \\ W_{0,j\cdot t+1}, & \ldots, & W_{m-1,m+j\cdot t}, \\ & \vdots & \\ W_{0,j\cdot t+t-1}, & \ldots, & W_{m-1,m+j\cdot t+t-2} \end{pmatrix}$$

Note that the last $(n \bmod t)\cdot m$ slots of D_{z-1} are padded with zeros. Then, the algorithm computes $[D_j] \leftarrow$ Enc(pk, D_j) for each D_j and outputs $[W] = ([D_0], \ldots, [D_{z-1}])$.

DecVec$(sk, [V])$. The vector decryption algorithm performs $V \leftarrow$ Dec$(sk, [V])$ and outputs a plaintext vector $V^* = (V_0^*, \ldots, V_{m-1}^*)$ where $V_j^* = \sum_{k=0}^{t-1} V_{j+k\cdot m}$.

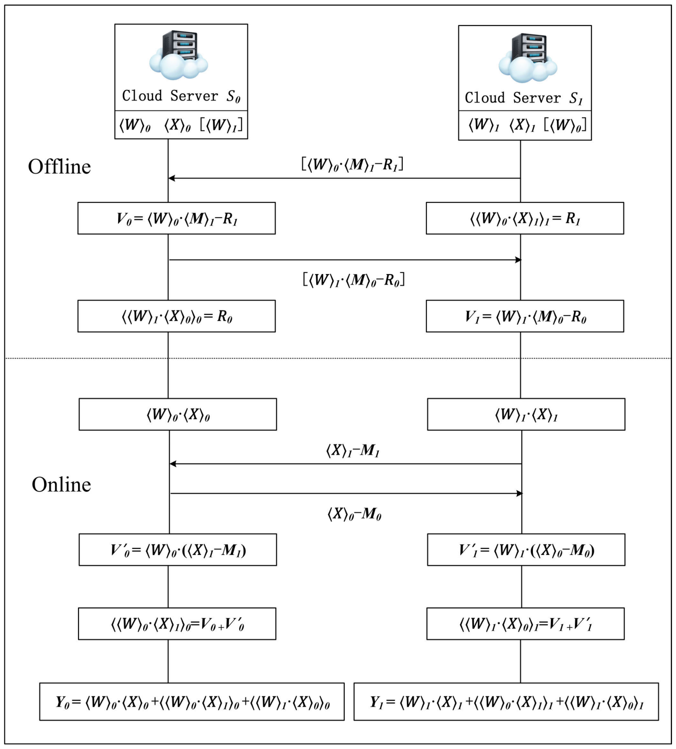

Cloud Server S_0
$\langle W\rangle_0$ $\langle X\rangle_0$ $[\langle W\rangle_1]$
Cloud Server S_1
$\langle W\rangle_1$ $\langle X\rangle_1$ $[\langle W\rangle_0]$
Offline
$[\langle W\rangle_0\cdot\langle M\rangle_1-R_1]$
$V_0=\langle W\rangle_0\cdot\langle M\rangle_1-R_1$
$\langle\langle W\rangle_0\cdot\langle X\rangle_1\rangle_1=R_1$
$[\langle W\rangle_1\cdot\langle M\rangle_0-R_0]$
$\langle\langle W\rangle_1\cdot\langle X\rangle_0\rangle_0=R_0$
$V_1=\langle W\rangle_1\cdot\langle M\rangle_0-R_0$
$\langle W\rangle_0\cdot\langle X\rangle_0$
$\langle W\rangle_1\cdot\langle X\rangle_1$
$\langle X\rangle_1-M_1$
Online
$\langle X\rangle_0-M_0$
$V'_0=\langle W\rangle_0\cdot(\langle X\rangle_1-M_1)$
$V'_1=\langle W\rangle_1\cdot(\langle X\rangle_0-M_0)$
$\langle\langle W\rangle_0\cdot\langle X\rangle_1\rangle_0=V_0+V'_0$
$\langle\langle W\rangle_1\cdot\langle X\rangle_0\rangle_1=V_1+V'_1$
$Y_0=\langle W\rangle_0\cdot\langle X\rangle_0+\langle\langle W\rangle_0\cdot\langle X\rangle_1\rangle_0+\langle\langle W\rangle_1\cdot\langle X\rangle_0\rangle_0$
$Y_1=\langle W\rangle_1\cdot\langle X\rangle_1+\langle\langle W\rangle_0\cdot\langle X\rangle_1\rangle_1+\langle\langle W\rangle_1\cdot\langle X\rangle_0\rangle_1$

Fig. 6. The flow of the FL protocol.

Add$(pk, [V], R)$. The vector addition algorithm calculates $[R] \leftarrow$ Enc(pk, R) and outputs $[V + R] \leftarrow$ CPTAdd$([V], [R])$.

Mul$([W], X)$. The matrix-vector multiplication algorithm first parses $[W]$ as $([D_0], \ldots, [D_{z-1}])$. Then it encodes X into $X^{(j)}$ $(0 \le j \le z)$ as follows, where we also omit mod n.

$$X^{(j)} = \begin{pmatrix} X_{j\cdot t}, & \ldots, & X_{m+j\cdot t-1}, \\ X_{j\cdot t+1}, & \ldots, & X_{m+j\cdot t}, \\ & \vdots & \\ X_{j\cdot t+t-1}, & \ldots, & X_{m+j\cdot t+t-2} \end{pmatrix}$$

For each $[D_j]$, it computes $C_j \leftarrow$ CPTScMult$([D_j], X^{(j)})$. At last, it obtains a ciphertext C of the matrix-vector product by invoking CPTAdd on all C_j.

We now analyze the noise in our method. We denote the noise of $[D_j]$ as η_0. CPTScMult makes the noise of C_j grow to $\eta_0 \cdot \eta_{mult}$. Finally, invoking CPTAdd on $(C_0, \ldots, C_{z-1})$ makes the noise of the output ciphertext grow to $\frac{n}{\lfloor N/m \rfloor} \cdot \eta_0 \cdot \eta_{mult}$.

4) Our Protocol: We now propose the FL protocol (recall that $[W_{1-b}]$ is sent to S_b in the model deployment phase). To reduce the latency of online prediction, we do not use our method directly for calculating $\langle W\rangle_0 \cdot \langle X\rangle_1$ and $\langle W\rangle_1 \cdot \langle X\rangle_0$. Instead, we utilize it in the offline precomputation phase which is independent of user's input.

Specifically, in the offline phase, S_b first selects an $n \times 1$ vector M_b and an $m \times 1$ vector R_b and calculates

$$[V_{1-b}] \leftarrow \texttt{Add}(pk_{1-b}, \texttt{Mul}(pk_{1-b}, [\langle W\rangle_{1-b}], M_b), -R_b).$$

Then, S_b sends $[V_{1-b}]$ to and receives $[V_b]$ from the other cloud server. At the end of the offline phase, S_b decrypts to obtain $V_b \leftarrow$ DecVec$(sk_b, [V_b])$.

