Fregata: Fast Private Inference with Unified Secure Two-Party Protocols

Xuanang Yang, Jing Chen, Yuqing Li, Kun He, Xiaojie Huang, Zikuan Jiang, Hao Bai, and Ruiying Du

***Abstract*—Private Inference (PI) safeguards client and server privacy when the client utilizes the server's model to make predictions. Existing PI solutions for Convolutional Neural Networks (CNNs) employ distinct cryptographic primitives to customize secure two-party protocols for linear and non-linear layers. This requires data to be converted into a specific form to switch between protocols, thus leading to a significant increase in inference latency. In this paper, we present Fregata, a fast PI scheme for CNNs by leveraging identical cryptographic primitives to calculate both linear and nonlinear layers. Specifically, our protocols utilize homomorphic encryption to obtain additive secret shares of matrix products during the offline phase, followed by lightweight multiplication and addition operations on these shares in the latency-sensitive online phase. Benefiting from uniformity, we accelerate inference from a holistic perspective by decoupling certain procedures of our protocols and executing them asynchronously. Moreover, to improve the efficiency of the offline phase, we elaborate a homomorphic matrix multiplication calculation method with reduced computation and communication complexity compared to existing approaches. Furthermore, we minimize inference latency by employing graphics processing units to parallelize the operations on the shares during the online phase. Experimental evaluations on popular CNN models such as SqueezeNet, ResNet, and DenseNet demonstrate that Fregata reduces 35-45 times inference latency over the state-of-the-art counterparts, accompanied by a 1.6-2.8 times decrease in communication overhead. In terms of total runtime, Fregata maintains a reduction of approximately 3 times.**

***Index Terms*—Private inference, convolutional neural networks.**

I. Introduction

CONVOLUTIONAL Neural Networks (CNNs) have exhibited remarkable performance in various fields including medical diagnosis [1] and facial recognition [2]. Many cloud service providers, such as Amazon [3] and Alibaba [4], monetize their trained CNN models by allowing clients to query them for predictions [5]. However, plaintext prediction raises privacy concerns, which have garnered widespread attention [6]–[9]. For one thing, since clients' data may contain sensitive personal information, it should be kept secret from the service provider [10]–[12]. For another, models are the property of the service provider who invests substantial amounts of data and computing resources in training them, and thus should be hidden from clients [13]–[15].

To address privacy concerns, Private Inference (PI) [16]–[18] has been developed to safeguard clients' data from the service provider while ensuring that model parameters are kept secret from clients during inference. In particular, PI based on secure 2-Party Computation (2PC) has shown exceptional performance in CNN inference [19], [20]. Typically, CNN makes predictions by feeding input data to a pipeline of alternating linear and nonlinear layers, and those PI solutions customize 2PC protocols for linear and nonlinear layers respectively based on their characteristics. In linear layers, most of PI [19]–[22] pack multiple plaintexts into a single Homomorphic Encryption (HE) ciphertext and then perform linear function calculations on the packed ciphertexts. In nonlinear layers, recent works [20], [23], [24] utilize Garbled Circuits (GC) or Oblivious Transfer (OT) to calculate the comparison operation in a privacy-preserving way. High inference latency is a major headache for PI, and many efforts have been devoted to accelerating inference. Some solutions [23], [25] put HE operations of linear layer protocols into the query-independent offline phase to lighten online prediction. Another type of work speeds up linear or nonlinear layer protocols by approximating linear [24] or nonlinear [8], [10], [23] calculations of CNNs to fit cryptographic primitives, reducing the model accuracy.

However, prior works accelerate linear and nonlinear layer protocols separately since they are based on different cryptographic primitives. Considering the alternation of linear and nonlinear layers in CNNs, the data processed by the previous layer protocol is required to be converted into a certain form before it is fed into the next layer protocol, greatly slowing down the inference. Even for the state-of-the-art PI scheme Cheetah [20], predicting an image from the ImageNet dataset with a ResNet50 model takes at least tens of seconds.

A natural question is whether it is feasible to use the same cryptographic primitive for both linear and nonlinear layer protocols, in order to reduce conversion cost between these layers and speed up inference from a holistic perspective. However, it is a challenging work. On one hand, HE constitutes state-of-the-art linear layer protocols, but it cannot efficiently conduct the comparison operation of nonlinear layers. On the other hand, GC and OT perform well in nonlinear layers, but their communication overhead for linear functions (e.g., matrix multiplication) is very high.

In this paper, we present Fregata, a fast PI scheme for CNNs via unified 2PC protocols that employ the same cryptographic primitives to calculate both linear and nonlinear layers. Specif-

Xuanang Yang, Jing Chen, Kun He, Xiaojie Huang, Zikuan Jiang, Hao Bai, and Ruiying Du 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; sdhxj66@whu.edu.cn; jlumos@whu.edu.cn; bh0036@whu.edu.cn; duraying@whu.edu.cn).

Yuqing Li is 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, and also with Wuhan University Shenzhen Research Institute, Shenzhen, 518057, China (e-mail: li.yuqing@whu.edu.cn).

TABLE I
A HIGH-LEVEL COMPARISON OF PI. "NATIVE MODEL" INDICATES THAT THE PI DOES NOT MODIFY THE MODEL; "ONLINE" DESCRIBES THE CRYPTOGRAPHIC PRIMITIVES UTILIZED IN THE ONLINE PHASE; "LINEAR" AND "NON-LINEAR" DEPICTS THE TECHNIQUES USED TO IMPLEMENT PRIVACY-PRESERVING LINEAR AND NON-LINEAR CALCULATIONS RESPECTIVELY; 'DATASET" MEANS THE LARGEST SCALE DATASET TESTED BY THE SOLUTION. THE INVOLVED CRYPTOGRAPHIC TECHNIQUES INCLUDE ASS, GC, HE, AND OT.

Solution	Native Model	Online	Linear	Nonlinear	Dataset
ENSEI [26]	✓	GC, HE	HE	GC	CIFAR-10
FALCON [27]	✓	GC, HE	HE	GC	CIFAR-10
Delphi [23]	✗	ASS, GC	ASS, HE	GC	CIFAR-100
CrypTFlow2 [25]	✓	ASS, OT	ASS, HE/OT	OT	ImageNet
GALA [19]	✓	ASS, GC	ASS,HE	GC	ImageNet
COINN [24]	✗	GC, OT	OT	GC	ImageNet
Cheetah [20]	✓	ASS,OT	ASS, HE	OT	ImageNet
Fregata	✔	ASS	ASS, HE	ASS, HE	ImageNet

ically, both our linear and nonlinear layer protocols leverage HE to derive Additive Secret Shares (ASS) of matrix products in the offline phase, followed by lightweight multiplication and addition operations on these shares in the online phase. This stands in contrast to existing 2PC PI approaches that use heavyweight OT or GC during latency-sensitive online prediction. Remarkably, the uniformity of these protocols liberates Fregata from the data format conversion tools like ABY [28] that is widely utilized in PI based on 2PC. Moreover, based on the uniformity, we speed up inference from a holistic perspective by decoupling certain procedures of linear and nonlinear layer protocols and executing them asynchronously—a departure from the synchronous execution in existing approaches.

Furthermore, we accelerate both the offline and online phases in our protocols. In the offline phase, we propose a homomorphic matrix encryption scheme that supports matrix multiplication with lower computational and communication complexity than Cheetah. It makes better use of ciphertext space than Cheetah which pads too many zeros in the ciphertexts. In the online phase, we employ Graphics Processing Units (GPUs) to parallelize operations on shares, dramatically reducing inference latency. Our protocols faithfully implement the calculation without modification, thus retaining the model's original characteristics. Table I summarizes the features of Fregata compared to others. To the best of our knowledge, Fregata is the first PI that tailors linear and nonlinear layer protocols utilizing identical cryptographic primitives and accelerates inference from a holistic perspective.

Our main contributions are summarized as follows.

- For the first time, we propose Fregata, a fast private inference scheme for CNNs featuring unified protocols. The protocols utilize the same cryptographic primitive for both linear and nonlinear layers, reducing conversion costs between these layers and facilitating the acceleration of inference from a holistic perspective.
- We propose a homomorphic matrix encryption scheme that supports matrix multiplication with reduced computational and communication complexity than the state-of-the-art algorithm to expedite the offline phase. Moreover, we exploit GPUs to parallelize the operations of ASS in the online phase, significantly reducing inference latency.

TABLE II
MAJOR NOTATIONS.

Notation	Description
$M_{i,j}$	Element in the i-th row and j-th column of a matrix M
$\langle a\rangle_0, \langle a\rangle_1$	A pair of additive secret shares of a
$\langle M_{i,j}\rangle_0, \langle M_{i,j}\rangle_1$	A pair of additive secret shares of $M_{i,j}$
$\hat{a}$	A plaintext polynomial
$\hat{a}_i$	The i-th coefficient of $\hat{a}$
$[a]$	A RLWE ciphertext of $\hat{a}$
N	The capacity of HE ciphertext
$\circ$	Matrix Hadamard product

- We implement Fregata and evaluate it using prevalent models, such as ResNet50, on datasets of ImageNet scale. The experimental results indicate that Fregata reduces $35-45\times$ inference latency over the state-of-the-art counterparts [20], coupled with a $1.6-2.8\times$ reduction in communication cost. Regarding total runtime, Fregata remains a decrease of about $3\times$.

II. PRELIMINARIES

We introduce the prediction process of CNNs, cryptographic primitives used in Fregata, and an abstraction of PI based on 2PC. Major notations we use are summarized in Table II.

A. Convolutional Neural Networks

CNNs make predictions by feeding input data into a pipeline of layers [29]. The first layer is the input layer which receives the input data, and its output is the input to the second layer. Similarly, the output of layer i is the input of layer $i+1$. The last layer is the output layer whose output is the prediction.

The layers within CNNs can be categorized into two fundamental types based on their computational characteristics: linear and non-linear.

1) Linear Layers: Linear layers encompass the Fully Connected (FC) layer, convolutional layer, and Batch Normalization (BN) layer, which can be calculated by matrix multiplication.

FC Layer. This layer takes as input a vector X and calculates the matrix-vector product $Y = W \cdot X$ as the output, where W is the learnable parameter matrix.

Convolutional Layer. Given a 3-dimension input matrix X, the convolutional layer calculates and outputs the convolution of X and a set of learnable 3-dimension filters F, denoted as $Y = F \odot X$. By flattening X and F respectively into 2D matrices X' and F', the convolution can be calculated by the matrix multiplication $Y = F' \cdot X'$ [30].

