# Efficient Single-Server Private Inference Outsourcing for Convolutional Neural Networks

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

***Abstract*—Private inference outsourcing ensures the privacy of both clients and model owners when model owners deliver inference services to clients through third-party cloud servers. Existing solutions either reduce inference accuracy due to model approximations or rely on the unrealistic assumption of non-colluding servers. Moreover, their efficiency falls short of HELiKs, a solution focused solely on client privacy protection. In this paper, we propose Skybolt, a single-server private inference outsourcing framework without resorting to model approximations, achieving greater efficiency than HELiKs. Skybolt is built upon efficient secure two-party computation protocols that safeguard the privacy of both clients and model owners. For the linear calculation protocol, we devise a ciphertext packing algorithm for homomorphic matrix multiplication, effectively reducing both computational and communication overheads. Additionally, our nonlinear calculation protocol features a lightweight online phase, involving only the addition and multiplication on secret shares. This stands in contrast to existing protocols, which entail resource-intensive techniques such as oblivious transfer. Extensive experiments on popular models, including ResNet50 and DenseNet121, show that Skybolt achieves a $5.4-7.3\times$ reduction in inference latency, accompanied by a $20.1-39.6\times$ decrease in communication cost compared to HELiKs.**

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

## I. Introduction

CONVOLUTIONAL Neural Networks (CNNs) have exhibited exceptional performance in computer vision tasks such as action recognition [1], target tracking [2], and object detection [3]. Fueled by advancements in cloud computing, a rising number of CNN model owners have begun offering inference services to clients by deploying their models on third-party cloud servers [4], [5], known as outsourcing inference services. This paradigm enables model owners to circumvent the cost of maintaining dedicated online services. For example, Sygic, a navigation company with over 200 million users, provides travel photo classification and recommendation functions by hosting its models on Amazon cloud servers [6].

Given potential sensitivity of clients' input and inference outcomes, it is crucial to protect them from disclosure to model owners or cloud servers [7], [8]. For instance, pathological data and diagnostic results within disease inference services are sensitive personal information under the General Data Protection Regulation (GDPR) [9]. To address this concern, Private Inference (PI) has been proposed, enabling inference without compromising client privacy through cryptographic techniques. Advanced methods [10], [11], [12] tailor secure two-party computation (2PC) protocols to the unique characteristics of linear and nonlinear calculations within CNNs, executing these protocols between the client and server. Among them, HELiKs [13] stands out as the state-of-the-art.

Regrettably, these protocols are unfit for outsourcing inference services as they deploy the model on the server in plaintext. Remarkably, models are the intellectual property of model owners, requiring substantial amounts of data and computing resources for training. Moreover, these models may have constituted competitive advantages in various business fields. Consequently, model owners aspire to conceal model parameters from both cloud servers and clients to preserve their commercial interests [14], [15].

A few PI solutions tackle the intricate challenge of preserving the privacy of both the client and model owner, yet they come with inherent limitations. For one thing, schemes utilizing Homomorphic Encryption (HE) to protect client input and model parameters [16], [17] suffer from reduced accuracy and high latency. Specifically, due to HE's inefficiency in handling nonlinear calculations, these methods approximate nonlinear functions with low-degree polynomials, which decreases inference accuracy. Additionally, they have to invoke time-consuming bootstrapping operations to alleviate the noise of HE ciphertext, resulting in high inference latency. For another, to overcome the limitations imposed by HE, the other methods [5], [14] opt to protect the client input by secretly sharing it between non-colluding servers. Nevertheless, fulfilling the stringent requirement of non-colluding servers is unrealistic and burdensome [18], [19]. Furthermore, these schemes remain less efficient than HELiKs, as their linear calculation methods involve high HE computation and communication

Received 24 July 2024; revised 27 December 2024; accepted 4 April 2025. Date of publication 9 April 2025; date of current version 6 October 2025. This work was supported in part by the National Key Research and Development Program of China under Grant 2022YFB3102100, in part by the National Natural Science Foundation of China under Grant 62172303 and Grant 62302343, in part by the Key Research Development Program of Hubei Province under Grant 2024BAB018, and in part by the Key Research and Development Program of Shandong Province under Grant 2022CXPT055. This article was recommended by Associate Editor T. Zheng. *(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; li.yuqing@whu.edu.cn; hekun@whu.edu.cn; sdhxj66@whu.edu.cn; jlumos@whu.edu.cn; duraying@whu.edu.cn; bh0036@whu.edu.cn).

Digital Object Identifier 10.1109/TCSVT.2025.3559101

TABLE I

A HIGH-LEVEL COMPARISON OF PI SCHEMES. "MODEL PRIVACY" MEANS THAT THE SCHEME PROTECTS MODEL PARAMETERS; "NATIVE MODEL" INDICATES THAT THE SOLUTION DOES NOT APPROXIMATE CNN INFERENCE; "NC SERVERS" REPRESENTS THAT THE SCHEME RELIES ON NON-COLLUDING SERVERS; "ONLINE" DESCRIBES THE CRYPTOGRAPHIC PRIMITIVES UTILIZED IN THE ONLINE PHASE; "LINEAR" AND "NON-LINEAR" DEPICT THE TECHNIQUES USED TO IMPLEMENT PRIVACY-PRESERVING LINEAR AND NON-LINEAR CALCULATIONS RESPECTIVELY; THE INVOLVED CRYPTOGRAPHIC TECHNIQUES INCLUDE ADDITIVELY SECRET SHARING (ASS), GC, HE, AND OT

| Solution | Model Privacy | Native Model | NC Servers | Online | Linear | Nonlinear |
|---|---|---|---|---|---|---|
| E2DM [16] | ✓ | ✗ | ✗ | HE | HE | Approximation |
| CDKS [17] | ✓ | ✗ | ✗ | HE | HE | Approximation |
| Delphi [10] | ✗ | ✗ | ✗ | ASS, GC | ASS, HE | GC, Approximation |
| OPPOCNNP [5] | ✓ | ✗ | ✓ | ASS, GC | ASS, HE | GC |
| Cheetah [20] | ✗ | ✓ | ✗ | ASS, OT | ASS, HE | OT |
| HELiKs [13] | ✗ | ✓ | ✗ | ASS, OT | ASS, HE | OT |
| Skybolt | ✔ | ✔ | ✗ | ASS | ASS, HE | ASS, HE |

complexity, and nonlinear calculations employ heavy Garbled Circuits (GC). Developing an efficient solution for outsourcing inference services that does not rely on non-colluding servers or approximation presents significant challenges.

In this paper, we present a single-server private CNN inference framework for outsourcing inference services, called Skybolt, which avoids approximation and even surpasses the efficiency of HELiKs. We achieve this by customizing efficient 2PC protocols that safeguard the privacy of both the client and model owner, allowing the faithful execution of CNN calculations without any approximations. In linear calculations, we propose a novel HE ciphertext packing algorithm for matrix multiplication. This algorithm eliminates the computationally expensive HE rotation operation and fully utilizes ciphertext space, resulting in decreased computation and communication complexity compared to HELiKs. Moreover, to alleviate the client's computational burden, we tailor an asymmetric protocol that offloads the computationally intensive HE matrix multiplication tasks to the server. To further reduce inference latency, we construct a lightweight online phase by shifting expensive HE calculations to a pre-computing offline phase. Within nonlinear calculations, our protocol features a lightweight online phase that only involves multiplication and addition on secret shares. This is in contrast to existing protocols that rely on resource-intensive cryptographic primitives such as GC and Oblivious Transfer (OT). Additionally, we leverage the computational capabilities of Graphics Processing Units (GPUs) to parallelize the multiplication and addition in the online phase of both linear and nonlinear calculation protocols, greatly decreasing inference latency.

Table I provides an overview of Skybolt's attributes in comparison to other schemes. To the best of our knowledge, Skybolt is the first private inference outsourcing that protects both the privacy of client and server without model approximations or relying on non-colluding servers, and only involves lightweight secret sharing in the latency-sensitive online phase.

Our principal contributions are outlined as follows:

- We present Skybolt, an efficient PI framework for outsourcing inference services. Skybolt eliminates the need for non-colluding servers and faithfully implements CNN inference without approximation.
- We propose 2PC protocols that protect the privacy of both the client and model owner. For the linear calculation protocol, we devise a packing algorithm for HE matrix multiplication, demonstrating reduced computation and communication complexity than existing methods. For nonlinear calculations, we craft a protocol that involves only lightweight addition and multiplication of secret shares during the online phase.
- We implement Skybolt and apply it to well-established models, such as ResNet50, on datasets of ImageNet scale. The performance evaluation demonstrates that Skybolt achieves a $5.4 - 7.3\times$ reduction in inference latency, accompanied by a $20.1 - 39.6\times$ decrease in communication cost compared to HELiKs.

## II. Related Work

Protecting users' data while ensuring its usability when uploaded to cloud servers has garnered significant attention [21], [22], [23]. With technological advancements, PI has been proposed to enable complex CNN inference on encrypted data. We classify PI into two classes based on whether CNN model parameters are concealed from the cloud server: (1) PI without model protection and (2) PI with model protection.