In the online phase, S_b sends $\langle X\rangle_b - M_b$ to S_{1-b} and receives $\langle X\rangle_{1-b} - M_{1-b}$. S_b computes $V'_b = \langle W\rangle_b \cdot (\langle X\rangle_{1-b} - M_{1-b})$ and $\langle\langle W\rangle_b \cdot \langle X\rangle_{1-b}\rangle_b = V_b + V'_b$ and sets $\langle\langle W\rangle_{1-b} \cdot \langle X\rangle_b\rangle_b = R_b$. Finally, S_b outputs

$$\langle Y\rangle_b \leftarrow \langle W\rangle_b \cdot \langle X\rangle_b + \langle\langle W\rangle_b \cdot \langle X\rangle_{1-b}\rangle_b + \langle\langle W\rangle_{1-b} \cdot \langle X\rangle_b\rangle_b.$$

It still satisfies $\langle Y\rangle_0 + \langle Y\rangle_1 = Y$. In our protocol, the latency-sensitive online phase only involves addition and multiplication of plaintext.

5) Security: We now prove the security of our FC layer protocol, which can be reduced to the security of the matrix-vector product calculation method proposed in Section VI-A.3. Therefore, we define the ideal functionality for matrix-vector product as $\langle\langle MV\rangle_0; \langle MV\rangle_1\rangle \leftarrow$ IMV$\{W; X\}$, where $\langle MV\rangle_0 + \langle MV\rangle_1 = W \cdot X$.

Theorem 1: Our method securely computes IMV *in the presence of semi-honest adversaries if the underlying PAHE scheme is semantically secure.*

(Proof sketch.) We first prove the security against semi-honest S_0. Note that S_0's view in an execution of our method π is $\text{view}_0^\pi(X, W) = \{X, r_S, [W]\}$, where X is S_0's input, r_S is S_0's randomness, and $[W]$ is the received message. The simulator $Sim_0(X, \texttt{IMV}\{W; X\})$ is constructed as follows. (1) Choose a random tape r_S^*. (2) Initialize a matrix W^* of the same size as W and choose each element in the matrix according to the distribution. (3) Encode each D_j from W^* and compute $[W^*] = ([D_0], \ldots, [D_{z-1}])$ as in EncMat of HME. (4) Output $(X, r_S^*, [W^*])$. Since PAHE is semantically secure, we can prove that $\{Sim_0(X, \texttt{IMV}\{W; X\})\}_{W,X\in\{0,1\}^*} \overset{c}{\equiv} \{\text{view}_0^\pi(X, W)\}_{W,X\in\{0,1\}^*}$ by a hybrid argument and complete the proof against semi-honest S_0.

We now prove the security against semi-honest S_1. The view of S_1 in an execution of our method is $\text{view}_1^\pi(X, W) = \{W, r_S, [W \cdot X - R]\}$, where W is S_1's input, r_S is S_1's randomness, and $[W \cdot X - R]$ is the received message. The simulator $Sim_1(W, \texttt{IMV}\{W; X\})$ is constructed as follows. (1) Choose a random tape r_S^*. (2) Initialize vectors X^* and R^* of the same size as X and R and choose each element in the vectors according to the distribution. (3) Compute $[W \cdot X^* - R^*]$ from W, X^*, and R^*. (4) Output $(X, r_S^*, [W \cdot X^* - R^*])$. Since PAHE is semantically secure, we can prove that $\{Sim_1(W, \texttt{IMV}\{W; X\})\}_{W,X\in\{0,1\}^*} \overset{c}{\equiv} \{\text{view}_1^\pi(X, W)\}_{W,X\in\{0,1\}^*}$ by a hybrid argument and complete the proof against semi-honest S_1.

Hence, S_{1-b} cannot learn about W_b or X_b when S_b and S_{1-b} use our method to secretly share $W_b \cdot X_{1-b}$ or $W_{1-b} \cdot X_b$. $W_b \cdot X_b$ is invisible to S_{1-b} because it is calculated by S_b locally. Finally, S_{1-b} cannot obtain any information about Y_b because it is the sum of $\langle W_0 \cdot X_1\rangle_b$, $\langle W_1 \cdot X_0\rangle_b$, and $\langle W\rangle_b \cdot \langle X\rangle_b$, all of which are kept secret from S_{1-b}. Therefore, FL guarantees that the cloud servers cannot learn about the matrix parameters, input, and output of the FC layer.

B. Convolutional Layer Protocol

In the convolutional layer, the servers S_0 and S_1 secretly share the filters F and the input data X, i.e., S_b takes $\langle F\rangle_b$ and $\langle X\rangle_b$, with the purpose of secretly sharing the convolution $Y = F \odot X$. Our secure two-party computation protocol for convolutional layer is denoted as $\{\langle Y\rangle_0; \langle Y\rangle_1\} \leftarrow \texttt{CL}\{\langle X\rangle_0, \langle F\rangle_0; \langle X\rangle_1, \langle F\rangle_1\}$, where $X = \langle X\rangle_0 + \langle X\rangle_1$, $F = \langle F\rangle_0 + \langle F\rangle_1$, and $Y = \langle Y\rangle_0 + \langle Y\rangle_1$.

We can express the computation of CL protocol as follows. $Y = F \odot X = (\langle F\rangle_0 + \langle F\rangle_1) \odot (\langle X\rangle_0 + \langle X\rangle_1) = \langle F\rangle_0 \odot \langle X\rangle_0 + \langle F\rangle_0 \odot \langle X\rangle_1 + \langle F\rangle_1 \odot \langle X\rangle_0 + \langle F\rangle_1 \odot \langle X\rangle_1$. Since $\langle F\rangle_0 \odot \langle X\rangle_0$ and $\langle F\rangle_1 \odot \langle X\rangle_1$ can be computed by S_0 and S_1 separately, we focus on the calculation method that outputs the additive shares of $\langle F\rangle_b \odot \langle X\rangle_{1-b}$. Then, S_b outputs $\langle Y\rangle_b$ in the CL protocol as follows.

$$\langle Y\rangle_b = \langle F\rangle_b \odot \langle X\rangle_b + \langle\langle F\rangle_0 \odot \langle X\rangle_1\rangle_b + \langle\langle F\rangle_1 \odot \langle X\rangle_0\rangle_b$$

In the state-of-the-art convolution calculation methods such as Gazelle and GALA, one participant encrypts X and sends it to the other who subsequently calculates and returns $[F \odot X - R]$ (similar to the flow in Figure 3a). As a result, these methods involve CPTPerm and require the participants to interact for two rounds. To eliminate the CPTPerm and reduce the number of communication rounds (as in the FL protocol), we construct a Homomorphic Filter Encryption (HFE) scheme that supports calculating the convolution through a ciphertext filter and a plaintext vector.

1) Homomorphic Filter Encryption: We first introduce our Homomorphic Filter Encryption (HFE) scheme which consists of five algorithms: Gen, EncFilter, DecVec, Add, and Conv.

- The key generation algorithm Gen takes as input a security parameter λ and outputs a pair of public-private keys (pk, sk).
- The filter encryption algorithm EncFilter takes as input a public key pk and a group of plaintext filters F and outputs a ciphertext $[F]$.
- The vector decryption algorithm DecVec takes as input a private key sk and a ciphertext vector $[V]$ and outputs a plaintext vector V.
- The vector addition algorithm Add takes as input a public key, a ciphertext $[V]$ and a plaintext vector R and outputs a ciphertext $[V + R]$.
- The convolution algorithm Conv takes as input a ciphertext $[F]$ and a plaintext matrix X and outputs a ciphertext $[F \odot X]$.