BN Layer. The BN layer performs linear transformations on the FC or convolutional layer's output to adjust it to amenable ranges. During CNN prediction, the calculation of the BN layer can be fused into the FC or convolutional layer without affecting the prediction result [31]. That is to say, a convolution layer and a BN layer behind it can be calculated by a matrix multiplication.

2) Nonlinear Layers: Non-linear layers serve the function of modeling the non-linear associations between input and output, manifesting main features, and diminishing data dimensions. This category incorporates the activation layer and max-pooling layer. The fundamental operation of both the common activation layer (e.g., the rectified linear unit layer) and the max-pooling layer is the comparison operation.

B. Cryptographic Primitives

1) ASS: With 2-out-of-2 ASS, we can generate a pair of secrets $(\langle a\rangle_0, \langle a\rangle_1)$ for a value a and share them between two participants, where $a = \langle a\rangle_0 + \langle a\rangle_1$. With only one of the secrets, each participant cannot learn about a. Among the cryptographic primitives used in PI such as HE, OT, and GC, ASS is the lightest. Its computation cost is very close to that in plaintext.

2) HE: HE enables the ciphertext of $f(x)$ to be calculated from the ciphertext of x without decryption [32], [33]. Our solution utilizes the HE based on Ring Learning With Errors (RLWE), which has proven prominent performance in PI [20].

RLWE-based HE operates on polynomials within the cyclotomic ring $\mathbb{Z}[x]/(x^N+1)$, where N is a power of 2. Two crucial moduli are employed: the plaintext modulus p and the ciphertext modulus q, where $0 < p \ll q$. The private key of the scheme is a secret polynomial $sk = \hat{s} \in \mathbb{Z}_q[x]/(x^N+1)$. The corresponding public key is generated by computing $pk = (\hat{r}\cdot\hat{s} + \hat{e}, \hat{r})$, where $\hat{r} \in \mathbb{Z}_q[x]/(x^N+1)$ is a random polynomial and $\hat{e} \in \mathbb{Z}_q[x]/(x^N+1)$ is a noise polynomial with coefficients sampled from a discrete Gaussian distribution of standard deviation χ_σ. The encryption process transforms a message $\hat{a} \in \mathbb{Z}_p[x]/(x^N+1)$ into a tuple of polynomials: $[a] = (\lfloor \frac{q}{p}\hat{a}\rceil + \hat{e_1}, 0) - \hat{r_1}\cdot pk \pmod q$, where $\hat{e_1} \in \mathbb{Z}_q[x]/(x^N+1)$ is another noise polynomial with coefficients sampled from χ_σ and $\hat{r_1} \in \mathbb{Z}_q[x]/(x^N+1)$ is a random polynomial with coefficients chosen uniformly from$\{0, \pm 1\}$. This ciphertext can then undergo various homomorphic operations that translate to polynomial additions and multiplications within the ciphertext domain. After computation, a RLWE ciphertext $(\hat{c}, \hat{b})$ is decrypted as: $\lfloor \frac{p}{q}(\hat{c} + \hat{b}\cdot\hat{s})\rceil$.

Below, we abstract the functions within RLWE-based HE used in Fregata, where $\hat{a}_i$ denotes the i-th coefficient of a polynomial $\hat{a}$. For the sake of clarity, the subsequent descriptions assume that plaintext and ciphertext polynomials reside in the rings $\mathbb{Z}_p[x]/(x^N+1)$ and $\mathbb{Z}_q[x]/(x^N+1)$, respectively.

- `KeyGen`. Given a security parameter λ, this key generation algorithm outputs a pair of public-private keys (pk, sk) and is denoted as $(pk, sk) \leftarrow \texttt{KeyGen}(1^\lambda)$.
- `Enc`. This encryption algorithm takes as input a public key pk and a plaintext $\hat{a}$, and outputs a ciphertext $[a]$, denoted as $[a] \leftarrow \texttt{Enc}(pk, \hat{a})$.
- `Dec`. This decryption algorithm takes as input a private key sk and a ciphertext $[a]$, and outputs the plaintext $\hat{a}$, denoted as $\hat{a} \leftarrow \texttt{Dec}(sk, [a])$.
- `Add`. With the input of two ciphertexts $[a]$ and $[b]$, this addition operation returns a ciphertext of $\hat{a} + \hat{b}$, denoted as $[a+b] \leftarrow \texttt{Add}([a], [b])$.
- `Mul`. With the input of a ciphertext $[a]$ and a plaintext $\hat{b}$, this multiplication operation returns a ciphertext $[v]$, where $\hat{v}_i = \hat{a}_i \cdot \hat{b}_i$, denoted as $[a \circ b] \leftarrow \texttt{Mul}([a], \hat{b})$.
- `PoMul`. Given a plaintext $\hat{a}$ and a ciphertext $[b]$, this polynomial multiplication operation returns a ciphertext of $\hat{a}\cdot\hat{b}$, denoted as $[a\cdot b] \leftarrow \texttt{PoMul}(\hat{a}, [b])$.

C. PI Based on 2PC

PI safeguards client data from the service provider and conceals model parameters from the client when the client utilizes the service provider's model for predictions. PI based on 2PC has demonstrated superior performance. These approaches tailor protocols for linear and nonlinear layers utilizing appropriate cryptographic primitives, based on the characteristics of individual layers. In each protocol, the client and server additively share the input and output of the layer. So that they can link the protocols sequentially to complete the inference. The security guarantees of all the protocols ensure the overall security of the entire privacy-preserving inference, as protocols culminating in secure re-sharing of outputs are universally composable [21].

III. Overview

In this section, we introduce the system model and describe our scheme at a high level.

A. System Model

There are two types of entities in our system: server and client. The server utilizes the CNN model to provide prediction services to clients, and clients send their input to the server and obtain the prediction results of CNN.

1) Threat Model: Consistent with prior works (e.g., CrypTFlow2 [25] and Cheetah [20]), we target a semi-honest threat model, where both entities honestly abide by our protocol but are curious about the other entity's private data. Specifically, the server attempts to learn the input value and prediction result and the client is curious about model parameters.

2) Security Goals: We aim to keep the client's input value and prediction result secret from the server while guaranteeing the server's model parameters are hidden from the client during inference. Our security definition follows the standard ideal/real world paradigm, which requires an adversary cannot distinguish its view in the real world from the ideal world.

As in prior works including CrypTFlow2 and Cheetah, our ideal function does not protect model structures that can be

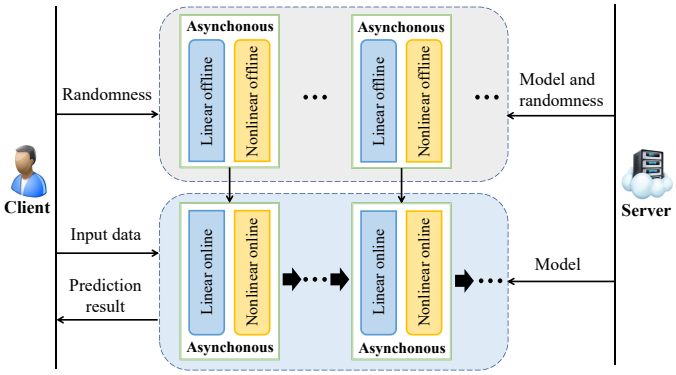


Fig. 1. The flow of our scheme, where the gray and blue segments denote the offline and online phases, respectively.

hidden by non-cryptographic techniques such as adding fake layers [22]. Also, the cryptographic technique cannot prevent attacks based purely on prediction results, such as model stealing attacks [20]. Orthogonal techniques (e.g., differential privacy [34], [35]) can be integrated into our solution to provide a stronger privacy guarantee.

B. Our Solution at a High Level

In our system, the client and server execute our 2PC linear and nonlinear layer protocols sequentially to complete the inference in a privacy-preserving manner. Current PI approaches adopt diverse cryptographic tools for linear and nonlinear layer protocols. This practice introduces frequent invocations of data format conversion tools like ABY and confines the acceleration of inference to the improvement of the respective linear and nonlinear layer protocols. In contrast, we craft unified protocols employing consistent cryptographic primitives. This approach not only eliminates the necessity for conversion tools but also provides the opportunity to accelerate inference from a holistic perspective. Benefiting from uniformity, we decouple specific procedures of our linear and nonlinear layer protocols, executing them asynchronously to expedite inference.

Each of our protocols is composed of an offline phase and an online phase. The offline phase is independent of the client's input and executed before the client makes queries. In this phase, the client and server collaboratively precompute some data from the model parameters and their random data. The online phase takes place when the client uploads data to the server for prediction, and its runtime determines the inference latency. Within this phase, the client and server perform lightweight addition and multiplication on the client's input and pre-computed data to derive the prediction result. Fig. 1 depicts the flow of our scheme.

IV. Unified 2PC Protocols for CNN Inference

We now detail our unified 2PC protocols for CNN inference. Given that linear layers can be computed through matrix multiplication, we devise a dedicated 2PC protocol specifically for matrix multiplication, elucidated in Section IV-A. For nonlinear layers, wherein the fundamental operation is comparison, we design a 2PC comparison protocol and explicate it in Section IV-B. In Section IV-C, we elaborate on

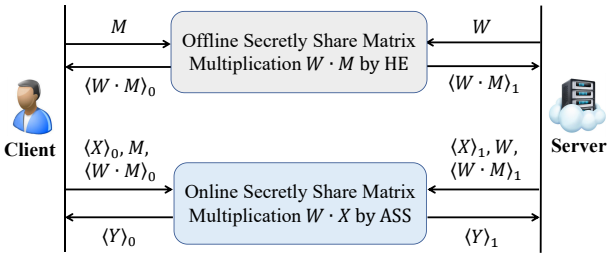


Fig. 2. Modular description of the offline and online phases of the `PMatMul` protocol.

expediting inference through the asynchronization of our linear and nonlinear layer protocols.

All our protocols follow the HE-SS offline-online paradigm. Specifically, the client and server utilize HE to additively share matrix products in the offline phase. In the online phase, they operate additions and multiplications on the shares to additively share the output of the layer.

In the following, we denote a 2PC protocol `Protocol` as $\{out_0; out_1\} \leftarrow \texttt{Protocol}\{in_0; in_1\}$, in which the client (resp. server) takes as input in_0 (resp. in_1) and obtains out_0 (resp. out_1). An algorithm `Algo` that takes as input in and outputs out is denoted as $out \leftarrow \texttt{Algo}(in)$. We use pk and sk to represent the public and private keys of the client. Specifically, the client computes $(pk, sk) \leftarrow \texttt{KeyGen}$ and sends pk to the server when joining the prediction service.