### A. PI Without Model Protection

Within this class, the cloud server holds the plaintext CNN model. Schemes based on 2PC exhibit superior performance by leveraging appropriate cryptographic primitives to tailor 2PC protocols that cater to the distinct characteristics of linear and nonlinear layers within CNNs. To enhance the efficiency of linear layer protocols, a line of work, including Gazelle [24], GALA [12], Cheetah [20], and HELiKs [13], pack multiple plaintexts into a single HE ciphertext, with dedicated packing algorithms to reduce HE operations. For nonlinear layer protocols, a series of research, such as Gazelle [24], CrypTFlow2 [10], and Cheetah [20], devise 2PC comparison protocols based on GC or OT.

Compared to them, our 2PC protocols additionally protect model parameters. Moreover, the online phase of our protocols only involves lightweight elementary operations of secret shares, amenable to GPU acceleration. Furthermore,

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 $M_{i,j}$ |
| $[M]$ | Ciphertext of a matrix $M$ |
| $N$ | The capacity of HE ciphertext |
| $\circ$ | Matrix element-wise product |

our GDPA exhibits reduced computation and communication complexity than existing algorithms.

## B. PI With Model Protection

This category ensures the secrecy of model parameters from cloud servers and is tailored for outsourcing inference services. One type of methodologies [16], [17] deploys the model that is encrypted by HE on the server for making inferences on the client's HE-encrypted input. Given the inefficiency of HE in computing nonlinear functions, these schemes approximate the ReLU activation with a square function and max-pooling with mean-pooling. Nevertheless, this approximation not only diminishes inference accuracy but also necessitates a retraining process to alleviate the decline. Notably, the inference latency in these approaches is considerable, as they have to invoke computationally expensive bootstrapping operations to mitigate the noise of ciphertext.

The utilization of non-colluding servers is an effective strategy for privacy-preserving outsourcing computation, which is employed in another category of schemes [5], [14]. In these approaches, the model owner securely deploys the model on non-colluding servers, and the client secretly shares its input between the servers for inference. However, the assumption of non-colluding servers is a strong requirement that is challenging and burdensome to fulfill.

Moreover, these solutions manifest inferior efficiency in comparison to HELiKs, which features state-of-the-art linear and nonlinear layer protocols. Skybolt, in contrast, achieves superior efficiency relative to HELiKs and does not rely on non-colluding servers and approximation.

## III. Preliminaries

In this section, we introduce an abstraction of CNNs and review the cryptographic primitives utilized in our solution (major notations we use are summarized in Table II).

### A. Convolutional Neural Networks

CNNs consist of a series of layers. During inference, the first layer receives the input data and generates its output which serves as the input to the second layer. Similarly, each layer performs specific calculations on its input and obtains the output that becomes the input to the subsequent layer. The iterative process persists until the last layer generates its output, which serves as the inference result.

According to the type of calculations involved, CNN layers can be classified into two distinct categories: linear layers and nonlinear layers.

**Linear Layers.** These layers conduct linear transformations on the input. They comprise the Fully Connected (FC) layer, convolutional layer, and batch normalization layer, which can be calculated by matrix multiplication [25].

**Nonlinear Layers.** Nonlinear layers in CNN encompass the max-pooling layer and activation layer. The max-pooling layer serves the dual purpose of highlighting predominant features in the input while concurrently reducing data dimensionality. The activation layer is typically positioned after linear layers and plays a pivotal role in modeling the non-linear relationship. The default and most common activation layer is the Rectified Linear Unit (ReLU) activation layer [26], [27]. Both the ReLU activation layer and the max-pooling layer operate fundamentally based on the comparison operation.

### B. Cryptographic Primitives

**Additive Secret Sharing (ASS).** In the context of 2-out-of-2 ASS, a pair of secrets denoted as $(\langle x\rangle_0, \langle x\rangle_1)$ is generated to represent a value $x$ and shared between two parties, where $x = \langle x\rangle_0 + \langle x\rangle_1$. The fundamental property of ASS ensures that a party possessing only one of these secrets cannot gain any knowledge about $x$. When considering cryptographic primitives commonly employed in PI, such as HE, OT, and GC, ASS stands out as a lightweight option. Its computational cost is close to that of plaintext operations.

**Packed Homomorphic Encryption (PHE).** HE [28], [29] enables the direct calculation of the ciphertext $[f(x)]$ from $[x]$ without decryption. For improved computational efficiency and space utilization, PHE empowers to pack multiple elements into a single ciphertext, permitting concurrent HE operations on these elements in a Single Instruction Multiple Data (SIMD) fashion [28]. The maximum number of elements that can be accommodated within a ciphertext is defined as ciphertext capacity, denoted as $N$. Skybolt utilizes the following functions supported by PHE.

- `Gen`. With the input of a security parameter $\lambda$, this key generation algorithm generates a pair of public-private keys $(pk, sk)$ as the output. This process is denoted as $(pk, sk) \leftarrow \texttt{Gen}(1^\lambda)$.
- `Enc`. Given a public key $pk$ and a plaintext vector $X = (x_0, \ldots, x_{n-1})$ $(n \leq N)$, this encryption algorithm outputs a ciphertext $C = [(x_0, \ldots, x_{n-1})]$, denoted as $C \leftarrow \texttt{Enc}(pk, X)$.
- `Dec`. Given a private key $sk$ and a ciphertext $C = [(x_0, \ldots, x_{n-1})]$, this decryption algorithm outputs the plaintext vector $X = (x_0, \ldots, x_{n-1})$. This process is denoted as $X \leftarrow \texttt{Dec}(sk, C)$.
- `PAdd`. This addition algorithm takes as input two ciphertexts $C_x = [(x_0, \ldots, x_{n-1})]$ and $C_y = [(y_0, \ldots, y_{n-1})]$, and returns $C = [(x_0 + y_0, \ldots, x_{n-1} + y_{n-1})]$, denoted as $C \leftarrow \texttt{PAdd}(C_x, C_y)$.
- `PMul`. This scalar multiplication algorithm takes as input a ciphertext $C_x = [(x_0, \ldots, x_{n-1})]$ and a plaintext vector $V = (v_0, \ldots, v_{n-1})$, and returns $C = [(x_0 \cdot v_0, \ldots, x_{n-1} \cdot v_{n-1})]$, denoted as $C \leftarrow \texttt{PMul}(C_x, V)$.

- PRot. This algorithm can rotate elements in a ciphertext. For instance, it can turn $[(x_0, x_1, x_2, x_3)]$ into $[(x_3, x_0, x_1, x_2)]$.

Prior work [12] has substantiated that the execution time of PRotexceeds that of PAddand PMulby approximately 34× and 56×, respectively.

## IV. Overview

We describe the system model and provide a high-level description of Skybolt.

### A. System Model

Considering the outsourcing inference service, our system consists of three entities: the model owner, the cloud server, and the client. The model owner utilizes the trained CNN model to offer inference services to the client via the cloud server. The client can submit data to the server and retrieve the CNN's inference results.

**Threat Model.** Similar to prior works [11], [20], our solution targets semi-honest adversaries, wherein all entities abide by the protocol honestly but may seek to obtain additional information from other participants. Specifically, the model owner is curious about the input and inference results of clients; the cloud server attempts to learn the model parameters as well as the input and inference results of clients; and the client harbors intentions of obtaining the model parameters.

**Security Goals.** Our security goals are two-fold. For one thing, the input and inference results of clients should be safeguarded from exposure to the model owner and cloud server. For another, the model parameters should remain undisclosed to clients and the cloud server. Our security definition adheres to the standard ideal/real-world paradigm, necessitating that an adversary's perspective in the real world remains indistinguishable from that in the ideal world.

Aligned with prior semi-honest PI schemes [11], [20], Skybolt is not devised to withstand attacks solely reliant on inference results, such as model stealing attacks. Nonetheless, supplementary techniques (e.g., differential privacy [30], [31]) can be synergistically integrated into our scheme to fortify the privacy safeguards. Specifically, differential privacy prevents users from inferring sensitive model information, such as training data, through repeated queries by adding noise to model parameters or outputs. This noise can be seamlessly incorporated in the following ways [32]: First, the model owner can add noise to the model parameters before deploying the model to the cloud server. The resulting perturbed model can then provide inference services through the private inference solution. This perturbation does not interfere with the cryptographic primitives, as the noise is added to the model prior to executing private inference. Second, the noise can be added to the model outputs by the cloud server. Specifically, after executing inference protocols, the model output is secretly shared between the client and server. The server can add the noise to its share, ensuring that the client receives the perturbed output and is unable to infer sensitive details from the model outputs.

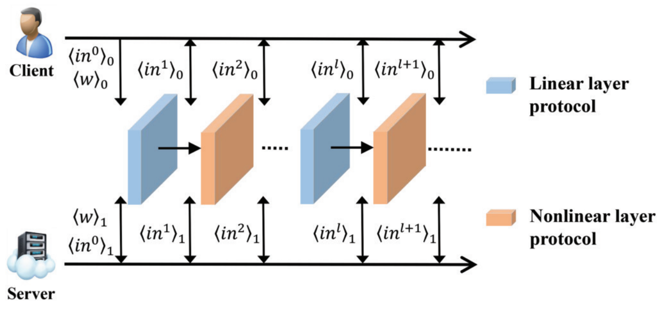


Fig. 1. The inference flow of Skybolt. $in^i$ denotes the input of the layer $i$, which is the output of layer $i-1$.$\langle in^i\rangle_0$ and $\langle in^i\rangle_1$ are a pair of secret shares of $in^i$.