We utilize PAHE to construct our homomorphic filter encryption scheme. In the convolutional layer, multiple filters with multiple channels convolute input data with the same number of channels. In the following, we denote the number of filters as m and the number of channels as n. The filters are denoted as $F = (F_0, \ldots, F_{m-1})$, where $F_i = (F_{i,0}, \ldots, F_{i,n-1})$ and $\mathrm{F}_{i,j}$ denotes the j-th channel of the i-th filter. We denote the width and height of $F_{i,j}$ by w and h. The input data is denoted as $X = (X_0, \ldots, X_{n-1})$, where X_j is the j-th channel of X. The convolution Y has m channels denoted as $Y = (Y_0, Y_1, \ldots, Y_{m-1})$) and each channel is $Y_i = F_i \odot X = \sum_{j=0}^{n-1} F_{i,j} \odot X_j$.

Our convolution calculation method is similar to that of the matrix-vector product. We first introduce how to calculate $F_{i,j} \odot X_j$. If the step size is 1, it can be computed as the sum of $w \cdot h$ intermediate elements, each of which is the element-wise product of a rotational variant of X_j and an offset variant of $F_{i,j}$ [15], denoted as $F_{i,j} \odot X_j = \sum_{k=0}^{w \cdot h-1} (F_{i,j}^{(k)} \cdot X_j^{(k)})$. Otherwise, we can decompose it into several convolutions with step size of 1 [24]. Next, we detail our HFE scheme.

$\texttt{Gen}(1^\lambda)$. The key generation algorithm simply outputs $(pk, sk) \leftarrow \texttt{KeyGen}(1^\lambda)$.

$\texttt{EncFilter}(pk, F)$. The filter encryption algorithm first computes the offset variant of F, i.e., $F^{(k)} = (F_0^{(k)}, \ldots, F_{m-1}^{(k)})$ $(0 \le k \le w \cdot h - 1)$, where $F_i^{(k)} = (F_{i,0}^{(k)}, \ldots, F_{i,n-1}^{(k)})$ $(0 \le i \le m-1)$. $F^{(k)}$ can be seen as an $m \times n$ matrix. For each $F^{(k)}$, it encodes $t = \lfloor (N/(m \cdot h \cdot w) \rfloor$ diagonals into a single vector $D_j^{(k)}$ and obtains $z = n/t$ vectors as in EncMat of HME. Then, the algorithm computes $[D_j^{(k)}] \leftarrow \texttt{Enc}(pk, D_j^{(k)})$ for each $D_j^{(k)}$ and outputs $[F] = ([F^{(0)}], \ldots, [F^{(w \cdot h-1)}])$, where $[F^{(k)}] = ([D_0^{(k)}], \ldots, [D_{z-1}^{(k)}])$.

$\texttt{DecVec}(sk, [V])$. The vector decryption algorithm performs $V \leftarrow \texttt{Dec}(sk, [V])$ and outputs a plaintext vector $V^* = (V_0^*, \ldots, V_{m \cdot w \cdot h-1}^*)$, where $V_j^* = \sum_{k=0}^{t} V_{j+k \cdot m \cdot w \cdot h}$.

$\texttt{Add}(pk, [V], R)$. The addition algorithm calculates $[R] \leftarrow \texttt{Enc}(pk, R)$ and outputs $[V + R] \leftarrow \texttt{CPTAdd}([V], [R])$.

$\texttt{Conv}([F], X)$. The convolution algorithm calculates $X^{(k)} = (X_0^{(k)}, \ldots, X_{n-1}^{(k)})$ $(0 \le k \le w \cdot h - 1)$, where $X^{(k)}$ can be seen as a vector. For each $X^{(k)}$, the algorithm encodes it into z vectors denoted as $X_{(j)}^{(k)}$ $(0 \le j \le z - 1)$ as in Mul of HME. Then, it parses $[F]$ as $([F^{(0)}], \ldots, [F^{(w \cdot h-1)}])$ and $[F^{(k)}]$ as $[(D_0^{(k)}, \ldots, D_{z-1}^{(k)})]$. For each $D_j^{(k)}$, it computes $C_j^{(k)} \leftarrow \texttt{CPTScMult}([D_j^{(k)}], X_{(j)}^{(k)})(0 \le j \le z-1)$. Finally, it invokes CPTAdd to add up all the $C_j^{(k)}$ $(0 \le j \le z - 1)$ $(0 \le k \le w \cdot h - 1)$ to obtain a ciphertext as the output.

The noise in the convolution algorithm is as follows. We denote the noise of $[D_j]$ as η_0. CPTScMult increases the noise of $C_j^{(k)}$ to $\eta_0 \cdot \eta_{mult}$. Finally, invoking CPTAdd on $C_j^{(k)}$ makes the noise of the output ciphertext grow to $\frac{m \cdot n}{N} \cdot w \cdot h \cdot \eta_0 \cdot \eta_{mult}$.

We compare our method with other state-of-the-art methods, including Gazelle and GALA, as shown in Table III. N denotes the number of slots in a ciphertext and u_h and u_w denotes the height and width of input data. Our method eliminates plenty of the most expensive CPTPerm without increasing other operations compared with state-of-the-art methods, which results in significant acceleration. The noise growth of our ciphertext is much less than that of GALA and Gazelle considering $\eta_{rot} > \eta_{mult} \gg \eta_0 \gg 1$.

2) Our Protocol: We now propose the CL protocol (recall that $[F_{1-b}]$ is sent to S_b in the model deployment phase). The process of CL is very similar to FL, and the only difference is to calculate convolutions instead of matrix-vector products. This is feasible because both convolution and matrix-product satisfy the associative law.

TABLE III
COMPARISON OF CONVOLUTION CALCULATION METHODS

Protocol	`CPTAdd`	`CPTScMult`	`CPTPerm`	Rounds	Noise Growth
Gazelle	$\frac{m(n\cdot w\cdot h-1)u_w\cdot u_h}{N}$	$\frac{m\cdot n\cdot w\cdot h\cdot u_w\cdot u_h}{N}$	$\frac{n(w\cdot h-1)u_w\cdot u_h}{N}+\frac{m\cdot n(N-u_w\cdot u_h)u_w\cdot u_h}{N^2}$	2	$\eta_\triangle+(n-\frac{n\cdot u_w\cdot u_h}{N})\eta_{rot}$
GALA	$\frac{m(n\cdot w\cdot h-1)u_w\cdot u_h}{N}$	$\frac{m\cdot n\cdot w\cdot h\cdot u_w\cdot u_h}{N}$	$\frac{n(w\cdot h-1)u_w\cdot u_h}{N}+\frac{m(N-u_w\cdot u_h)}{N}$	2	$\eta_\triangle$
Pio	$\frac{m(n\cdot w\cdot h-1)u_w\cdot u_h}{N}$	$\frac{m\cdot n\cdot w\cdot h\cdot u_w\cdot u_h}{N}$	0	1	$\frac{m\cdot n}{N}\cdot w\cdot h\cdot\eta_0\cdot\eta_{mult}$

$\eta_\triangle = n\cdot w\cdot h\cdot\eta_0\cdot\eta_{mult}+n(w\cdot h-1)\eta_{mult}\cdot\eta_{rot}$

To reduce the online prediction latency, we use HFE for the precomputation of the offline phase. Specifically, in the offline phase, S_b first selects a random matrix M_b of the same size as X and calculates the convolution of $[\langle F\rangle_{1-b}]$ and M_b. Then, S_b selects a random matrix R_b of the same size as $\langle F\rangle_{1-b}\odot M_b$ and utilizes it to mask $\langle F\rangle_{1-b}\odot M_b$. That is to say, S_b performs

$$[V_{1-b}] \leftarrow \mathtt{Add}(\mathtt{Conv}([\langle F\rangle_{1-b}], M_b), -R_b).$$

And then, S_b sends $[V_{1-b}]$ to S_{1-b} and receives $[V_b]$. Finally, S_b decrypts to obtain $V_b \leftarrow \mathtt{DecVec}(sk_b, [V_b])$. Indeed, V_b is equal to $\langle F\rangle_b \odot M_{1-b} - R_{1-b}$.