A. Linear Layer Protocol

1) Design Goal: We design `PMatMul`, a 2PC matrix multiplication protocol for linear layers. Specifically, the client and server additively share the $n \times k$ input matrix X, and the $m \times n$ parameter matrix W is held by the server. After executing our protocol, they additively share the $m \times k$ matrix multiplication $Y = W \cdot X$. Our protocol is denoted as $\{\langle Y \rangle_0; \langle Y \rangle_1\} \leftarrow \texttt{PMatMul}\{\langle X \rangle_0; W, \langle X \rangle_1\}$, where $X = \langle X \rangle_0 + \langle X \rangle_1$ and $Y = \langle Y \rangle_0 + \langle Y \rangle_1$.

Remarkably, we calculate the convolution through matrix multiplication. Conventionally, convolution operations entail sliding filters across an input feature map and calculating the inner product between the filter and local patches of the feature map. By representing the filters as rows of a matrix W and the patches as columns of another matrix X, convolution can be computed through $W \cdot X$ without altering the computational complexity [30]. This approach enables efficiency improvement through parallel acceleration, albeit at the potential cost of increased memory usage. In case the memory is not enough, we split the large matrices into submatrices and add the multiplication of corresponding submatrices to obtain the final result. This allows us to avoid running out of memory while still benefiting from parallel acceleration offered by matrix multiplication.

2) Protocol Flow: Fig. 2 is the modular description of the `PMatMul` protocol. In the offline phase, the client generates an $n \times k$ random matrix M and secretly shares $W \cdot M$ with the server through HE. In the online phase, the client and server operate lightweight addition and multiplication on the shares to additively share the output $Y = W \cdot X$.

We now describe the offline phase. The client first encrypts the matrix M and sends the ciphertext $[M]$ to the server, who homomorphically calculates the matrix multiplication $[W \cdot M]$. Then, the server generates an $m \times k$ random matrix R and uses it to mask $[W \cdot M]$ by adding $[W \cdot M]$ and $[-R]$ homomorphically. The server sends $[W \cdot M - R]$ to the client and sets $\langle W \cdot M \rangle_1 = R$. Finally, the client decrypts to obtain $\langle W \cdot M \rangle_0 = W \cdot M - R$. It satisfies $\langle W \cdot M \rangle_0 + \langle W \cdot M \rangle_1 = W \cdot M$.

Then, we describe the online phase. The client computes and sends $\langle X \rangle_0 - M$ to the server. The server then simply calculates $X - M = (\langle X \rangle_0 - M) + \langle X \rangle_1$ and $\langle Y \rangle_1 = W \cdot (X - M) + \langle W \cdot M \rangle_1$. The client finally sets $\langle Y \rangle_0 = \langle W \cdot M \rangle_0$. It satisfies $\langle Y \rangle_0 + \langle Y \rangle_1 = W \cdot X = Y$. The online phase only involves lightweight addition and multiplication on the shares.

3) Homomorphic Matrix Multiplication: A long line of PI [16], [20]–[22], [36] focuses on the homomorphic matrix multiplication methods, which can be applied in the offline phase of `PMatMul`. Specifically, these methodologies craft encoding techniques to encode multiple elements of the matrix into a single ciphertext, and design algorithms to minimize HE operations and ciphertext transmissions. The state-of-the-art approach, Cheetah [20], encodes elements of two matrices into two respective polynomials and calculates homomorphic matrix multiplication through homomorphic polynomial multiplication. Nevertheless, this algorithm can only compute the multiplication of a multidimensional and a one-dimensional matrix through a single polynomial multiplication. Calculating the multiplication of multidimensional matrices requires repeated applications of Cheetah, resulting in complexity scaling linearly with the matrix dimension k. To overcome this limitation and improve efficiency, we devise a Homomorphic Matrix Encryption (HME) scheme that supports direct computation of multidimensional matrix multiplication through a single polynomial multiplication. HME leverages the ciphertext space more efficiently by computing multiplications of matrices in as many dimensions as possible within a single polynomial multiplication, reducing the number of HE operations and ciphertext transmissions compared to existing methods.

HME Construction. We utilize the RLWE-based HE to construct our homomorphic matrix encryption scheme. It consists of five algorithms: `Gen`, `EncMat`, `DecMat`, `AddMat`, and `MulMat`. Fig. 3 depicts an example of using HME for matrix multiplication, where $m = 3$, $n = 2$, $k = 2$, $N = 16$, and $p = 2^8$.

$\texttt{Gen}(1^\lambda)$. The key generation algorithm `Gen` takes as input a security parameter λ and outputs $(pk, sk) \leftarrow \texttt{KeyGen}(1^\lambda)$.

$\texttt{EncMat}(pk, M)$. Given a public key pk and an $n \times k$ plaintext matrix M, the matrix encryption algorithm `EncMat` encodes M into a polynomial $\hat{M}$, where $\hat{M}_{n \cdot k - j \cdot n - i - 1} \equiv M_{i,j} \mod p$ $(0 \le i \le n-1, 0 \le j \le k-1)$, and sets other coefficients of $\hat{M}$ to 0. Finally, it outputs $[M] \leftarrow \texttt{Enc}(pk, \hat{M})$.

$\texttt{DecMat}(sk, [P], m, n, k)$. The matrix decryption algorithm `DecMat` takes as input a private key sk, a ciphertext polynomial $[P]$, and three integer parameters m, n, k related to matrix dimensions. It performs $\hat{P} \leftarrow \texttt{Dec}(sk, [P])$ and outputs a matrix Y, where $Y_{i,j} = \hat{P}_{i \cdot n \cdot k - j \cdot n + k \cdot n - 1}$ $(0 \le i \le m-1, 0 \le j \le k-1)$.

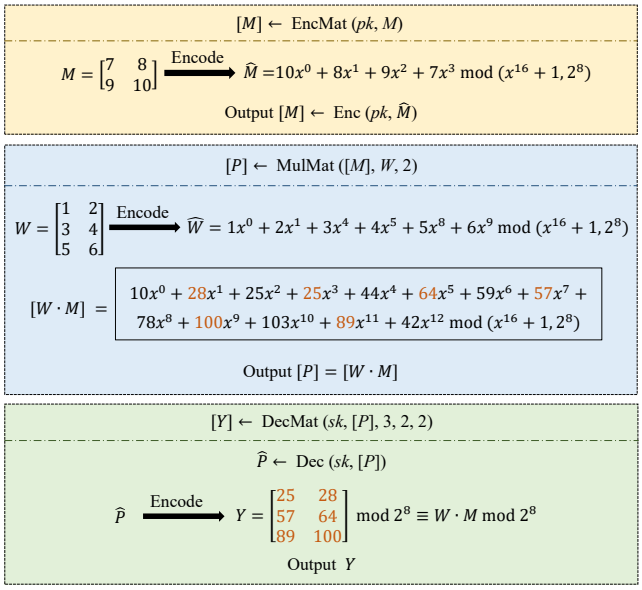


Fig. 3. An example of using HME for matrix multiplication, where the box on "$10x^0 + 28x^1 + ... + 42x^{12}$" represents the ciphertext of the polynomial "$10x^0 + 28x^1 + ... + 42x^{12}$".

$\texttt{AddMat}(pk, [P], R)$. With the input of a public key pk, a ciphertext polynomial $[P]$, and an $m \times k$ plaintext matrix R, the matrix addition algorithm `AddMat` first encodes R into a polynomial $\hat{R}$ of the same degree as $\hat{P}$, where $\hat{R}_{i \cdot n \cdot k - j \cdot n + k \cdot n - 1} \equiv R_{i,j} \mod p$ $(0 \le i \le m-1, 0 \le j \le k-1)$, and set other coefficients of $\hat{R}$ to random values. Then, it encrypts $[R] \leftarrow \texttt{Enc}(pk, \hat{R})$. Finally, it outputs $[P + R] \leftarrow \texttt{Add}([P], [R])$.

$\texttt{MulMat}([M], W, k)$. The matrix multiplication algorithm `MulMat` takes as input a ciphertext polynomial $[M]$, an $m \times n$ plaintext matrix W, and an integer parameter k related to the matrix dimension. It encodes W into a polynomial $\hat{W}$, where $\hat{W}_{i \cdot n \cdot k + j} \equiv W_{i,j} \mod p$ $(0 \le i \le m-1, 0 \le j \le n-1)$, and sets other coefficients of $\hat{W}$ to 0. Finally, it outputs $[P] \leftarrow \texttt{PoMul}(\hat{W}, [M])$.

Principle of HME. HME implements matrix multiplication by encoding matrix elements into polynomials and performing polynomial multiplication homomorphically. In this process, matrix multiplication $W \cdot M$ is achieved by computing the inner product of each row of W and each column of M. To compute the inner product, HME encodes a row of W and a column of M into continuous coefficients of two respective polynomials. It then calculates the product of these polynomials and extracts the inner product from the middle coefficient of the resulting polynomial. To calculate multiple inner products through a polynomial multiplication, HME encodes different rows of W into different segments of polynomial coefficients, ensuring sufficient degree space in each segment for inner product computation. We give the homomorphic proof of HME in the Appendix.

Complexity Comparison. Table III shows the computation and communication complexity comparison of our HME and other methods. `Perm` denotes the homomorphic operation that can permute elements in a packed ciphertext. Its computational overhead is much higher than `Add` and `Mul`, and we suc-

TABLE III
COMPLEXITY COMPARISON OF HOMOMORPHIC MATRIX MULTIPLICATION CALCULATION METHODS.

Method	`Add`	`Mul`	`Perm`	Communication
Gazelle	$log(\lceil \frac{N}{m} \rceil) \cdot k + \lceil \frac{m \cdot n}{N} \rceil \cdot k$	$\lceil \frac{m \cdot n}{N} \rceil \cdot k$	$log(\lceil \frac{N}{m} \rceil) \cdot k + \lceil \frac{m \cdot n}{N} \rceil \cdot k - k$	$\lceil \frac{m}{N} \rceil \cdot k$
Cheetah	$\lceil \frac{m \cdot n}{N} \rceil \cdot k$	$\lceil \frac{m \cdot n}{N} \rceil \cdot k$	0	$\lceil \frac{m \cdot n}{N} \rceil \cdot k$
Fregata	$\lceil \frac{m \cdot n \cdot k}{N} \rceil$	$\lceil \frac{m \cdot n \cdot k}{N} \rceil$	0	$\lceil \frac{m \cdot n \cdot k}{N} \rceil$

Protocol 1 `PMatMul`

Input: The client inputs an $n \times k$ matrix $\langle X \rangle_0$. The server inputs an $m \times n$ matrix W and an $n \times k$ matrix $\langle X \rangle_1$.