Moreover, our primary focus does not center on concealing model structures, as this can be achieved through non-cryptographic strategies, such as the inclusion of dummy layers and parameters.

### B. Our Solution at a High Level

Existing PI approaches for outsourcing inference services either sacrifice inference accuracy due to model approximations or rely on the unrealistic assumption of non-colluding servers, both of which exhibit lower efficiency compared to HELiKs. To address these limitations, we present Skybolt, an efficient single-server PI framework that eschews approximations. Specifically, we safeguard client input by transmitting its secret share to the server, and complete inference by executing our tailored 2PC protocols between the client and server. These protocols ensure the privacy of both the client and model owner, specifically catering to the unique characteristics of both linear and non-linear layers within CNN. Notably, the protocols even surpass the efficiency of HELiKs's protocols, which lack protection for model parameters. In the following, we provide a high-level description of Skybolt.

Consistent with existing single-server PI schemes [12], [16], [17], [20], the client conducts a one-time setup phase when joining the inference service. This phase is independent of the client's input and only needs to be executed once per client. In this phase, the client generates a pair of public-private keys by invoking $(pk, sk) \leftarrow \mathtt{Gen}(1^\lambda)$, sends $pk$ to the server, and receives secret shares of model parameters. These shares are generated by the model owner, who selects a random value $r$ for each model parameter $w$, and then sends the shares $\langle w\rangle_0 = w - r$ and $\langle w\rangle_1 = r$ to the client and server, respectively. This will not pose a significant burden on the client's storage, given that well-established models like ResNet50 (with over 25 million parameters) only require about 100MB of memory.

When the client makes queries, the client and server complete the inference phase by conducting our 2PC protocols, which are tailored to suit the distinct characteristics of both linear and nonlinear layers within CNN. Specifically, each protocol starts with the client and server secretly sharing the input to the layer, and ends with them secretly sharing the layer's output. This paradigm enables the client and server to stitch our protocols sequentially to complete inference layer by layer, as shown in Fig. 1, while safeguarding the intermediate data from leakage to either the client or server. Initially, the client generates a random matrix $R$ of the same size as its input $in^0$ and then sends $\langle in^0\rangle_1 = in^0 - R$ to the server while retaining $\langle in^0\rangle_0 = R$. Finally, the output of the last layer is shared between the client and server, and the server sends its share to the client who recovers the inference result. Our protocols guarantee that the client remains oblivious to model parameters, and the server can learn nothing about the client's data and model parameters.

## V. 2PC Protocols for CNN Inference

In this section, we describe our 2PC protocols utilized in the inference phase of Skybolt.

### A. Linear Layer Protocol

Linear layers perform linear transformations on the input, typically achieved by matrix multiplication. To implement privacy-preserving computation, we customize a 2PC protocol for matrix multiplications, denoted as `PMulMat`.

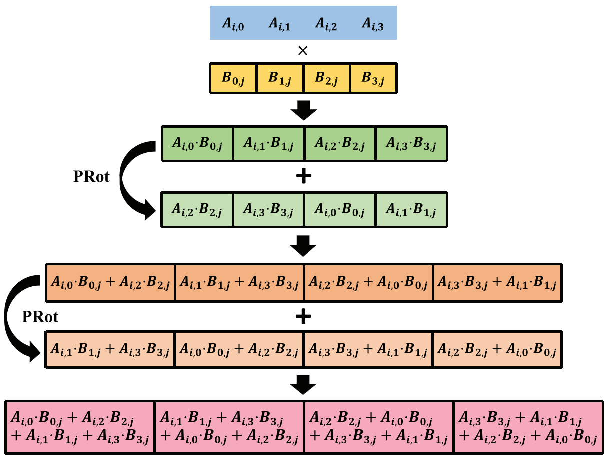
Recall that in linear layers, the client and server secretly share the parameter matrix $W$ and input matrix $X$. That is, the client and server hold $(\langle W\rangle_0, \langle X\rangle_0)$ and $(\langle W\rangle_1, \langle X\rangle_1)$ respectively. `PMulMat`enables them to jointly compute the secret shares $\langle Y\rangle_0$ and $\langle Y\rangle_1$ of the output matrix $Y = W \cdot X$, while ensuring the privacy of $W$, $X$, and $Y$. In the following of this subsection, we represent the dimensions of $W$ and $X$ as $(m, n)$ and $(n, k)$, respectively.

The computation of $Y$ can be expressed as: $Y = W \cdot X = (\langle W\rangle_0 + \langle W\rangle_1)(\langle X\rangle_0 + \langle X\rangle_1) = \langle W\rangle_0 \cdot \langle X\rangle_0 + \langle W\rangle_0 \cdot \langle X\rangle_1 + \langle W\rangle_1 \cdot \langle X\rangle_0 + \langle W\rangle_1 \cdot \langle X\rangle_1$. Since $\langle W\rangle_0 \cdot \langle X\rangle_0$ and $\langle W\rangle_1 \cdot \langle X\rangle_1$ can be calculated by the client and server respectively, the crux lies in the calculation of the cross-product terms $\langle W\rangle_0 \cdot \langle X\rangle_1$ and $\langle W\rangle_1 \cdot \langle X\rangle_0$. Note that if either the client or server acquires any of the cross-product terms, it can learn the counterpart's share of $W$ or $X$, thereby deducing $W$ or $X$. To address this security concern, we focus on the method that enables the client and server to secretly share these cross-product terms. This ensures that they can add their respective shares of the terms and $\langle W\rangle_0 \cdot \langle X\rangle_0$ or $\langle W\rangle_1 \cdot \langle X\rangle_1$ to secretly share $Y$.

We utilize HE and ASS techniques to secretly share the cross-product terms. This method enables two parties, $P_0$ and $P_1$, who possess the matrices $A$ and $B$ respectively, to share $D = A \cdot B$ without revealing $A$, $B$, or $D$. Specifically, $P_1$ first homomorphically encrypts $B$ and transmits the ciphertext $[B]$ to $P_0$. Then, $P_0$ performs HE matrix multiplication on $A$ and $[B]$ to obtain a ciphertext $[A \cdot B]$. Subsequently, $P_0$ homomorphically masks the product ciphertext with a randomly generated matrix $R$ and sends $[A \cdot B - R]$ to $P_1$ for decryption. In this way, $P_0$ and $P_1$ obtain $A \cdot B - R$ and $R$, respectively, to share $D$.

Building upon this method, we present further optimizations to enhance its overall efficiency.

*1) Improving He Matrix Multiplication:* Numerous studies [11], [12], [13], [20], [24], [33] have explored leveraging the PHE technique and crafting packing algorithms to improve the HE matrix multiplication, which dominates their total runtime and communication cost. For example, the naive



$A_{i,0}$ $A_{i,1}$ $A_{i,2}$ $A_{i,3}$
×
$B_{0,j}$ $B_{1,j}$ $B_{2,j}$ $B_{3,j}$
$A_{i,0}\cdot B_{0,j}$ $A_{i,1}\cdot B_{1,j}$ $A_{i,2}\cdot B_{2,j}$ $A_{i,3}\cdot B_{3,j}$
PRot
+
$A_{i,2}\cdot B_{2,j}$ $A_{i,3}\cdot B_{3,j}$ $A_{i,0}\cdot B_{0,j}$ $A_{i,1}\cdot B_{1,j}$
$A_{i,0}\cdot B_{0,j}+A_{i,2}\cdot B_{2,j}$ $A_{i,1}\cdot B_{1,j}+A_{i,3}\cdot B_{3,j}$ $A_{i,2}\cdot B_{2,j}+A_{i,0}\cdot B_{0,j}$ $A_{i,3}\cdot B_{3,j}+A_{i,1}\cdot B_{1,j}$
PRot
+
$A_{i,1}\cdot B_{1,j}+A_{i,3}\cdot B_{3,j}$ $A_{i,0}\cdot B_{0,j}+A_{i,2}\cdot B_{2,j}$ $A_{i,3}\cdot B_{3,j}+A_{i,1}\cdot B_{1,j}$ $A_{i,2}\cdot B_{2,j}+A_{i,0}\cdot B_{0,j}$
$A_{i,0}\cdot B_{0,j}+A_{i,2}\cdot B_{2,j}+A_{i,1}\cdot B_{1,j}+A_{i,3}\cdot B_{3,j}$ $A_{i,1}\cdot B_{1,j}+A_{i,3}\cdot B_{3,j}+A_{i,0}\cdot B_{0,j}+A_{i,2}\cdot B_{2,j}$ $A_{i,2}\cdot B_{2,j}+A_{i,0}\cdot B_{0,j}+A_{i,3}\cdot B_{3,j}+A_{i,1}\cdot B_{1,j}$ $A_{i,3}\cdot B_{3,j}+A_{i,1}\cdot B_{1,j}+A_{i,2}\cdot B_{2,j}+A_{i,0}\cdot B_{0,j}$

Fig. 2. An example of calculating the inner product of $A_i$ and $B_j$ homomorphically by "rotate-and-sum".

method [24] packs each column of $B$ into a single ciphertext $[B_j]$ $(0 \le j < k)$ and encodes each row of $A$ into a single vector $A_i$ $(0 \le i < m)$. To derive the ciphertext $[A \cdot B]$, it computes the inner product for each $A_i$ and $[B_j]$ homomorphically. To calculate the inner product, it first invokes `PMul`to compute the element-wise product of $A_i$ and $[B_j]$, obtaining a ciphertext $C_{i,j} = [A_{i,0} \cdot B_{0,j}, A_{i,1} \cdot B_{1,j}, \ldots, A_{i,n-1} \cdot B_{n-1,j}]$. Here, a key challenge arises: direct summation of elements within $C_{i,j}$ is not natively supported by PHE. Consequently, it has to execute the "rotate-and-sum" process $log(n)$ times as shown in Fig. 2. This approach necessitates a total of $m \cdot \log(n) \cdot k$ executions of `PAdd`and `PRot`operations, along with $m \cdot k$ `PMul` operations, transmitting $2 \cdot m \cdot k$ ciphertexts in total.