In the online phase, S_b sends $\langle X\rangle_b - M_b$ to S_{1-b} and receives $\langle X\rangle_{1-b} - M_{1-b}$. Then, S_b computes $V'_b = \langle F\rangle_b \odot (\langle X\rangle_{1-b} - M_{1-b})$ and $\langle\langle F\rangle_b \odot \langle X\rangle_{1-b}\rangle_b = V_b + V'_b = \langle F\rangle_b \odot M_{1-b} - R_{1-b} + \langle F\rangle_b \odot (\langle X\rangle_{1-b} - M_{1-b}) = \langle F\rangle_b \odot \langle X\rangle_{1-b} - R_{1-b}$. And then, S_b sets $\langle\langle F\rangle_{1-b} \odot \langle X\rangle_b\rangle_b = R_b$. In this way, it satisfies that $\langle\langle F\rangle_b \odot \langle X\rangle_{1-b}\rangle_b + \langle\langle F\rangle_b \odot \langle X\rangle_{1-b}\rangle_{1-b} = \langle F\rangle_b \odot \langle X\rangle_{1-b}$. Finally, S_b outputs

$$\begin{aligned}\langle Y\rangle_b \leftarrow \langle F\rangle_b \odot \langle X\rangle_b + \langle\langle F\rangle_b \odot \langle X\rangle_{1-b}\rangle_b \\ + \langle\langle F\rangle_{1-b} \odot \langle X\rangle_b\rangle_b.\end{aligned}$$

According to the associative law of convolution, it satisfies $\langle Y\rangle_0 + \langle Y\rangle_1 = F \odot X$. In our protocol, the latency-sensitive online phase of `CL` only involves the addition and multiplication of plaintext.

The security of our convolutional layer protocol can be proved under the semantic security of the PAHE scheme. We omit the proof since it is similar to that in Section VI-A.4.

C. Activation Layer Protocol

In this protocol, we only consider the computation of ReLU function, which is the default and most common activation function in CNNs [30], [31]. The secure activation layer computation protocol is denoted as $\{\langle Y\rangle_0; \langle Y\rangle_1\} \leftarrow \mathtt{AL}\{\langle X\rangle_0; \langle X\rangle_1\}$, where $X = \langle X\rangle_0 + \langle X\rangle_1$, $Y = \langle Y\rangle_0 + \langle Y\rangle_1$, and $Y = \mathtt{ReLU}(X)$.

We employ GC to implement this protocol. S_0 inputs $\langle X\rangle_0$ and S_1 inputs $\langle X\rangle_1$ and a random matrix R. The garbled circuit reconstructs X and calculates $Y \leftarrow \mathtt{ReLU}(X) = max(0, X)$, after which return $\langle Y\rangle_0 = Y - R$ to S_0 as the output. S_1 takes R as the output $\langle Y\rangle_1$. In this way, S_0 and S_1 secretly share Y. The uniformly chosen R and GC guarantee that S_0 and S_1 can learn nothing about X and Y.

D. Pooling Layer Protocol

We introduce our secure two-party computation protocol for pooling layer including max-pooling and mean-pooling layer. In the pooling layer, S_b takes a secret share $\langle X\rangle_b$, aiming at secretly sharing the output Y, whose element is the maximum or average of a subarea of X. Our secure pooling layer computation protocol is denoted as $\{\langle Y\rangle_0; \langle Y\rangle_1\} \leftarrow \mathtt{PL}\{\langle X\rangle_0; \langle X\rangle_1\}$, where $X = \langle X\rangle_0 + \langle X\rangle_1$ and $Y = \langle Y\rangle_0 + \langle Y\rangle_1$.

1) Max-Pooling: Similar to our activation layer protocol, we employ GC to compute maximum value of the elements in corresponding subarea of X. GC and ASS guarantee that non-colluding servers cannot learn information about the input or output of the max-pooling layer.

2) Mean-Pooling: It can be easily calculated by having each server conduct mean-pooling on their respective shares.

3) Remark: We prove the security of our linear layer protocols and the security of our non-linear protocols is directly guaranteed by GC. Specifically, our protocols guarantee that the cloud servers cannot learn about the layers' model parameters, input, and output, all of which are kept secretly shared. Our inference phase is to connect these protocols sequentially and is secure since protocols ending with secure re-sharing of outputs are universally composable [40]. Additionally, ASS and HE keep the model parameters and the client's input data secret from the cloud servers during the model deployment phase and data upload phase. Hence, in our solution, the cloud servers can learn nothing about the model parameters and the client's input and prediction results.

VII. EVALUATION

We first describe the implementation of our solution. Then, we compare the runtime and communication cost of our calculation methods for matrix-vector product and convolution with that of the naive method and state-of-the-art methods including Gazelle and GALA. Finally, we evaluate the performance of our solution on CNN models.

A. Implementation

We utilize the PAHE based on Brakerski-Fan-Vercauteren (BFV) scheme [41], [42]. We use the GC which incorporates the Half-Gates optimization [43] into JustGarble [44] and achieves a good performance. We set the cryptographic parameters in conformity to GALA including: (1) A 128-bit security level parameters for HE and GC schemes. (2) A plaintext modulus m_p of 20 bits. (3) A 60-bit pseudo-Mersenne prime m_q which is slightly smaller than the native machine word on a 64 bit machine to be the ciphertext modulus. (4) The capacity of the ciphertext $N = 2048$. To compare with other solutions, we reimplement GALA and use the source code of ABY [45], Gazelle, and CrypTFlow2. We perform the evaluations in a

TABLE IV

COMPUTATION COMPLEXITY OF MATRIX-VECTOR PRODUCT MICROBENCHMARKS

Protocol	MV1: $(m \times n)$:(128×80) CPTAdd	CPTScMult	CPTPerm
Naive	1023	256	896
Gazelle	11	8	11
GALA	7	8	7
Pio	**4**	**5**	**0**
Protocol	MV2: $(m \times n)$:(128×528) CPTAdd	CPTScMult	CPTPerm
Naive	1407	256	1280
Gazelle	67	64	67
GALA	63	64	63
Pio	**32**	**33**	**0**
Protocol	MV3: $(m \times n)$:(256×136) CPTAdd	CPTScMult	CPTPerm
Naive	2303	512	2048
Gazelle	34	32	34
GALA	31	32	31
Pio	**16**	**17**	**0**
Protocol	MV4: $(m \times n)$:(512×516) CPTAdd	CPTScMult	CPTPerm
Naive	5631	1024	5120
Gazelle	257	256	257
GALA	255	256	255
Pio	**128**	**129**	**0**