Output: The client and server outputs matrices $\langle Y \rangle_0$ and $\langle Y \rangle_1$ of dimensions $m \times k$ respectively.

Offline phase:

1: The client generates an $n \times k$ random matrix M and computes $[M] \leftarrow \texttt{EncMat}(pk, M)$. Then, the client sends $[M]$ to the server.
2: The server generates a $m \times k$ random matrix R and sets $\langle W \cdot M \rangle_1 = R$. Then, the server calculates $[W \cdot M - R] \leftarrow \texttt{AddMat}(pk, \texttt{MulMat}([M], W, k), -R)$ and sends it to the client.
3: The client decrypts to get $\langle W \cdot M \rangle_0 \leftarrow \texttt{DecMat}(sk, [W \cdot M - R], m, n, k)$.

Online phase:

1: The client computes and sends $V = \langle X \rangle_0 - M$ to the server and sets $\langle Y \rangle_0 = \langle W \cdot M \rangle_0$.
2: The server calculates $X - M = V + \langle X \rangle_1$ and $\langle Y \rangle_1 = W \cdot (X - M) + \langle W \cdot M \rangle_1$.

cessfully avoid it. Compared with Cheetah, our computation and communication complexity $\lceil \frac{m \cdot n \cdot k}{N} \rceil$ is usually lower than $\lceil \frac{m \cdot n}{N} \rceil \cdot k$ in practice since the ciphertext capacity N is greater than $m \cdot n$ in most cases of PI. Similarly, our communication complexity is typically not higher than that of Gazelle. This is because our algorithm makes better use of ciphertext space.

We utilize HME to construct the offline phase of our `PMatMul` protocol, and the entire flow of `PMatMul` is described in Protocol 1.

B. Nonlinear Layer Protocol

1) Design Goal: The fundamental operation of nonlinear layers is identifying the maximum value. Toward this, we devise a 2PC protocol that can compare the max value in a privacy-preserving way.

In the protocol, the client and server start by additively sharing input $m \times k$ matrices E and F. After executing the protocol, they obtain $\langle Y \rangle_0$ and $\langle Y \rangle_1$ respectively, where $\langle Y_{i,j} \rangle_b = \texttt{max}(\langle E_{i,j} \rangle_b, \langle F_{i,j} \rangle_b)$ $(i \in [0, m-1], j \in [0, k-1]$, and $b \in [0, 1])$. Our 2PC comparison protocol is denoted as $\{\langle Y \rangle_0; \langle Y \rangle_1\} \leftarrow \texttt{PMax}\{\langle E \rangle_0, \langle F \rangle_0; \langle E \rangle_1, \langle F \rangle_1\}$. Notably, to improve efficiency, `PMax` parallelize the element-wise comparison between E and F by packing multiple elements into a single ciphertext in the offline phase and employ GPU in the online phase.

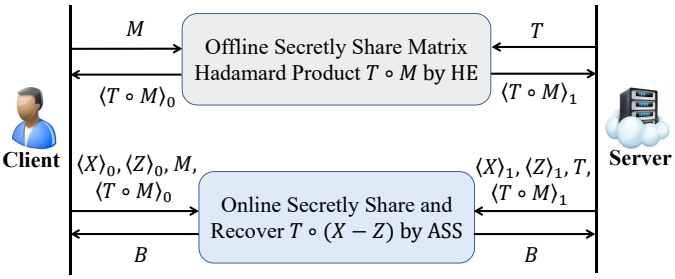


Fig. 4. Modular description of the offline and online phases of the `PMax` protocol.

Many prior works [8], [20]–[23], [25] make efforts to lessen the computation and communication cost of the comparison protocol. However, they either involve expensive cryptographic primitives such as GC and OT in their latency-sensitive online prediction or adopt approximation that reduces the model accuracy. In contrast, our protocol only utilizes the lightest ASS in the online phase and faithfully implements the comparison without approximation.

2) Protocol Flow: The `PMax` protocol also follows the HE-SS offline-online paradigm. The details of `PMax` are shown in Fig. 4. Specifically, the client and server first additively share $E - F$ by calculating $\langle E - F \rangle_0 = \langle E \rangle_0 - \langle F \rangle_0$ and $\langle E - F \rangle_1 = \langle E \rangle_1 - \langle F \rangle_1$, respectively. Then, the server generates a random matrix T of dimensions $m \times k$, where each element is a positive value. And then, the client interacts with the server to additively share $T \circ (E - F)$. This procedure is similar to the mechanism of sharing $W \cdot X$ within the `PMatMul` protocol, and the sole distinction lies in homomorphically calculating matrix Hadamard product rather than matrix multiplication during the offline phase. Finally, the client and server ascertain shares of the max value through the Hadamard product. Protocol 2 details the `PMax` protocol, where the $\bmod p$ operation is omitted in encoding matrix elements into polynomial coefficients for clarity.

C. Asynchronization of Linear and Nonlinear Layer Protocols

The disparity of linear and nonlinear layer protocols within existing PI schemes necessitates the conversion of the output from the former protocol to a specific format before feeding it into the subsequent protocol. This approach introduces conversion cost and mandates the synchronous execution of protocols. `PMatMul` and `PMax` bridge the gap by elaborating both in the HE-SS offline-online paradigm, eliminating the need for conversion tools. To further expedite inference, we execute certain procedures of `PMatMul` and `PMax` in asynchronous modalities. Specifically, specific procedures of `PMax`

Protocol 2 PMax

Input: The client inputs matrices $\langle E\rangle_0$ and $\langle F\rangle_0$ of dimensions $m \times k$. The server inputs two $m \times k$ matrices $\langle E\rangle_1$ and $\langle F\rangle_1$.

Output: The client and server output $\langle Y\rangle_0$ and $\langle Y\rangle_1$ of dimensions $m \times k$ respectively.

Offline phase:

1: The client generates an $m \times k$ random matrix M' and encodes it into a polynomial $\hat{M}'$ in raster scan fashion. Then, the client computes $[M'] \leftarrow \texttt{Enc}(pk, \hat{M}')$ and sends it to the server.

2: The server generates two $m \times k$ random matrices R' and T, where all elements of T are positive. Then, the server encodes them into polynomials $\hat{R}'$ and $\hat{T}$ in raster scan fashion respectively. And then, the server calculates $[-R'] \leftarrow \texttt{Enc}(pk, -\hat{R}')$ and $[T \circ M' - R'] \leftarrow \texttt{Add}(\texttt{Mul}([M'], \hat{T}), [-R'])$. Finally, the server sends $[T \circ M' - R']$ to the client and sets $\langle T \circ M'\rangle_1 = R'$.

3: The client decrypts to get $\hat{T} \circ \hat{M}' - \hat{R}' \leftarrow \texttt{Dec}(sk, [T \circ M' - R'])$ and encodes coefficients of $\hat{T} \circ \hat{M}' - \hat{R}'$ into an $m \times k$ matrix $\langle T \circ M'\rangle_0$ in raster scan fashion.

Online phase:

1: The client computes $\langle E - F\rangle_0 = \langle E\rangle_0 - \langle F\rangle_0$ and sends $V' = \langle E - F\rangle_0 - M'$ to the server.

2: The server calculates $E - F - M' = \langle E\rangle_1 - \langle F\rangle_1 + V'$ and $U = T \circ (E - F - M') + \langle T \circ M'\rangle_1$, and then sends U to the client.

3: The client computes $T \circ (E - F) = U + \langle T \circ M'\rangle_0$ and transmits an $m \times k$ matrix G to the server, wherein $G_{i,j}$ is a randomly generated positive value if $(T \circ (E - F))_{i,j}$ is positive, and a non-positive random value otherwise.

4: The client and server outputs $\langle Y\rangle_b$ ($b \in \{0, 1\}$) respectively, where $\langle Y_{i,j}\rangle_b = \langle E_{i,j}\rangle_b$ if $G_{i,j}$ is positive and $\langle Y_{i,j}\rangle_b = \langle F_{i,j}\rangle_b$ otherwise.

can proceed without waiting for the data of the PMatMul protocol, enabling simultaneous execution of specific procedures of PMatMul and PMax. For instance, the client can encrypt M' while transmitting $[M]$. The comparison between synchronous and asynchronous execution of PMatMul and PMax is illustrated in Fig. 5. Notably, the upper and lower portions of Fig. 5a depict the processes of the PMatMul and PMax protocols, respectively.

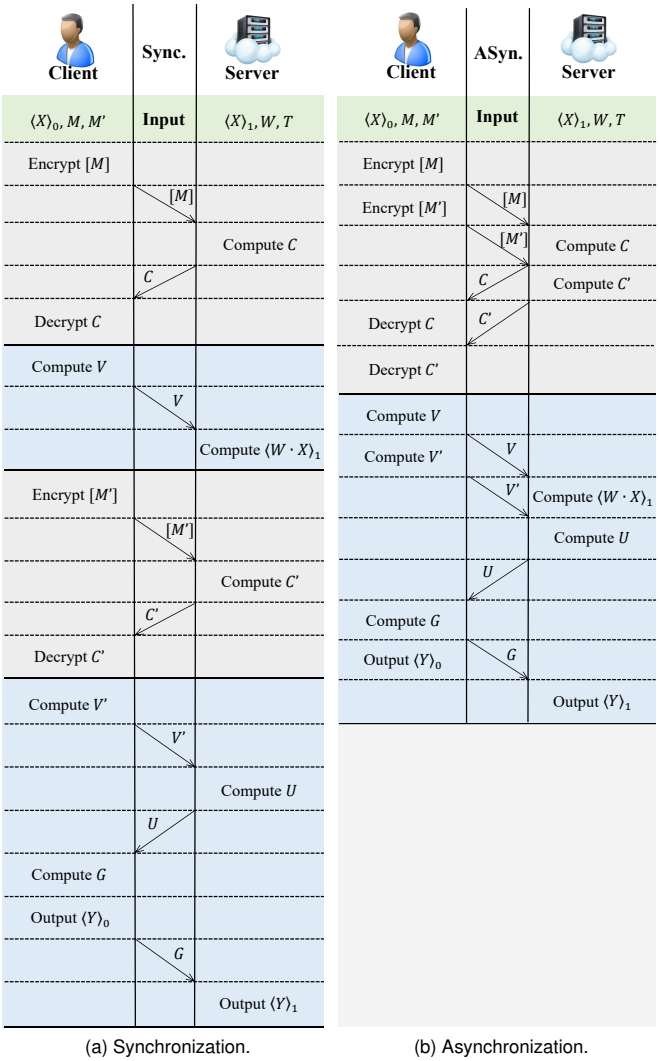


Fig. 5. The comparison between synchronous and asynchronous execution of PMatMul and PMax, with the gray and blue segments denoting the offline and online phases, respectively. The upper and lower portions of subfigure (a) describe the flows of the PMatMul and PMax protocols, respectively.