A long line of works concentrate on redesigning packing algorithms to minimize HE operations, particularly the resource-intensive operation `PRot`. However, the vast majority of them, including HELiKs, still rely on `PRot`to align ciphertext vectors with plaintext vectors and aggregate elements within the same ciphertext. Cheetah [20] avoids `PRot`by packing both $A$ and $B$ into polynomials and deriving $A \cdot B$ from coefficients of the polynomial product. Nevertheless, this approach invokes extra HE extraction operations to isolate desired coefficients into individual ciphertexts, in order to prevent potential information leakage from other coefficients. This significantly increases the communication overhead.

*a) Greedy Diagonal Packing Algorithm:* This motivates us to tailor a packing algorithm to eliminate both `PRot`and extraction, thereby reducing the overhead of calculating the HE matrix multiplication, called GDPA. To avoid extraction, GDPA does not derive matrix multiplication from polynomial products. Instead, it packs both $A$ and $B$ into vectors in a diagonal order. This diagonal packing inherently aligns ciphertext vectors with plaintext vectors, while ensuring that elements to be aggregated within the inner product calculation are distributed across multiple ciphertexts, thereby eliminating the need for `PRot`. Moreover, GDPA makes better use of ciphertext space by greedily accommodating as many diagonals as possible in a single ciphertext, unlike existing algorithms that underutilize ciphertext slots.

We now detail GDPA. Considering that matrix multiplication does not satisfy commutative law, we devise different encoding methods for the multiplicand matrix $A$ and the multiplier matrix $B$, denoted as $\mathtt{Pack}^{0}$ and $\mathtt{Pack}^{1}$ respectively. In the following, for simplicity of presentation, we assume that $(m \cdot k) \mid N$ and $N \mid (m \cdot n \cdot k)$, and set $z = \frac{m \cdot n \cdot k}{N}$ and $t = \frac{N}{m \cdot k}$. This can be easily satisfied by padding matrices with zeros and disusing the last $N \bmod (m \cdot n \cdot k)$ slots of ciphertexts. For cases with $m \cdot k > N$, we can split matrices into submatrices and process them independently.

$\mathtt{Pack}^{0}$. It encodes $A$ diagonally into $z$ vectors of length $N$, denoted as $(A^{(0)}, A^{(1)}, \ldots, A^{(z-1)})$. Specifically, $A^{(i)}(0 \leq i < z)$ is composed of $k$ replicas of a subvector. The subvector of $A^{(i)}$ consists of $t$ adjacent diagonals of $A$ as follows, in which we omit mod $n$ in the second element of subscripts.

$$\begin{pmatrix} A_{0,i\cdot t}, & A_{1,i\cdot t+1} & \ldots, & A_{m-1,i\cdot t+m-1}, \\ A_{0,i\cdot t+1}, & A_{1,i\cdot t+2} & \ldots, & A_{m-1,i\cdot t+m}, \\ & \vdots & & \\ A_{0,i\cdot t+t-1}, & A_{1,i\cdot t+t} & \ldots, & A_{m-1,i\cdot t+t+m-2} \end{pmatrix}$$

$\mathtt{Pack}^{1}$. It encodes $B$ in diagonal order into $z$ vectors of length $N$, denoted as $(B^{(0)}, B^{(1)}, \ldots, B^{(z-1)})$. $B^{(i)}(0 \leq i < z)$ is composed of $k$ subvectors and each subvector encodes $t$ diagonals of $B$. The $j$-th $(0 \leq j < k)$ subvector of $B^{(i)}$ is as follows, wherein we omit mod $n$ and mod $k$ in the first and second elements of subscripts, respectively.

$$\begin{pmatrix} B_{i\cdot t,j}, & B_{i\cdot t+1,j+1}, & \ldots, & B_{i\cdot t+m-1,j+m-1}, \\ B_{i\cdot t+1,j}, & B_{i\cdot t+2,j+1}, & \ldots, & B_{i\cdot t+m,j+m-1}, \\ & \vdots & & \\ B_{i\cdot t+t-1,j}, & B_{i\cdot t+t,j+1}, & \ldots, & B_{i\cdot t+t+m-2,j+m-1} \end{pmatrix}$$

We use the above algorithms to pack $A$ and $B$ into plaintext vectors and ciphertext vectors respectively, and derive the matrix multiplication $D = A \cdot B$ by performing HE operations on the vectors. First, for $0 \leq i < z$, we calculate the element-wise product of $A^{(i)}$ and $[B^{(i)}]$ to obtain a ciphertext $[D^{(i)}]$ using PMul. Then, we perform element-wise addition on all $[D^{(i)}](0 \leq i < z)$ to obtain a ciphertext $[V]$ by invoking PAdd. The vector $V$ can be decomposed into $k$ subvectors. The $j$-th subvector (denoted as $V^{j}$) is described in Equation 1, as shown at the bottom of the page, (where modular operations are omitted in the subscripts for clarity). Notably, aggregating every $m$-th element of $V^{j}$ can yield a diagonal of the product matrix $D$, i.e., $D_{i,i+j \bmod k} = \sum_{l=0}^{t-1} V^{j}_{i+m\cdot l}$ $(0 \leq i < m)$.

To circumvent the computationally expensive PRotoperation required for this aggregation, we strategically perform it in plaintext after decryption. This is feasible because our protocol uses a random matrix $R$ to mask $V$, allowing $P_0$ and $P_1$ to independently perform the aggregation on $R$ and $V - R$ without revealing the matrices. This approach does not increase communication complexity compared to existing methods, as the product elements in their transmitted ciphertexts are sparser. By packing both $A$ and $B$ diagonally, GDPA ensures that elements to be added together appear in different ciphertexts whenever possible, thus minimizing the number of elements requiring aggregation and reducing communication complexity. An example of using GDPA to calculate matrix multiplication is shown in Fig. 3, where the ciphertext capacity $N$ is 12, and the matrices dimensions $(m, n, k)$ are $(3, 2, 4)$.

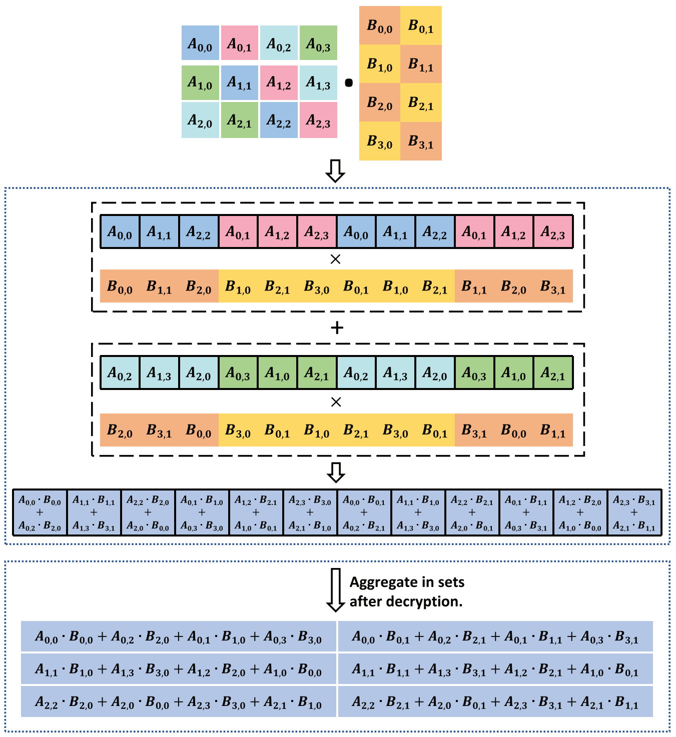


Fig. 3. An example of utilizing GDPA to calculate the matrix multiplication, where grids with solid boundaries represent ciphertext. "+" and "×" denote PAdd and PMul respectively.