TABLE V

MATRIX-VECTOR PRODUCT MICROBENCHMARKS. "COMM" DENOTES THE COMMUNICATION COST, "SET" DENOTES THE SETUP PART, AND "INF" DENOTES THE INFERENCE PART

Matrix	Protocol	LAN (ms)		WAN (ms)		Comm (MB)	
		Set	Inf	Set	Inf	Set	Inf
MV1	Naive	175.3	686.4	1331.8	742.3	29.44	0.1
	Gazelle	19.3	14.6	161.6	89.5	2.53	0.1
	CTF2 (OT)	185.2	31.3	227.9	187.6	**0.48**	0.29
	GALA	13.1	5.6	84.7	83.7	1.61	0.1
	Pio	**4.5**	**1.8**	**47.1**	**28.9**	0.55	**0.03**
MV2	Naive	176.7	1059.4	1290.8	1143.4	30.13	0.1
	Gazelle	100.7	41.9	706.9	112.9	15.37	0.1
	CTF2 (OT)	187.8	71.6	226.5	815.3	**0.48**	1.95
	GALA	96.7	19.8	671.3	97.5	14.45	0.1
	Pio	**24.5**	**5.6**	**135.7**	**32.2**	2.37	**0.03**
MV3	Naive	281.3	1526.4	2493.1	1609.6	58.88	0.1
	Gazelle	50.2	25.3	364.2	103.8	7.74	0.1
	CTF2 (OT)	183.7	49.2	229.3	418.7	**0.48**	0.91
	GALA	47.1	9.8	337.4	86.6	7.05	0.1
	Pio	**15.6**	**3.3**	**88.1**	**30.9**	1.33	**0.03**
MV4	Naive	665.4	4976.5	5049.2	5046.5	117.99	0.1
	Gazelle	381.7	130.0	2577.9	213.1	58.95	0.1
	CTF2 (OT)	189.1	196.2	**231.6**	2616.7	**0.48**	7.19
	GALA	371.6	56.2	2509.4	130.4	58.49	0.1
	Pio	**88.9**	**19.7**	439.1	**48.2**	8.61	**0.03**

LAN network setting with an average bandwidth of 950mps and round-trip time around 0.25ms and a WAN setting with a bandwidth of 150mps and round-trip time around 31ms. All experiments are conducted on HUAWEI cloud servers with 2.8 GHz Intel Xeon CPU and 16 GB RAM.

B. Linear Calculations Benchmarks

We elaborate efficient linear calculation methods for matrix-vector product (in Section VI-A.3) and convolution (in Section VI-B.1). We run benchmarks to compare our methods with that of the naive method, Gazelle, and GALA. We also compare our calculation method for matrix-vector product with CrypTFlow2 (OT) which is the state-of-the-art OT-based solution without modifying the model. In the following, we use CTF2 to represent CrypTFlow2.

For ease of analysis, we divide these methods into two parts: setup part and inference part. The former includes key generation and transmission of the above methods and matrix/filter encryption and transmission of our method. The latter includes the process described in Figure 3. Note that the setup part is independent of clients' inputs and occurs only once between the two participants.

1) Matrix-Vector Product Benchmarks: We run four benchmarks with the following matrix size (denoted as MV1 to MV4): 128×80, 128×528, 256×136, and 512×516. Table IV shows the computational complexity of the naive method, Gazelle, GALA, and Pio. It can be seen that Pio eliminates the most expensive `CPTPerm` operations, which is 255 for a 512×1024 parameter matrix. Moreover, we make better use of the slots of ciphertext and involve fewer `CPTAdd` and `CPTScMult` than GALA which fills too many zeros.

Table V shows the microbenchmarks comparison. Although the matrix needs to be encrypted and transferred in the setup part of Pio, Pio is $3-4\times$ and $2-6$ faster than GALA in the LAN and WAN settings respectively, and the communication cost is $3-7\times$ smaller. This is because we do not need to generate and transfer the keys for `CPTPerm`. In the inference part, Pio is about $3\times$ faster than GALA in both the LAN and WAN settings, for we remove the expensive `CPTPerm` operations, make better use of the slots, and reduce the number of communication rounds to 1. The communication cost is about $3\times$ smaller than GALA. Although the communication cost of CTF2 (OT) is $1-17\times$ smaller than Pio in the setup part, it is $10-240\times$ larger in the inference part. Pio is $10-17\times$ faster than CTF2 (OT) in the LAN setting and $7-54\times$ faster in the WAN setting in the inference part.

2) Convolution Benchmarks: Table VI depicts the computational complexity of Gazelle, GALA, and Pio in calculating convolution. Compared with Gazelle and GALA, Pio removes expensive `CPTPerm` operation $\frac{n(w \cdot h-1)u_w \cdot u_h}{N} + \frac{m(N-u_w \cdot u_h)}{N}$ times which increases with the number of channels and the size of the convolution kernel. Also, Pio involves fewer `CPTAdd` and `CPTScMult` for we make better use of the slots.

Table VII depicts the microbenchmarks comparison. Although the filters need to be encrypted and transferred in the setup part of Pio, Pio does not need to generate and transfer the keys for `CPTPerm`. The experimental results show that Pio is $1-2.5\times$ and $2-6.1\times$ faster than GALA in the LAN and WAN settings respectively, and the communication cost is $2.1-9.5\times$ smaller. In the inference part, Pio is $3-4\times$ and $2.6-3.2\times$ faster in the LAN and WAN settings respectively, and the communication overhead is $3.3-7.7\times$ smaller, for we avoid the expensive `CPTPerm`, make better use of the slots, and reduce the number of communication rounds.

TABLE VI
COMPUTATION COMPLEXITY OF CONVOLUTION MICROBENCHMARKS

Protocol	CPTAdd	CPTScMult	CPTPerm
Conv1: $(n \times u_w \times u_h)$:$(1 \times 32 \times 32)$, $(m \times w \times h)$:$(1 \times 3 \times 3)$			
Gazelle	8	9	8
GALA	8	9	8
Pio	8	9	**0**
Conv2: $(n \times u_w \times u_h)$:$(1 \times 16 \times 16)$, $(m \times w \times h)$:$(5 \times 3 \times 3)$			
Gazelle	71	72	15
GALA	71	72	15
Pio	**24**	**27**	**0**
Conv3: $(n \times u_w \times u_h)$:$(5 \times 28 \times 28)$, $(m \times w \times h)$:$(1 \times 5 \times 5)$			
Gazelle	147	150	27
GALA	147	150	25
Pio	**74**	**75**	**0**
Conv4: $(n \times u_w \times u_h)$:$(5 \times 32 \times 32)$, $(m \times w \times h)$:$(1 \times 3 \times 3)$			
Gazelle	53	54	11
GALA	53	54	9
Pio	**26**	**27**	**0**

TABLE VII
CONVOLUTION MICROBENCHMARKS. "COMM" DENOTES THE COMMUNICATION COST, "SET" DENOTES THE SETUP PART, AND "INF" DENOTES THE INFERENCE PART