V. Security Analysis

We now prove the security of the PMatMul and PMax protocols proposed in Section IV.

A. Security of PMatMul Protocol

The security of the PMatMul protocol can be reduced to the security of the matrix multiplication in the offline phase as shown in Theorem 1. We define the ideal matrix multiplication functionality as $\{\langle W \cdot M\rangle_0; \langle W \cdot M\rangle_1\} \leftarrow \texttt{IMMul}\{M; W\}$, where $\langle W \cdot M\rangle_0 + \langle W \cdot M\rangle_1 = W \cdot M$.

Theorem 1. *The offline phase of the* PMatMul *protocol implements* IMMul *in the presence of semi-honest adversaries if the underlying RLWE-based HE scheme is semantically secure.*

Proof: We first prove the security against a semi-honest server. The server's view in an execution of PMatMul's offline phase π is $\text{view}_0^\pi(W, M) = \{W, r_S, [M]\}$, where W is the server's input, r_S is the server's randomness, and $[M]$ is the received message. We construct the simulator $Sim_0(W, \texttt{IMMul}\{M; W\})$ as follows. (1) Choose a random tape r_S^*. (2) Initialize a matrix M^* of the same size as M and choose each element in the matrix according to the distribution. (3) Compute $[M^*] \leftarrow \texttt{EncMat}(pk, M^*)$. (4) Output $(W, r_S^*, [M^*])$. If the RLWE-based HE is semantically secure, we can prove that $\{Sim_0(W, \texttt{IMMul}\{M; W\})\}_{W,M\in\{0,1\}^*} \overset{c}{\equiv} \{\text{view}_0^\pi(W, M)\}_{W,M\in\{0,1\}^*}$ by a hybrid argument and complete the proof against a semi-honest server.

We next prove the security against a semi-honest client. The view of the client in an execution of PMatMul's offline phase is $\text{view}_1^\pi(W, M) = \{M, r_C, [W \cdot M - R]\}$,

where M is the client's input, r_C is the client's randomness, and $[W \cdot M - R]$ is the received message. The simulator $Sim_1(M, \texttt{IMMul}\{M; W\})$ is constructed as follows. (1) Choose a random tape r_C^*. (2) Initialize matrix W^* of the same size as W and R^* of the same size as R and choose each element in the vectors according to the distribution. (3) Compute $[W^* \cdot M - R^*]$ from W^*, M, and R^*. (4) Output $(M, r_C^*, [W^* \cdot M - R^*])$. If the RLWE-based HE is semantically secure, we can prove that $\{Sim_1(M, \texttt{IMMul}\{M; W\})\}_{W,M\in\{0,1\}^*} \stackrel{c}{\equiv} \{\text{view}_1^\pi(W, M)\}_{W,M\in\{0,1\}^*}$ by a hybrid argument and complete the proof against semi-honest the client. ∎

B. Security of PMax Protocol

The security of the PMax protocol can be reduced to the security of the matrix Hadamard product in the offline phase as shown in Theorem 2. We define the ideal matrix Hadamard product functionality as $\{\langle T \circ M\rangle_0; \langle T \circ M\rangle_1\} \leftarrow \texttt{IMHP}\{M; T\}$, where $\langle T \circ M\rangle_0 + \langle T \circ M\rangle_1 = T \circ M$.

Theorem 2. *The offline phase of the* PMax *protocol implements* IMHP *in the presence of semi-honest adversaries if the underlying RLWE-based HE scheme is semantically secure.*

Proof: We first prove the security against a semi-honest server. The server's view in an execution of PMax's offline phase π is $\text{view}_0^\pi(T, M') = \{T, r_S, [M']\}$, where T is the server's input, r_S is the server's randomness, and $[M']$ is the received message. The simulator $Sim_0(T, \texttt{IMHP}\{M'; T\})$ is constructed as follows. (1) Choose a random tape r_S^*. (2) Initialize a matrix M^* of the same size as M' and choose each element in the matrix according to the distribution. (3) Encode M^* into a polynomial $\hat{M}^*$ in raster scan fashion and compute $[M^*] \leftarrow \texttt{Enc}(pk, \hat{M}^*)$. (4) Output $(T, r_S^*, [M^*])$. Since the RLWE-based HE is semantically secure, we can prove that $\{Sim_0(T, \texttt{IMHP}\{M'; T\})\}_{T,M'\in\{0,1\}^*} \stackrel{c}{\equiv} \{\text{view}_0^\pi(T, M')\}_{M'\in\{0,1\}^*}$ by a hybrid argument and complete the proof against a semi-honest server.

In the following, we prove the security against a semi-honest client. The view of the client in an execution of PMax's offline phase is $\text{view}_1^\pi(T, M') = \{M', r_C, [T \circ M' - R']\}$, where M' is the client's input, r_C is the client's randomness, and $[T \circ M' - R']$ is the received message. The simulator $Sim_1(M', \texttt{IMHP}\{M'; T\})$ is constructed as follows. (1) Choose a random tape r_C^*. (2) Initialize vectors T^* and R^* of the same size as T and R', and choose each element in the vectors according to the distribution. (3) Compute $[T^* \circ M' - R^*]$ from T^*, M' and R^*. (4) Output $(M', r_C^*, [T^* \circ M' - R^*])$. Since the RLWE-based HE is semantically secure, we can prove that $\{Sim_1(M', \texttt{IMHP}\{M'; T\})\}_{T,M'\in\{0,1\}^*} \stackrel{c}{\equiv} \{\text{view}_1^\pi(T, M')\}_{T,M'\in\{0,1\}^*}$ by a hybrid argument and complete the proof against a semi-honest the client. ∎

VI. Evaluation

We conduct experiments on Fregata and compare it with state-of-the-art PI frameworks.

A. Evaluation Setup

Implementation. We implement Fregata on top of the SEAL library [37]. We set the SEAL parameters consistent with Cheetah: $N = 4096, q \approx 2^{102}, p = 2^{37}, \sigma = 3.2$, where σ signifies the standard deviation of discrete Gaussian distribution.

The online phases of both PMatMul and PMax involve only elementary operations: addition and multiplication of additive secret shares. The inherent simplicity of these operations makes them highly amenable to GPU acceleration. To reduce inference latency, we utilize GPUs to parallelize these operations, programming them in CUDA C++. Specifically, we craft kernel functions to transform the originally serial loops for matrix multiplication and addition of secret shares into parallel operations executable on the GPU. Considering that CUDA C++ does not natively support modular arithmetic, we set an integer type value as the modulus and perform modular operations after multiplication or addition, which is also written in kernel functions for parallel execution. This approach significantly improves performance compared to serial execution on the CPU. Furthermore, we optimize memory management and kernel launches by meticulously selecting grid and block sizes, ensuring efficient workload distribution across the GPU.

Baselines. To compare with state-of-the-art PI solutions, we use the source code of Gazelle [22], CrypTFlow2 [25], and Cheetah [20], and reimplement GALA [19] based on Gazelle. For fairness, all solutions use the same version of SEAL and apply HEXL acceleration.

Testbed Environment. We run the client-side program on a desktop (Intel Core i7 CPU with 6 3.19 GHz cores and 15.8GB memory) and the server-side program on a server (Intel Xeon with 14 2.40 GHz cores, 128GB memory, and NVIDIA 4090 GPU). Both machines use Ubuntu 18.04 and run on a LAN network with a bandwidth of 460 MBps.

Datasets and CNN Architectures. We conduct a comparative analysis with other counterparts across a diverse set of CNN architectures, encompassing SqueezeNet, ResNet (characterized by over 23 million parameters), and DenseNet. This evaluation spans classical datasets such as CIFAR and ImageNet.

B. Microbenchmarks

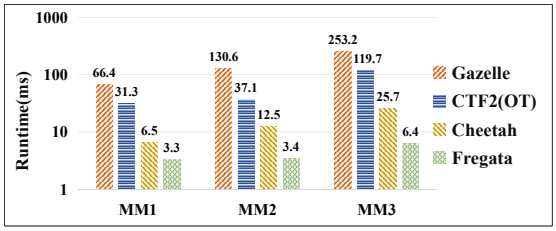
1) Evaluation on Matrix Multiplications: To validate the effectiveness of HME, we conduct a comparative evaluation with prevalent methods across three matrix multiplications with the following dimensions (m, n, k): $(64, 32, 32)$, $(16, 64, 64)$, $(32, 32, 128)$.

Table IV details the computation and communication complexity of these methods. Fregata avoids the most expensive Perm operation. Additionally, it incurs fewer HE addition and multiplication, and reduces ciphertext transmission compared to existing methods. This enhancement stems from our HME, which maximizes the utilization of ciphertext space.

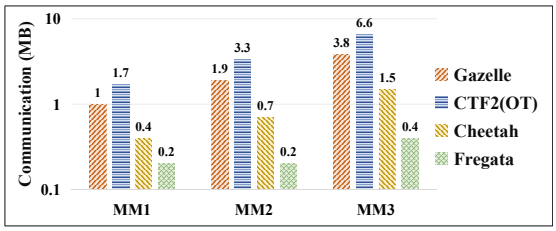
The comparison of practical runtime and communication overhead is illustrated in Fig. 6a and Fig. 6b respectively, where CTF(OT) represents an advanced matrix multiplication

TABLE IV
COMPUTATION AND COMMUNICATION COMPLEXITY COMPARISON OF MATRIX MULTIPLICATION.

Method	MM1: $(m \times n \times k)$:$(64 \times 32 \times 32)$			
	`Add`	`Mul`	`Perm`	Communication
Gazelle	224	32	192	32
Cheetah	32	32	**0**	32
Fregata	**16**	**16**	**0**	**16**
Method	MM2: $(m \times n \times k)$:$(16 \times 64 \times 64)$			
	`Add`	`Mul`	`Perm`	Communication
Gazelle	576	64	512	64
Cheetah	64	64	**0**	64
Fregata	**16**	**16**	**0**	**16**
Method	MM3: $(m \times n \times k)$:$(32 \times 32 \times 128)$			
	`Add`	`Mul`	`Perm`	Communication
Gazelle	1024	128	896	128
Cheetah	128	128	**0**	128
Fregata	**32**	**32**	**0**	**32**



(a) Runtime.



(b) Communication.

Fig. 6. Performance comparison of matrix multiplication.

TABLE V
PERFORMANCE COMPARISON OF FC LAYERS.