*b) Complexity Analysis:* When multiplying an $m \times n$ matrix $W$ with an $n \times k$ matrix $X$, GDPA encodes the matrices into $z$ ciphertext vectors and $z$ plaintext vectors respectively, where $z = \lceil \frac{m \cdot n \cdot k}{N} \rceil$. It then performs $z$ instances of PMul on corresponding ciphertext and plaintext vectors. Subsequently, PAdd is applied to all resulting ciphertexts, requiring $z - 1$ instances of PAdd. The overall complexity of GDPA is $\mathcal{O}\left(\lceil \frac{m \cdot n \cdot k}{N} \rceil\right)$, with both PMul and PAdd operations scaling accordingly. No PRot operations are required. For the lower bound of the complexity, we consider the best-case scenario where $N > m \cdot n \cdot k$, which results in only a single PRot operation and no PAdd operations.

$$\begin{pmatrix} \sum_{i=0}^{z-1} A_{0,i\cdot t} \cdot B_{i\cdot t,j}, & \sum_{i=0}^{z-1} A_{1,i\cdot t+1} \cdot B_{i\cdot t+1,j+1}, & \ldots, & \sum_{i=0}^{z-1} A_{m-1,i\cdot t+m-1} \cdot B_{i\cdot t+m-1,j+m-1}, \\ \sum_{i=0}^{z-1} A_{0,i\cdot t+1} \cdot B_{i\cdot t+1,j}, & \sum_{i=0}^{z-1} A_{1,i\cdot t+2} \cdot B_{i\cdot t+2,j+1}, & \ldots, & \sum_{i=0}^{z-1} A_{m-1,i\cdot t+m} \cdot B_{i\cdot t+m,j+m-1}, \\ & \vdots & & \\ \sum_{i=0}^{z-1} A_{0,i\cdot t+t-1} \cdot B_{i\cdot t+t-1,j}, & \sum_{i=0}^{z-1} A_{1,i\cdot t+t} \cdot B_{i\cdot t+t,j+1}, & \ldots, & \sum_{i=0}^{z-1} A_{m-1,i\cdot t+t+m-2} \cdot B_{i\cdot t+t+m-2,j+m-1} \end{pmatrix} \quad (1)$$

TABLE III

COMPLEXITY COMPARISON OF MATRIX MULTIPLICATION CALCULATION METHODS

| Algorithm | Add | Multiply | Rotate | Extract | Communication |
|---|---|---|---|---|---|
| Naive | $m \cdot log(n) \cdot k$ | $m \cdot k$ | $m \cdot log(n) \cdot k$ | 0 | $2 \cdot m \cdot k$ |
| Cheetah | $\lceil \frac{m \cdot n}{N} \rceil \cdot k - 1$ | $\lceil \frac{m \cdot n}{N} \rceil \cdot k$ | 0 | $m \cdot k$ | $\lceil \frac{m}{N} \rceil \cdot k + m \cdot k$ |
| HELiKs | $\lceil \frac{m \cdot n}{N} \rceil \cdot k - 1$ | $\lceil \frac{m \cdot n}{N} \rceil \cdot k$ | $\lceil \frac{m \cdot n}{N} \rceil \cdot k - 1$ | 0 | $\lceil \frac{n}{N} \rceil \cdot k + \lceil \frac{m \cdot n}{N} \rceil \cdot k$ |
| GDPA | $\lceil \frac{m \cdot n \cdot k}{N} \rceil - 1$ | $\lceil \frac{m \cdot n \cdot k}{N} \rceil$ | 0 | 0 | $2 \cdot \lceil \frac{m \cdot n \cdot k}{N} \rceil$ |

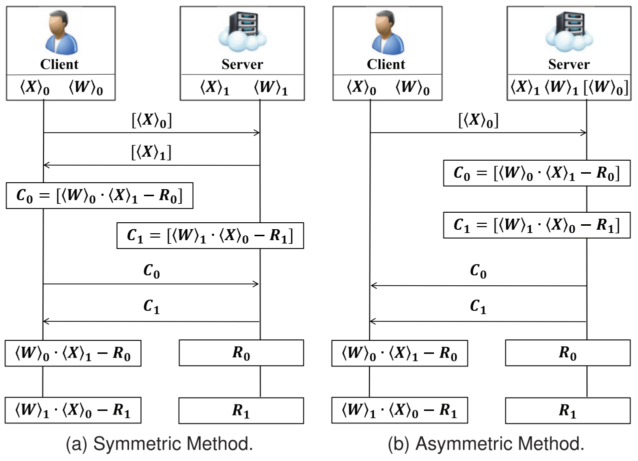

Client
Server
$\langle X\rangle_0$ $\langle W\rangle_0$
$\langle X\rangle_1$ $\langle W\rangle_1$
$[\langle X\rangle_0]$
$[\langle X\rangle_1]$
$C_0 = [\langle W\rangle_0 \cdot \langle X\rangle_1 - R_0]$
$C_1 = [\langle W\rangle_1 \cdot \langle X\rangle_0 - R_1]$
$C_0$
$C_1$
$\langle W\rangle_0 \cdot \langle X\rangle_1 - R_0$
$R_0$
$\langle W\rangle_1 \cdot \langle X\rangle_0 - R_1$
$R_1$
(a) Symmetric Method.
Client
Server
$\langle X\rangle_0$ $\langle W\rangle_0$
$\langle X\rangle_1$ $\langle W\rangle_1$ $[\langle W\rangle_0]$
$[\langle X\rangle_0]$
$C_0 = [\langle W\rangle_0 \cdot \langle X\rangle_1 - R_0]$
$C_1 = [\langle W\rangle_1 \cdot \langle X\rangle_0 - R_1]$
$C_0$
$C_1$
$\langle W\rangle_0 \cdot \langle X\rangle_1 - R_0$
$R_0$
$\langle W\rangle_1 \cdot \langle X\rangle_0 - R_1$
$R_1$
(b) Asymmetric Method.

Fig. 4. The methods for secretly sharing cross-product terms.

We compare GDPA with existing packing algorithms in terms of computation and communication complexity, as shown in Table III. GDPA avoids the most expensive `PRot`operation by eliminating the need for summing elements within the same ciphertext or aligning vectors. While Cheetah also eliminates `PRot`, it requires additional $m \cdot k$ extraction operations and exhibits elevated communication complexity. Moreover, considering that the ciphertext capacity $N$ is typically a multiple of $m \cdot n$ in PI, the quantities $\lceil \frac{m}{N} \rceil \cdot k$, $\lceil \frac{n}{N} \rceil \cdot k$, $\lceil \frac{m \cdot n}{N} \rceil \cdot k$, and $m \cdot k$ are typically several times larger than $\lceil \frac{m \cdot n \cdot k}{N} \rceil$. Consequently, the computation and communication complexity of GDPA tends to be lower than Cheetah and HELiKs. This stems from GDPA's strategy of greedily packing as many diagonals as possible within a single ciphertext.

Notably, the efficiency gains offered by GDPA are not limited to our protocol; it can serve as a general drop-in replacement for existing PI systems including HELiKs, which do not safeguard the model.

2) *Alleviating Burden on the Client:* The current method for sharing cross-product terms places the resource-intensive HE calculation of $[\langle W\rangle_0 \cdot \langle X\rangle_1 - R_0]$ on the client, as shown in Fig. 4a. Given the inherent computational capability gap between the client and the server, this approach may introduce a performance bottleneck.

*a) Asymmetric Protocol:* To address this limitation, we customize an asymmetric protocol that offloads the heavy HE calculations for masked matrix multiplication to the server, as illustrated in Fig. 4b. Specifically, to eliminate the client-side computation of $[\langle W\rangle_0 \cdot \langle X\rangle_1 - R_0]$, the client computes the ciphertext $[\langle W\rangle_0]$ and transfers it to the server during the one-time setup phase, eliminating the transmission $[\langle X\rangle_1]$ in the inference phase. This approach not only reduces one round of communication per linear layer during the inference phase but also allows for further efficiency gains: by precomputing $[\langle W\rangle_0]$, the server can calculate $[\langle W\rangle_0 \cdot \langle X\rangle_1 - R_0]$ concurrently while the client generates and sends the ciphertext $[X_0]$.

*b) Homomorphic Matrix Encryption Scheme:* Considering that the asymmetric protocol involves the multiplication of a ciphertext multiplicand matrix with a plaintext multiplier matrix (and vice versa), we customize a HME scheme that supports both functionalities, which is built upon GDPA. The HME comprises five algorithms: `GenKey`, `EMat`, `DMat`, `AddMat`, and `MulMat`.

`GenKey`$(1^\lambda)$. With the input of a security parameter $\lambda$, the key generation algorithm invokes $(pk, sk) \leftarrow \texttt{Gen}(1^\lambda)$ to obtain a pair of public-private keys as the output.

`EMat`$(pk, B, b)$. The matrix encryption algorithm takes as input a public key $pk$, a matrix $B$, and an integer parameter $b \in \{0, 1\}$ used to determine the packing algorithm. It first invokes the encoding algorithm $\texttt{Pack}^{\texttt{b}}$ to encode $B$ into $z$ vectors denoted as $(B^{(0)}, B^{(1)}, \ldots, B^{(z-1)})$. Then, for each $B^{(i)}(0 \le i < z)$, the algorithm performs $[B^{(i)}] \leftarrow \texttt{Enc}(pk, B^{(i)})$ and outputs $[B] = ([B^{(0)}], \ldots, [B^{(z-1)}])$.