Convolution	Protocol	LAN (ms)		WAN (ms)		Comm (MB)	
		Set	Inf	Set	Inf	Set	Inf
Conv1	Gazelle	25.9	16.4	146.6	101.4	2.53	0.1
	GALA	25.5	15.2	110.3	91.6	1.61	0.1
	Pio	**10.1**	**5.1**	**53.6**	**35.3**	0.55	**0.03**
Conv2	Gazelle	100.8	34.3	715.0	121.9	15.37	0.1
	GALA	88.2	32.1	669.9	102.7	14.45	0.1
	Pio	**39.3**	**8.0**	**153.1**	**35.6**	2.37	**0.03**
Conv3	Gazelle	84.9	42.7	405.9	118.6	7.74	0.1
	GALA	78.0	30.5	372.8	109.2	7.05	0.1
	Pio	**75.1**	**10.2**	**139.7**	**41.0**	1.33	**0.03**
Conv4	Gazelle	28.1	22.3	2329.8	106.6	58.95	0.1
	GALA	25.2	16.9	2309.8	94.8	58.49	0.1
	Pio	**21.8**	**5.9**	**377.4**	**30.1**	8.61	**0.03**

3) Summary: The benchmarks of matrix-vector product and convolution prove that the methods in Pio for privacy-preserving linear calculations outperform other state-of-the-art HE-based solutions (e.g., GALA) in terms of runtime and communication in the setup part and inference part. Compared with the state-of-the-art OT-based solution CTF2 (OT), Pio is more efficient in the inference part. Although the communication cost is higher in the setup part, Pio is extremely parsimonious in the inference part. Note that our methods can replace these methods to raise the efficiency of privacy-preserving inference solutions without model privacy.

C. CNN Benchmarks

We conclude experiments on realistic CNNs for MNIST and CIFAR-10 image classification tasks. For a fair comparison, we extend Gazelle, CrypTFlow2, and GALA to the outsourced PaaS paradigm based on two non-colluding servers and compare them with Pio. For ease of representation, we add "-OS" after their names to denote their extended versions.

We now introduce our universal framework that can smoothly extend secure two-party inference solutions to the outsourced scenario. In secure two-party inference solutions with plaintext models, the client and server execute secure two-party computation protocols sequentially to carry out CNN layer by layer. In all protocols, the client and server start by secretly sharing the layer's input and end by secretly sharing the layer's output. The FC layer protocols can be abstracted as $\{\langle W \cdot X\rangle_0; \langle W \cdot X\rangle_1\} \leftarrow \mathtt{FL}'\{\langle X\rangle_0; W, \langle X\rangle_1\}$, where $X = \langle X\rangle_0 + \langle X\rangle_1$ and is the input to the FC layer, and W is the parameter matrix. Specifically, the client (resp. server) inputs $\langle X\rangle_0$ (resp. W and $\langle X\rangle_1$) to the $\mathtt{FL}$ protocol and finally secretly shares the output $W \cdot X$. Similarly, the convolutional layer, activation layer, and pooling layer protocols can be represented as $\{\langle F \odot X\rangle_0; \langle F \odot X\rangle_1\} \leftarrow \mathtt{CL}'\{\langle X\rangle_0; F, \langle X\rangle_1\}$, $\{\langle Y\rangle_0; \langle Y\rangle_1\} \leftarrow \mathtt{AL}'\{\langle X\rangle_0; \langle X\rangle_1\}$, and $\{\langle Y\rangle_0; \langle Y\rangle_1\} \leftarrow \mathtt{PL}'\{\langle X\rangle_0; \langle X\rangle_1\}$, respectively. Their extended versions for the outsourced scenario also consist of three phases: model deployment, data upload, and inference.

1) Model Deployment: In the beginning, S_b ($b \in \{0, 1\}$) generates public-private keys and sends the public key to S_{1-b}. The deployment of the activation layer and pooling layer is consistent with Pio. For the FC layer, the model owner generates secret shares of the parameter matrix W and sends W_b to S_b. For the convolutional layer, the model owner generates and sends the filters' secret share F_b to S_b.

2) Data Upload: A client generates secret shares of his/her data X and sends X_b to S_b.

3) Inference: The cloud servers that secretly share model parameters and client input carry out CNN layer by layer. For each layer, they secretly share the input and aim at secretly sharing the output. (1) FC layer: The cloud servers execute $\mathtt{FL}'$ to secretly share $\langle W\rangle_0 \cdot X$ and $\langle W\rangle_1 \cdot X$ and add up their respective shares to secretly share $W \cdot X$. Specifically, S_0 (resp. S_1) input X_0 (resp. W_1 and X_1) to the $\mathtt{FL}'$ protocol to secretly share $\langle W\rangle_1 \cdot X$, denoted as $\{\langle\langle W\rangle_1 \cdot X\rangle_0; \langle\langle W\rangle_1 \cdot X\rangle_1\} \leftarrow \mathtt{FL}'\{\langle X\rangle_0; \langle W\rangle_1, \langle X\rangle_1\}$. Then, S_1 (resp. S_0) input X_1 (resp. W_0 and X_0) to $\mathtt{FL}'$ to secretly share $\langle W\rangle_0 \cdot X$, denoted as $\{\langle\langle W\rangle_0 \cdot X\rangle_1; \langle\langle W\rangle_0 \cdot X\rangle_0\} \leftarrow \mathtt{FL}'\{\langle X\rangle_1; \langle W\rangle_0, \langle X\rangle_0\}$. Finally, S_b adds up $\langle\langle W\rangle_0 \cdot X\rangle_b$ and $\langle\langle W\rangle_1 \cdot X\rangle_b$ as the output Y_b. It satisfies $\langle Y\rangle_0 + \langle Y\rangle_1 = \langle\langle W\rangle_0 \cdot X\rangle_0 + \langle\langle W\rangle_1 \cdot X\rangle_0 + \langle\langle W\rangle_0 \cdot X\rangle_1 + \langle\langle W\rangle_1 \cdot X\rangle_1 = W \cdot X$. (2) Convolution layer: similar to the FC layer and the only difference is that S_0 and S_1 run $\mathtt{CL}'$ instead of $\mathtt{FL}'$. (3) Activation/pooling layer: S_0 and S_1 simply execute $\{\langle Y\rangle_0; \langle Y\rangle_1\} \leftarrow \mathtt{AL}'\{\langle X\rangle_0; \langle X\rangle_1\}$ or $\{\langle Y\rangle_0; \langle Y\rangle_1\} \leftarrow \mathtt{PL}'\{\langle X\rangle_0; \langle X\rangle_1\}$.

Also, we implement a solution for the outsourced PaaS paradigm based on the generic MPC framework ABY [45], called ABY-OS. Specifically, the model parameters and the client's input are secretly shared between S_0 and S_1 in the model deployment phase and data upload phase, respectively. In the inference phase, S_0 and S_1 carry out CNN layer by layer, and the inputs and outputs of each layer are secretly shared between them. They utilize Beaver's triplets to calculate matrix-vector product and convolution and use GC to calculate

TABLE VIII
CNN ARCHITECTURE

CNN	Architecture	Source
A	FC1(128×784) $\rightarrow$ square $\rightarrow$ FC2(128×128) $\rightarrow$ square $\rightarrow$ FC3(10×128)	SecureML [28]
B	Conv1($1 \times 28 \times 28$, $5 \times 5 \times 5$) $\rightarrow$ square $\rightarrow$ mean-pooling $\rightarrow$ square $\rightarrow$ FC4(10×100)	CryptoNets [21]
C	Conv1($1 \times 28 \times 28$, $5 \times 5 \times 5$) $\rightarrow$ ReLU $\rightarrow$ FC5(100×865) $\rightarrow$ ReLU $\rightarrow$ FC4(10×100)	Deepsecure [23]

TABLE IX
MNIST BENCHMARK. "COMM" DENOTES THE COMMUNICATION COST

CNN	Protocol	LAN (ms)	WAN (ms)	Comm (MB)
A	ABY-OS	169.5	2616.7	16.8
	Gazelle-OS	122.7	463.7	0.9
	GALA-OS	42.5	406.9	0.9
	Pio	**13.2**	**104.6**	**0.3**
B	ABY-OS	497.8	9819.3	62.4
	Gazelle-OS	102.2	385.7	0.7
	GALA-OS	42.4	331.5	0.7
	Pio	**10.7**	**129.8**	**0.4**
C	ABY-OS	5764.9	37694.1	220.4
	Gazelle-OS	253.7	1488.4	2.9
	GALA-OS	104.6	1295.4	2.9
	Pio	**45.7**	**656.4**	**2.1**

max-pooling and ReLU. The Beaver's triplets are generated by HE in the offline phase, and we use CPT to accelerate it.