	Protocol	Runtime(ms)			Communication(KB)		
		Online	Offline	Total	Online	Offline	Total
FC1	Fregata	**0.02**	0.33	**0.35**	0.1	**11.4**	**11.6**
	Cheetah	0.1	**0.31**	0.41	0.1	**11.4**	**11.6**
	CrypTFlow2	0.11	1.06	1.17	0.1	51.7	51.8
FC2	Fregata	**0.07**	**0.58**	**0.65**	0.5	**22.9**	**23.4**
	Cheetah	0.47	0.67	1.14	0.5	**22.9**	**23.4**
	CrypTFlow2	0.46	3.93	4.39	0.5	103.5	103.9
FC3	Fregata	**0.92**	**6.58**	**7.51**	7.8	**366.1**	**373.9**
	Cheetah	4.64	6.65	11.29	7.8	**366.1**	**373.9**
	CrypTFlow2	4.58	58.37	62.95	7.8	1655.2	1663.1

calculation method based on OT [25]. Fregata demonstrates an approximate $2-4\times$ enhancement over Cheetah, coupled with a reduction in communication cost by $2-4\times$, attributable to the superior utilization of ciphertext space.

2) Evaluation on Linear Layers: We evaluate the performance of our `PMatMul` protocol against the linear protocols within CrypTFlow2 and Cheetah on both FC and convolutional layers. For FC layers, we used parameter matrices of dimensions 128×16, 64×128, and 1024×128, referred to as FC1 to FC3. For convolutional layers, the input feature maps' dimensions, filter counts, filter sizes, and stride lengths are set to $(16 \times 16 \times 32, 32, 1 \times 1, 1)$, $(32 \times 32 \times 64, 32, 5 \times 5, 2)$, and $(64 \times 64 \times 128, 128, 3 \times 3, 2)$, denoted as Conv1 to Conv3. Since Cheetah's and CrypTFlow2's linear protocols can be extended to the online-offline paradigm like `PMatMul`, we compare the online-offline versions of these protocols for a fair comparison.

TABLE VI
PERFORMANCE COMPARISON OF CONVOLUTIONAL LAYERS.

	Protocol	Runtime(ms)			Communication(MB)		
		Online	Offline	Total	Online	Offline	Total
Conv1	Fregata	**1.2**	**7.3**	**8.5**	0.02	**0.55**	**0.57**
	Cheetah	8.9	13.7	22.6	0.02	0.73	0.75
	CrypTFlow2	8.8	96.1	104.9	0.02	3.31	3.33
Conv2	Fregata	**13.8**	**21.7**	**35.5**	0.13	**1.46**	**1.59**
	Cheetah	69.5	105.7	175.2	0.13	5.86	5.98
	CrypTFlow2	70.7	873.8	944.5	0.13	26.48	26.61
Conv3	Fregata	**289.4**	**694.5**	**983.9**	4.02	**46.86**	**50.87**
	Cheetah	2231.8	3271.6	5503.4	4.02	187.43	191.45
	CrypTFlow2	2243.1	27169.4	29412.5	4.02	847.46	851.48

TABLE VII
PERFORMANCE COMPARISON OF ACTIVATION LAYERS.

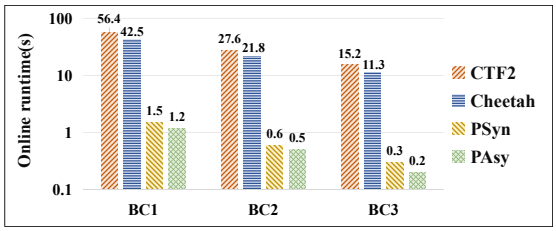
Network	Protocol	Runtime(S)			Communication(GB)		
		Online	Offline	Total	Online	Offline	Total
SqueezeNet	Fregata	**0.1**	1.3	**1.4**	**0.05**	0.12	0.17
	Cheetah	17.9	**0**	17.9	0.13	**0**	**0.13**
	CrypTFlow2	18.8	**0**	18.8	2.22	**0**	2.22
ResNet50	Fregata	**0.3**	7.2	**7.5**	**0.19**	0.57	0.76
	Cheetah	71.8	**0**	71.8	0.59	**0**	**0.59**
	CrypTFlow2	80.4	**0**	80.4	9.66	**0**	9.66
DenseNet121	Fregata	**0.5**	14.6	**15.1**	**0.28**	1.00	1.28
	Cheetah	126.2	**0**	126.2	0.99	**0**	**0.99**
	CrypTFlow2	134.9	**0**	134.9	16.18	**0**	16.18

The detailed performance comparisons for FC and convolutional layers are presented in Table V and Table VI, respectively. Fregata achieves significant latency reductions $(4.5-7.7\times)$ compared to Cheetah in the online phase due to efficient GPU parallelization. During the offline phase, Fregata exhibits comparable runtime and communication costs to Cheetah for FC layers due to their shared underlying complexity. However, for convolutional layers, Fregata demonstrates substantial improvements in both runtime $(1.9-4.9\times)$ and communication cost $(1.3-4\times)$ compared to Cheetah. This stems from the superior ciphertext space utilization offered by HME. This benefit is particularly pronounced in strided convolutions, where Cheetah's packing method has to calculate unnecessary inner products, whereas HME only computes the essential ones through matrix multiplication.

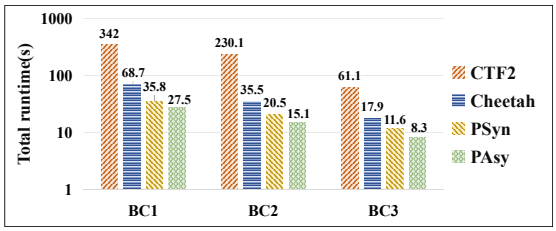
3) Evaluation on Nonlinear Layers: To validate the performance of the `PMax` protocol, we evaluate it and the comparison protocols within CrypTFlow2 and Cheetah in terms of runtime and communication overhead. Specifically, we invoke these protocols to calculate ReLU and max-pooling layers within the architectures of SqueezeNet, ResNet50, and DenseNet121.

TABLE VIII
PERFORMANCE COMPARISON OF MAX-POOLING LAYERS.

Network	Protocol	Runtime(S)			Communication(GB)		
		Online	Offline	Total	Online	Offline	Total
SqueezeNet	Fregata	**0.1**	1.7	**1.8**	**0.05**	0.18	0.23
	Cheetah	14.7	**0**	14.7	0.17	**0**	**0.17**
	CrypTFlow2	21.8	**0**	21.8	2.94	**0**	2.94
ResNet50	Fregata	**0.1**	1.1	**1.2**	**0.02**	0.12	0.14
	Cheetah	7.4	**0**	7.4	0.1	**0**	**0.1**
	CrypTFlow2	13.7	**0**	13.7	1.74	**0**	1.74
DenseNet121	Fregata	**0.1**	1.0	**1.1**	**0.02**	0.11	0.13
	Cheetah	7.8	**0**	7.8	0.1	**0**	**0.1**
	CrypTFlow2	13.8	**0**	13.8	1.74	**0**	1.74



(a) Online runtime.



(b) Total runtime.

Fig. 7. Performance comparison of linear and nonlinear layers. CTF2 refers to CrypTFlow2 [25].

The performance comparison of activation layers are summarized in Table VII. In the online phase, Fregata is $179-266\times$ faster than Cheetah, since it only involves lightweight addition and multiplication of ASS and we adopt GPU to paralellize such operations. Regarding total runtime (offline + online), Fregata remains $8-13\times$ faster. Although our total communication cost is slightly exceeds that of Cheetah, our communication overhead is $2.6-3.5\times$ smaller during the resource-sensitive online phase.

Table VIII presents the performance comparison of max-pooling layers. Fregata is $74-147\times$ faster than Cheetah, accompanied with communication cost reduced by $4-7\times$, owing to its reliance on lightweight ASS and the utilization of GPU. As for total runtime, Fregata achieves a speedup of $6.2-8.2\times$ over Cheetah.

4) Evaluation on Linear and Nonlinear Layers: We propose an asynchronous computation scheme to expedite the inference. To demonstrate its efficacy, we conduct evaluations on connected linear and activation layers, comparing it with CrpyTFlow2, Cheetah, and synchronous executions of `PMatMul` and `PMax`. For ease of representation, we refer to synchronous and asynchronous execution as `PSyn` and `PAsy`, respectively. The matrix dimensions in linear layers (from BC1 to BC3) are set as follows: $(1024, 128, 4096)$, $(512, 256, 4096)$, and $(512, 128, 2048)$.

Fig. 7a and Fig. 7b provide a visual representation of online runtime and total runtime. `PAsy` exhibits significant performance advantages over Cheetah. In the online phase, `PAsy` achieves a remarkable speedup, ranging from $36-50\times$ faster compared to Cheetah. This improvement extends to the total runtime, where `PAsy` is $2.5\times$ faster. In comparison to `PSyn`, our asynchronous computation scheme introduces enhancements. Specifically, we observe a $1.3\times$ speedup in online runtime and a $1.3-1.4\times$ speedup in total runtime. These advancements stem from our strategy to decouple specific procedures of `PMatMul` and `PMax`, executing them asynchronously.