`DMat`$(sk, [V])$. This matrix decryption algorithm takes as input a private key $sk$ and a ciphertext $[V]$. It first computes $V \leftarrow \texttt{Dec}(sk, [V])$ and partitions $V$ into $k$ subvectors of length $\frac{N}{k}$, denoted as $V^0, \ldots, V^{k-1}$. Then, it encodes these subvectors into a matrix $D$ as the output, where $D_{i,i+j \bmod k} = \sum_{l=0}^{t-1} V^j_{i+m \cdot l}$ for $0 \le i < m$ and $0 \le j < k$.

`AddMat`$(pk, [V], R)$. Given a public key $pk$, a ciphertext $[V]$ and an matrix $R$, the addition algorithm first encodes $R$ into $k$ vectors of length $\frac{N}{k}$, denoted as $R^0, \ldots, R^{k-1}$, where $R_{i,i+j \bmod k} = \sum_{l=0}^{t-1} R^j_{i+m \cdot l}$ for $0 \le i < m$ and $0 \le j < k$. Then, it concatenates these vectors, $R^0$ to $R^{k-1}$, to form a vector $R'$. Finally, it computes $[R'] \leftarrow \texttt{Enc}(pk, R')$ and outputs $\texttt{PAdd}([V], [R'])$.

`MulMat`$([B], A, b)$. With the input of a ciphertext matrix $[B]$ and a plaintext matrix $A$, and an integer parameter $b \in \{0, 1\}$, the matrix multiplication algorithm first parses the encrypted matrix $[B]$ as $([B_0], \ldots, [B_{z-1}])$. Then, it encodes the plaintext matrix $A$ into $z$ vectors denoted as $(A^{(0)}, A^{(1)}, \ldots, A^{(z-1)})$ by invoking $\texttt{Pack}^{1-\texttt{b}}$. And then, for each $[B^{(i)}]$, the algorithm performs $C_i \leftarrow \texttt{PMul}([B^{(i)}], A^{(i)})$. Finally, it invokes `PAdd`on all $C_i$ to obtain a ciphertext $C$ of the matrix multiplication as the output.

The complete process of the `PMulMat`protocol that utilizes GDPA and HME is delineated in Protocol 1. Note that $[\langle W\rangle_0]$ is calculated by the client performing $[\langle W\rangle_0] \leftarrow$

**Protocol 1** $\texttt{PMulMat}$

```
Input: The client inputs an $m \times n$ matrix $\langle W\rangle_0$ and an $n \times k$ matrix $\langle X\rangle_0$. The server inputs an $m \times n$ matrix $\langle W\rangle_1$, an $n \times k$ matrix $\langle X\rangle_1$, and a ciphertext matrix $[\langle W\rangle_0]$.
Output: The client and server output $m \times k$ matrices $\langle Y\rangle_0$ and $\langle Y\rangle_1$ respectively, where $\langle Y\rangle_0 + \langle Y\rangle_1 = (\langle W\rangle_0 + \langle W\rangle_1) \cdot (\langle X\rangle_0 + \langle X\rangle_1)$.
1: The client computes $[\langle X\rangle_0] \leftarrow \texttt{EMat}(pk, \langle X\rangle_0, 1)$ and sends it to the server. Meanwhile, the server generates two $m \times k$ random matrices $R_0$ and $R_1$, and calculates $[\langle W\rangle_0 \cdot \langle X\rangle_1 - R_0] \leftarrow \texttt{AddMat}(pk, \texttt{MulMat}([\langle W\rangle_0], \langle X\rangle_1, 0), -R_0)$, and then sends it to the client.
2: After receiving $[\langle X\rangle_0]$, the server computes $[\langle W\rangle_1 \cdot \langle X\rangle_0 - R_1] \leftarrow \texttt{AddMat}(pk, \texttt{MulMat}([\langle X\rangle_0], \langle W\rangle_1, 1), -R_1)$ and sends it to the client. Then, the server sets $\langle\langle W\rangle_0 \cdot \langle X\rangle_1\rangle_1 = R_0$ and $\langle\langle W\rangle_1 \cdot \langle X\rangle_0\rangle_1 = R_1$.
3: The client decrypts to get $\langle\langle W\rangle_0 \cdot \langle X\rangle_1\rangle_0 \leftarrow \texttt{DMat}(sk, [\langle W\rangle_0 \cdot \langle X\rangle_1 - R_0])$ and $\langle\langle W\rangle_1 \cdot \langle X\rangle_0\rangle_0 \leftarrow \texttt{DMat}(sk, [\langle W\rangle_1 \cdot \langle X\rangle_0 - R_1])$.
4: The client outputs $Y_0 = \langle W\rangle_0 \cdot \langle X\rangle_0 + \langle\langle W\rangle_0 \cdot \langle X\rangle_1\rangle_0 + \langle\langle W\rangle_1 \cdot \langle X\rangle_0\rangle_0$. The server outputs $Y_1 = \langle W\rangle_1 \cdot \langle X\rangle_1 + \langle\langle W\rangle_0 \cdot \langle X\rangle_1\rangle_1 + \langle\langle W\rangle_1 \cdot \langle X\rangle_0\rangle_1$.
```

$\texttt{EMat}(pk, \langle W\rangle_0, 0)$ and sent to the server during the one-time setup phase.

*3) Constructing Lightweight Online Phase:* To reduce inference latency, we extend $\texttt{PMulMat}$to the online-offline paradigm, thereby creating a lightweight online phase. This improvement offloads the HE operations to the preprocessing offline phase that is independent of the client's input and executed prior to the client's query. Our online phase only involves addition and multiplications on secret shares.

During the offline phase, the client and server independently generate two random matrices, $\langle S\rangle_0$ and $\langle S\rangle_1$, with the same dimensions as $X$. They then apply our method of secretly sharing cross-product terms to share $\langle W\rangle_1 \cdot \langle S\rangle_0$ and $\langle W\rangle_0 \cdot \langle S\rangle_1$. As a result, the client obtains $\langle\langle W\rangle_1 \cdot \langle S\rangle_0\rangle_0$ and $\langle\langle W\rangle_0 \cdot \langle S\rangle_1\rangle_0$, while the server obtains $\langle\langle W\rangle_1 \cdot \langle S\rangle_0\rangle_1$ and $\langle\langle W\rangle_0 \cdot \langle S\rangle_1\rangle_1$. It satisfies $\langle\langle W\rangle_1 \cdot \langle S\rangle_0\rangle_0 + \langle\langle W\rangle_1 \cdot \langle S\rangle_0\rangle_1 = \langle W\rangle_1 \cdot \langle S\rangle_0$ and $\langle\langle W\rangle_0 \cdot \langle S\rangle_1\rangle_0 + \langle\langle W\rangle_0 \cdot \langle S\rangle_1\rangle_1 = \langle W\rangle_1 \cdot \langle S\rangle_0$.

In the online phase, the client and server aim to secretly share the cross-product terms $\langle W\rangle_1 \cdot \langle X\rangle_0$ and $\langle W\rangle_0 \cdot \langle X\rangle_1$. To achieve this, the client sends $\langle X\rangle_0 - \langle S\rangle_0$ to the server. The server then computes $\langle\langle W\rangle_1 \cdot \langle X\rangle_0\rangle_1 = \langle W\rangle_1 \cdot (\langle X\rangle_0 - \langle S\rangle_0) + \langle\langle W\rangle_1 \cdot \langle S\rangle_0\rangle_1$. Subsequently, the client sets $\langle\langle W\rangle_1 \cdot \langle X\rangle_0\rangle_0 = \langle\langle W\rangle_1 \cdot \langle S\rangle_0\rangle_0$. This results in the equality $\langle\langle W\rangle_1 \cdot \langle X\rangle_0\rangle_0 + \langle\langle W\rangle_1 \cdot \langle X\rangle_0\rangle_1 = \langle W\rangle_1 \cdot \langle X\rangle_0$, thereby securely sharing the cross-product term $\langle W\rangle_1 \cdot \langle X\rangle_0$. Similarly, the client and server can secretly share the cross-product term $\langle W\rangle_0 \cdot \langle X\rangle_1$ in reverse.

*Remarks:* The online phase of the $\texttt{PMulMat}$protocol is limited to fundamental operations, specifically the addition and multiplication on secret shares. These basic operations are well-suited for parallel processing on GPUs, which we leverage to significantly reduce online inference latency. We achieve this by developing kernel functions that convert the originally serial loops for matrix multiplication and addition on secret shares into parallel GPU operations suitable for GPU execution. Additionally, we enhance memory management and kernel execution by carefully selecting grid and block sizes. This approach ensures efficient workload distribution across the GPU. Our GPU acceleration is implemented using CUDA C++, which lacks support for modular arithmetic. To address this, we designate an integer value as the modulus and conduct modular operations subsequent to multiplication or addition, all within kernel functions designed for parallel execution. Moreover, since we set the modulus $p$ to a power of 2, we implement the modular operation through a lightweight bitwise AND operation with $p - 1$, further reducing the inference latency.