We implement the above extended versions on top of their respective source codes and compare them with Pio.

4) MNIST Benchmarks: Previous privacy-preserving inference work (e.g. CryptoNets [21], MiniONN [22], Gazelle, and GALA) have been applied to some CNNs to complete MNIST classification task. We adopt the CNNs described in Table VIII and compare Pio with ABY-OS, Gazelle-OS, and GALA-OS in terms of total (offline and online) inference runtime and communication cost between the non-colluding servers.

The empirical performance evaluation in Table IX indicates that Pio is obviously parsimonious in both runtime and bandwidth. Specifically, the runtime of Pio is $2-4\times$ faster than GALA-OS and the communication cost is $1.4-3\times$ smaller, which benefits from our efficient and low communication protocols for linear calculation.

5) CIFAR-10 Benchmarks: To compare the performance of other solutions and Pio in realistic scenarios, we apply them on popular neural networks, including AlexNet, VGG, ResNet-18, ResNet-50, ResNet-101, and ResNet-152 for CIFAR-10 tasks.

The experimental results are shown in Table X. In popular models, Pio is $2.5-4.2\times$ faster than GALA-OS in the LAN setting and $2.6-3.2\times$ faster in the WAN setting. The communication cost of GALA-OS is $1.7-2.4\times$ of Pio. Pio saves about 459.1 megabytes than GALA-OS for each inference on ResNet-152. Compared to CTF2 (OT)-OS, our solution is $4.3-6.4\times$ and $21-28\times$ faster in the LAN and WAN settings respectively, and our communication cost is $79-170\times$ smaller.

Our solution does not introduce approximation and the only possible loss of accuracy comes from the replacement of fixed point numbers for floating point numbers. GALA and CrypTFlow2 have proved that this accuracy loss is negligible

TABLE X
CIFAR-10 BENCHMARK. "COMM" DENOTES THE COMMUNICATION COST

CNN	Protocol	LAN (s)	WAN (s)	Comm (MB)	Accuracy (%)
AlexNet	Gazelle-OS	14.6	18.5	23.27	97.32
	GALA-OS	6.2	9.7	23.27	**97.33**
	Pio	**2.2**	**3.4**	**9.52**	97.32
VGG	Gazelle-OS	22.7	27.9	30.44	**98.04**
	GALA-OS	8.6	13.6	30.44	98.02
	Pio	**3.5**	**5.4**	**12.72**	98.03
Res-18	Gazelle-OS	52.8	63.3	68.29	98.22
	CTF2 (OT)-OS	26.7	290.4	6324.43	98.21
	CTF2 (HE)-OS	29.9	68.7	259.21	**98.25**
	GALA-OS	17.1	28.1	68.29	98.24
	Pio	**5.8**	**10.4**	**37.06**	98.23
Res-50	Gazelle-OS	334.2	398.5	395.72	98.31
	CTF2 (OT)-OS	65.3	870.9	19153.81	98.34
	CTF2 (HE)-OS	69.4	249.6	1276.43	98.33
	GALA-OS	42.4	102.1	395.72	98.33
	Pio	**10.0**	**32.2**	**208.36**	**98.36**
Res-101	Gazelle-OS	597.1	698.4	786.31	98.45
	CTF2 (OT)-OS	136.5	1710.6	38257.49	98.44
	CTF2 (HE)-OS	139.8	531.2	2661.54	**98.47**
	GALA-OS	91.5	207.7	786.31	98.46
	Pio	**26.2**	**78.1**	**446.34**	**98.47**
Res-152	Gazelle-OS	807.0	918.3	1146.57	98.77
	CTF2 (OT)-OS	216.6	2407.4	54168.43	**98.84**
	CTF2 (HE)-OS	209.5	716.0	3814.34	98.79
	GALA-OS	131.6	298.9	1146.57	98.80
	Pio	**42.0**	**114.8**	**687.48**	98.83

in their experiments. We apply other solutions and Pio on the same models and compare the average accuracy of ten tests. The experimental results show that our accuracy is very close to other solutions.

VIII. CONCLUSION

We propose an efficient privacy-preserving CNN inference solution for the outsourced PaaS paradigm, which protects the inputs and prediction results of clients and model parameters. In particular, our linear calculations outperform the state-of-the-art solution GALA, since we avoid the expensive permutation operations. Experimental results show that our solution is 2.5–4.2 times faster than GALA on realistic CNN models including AlexNet, VGG, and ResNet. Compared to state-of-the-art OT-based solution CrypTFlow2, Pio is 21–28 times faster in the WAN setting and communication cost is $79-170\times$ smaller.

REFERENCES

[1] W. Wang et al., "Rafiki: Machine learning as an analytics service system," *Proc. VLDB Endowment*, vol. 12, no. 2, pp. 128–140, Oct. 2018.

[2] [Online]. Available: https://aws.amazon.com

[3] [Online]. Available: https://ai.aliyun.com

[4] [Online]. Available: https://cloud.tencent.com

[5] [Online]. Available: https://www.sygic.com

[6] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar, "Chameleon: A hybrid secure computation framework for machine learning applications," in *Proc. Asia Conf. Comput. Commun. Secur.*, May 2018, pp. 707–721.

[7] L. K. L. Ng and S. S. M. Chow, "GForce: GPU-friendly oblivious and rapid neural network inference," in *Proc. USENIX Secur. Symp.* Berkeley, CA, USA: USENIX Association, 2021, pp. 1–18.

[8] *Proposal for a Regulation of the European Parliament and of the Council on the Protection of Individuals With Regard to the Processing of Personal Data and on the Free Movement of Such Data (General Data Protection Regulation), COM (2012) 11 Final*, 2012.

[9] M. Barni, C. Orlandi, and A. Piva, "A privacy-preserving protocol for neural-network-based computation," in *Proc. 8th Workshop Multimedia Secur.*, Sep. 2006, pp. 146–151.

[10] M. Li, Y. Yan, Q. Wang, M. Du, Z. Qin, and C. Wang, "Secure prediction of neural network in the cloud," *IEEE Netw.*, vol. 35, no. 1, pp. 251–257, Jan. 2021.

[11] F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A unified embedding for face recognition and clustering," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 815–823.

[12] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. ICLR*, 2015, pp. 1–14.