C. End-to-end Inference Evaluation

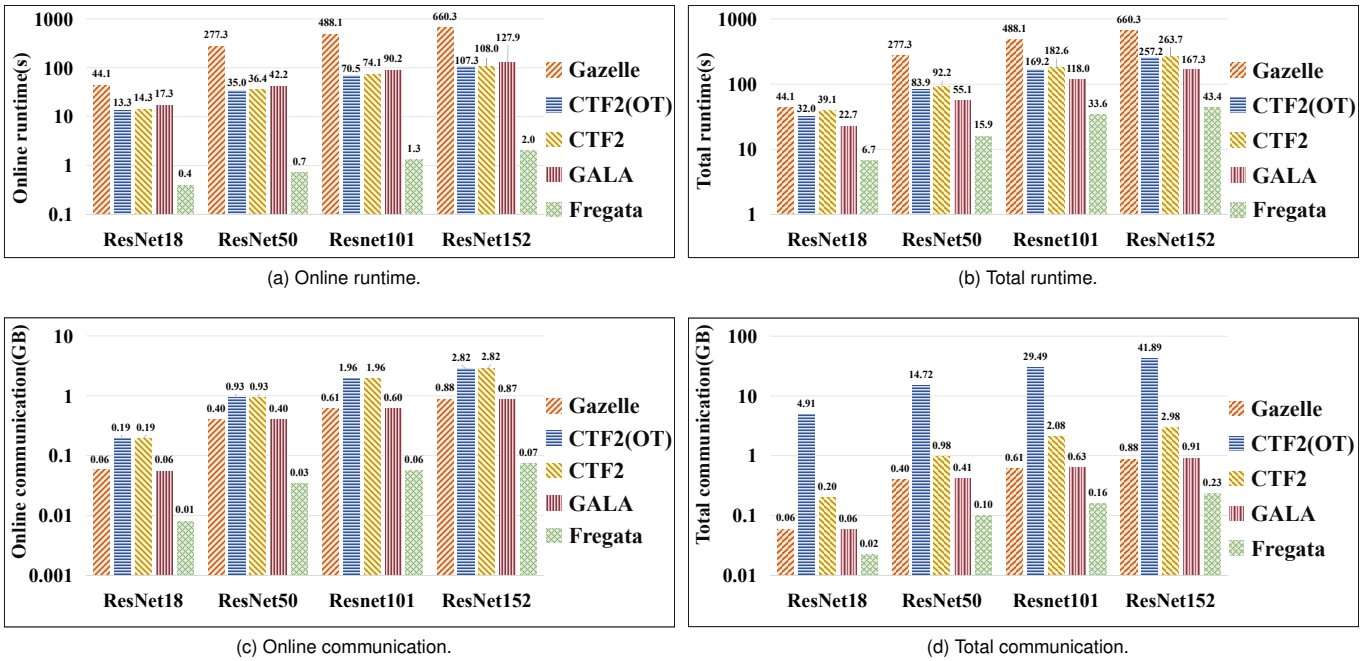
1) Evaluation on CIFAR-10 Dataset: The CIFAR-10 serves as a widely recognized dataset in the field of computer vision. In the CIFAR-10 challenge, images are presented with three channels to denote color, and the primary objective is to classify these images into 10 distinct classes. These classes encompass a variety of entities, including but not limited to ships, birds, frogs, trucks, and more.

We execute Gazelle, CrypTFlow2, CTF2(OT), GALA, and Fregata on ResNet18, ResNet50, and ResNet101, specifically for CIFAR-10 tasks. Fig. 8 delineate the online and total runtime, along with associated communication cost. During the latency-sensitive online phase, Fregata demonstrates a remarkable $34-64\times$ acceleration compared to CTF(OT), accompanied by a $6.8-11.9\times$ reduction in communication cost over GALA. Moreover, in comparison to GALA, Fregata sustains a $3.4-3.9\times$ acceleration in terms of total runtime and achieves an overall communication cost improvement of $2.6-4.1\times$. This performance improvement is attributed to the efficacy of our unified protocols and asynchronous computation scheme.

Fregata refrains from approximation in CNN inference, and the only possible accuracy diminution stems solely from the substitution of floating-point numbers with fixed-point numbers—a necessary procedure in PI schemes. This inevitable substitution has been extensively explored in prior works [19], [20], [25], consistently affirming the insignificance of the incurred loss. To substantiate this claim, we compare the accuracy of Fregata against Gazelle, CrypTFlow2, CTF2(OT), GALA, and the plaintext model. The results in Fig. 9 prove that the accuracy of these PI frameworks are very close, with negligible accuracy loss.

2) Evaluation on ImageNet Dataset: ImageNet is also a popular dataset in the realm of image classification tasks, which is much more complex than the CIFAR-10 dataset. In the ImageNet challenge, images are categorized into thousands of classes, spanning a spectrum from animals and natural landscapes to artificial structures and abstract concepts. Notably, the resolution of ImageNet images is usually higher than that of cifar, necessitating more computations in the model prediction process.

Fig. 10 shows the end-to-end comparison of CrypTFlow2, Cheetah, and Fregata on SqueezeNet, ResNet50, and



(a) Online runtime. (b) Total runtime.

(c) Online communication. (d) Total communication.

Fig. 8. Performance comparison of CIFAR-10 dataset. CTF2 refers to CrypTFlow2 [25].

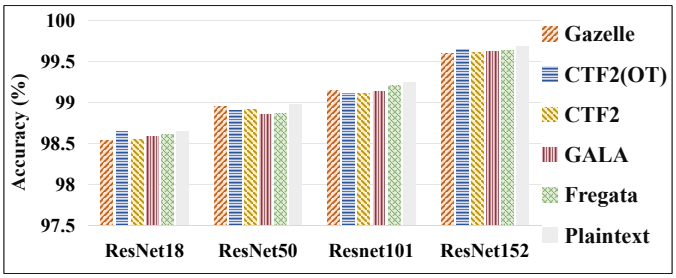


Fig. 9. Comparison of accuracy. CTF2 refers to CrypTFlow2 [25].

DenseNet121 for ImageNet tasks. Compared to Cheetah, Fregata exhibits a remarkable speed improvement of $35-45\times$ in the online phase, significantly reducing inference latency. Simultaneously, it achieves a communication cost reduction of $1.6-2.8\times$. Regarding total runtime and communication cost, Fregata maintains a speed advantage of $3\times$ and a communication cost improvement of $1.1\times$. This owes to the efficiency of our unified protocols and our holistic approach to accelerating inference. Significantly, the prediction latency is determined by the runtime of online phase.

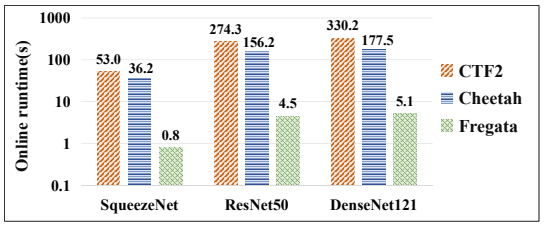
VII. Related Work

According to the cryptographic primitives employed, we categorize PI for CNNs into three types: *PI based on HE*, *PI based on GC*, and *PI based on 2PC*.

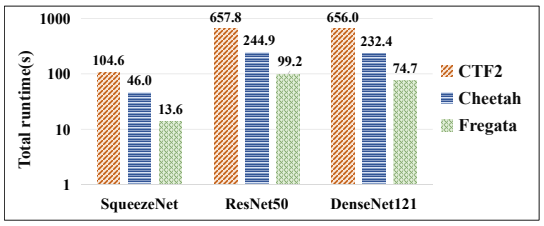
PI Based on HE. In this category [10], [16], [38], [39], clients transmit HE-encrypted data to the server who performs inference homomorphically. To improve efficiency, E2DM [36] elaborate a packing algorithm, minimizing HE operations in linear calculations. Despite this optimization, it suffers from notable inference latency, since it has to invoke the heavy bootstrapping operation to alleviate the ciphertext noise that accumulates with HE operations. Moreover, considering the inefficiency of HE in computing nonlinear functions, this category approximates them with low-degree polynomials. This approach compromises model accuracy and necessitates an additional retraining process to mitigate the decline.

PI Based on GC. GC is one of the generic secure two-party computation protocols, which is utilized by DeepSecure [40] to implement secure two-party inference. To enhance efficiency, XONN [11] binarizes the CNN and employs lightweight circuit operations to complete the inference of binarized CNN. However, this methodology diminishes model accuracy. Moreover, this type of PI incurs a large communication overhead since the communication of GC's each binary gate scales linearly with the security parameter.

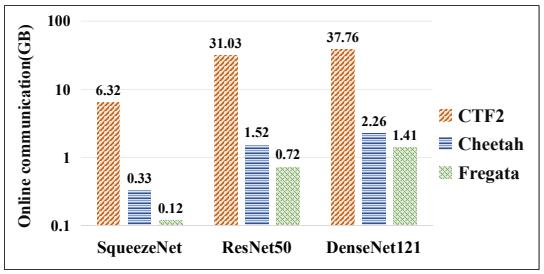
PI Based on 2PC. To address the challenges posed by the aforementioned categories, PI based on 2PC takes advantage of different cryptographic primitives. They customize 2PC protocols for linear and nonlinear layers based on appropriate techniques respectively. For expediting linear layer protocols, Gazelle [22], GALA [19], and Cheetah [20] elaborate packing algorithms for homomorphic linear calculations. To reduce inference latency, Delphi [23] puts HE operations of the linear layer protocol into the offline phase. To accelerate nonlinear layer protocols, CrypTFlow2 [25] and Cheetah utilize OT to perform the comparison operation in a privacy-preserving, outperforming previous methodologies. The proposed schemes employ distinct cryptographic tools for constructing linear and nonlinear layer protocols, relying on tools such as ABY [28] for data format conversion, necessitating the execution of linear and nonlinear layer protocols in synchronous modalities. In contrast, we design unified linear and nonlinear layer protocols to reduce conversion cost and expedite inference by



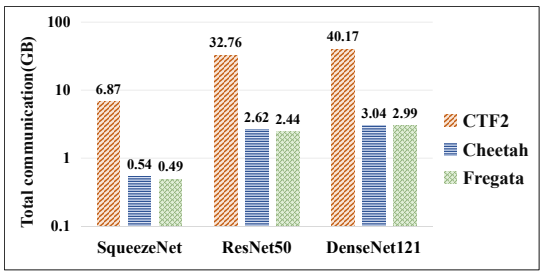
(a) Online runtime.



(b) Total runtime.



(c) Online communication.



(d) Total communication.

Fig. 10. Performance comparison of ImageNet dataset. CTF2 refers to CrypTFlow2 [25].

asynchronous computation. Furthermore, the online phases of our protocols exclusively involve lightweight ASS, enabling the utilization of GPU for additional acceleration.

VIII. Conclusion

In this paper, we present a fast PI based on unified 2PC protocols that leverage identical cryptographic primitives to calculate linear and nonlinear layers. Our protocols consist of an offline phase that utilizes HE to secretly share matrix products and an online phase that only involves lightweight operations on the shares. Benefiting from uniformity, we decouple certain procedures of linear and nonlinear layer protocols and execute them asynchronously. Moreover, we propose a homomorphic matrix encryption scheme that supports matrix multiplication with lower complexity than Cheetah to expedite the offline phase and employ GPU to speed up the online phase. Our experimental assessments on well-established CNN architectures, such as SqueezeNet, ResNet, and DenseNet, reveal a $35-45\times$ reduction in inference latency compared to Cheetah, with a corresponding $1.6-2.8\times$ decrease in communication cost. Additionally, Fregata maintains an approximate $3\times$ improvement in total runtime.

Acknowledgments

This research was supported in part by the National Key R&D Program of China under grant No. 2022YFB3102100, the Fundamental Research Funds for the Central Universities under grants No. 2042022kf1195, 2042023kf0120, the National Natural Science Foundation of China under grants No. 62076187, 62172303, 62302343, the Key R&D Program of Hubei Province under grant No. 2022BAA039, Key R&D Program of Shandong Province under grant No. 2022CXPT055, and the Guangdong Basic and Applied Basic Research Foundation under grant No. 2022A1515110396. The corresponding author is Jing Chen.