## B. Nonlinear Layer Protocol

The basic operation of nonlinear layers within CNN is the comparison operation. To calculate nonlinear layers in a privacy-preserving way, we devise a 2PC comparison protocol that can conduct element-wise comparison of two secretly shared matrices without disclosing them, denoted as $\texttt{PCmp}$. In $\texttt{PCmp}$, the client and server start by secretly sharing two $m \times k$ matrices $A$ and $B$. That is, they hold $(\langle A\rangle_0, \langle B\rangle_0)$ and $(\langle A\rangle_1, \langle B\rangle_1)$, respectively. $\texttt{PCmp}$empowers them to obtain $\langle Y\rangle_0$ and $\langle Y\rangle_1$, respectively, where $\langle Y_{i,j}\rangle_b = \texttt{max}(\langle A_{i,j}\rangle_b, \langle B_{i,j}\rangle_b)$ $(0 \le i < m, 0 \le j < k, b \in [0, 1])$.

Many prior research make efforts to improve privacy-preserving nonlinear calculations. One category of studies opts to approximate nonlinear calculations with low-degree polynomials to suit cryptographic primitives [10], [34], [35]. Unfortunately, this approach diminishes inference accuracy, necessitating model retraining to mitigate the decline. Another type of research constructs the comparison protocol utilizing cryptographic primitives [11], [20], [24]. However, these protocols involve resource-intensive GC or OT during online inference, which dominates their inference latency. Moreover, the comparison protocols in existing PI schemes rely on cryptographic primitives different from those in linear layer protocols. Consequently, these protocols necessitate the utilization of data format conversion tools, such as the ABY framework [36], to bridge the gap, incurring extra data conversion overhead.

This motivates us to devise online lightweight nonlinear layer protocols that leverage the identical cryptographic primitives as our $\texttt{PMulMat}$protocol. To this end, we utilize HE to secretly share some randomness in the offline phase and perform lightweight operations on the shares during the online phase to accomplish the comparison. Remarkably, the online phase of $\texttt{PCmp}$exclusively involves basic addition and multiplication on secret shares. We leverage GPUs to parallelize these operations as done in $\texttt{PMulMat}$, significantly reducing inference latency. The complete process of the $\texttt{PCmp}$protocol is illustrated in Protocol 2, where $V$ and $R$ are $m \cdot k$ random vectors generated by the client and server respectively. Additionally, $U$ is also a random vector generated by the server, whose elements are random positive values. In Protocol 2, we assume that $N > m \cdot k$. If the assumption

**Protocol 2** PCmp

**Input:** The client inputs $\langle A\rangle_0$, $\langle B\rangle_0$, and $V$. The server inputs $\langle A\rangle_1$, $\langle B\rangle_1$, $U$, and $R$.

**Output:** The client and server output $\langle Y\rangle_0$ and $\langle Y\rangle_1$ respectively.

**Offline phase:**

```
1: The client computes $[V] \leftarrow \mathtt{Enc}(pk, V)$ and sends $[V]$ to the server.
2: The server first calculates the element-wise product of $V$ and $U$ homomorphically by invoking $[V \circ U] \leftarrow \mathtt{PMul}([V], U)$. Then, it masks $V \circ U$ by performing $[V \circ U - R] \leftarrow \mathtt{PAdd}([V \circ U], \mathtt{Enc}(pk, R))$ and sends $[V \circ U - R]$ to the client.
3: The client decrypts to get $\langle V \circ U\rangle_0 = \mathtt{Dec}(sk, [V \circ U - R])$. The server sets $\langle V \circ U\rangle_1 = R$. It satisfies $\langle V \circ U\rangle_0 + \langle V \circ U\rangle_1 = V \circ U$.
```

**Online phase:**

```
1: The client encodes $V$ and $\langle V \circ U\rangle_0$ into $m \times n$ matrix $V'$ and $\langle V \circ U\rangle'_0$ in a row-wise manner, respectively. Then, it computes and sends $\langle A\rangle_0 - \langle B\rangle_0 - V'$ to the server.
2: The server encodes the elements of $U$ and $\langle V \circ U\rangle_1$ into $m \times n$ matrix $U'$ and $\langle V \circ U\rangle'_1$ in a row-wise manner respectively. Then, it computes and sends $(\langle A\rangle_0 - \langle B\rangle_0 - V') \circ U' + \langle V \circ U\rangle'_1$ to the client.
3: The client performs $(A - B) \circ U' = (\langle A\rangle_0 - \langle B\rangle_0 - V') \circ U + \langle V \circ U\rangle'_1 + \langle V \circ U\rangle'_0$ and transmits an $m \times k$ matrix $Z$ to the server, where $Z_{i,j}$ is a randomly generated positive value if $((A - B) \circ U')_{i,j} > 0$, and a non-positive random value otherwise.
4: The client and server outputs $\langle Y\rangle_b$ ($b \in \{0, 1\}$), where $\langle Y_{i,j}\rangle_b = \langle A_{i,j}\rangle_b$ if $Z_{i,j} > 0$ and $\langle Y_{i,j}\rangle_b = \langle B_{i,j}\rangle_b$ otherwise.
```

is not satisfied, the matrices $A$ and $B$ can be subdivided into appropriate submatrices.

## VI. Security Analysis

We now prove the security of the PMulMatand PCmpprotocols introduced in Section V.

### A. *Security of PMulMat*

We now prove the security of PMulMat, which can be reduced to the security of sharing $\langle W\rangle_1 \cdot \langle S\rangle_0$ and $\langle W\rangle_0 \cdot \langle S\rangle_1$ in the offline phase. Since our processes of sharing $\langle W\rangle_1 \cdot \langle S\rangle_0$ and $\langle W\rangle_0 \cdot \langle S\rangle_1$ are irrelevant, we prove the security of them separately in Section VI-A1 and Section VI-A2.

*1) Sharing $\langle \mathbf{W}\rangle_1 \cdot \langle \mathbf{S}\rangle_0$:* We define the ideal matrix multiplication sharing functionality as $\{\langle W \cdot S\rangle_0; \langle W \cdot S\rangle_1\} \leftarrow \mathtt{IMulMat}\{S; W\}$, where $\langle W \cdot S\rangle_0 + \langle W \cdot S\rangle_1 = W \cdot S$.

*Theorem 1:* Our method of sharing $\langle W\rangle_1 \cdot \langle S\rangle_0$ securely computes IMulMat in the presence of semi-honest adversaries if the underlying PHE scheme is semantically secure.

*Proof:* We first prove the security against a semi-honest server. The server's view in an execution of our method $\pi$ is $\mathrm{view}_0^{\pi}(\langle W\rangle_1, \langle S\rangle_0) = \{\langle W\rangle_1, R_S, [\langle S\rangle_0]\}$, where $\langle W\rangle_1$ is the server's input, $R_S$ is the server's randomness, and $[\langle S\rangle_0]$ is the received message. We construct the simulator $Sim_0(\langle W\rangle_1, \mathtt{IMulMat}\{\langle S\rangle_0; \langle W\rangle_1\})$ as follows. (1) Choose a random tape $R_S^*$. (2) Initialize a matrix $S^*$ of the same size as $\langle S\rangle_0$ and choose each element in the matrix according to the distribution. (3) Compute $[S^*] \leftarrow \mathtt{EMat}(pk, S^*, 1)$. (4) Output $(\langle W\rangle_1, R_S^*, [S^*])$. If the underlying PHE is semantically secure, we can prove that $\{Sim_0(\langle W\rangle_1, \mathtt{IMulMat}\{\langle S\rangle_0; \langle W\rangle_1\})\}_{\langle W\rangle_1, \langle S\rangle_0 \in \{0,1\}^*} \overset{c}{\equiv} \{\mathrm{view}_0^{\pi}(\langle W\rangle_1, \langle S\rangle_0)\}_{\langle W\rangle_1, \langle S\rangle_0 \in \{0,1\}^*}$ 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 our method is $\mathrm{view}_1^{\pi}(\langle W\rangle_1, \langle S\rangle_0) = \{\langle S\rangle_0, R_C, [\langle W\rangle_1 \cdot \langle S\rangle_0 - R_0]\}$, where $\langle S\rangle_0$ is the client's input, $R_C$ is the client's randomness, and $[\langle W\rangle_1 \cdot \langle S\rangle_0 - R_0]$ is the received message. The simulator $Sim_1(\langle S\rangle_0, \mathtt{IMulMat}\{\langle S\rangle_0; \langle W\rangle_1\})$ is constructed as follows. (1) Choose a random tape $R_C^*$. (2) Initialize matrix $W^*$ of the same size as $\langle W\rangle_1$ and $R^*$ of the same size as $R_0$ and choose each element in the vectors according to the distribution. (3) Compute $[W^* \cdot \langle S\rangle_0 - R^*]$ from $W^*$, $\langle S\rangle_0$, and $R^*$. (4) Output $(\langle S\rangle_0, R_C^*, [W^* \cdot \langle S\rangle_0 - R^*])$. If the underlying PHE is semantically secure, we can prove that $\{Sim_1(\langle S\rangle_0, \mathtt{IMulMat}\{\langle S\rangle_0; \langle W\rangle_1\})\}_{\langle W\rangle_1, \langle S\rangle_0 \in \{0,1\}^*} \overset{c}{\equiv} \{\mathrm{view}_1^{\pi}(\langle W\rangle_1, \langle S\rangle_0)\}_{\langle W\rangle_1, \langle S\rangle_0 \in \{0,1\}^*}$ by a hybrid argument and complete the proof against semi-honest the client. ■

*2) Sharing $\langle \mathbf{W}\rangle_0 \cdot \langle \mathbf{S}\rangle_1$:* The ideal matrix multiplication sharing functionality is defined as $\{\langle W \cdot S\rangle_0; \langle W \cdot S\rangle_1\} \leftarrow \mathtt{IMulMat}'\{W; S\}$, where $\langle W \cdot S\rangle_0 + \langle W \cdot S\rangle_1 = W \cdot S$.