[13] A. Brutzkus, R. Gilad-Bachrach, and O. Elisha, "Low latency privacy preserving inference," in *Proc. ICML*, 2019, pp. 1–10.

[14] S. U. Hussain, M. Javaheripi, M. Samragh, and F. Koushanfar, "COINN: Crypto/ML codesign for oblivious inference via neural networks," in *Proc. CCS*. New York, NY, USA: ACM, 2021, pp. 3266–3281.

[15] Q. Zhang, C. Xin, and H. Wu, "GALA: Greedy computation for linear algebra in privacy-preserved neural networks," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2021, pp. 1–16.

[16] H. Chaudhari, R. Rachuri, and A. Suresh, "Trident: Efficient 4PC framework for privacy preserving machine learning," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2020, pp. 1–26.

[17] J. P. K. Ma, R. K. H. Tai, Y. Zhao, and S. S. M. Chow, "Let's stride blindfolded in a forest: Sublinear multi-client decision trees evaluation," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2021, pp. 1–18.

[18] X. Jiang, M. Kim, K. Lauter, and Y. Song, "Secure outsourced matrix computation and application to neural networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2018, pp. 1209–1222.

[19] H. Chen, W. Dai, M. Kim, and Y. Song, "Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2019, pp. 1–30.

[20] M. Li, S. S. M. Chow, S. Hu, Y. Yan, M. Du, and Z. Wang, "Optimizing privacy-preserving outsourced convolutional neural network predictions," 2020, *arXiv:2002.10944*.

[21] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. E. Lauter, M. Naehrig, and J. Wernsing, "CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy," in *Proc. ICML*, 2016, p. 210.

[22] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via MiniONN transformations," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 1–13.

[23] B. D. Rouhani, M. S. Riazi, and F. Koushanfar, "DeepSecure: Scalable provably-secure deep learning," in *Proc. 55th ACM/ESDA/IEEE Design Autom. Conf. (DAC)*, Jun. 2018, pp. 1–6.

[24] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *Proc. USENIX Secur. Symp.* Berkeley, CA, USA: USENIX Association, 2018, pp. 1–19.

[25] M. S. Riazi, M. Samragh, H. Chen, K. Laine, K. E. Lauter, and F. Koushanfar, "XONN: XNOR-based oblivious deep neural network inference," in *Proc. USENIX Secur. Symp.* Berkeley, CA, USA: USENIX Association, 2019, pp. 1–18.

[26] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "Delphi: A cryptographic inference system for neural networks," in *Proc. USENIX Secur. Symp.*, Nov. 2020, pp. 27–30.

[27] D. Rathee et al., "CrypTFlow2: Practical 2-party secure inference," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2020, pp. 1–18.

[28] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2017, pp. 19–38.

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

[30] C. Szegedy et al., "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.

[31] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017.

[32] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds," in *Proc. ASIACRYPT*, 2016, pp. 3–33.

[33] C. Gentry, S. Halevi, and N. P. Smart, "Fully homomorphic encryption with polylog overhead," in *Proc. EUROCRYPT*. Cham, Switzerland: Springer, 2012, pp. 1–18.

[34] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proc. EUROCRYPT*. Berlin, Germany: Springer, 1999, pp. 223–238.

[35] N. P. Smart and F. Vercauteren, "Fully homomorphic SIMD operations," *Designs, Codes Cryptography*, vol. 71, no. 1, pp. 57–81, Apr. 2014.

[36] Z. Brakerski, C. Gentry, and S. Halevi, "Packed ciphertexts in LWE-based homomorphic encryption," in *Proc. PKC*. Cham, Switzerland: Springer, 2013, pp. 1–12.

[37] P. Mohassel and P. Rindal, "ABY^3: A mixed protocol framework for machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2018, pp. 1–40.

[38] A. Patra and A. Suresh, "BLAZE: Blazing fast privacy-preserving machine learning," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2020, pp. 1–28.

[39] B. Wang and N. Z. Gong, "Stealing hyperparameters in machine learning," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2018, pp. 36–52.

[40] D. Bogdanov, S. Laur, and J. Willemson, "Sharemind: A framework for fast privacy-preserving computations," in *Proc. ESORICS*. Berlin, Germany: Springer, 2008, pp. 192–206.

[41] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *IACR Cryptol. ePrint Arch.*, 2012, p. 144.

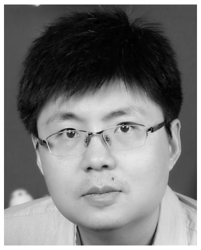
[42] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical GapSVP," in *Proc. CRYPTO*. Cham, Switzerland: Springer, 2012, pp. 1–19.

[43] S. Zahur, M. Rosulek, and D. Evans, "Two halves make a whole reducing data transfer in garbled circuits using half gates," in *Proc. EUROCRYPT*. Cham, Switzerland: Springer, 2015, pp. 1–28.

[44] M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway, "Efficient garbling from a fixed-key blockcipher," in *Proc. IEEE Symp. Secur. Privacy*, May 2013, pp. 478–492.

[45] D. Demmler, T. Schneider, and M. Zohner, "ABY—A framework for efficient mixed-protocol secure two-party computation," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2015, pp. 1–15.

Xuanang Yang received the B.E. degree in software engineering from Wuhan University, China, in 2019, where he is currently pursuing the Ph.D. degree with the School of Cyber Science and Engineering. His current research interests include privacy-preserving machine learning and distributed machine learning.


Jing Chen (Senior Member, IEEE) received the Ph.D. degree in computer science from the Huazhong University of Science and Technology, Wuhan. He has been a Full Professor with Wuhan University since 2015. He has published more than 100 research papers in many international journals and conferences, such as IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE TRANSACTIONS ON COMPUTERS, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, USENIX Security, CCS, and INFOCOM. His current research interests include computer science, network security, and cloud security. He acts as a reviewer for many journals and conferences, such as IEEE TRANSACTIONS ON INFORMATION FORENSICS, IEEE TRANSACTIONS ON COMPUTERS, and IEEE/ACM TRANSACTIONS ON NETWORKING.

Kun He (Associate Member, IEEE) received the Ph.D. degree in computer science from the Computer School, Wuhan University. He is currently an Associate Professor with Wuhan University. He has published more than 30 research papers in many international journals and conferences, such as IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, IEEE TRANSACTIONS ON MOBILE COMPUTING, USENIX Security, CCS, and INFOCOM. His current research interests include cryptography, network security, mobile computing, and cloud computing.

Hao Bai received the B.E. degree in information security from Wuhan University, Wuhan, China, in 2022, where he is currently pursuing the M.S. degree with the School of Cyber Science and Engineering. His current research interests include distributed machine learning.

Cong Wu received the Ph.D. degree from the School of Cyber Science and Engineering, Wuhan University. His research outcomes have appeared in USENIX Security and ACM CCS. His current research interests include systems and mobile security.

Ruiying Du received the B.S., M.S., and Ph.D. degrees in computer science from Wuhan University, Wuhan, China, in 1987, 1994, and 2008, respectively. She is currently a Professor with the School of Cyber Science and Engineering, Wuhan University. She has published more than 80 research papers in many international journals and conferences, such as IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, *International Journal of Parallel and Distributed Systems*, INFOCOM, SECON, TrustCom, and NSS. Her current research interests include network security, wireless networks, cloud computing, and mobile computing.