Appendix
Homomorphic Property Proof of HME

We first formally prove that HME calculates homomorphic matrix multiplication correctly. For clarity, we write $\pi(i,j) = i \cdot n \cdot k - j \cdot n + k \cdot n - 1$.

Theorem 3. *Given an $m \times n$ plaintext matrix W and a ciphertext $[M]$ of an $n \times k$ matrix M, where $[M] \leftarrow \texttt{EncMat}(pk, M)$, the following equality holds:* $\texttt{DecMat}(sk, \texttt{MulMat}([M], W, k), m, n, k) \equiv W \cdot M \bmod p$.

Proof: Let $[P]$ denote the result of $\texttt{MulMat}([M], W, k)$. The DecMat algorithm first computes $\hat{P} \leftarrow \texttt{Dec}(sk, [P])$ and then extracts the coefficient of $\hat{P}$ at position $\pi(i,j)$ as $(W \cdot M)_{i,j}$, where $0 \le i < m$ and $0 \le j < k$. Our objective is to prove the equivalence: $(W \cdot M)_{i,j} \equiv \hat{P}\pi(i,j) \bmod p$. $\hat{P}$ is calculated from $\hat{W} \cdot \hat{M}$ within the MulMat algorithm, where the coefficients of $\hat{W}$ and $\hat{M}$ are 0 except for $\hat{W}_{i \cdot n \cdot k + j} \equiv W_{i,j} \bmod p$ $(0 \le i < m, 0 \le j < n)$ and $\hat{M}_{n \cdot k - j \cdot n - i - 1} \equiv M_{i,j} \bmod p$ $(0 \le i < n, 0 \le j < k)$. Hence, we have $\hat{P}_{\pi(i,j)} \equiv \sum_{0 \le t \le \pi(i,j)} \hat{W}_t \cdot \hat{M}_{\pi(i,j)-t} - \sum_{\pi(i,j) < t < N} \hat{W}_t \cdot \hat{M}_{N+t-\pi(i,j)} \bmod p$. Since $\hat{M}_i = 0$ for $i \ge n \cdot k$ and $\hat{W}_j = 0$ for $j \notin [u \cdot n \cdot k, u \cdot n \cdot k + n)$, where u is an integer with $0 \le u < m$, it follows that $\hat{P}_{\pi(i,j)} \equiv \sum_{0 \le v < n} \hat{W}_{i \cdot n \cdot k + v} \cdot \hat{M}_{n \cdot k - j \cdot n - v - 1} \bmod p \equiv \sum_{0 \le v < n} W_{i,v} \cdot M_{v,j} \bmod p$, which precisely equals $(W \cdot M)_{i,j}$. ∎

We then prove the homomorphic property of $\texttt{DecMat}(sk, \texttt{AddMat}(pk, [W \cdot M], R) \equiv W \cdot M + R \bmod p$, which is utilized in the offline phase of PMatMul.

Theorem 4. *Given a ciphertext $[W \cdot M]$ obtained from $[W \cdot M] \leftarrow \texttt{MulMat}(\texttt{EncMat}(pk, M), W, k)$ and an $m \times k$ plaintext matrix R, where W and M are $m \times n$ and $n \times k$ matrices respectively, the following equality holds:* $\texttt{DecMat}(sk, \texttt{AddMat}(pk, [W \cdot M], R) \equiv W \cdot M + R \bmod p$.

Proof: For clarity, let $[P]$ denote $[W \cdot M]$ and let $[Q]$ represent the result of $\texttt{AddMat}(pk, [P], R)$. The AddMat algorithm encodes $R_{i,j}$ $(0 \le i < m, 0 \le j < k)$ into the $\pi(i,j)$-th coefficient of $\hat{R}$ and outputs $[Q] \leftarrow \texttt{Add}([P], [R])$. Recalling the proof of Theorem 3, we know that $\hat{P}\pi(i,j) \equiv$

$(W \cdot M)_{i,j} \bmod p$. Hence, we have $\hat{Q}_{\pi(i,j)} \equiv \hat{P}_{\pi(i,j)} + \hat{R}_{\pi(i,j)} \bmod p \equiv (W \cdot M)_{i,j} + R_{i,j} \bmod p$. Upon invocation of `DecMat`, $\hat{Q}_{\pi(i,j)}$ is assigned to the corresponding element $Y_{i,j}$ of the output matrix Y. Consequently, we can conclude that $Y \equiv W \cdot M + R \bmod p$. ■

REFERENCES

[1] R. Fakoor, F. Ladhak, A. Nazi, and M. Huber, "Using deep learning to enhance cancer diagnosis and classification," in *Proc. of ICML WHEALTH workshop*, 2013.

[2] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proc. of CVPR*. IEEE Computer Society, 2015.

[3] https://aws.amazon.com/rekognition/?nc1=h_ls.

[4] https://www.alibabacloud.com/product/imagesearch?spm=a3c0i.7911826.6791778070.268.44193870O9D4Mx.

[5] W. Wang, J. Gao, M. Zhang, S. Wang, G. Chen, T. K. Ng, B. C. Ooi, J. Shao, and M. Reyad, "Rafiki: Machine learning as an analytics service system," *Proc. VLDB Endow.*, vol. 12, no. 2, pp. 128–140, 2018.

[6] M. Barni, C. Orlandi, and A. Piva, "A privacy-preserving protocol for neural-network-based computation," in *Proc. of workshop on Multimedia & Security, MM&Sec*. ACM, 2006.

[7] L. L. Ng and S. M. Chow, "Sok: Cryptographic neural-network computation," in *Proc. of S&P*. IEEE Computer Society, 2023.

[8] Z. Ghodsi, N. K. Jha, B. Reagen, and S. Garg, "Circa: Stochastic relus for private deep learning," in *Proc. of NeurIPS*. Curran Associates, Inc., 2021.

[9] 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. of NDSS*. The Internet Society, 2021.

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

[11] 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. of USENIX Security Symposium*. USENIX Association, 2019.

[12] N. Koti, A. Patra, R. Rachuri, and A. Suresh, "Tetrad: Actively secure 4pc for secure training and inference," in *Proc. of NDSS*. The Internet Society, 2022.

[13] N. Koti, M. Pancholi, A. Patra, and A. Suresh, "SWIFT: super-fast and robust privacy-preserving machine learning," in *Proc. of USENIX Security Symposium*. USENIX Association, 2021.

[14] M. Li, S. S. M. Chow, S. Hu, Y. Yan, C. Shen, and Q. Wang, "Optimizing privacy-preserving outsourced convolutional neural network predictions," *IEEE Transactions on Dependable and Secure Computing.*, vol. 19, no. 3, pp. 1592–1604, 2022.

[15] X. Yang, J. Chen, K. He, H. Bai, C. Wu, and R. Du, "Efficient privacy-preserving inference outsourcing for convolutional neural networks," *IEEE Transactions on Information Forensics and Security*, pp. 1–1, 2023.

[16] 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. of ICML*. JMLR.org, 2016.

[17] 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. of AsiaCCS*. ACM, 2018.

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

[19] Q. Zhang, C. Xin, and H. Wu, "GALA: greedy computation for linear algebra in privacy-preserved neural networks," in *Proc. of NDSS*. The Internet Society, 2021.

[20] Z. Huang, W. Lu, C. Hong, and J. Ding, "Cheetah: Lean and fast secure two-party deep neural network inference," in *Proc. of USENIX Security Symposium*. USENIX Association, 2022.

[21] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via minionn transformations," in *Proc. of CCS*. ACM, 2017.

[22] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *Proc. of USENIX Security Symposium*. USENIX Association, 2018.

[23] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "Delphi: A cryptographic inference service for neural networks," in *Proc. of USENIX Security Symposium*. USENIX Association, 2020.

[24] S. U. Hussain, M. Javaheripi, M. Samragh, and F. Koushanfar, "COINN: crypto/ml codesign for oblivious inference via neural networks," in *Proc. of CCS*. ACM, 2021.

[25] D. Rathee, M. Rathee, N. Kumar, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "Cryptflow2: Practical 2-party secure inference," in *Proc. of CCS*. ACM, 2020.

[26] S. Bian, T. Wang, M. Hiromoto, Y. Shi, and T. Sato, "ENSEI: efficient secure inference via frequency-domain homomorphic convolution for privacy-preserving visual recognition," in *Proc. of CVPR*. Computer Vision Foundation / IEEE, 2020.

[27] S. Li, K. Xue, B. Zhu, C. Ding, X. Gao, D. S. L. Wei, and T. Wan, "FALCON: A fourier transform based approach for fast and secure convolutional neural network predictions," in *Proc. of CVPR*. Computer Vision Foundation / IEEE, 2020.

[28] D. Demmler, T. Schneider, and M. Zohner, "ABY - A framework for efficient mixed-protocol secure two-party computation," in *Proc. of NDSS*. The Internet Society, 2015.

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

[30] P. Kim, "Convolutional neural network," in *MATLAB deep learning*. Springer, 2017, pp. 121–147.

[31] X. Ding, X. Zhang, N. Ma, J. Han, G. Ding, and J. Sun, "Repvgg: Making vgg-style convnets great again," in *Proc. of CVPR*. Computer Vision Foundation / IEEE, 2021.

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

[33] C. Gentry, S. Halevi, and N. P. Smart, "Fully homomorphic encryption with polylog overhead," in *Proc. of EUROCRYPT*. Springer, 2012.

[34] B. Jayaraman and D. Evans, "Evaluating differentially private machine learning in practice," in *Proc. of USENIX Security Symposium*. USENIX Association, 2019.

[35] M. Abadi, A. Chu, I. J. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proc. of CCS*. ACM, 2016.

[36] X. Jiang, M. Kim, K. E. Lauter, and Y. Song, "Secure outsourced matrix computation and application to neural networks," in *Proc. of CCS*. ACM, 2018.

[37] SEAL, https://github.com/microsoft/SEAL, 2020.

[38] E. Hesamifard, H. Takabi, and M. Ghasemi, "Cryptodl: Deep neural networks over encrypted data," *CoRR*, vol. abs/1711.05189, 2017.

[39] E. Chou, J. Beal, D. Levy, S. Yeung, A. Haque, and L. Fei-Fei, "Faster cryptonets: Leveraging sparsity for real-world encrypted inference," *CoRR*, vol. abs/1811.09953, 2018.

[40] B. D. Rouhani, M. S. Riazi, and F. Koushanfar, "Deepsecure: scalable provably-secure deep learning," in *Proc. of DAC*. ACM, 2018.