*Theorem 2:* Our method of sharing $\langle W\rangle_0 \cdot \langle S\rangle_1$ securely computes IMulMat'in the presence of semi-honest adversaries if the underlying PHE scheme is semantically secure.

The only difference between sharing $\langle W\rangle_1 \cdot \langle S\rangle_0$ and $\langle W\rangle_0 \cdot \langle S\rangle_1$ is whether the client encrypts and transmits the share of the multiplicand matrix $\langle W\rangle_0$ or multiplier matrix $\langle S\rangle_1$. The security proof is very similar and we omit it to avoid redundancy.

### B. *Security of PCmp*

The security of PCmpcan be reduced to the security of sharing the vector element-wise product $S \circ U$ in the offline phase. We define the ideal vector element-wise product functionality as $\{\langle S \circ U\rangle_0; \langle S \circ U\rangle_1\} \leftarrow \mathtt{IVEP}\{S; U\}$, where $\langle S \circ U\rangle_0 + \langle S \circ U\rangle_1 = S \circ U$.

*Theorem 3:* Our method of sharing $S \circ U$ securely computes IVEPin the presence of semi-honest adversaries if the underlying PHE scheme is semantically secure.

*Proof:* We first prove the security against a semi-honest server. The server's view in an execution of our method $\pi$ is $\mathrm{view}_0^{\pi}(V, U) = \{U, R_S, [V]\}$, where $U$ is the server's input, $R_S$ is the server's randomness, and $[V]$ is the received message. We construct the simulator $Sim_0(U, \mathtt{IVEP}\{U; V\})$ as follows. (1) Choose a random tape $R_S^*$. (2) Initialize a matrix $V^*$ of the same size as $V$ and choose each element in the matrix according to the distribution. (3) Compute $[V^*] \leftarrow \mathtt{Enc}(pk, V^*)$. (4) Output $(U, R_S^*, [V^*])$. If the underlying PHE is semantically secure, we can prove that $\{Sim_0(U, \mathtt{IVEP}\{U; V\})\}_{V,U \in \{0,1\}^*} \overset{c}{\equiv} \{\mathrm{view}_0^{\pi}(V, U)\}_{V,U \in \{0,1\}^*}$ 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 our method is $\text{view}_1^{\pi}(U, V) = \{V, R_C, [V \circ U - R]\}$, where $V$ is the client's input, $R_C$ is the client's randomness, and $[V \circ U - R]$ is the received message. The simulator $Sim_1(V, \texttt{IVEP}\{V; U\})$ is constructed as follows. (1) Choose a random tape $R_C^*$. (2) Initialize vector $U^*$ of the same size as $U$ and $R^*$ of the same size as $R$ and choose each element in the vectors according to the distribution. (3) Compute $[V \circ U^* - R^*]$ from $V$, $U^*$, and $R^*$. (4) Output $(V, R_C^*, [V \circ U^* - R^*])$. If the underlying PHE is semantically secure, we can prove that $\{Sim_1(V, \texttt{IVEP}\{V; U\})\}_{U,V\in\{0,1\}^*} \overset{c}{\equiv} \{\text{view}_1^{\pi}(U, V)\}_{U,V\in\{0,1\}^*}$ by a hybrid argument and complete the proof against semi-honest the client. ■

*Remarks:* The security of Skybolt relies on Random Number Generation (RNG), which is consistent with other private inference schemes, including GALA, Cheetah, and HELiKs. We employ several strategies to enhance security in case of poor-quality RNG: 1) Cryptographically secure random number generators (CSPRNGs) [37], such as OpenSSL's RAND-bytes, can be utilized to ensure high-quality randomness, even in low-entropy environments. 2) To further improve randomness, external entropy sources like hardware RNGs [38] (e.g., Intel's RDRAND) or operating system entropy pools [39] can be leveraged. 3) Merging outputs from different RNGs (CSPRNGs and external sources) helps mitigate risks from any single low-quality entropy source [40].

## VII. Evaluation

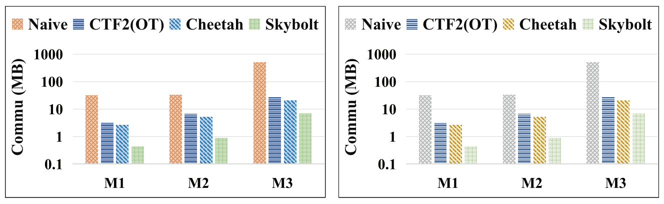
We implement Skybolt and perform experimental comparisons with state-of-the-art PI schemes.

### A. Evaluation Setup

*1) Implementation:* We instantiate Skybolt utilizing the SEAL library [41], configuring the parameters as follows: (1) The security parameter $\lambda$ is chosen to be 128 bits. (2) The plaintext modulus is set to $2^{37}$. (3) The ciphertext modulus is chosen to be $2^{105}$. (4) The ciphertext capacity is configured to 4096. To reduce inference latency, we leverage the parallel processing capabilities of GPUs to parallelize the online phase, implemented using CUDA C++. To compare with other counterparts, we acquire the source code of Gazelle [24], CrypTFlow2 [11], Cheetah [20], and HELiKs [13], and reimplement GALA [12] based on Gazelle. Notably, alongside the HE-based linear calculation method, CrypTFlow2 proposes an advanced OT-based method, and we denote them as CTF2(HE) and CTF2(OT) respectively.

*2) Testbed Environment:* All the evaluations are conducted on a laptop (Intel Core i5 CPU with 4 2.2 GHz cores and 8 GB memory) serving as the client, and a machine (Intel Xeon with 14 2.4 GHz cores, 128 GB memory and NVIDIA 4090 GPU) as the server. The two machines are configured in a LAN network setting with a bandwidth of 350 MBps.

*3) Datasets and Model Architectures:* We perform a comparative evaluation with other counterparts across diverse well-established CNN architectures, such as SqueezeNet, ResNet (with over 23 million parameters), and DenseNet.



(a) Runtime. (b) Communication costs.

Fig. 5. Runtime and communication cost comparison of matrix multiplication. The matrices dimensions within M1 to M3 are $(m, n, k)$: (32, 32, 16), (16, 64, 32), (64, 32, 128), respectively.

TABLE IV
Runtime and Communication Cost Comparison of Nonlinear Layers

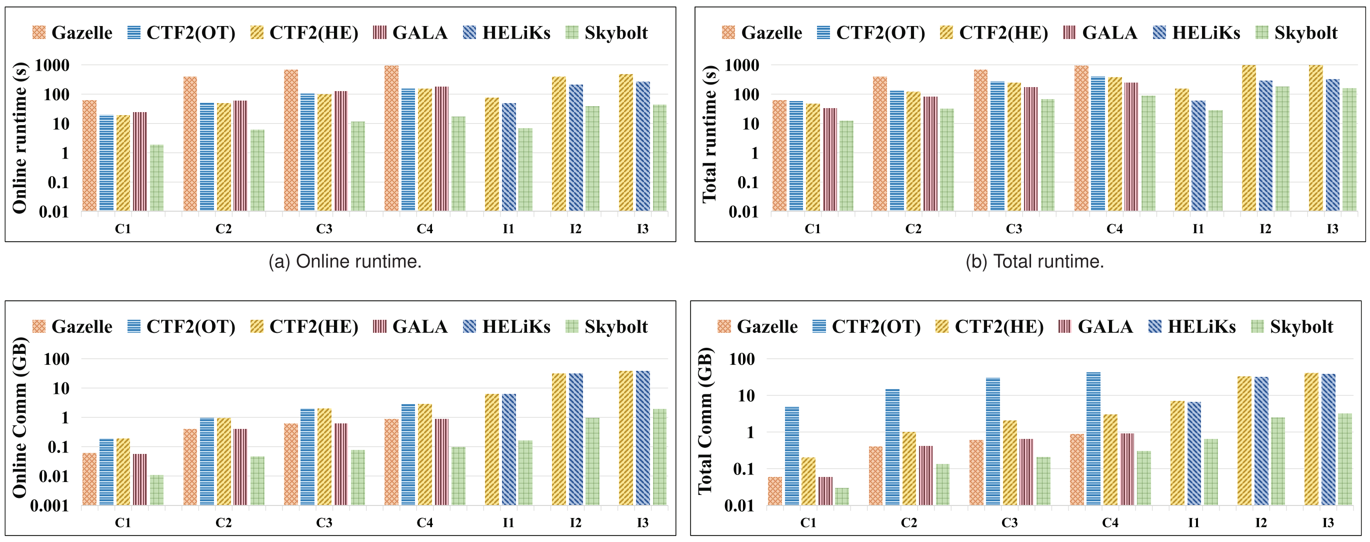
| Network | Protocol | Runtime (S) | | | Communication (GB) | | |
|---|---|---|---|---|---|---|---|
| | | Online | Offline | Total | Online | Offline | Total |
| N1 | CrypTFlow2 | 27.6 | **0** | 27.6 | 2.22 | **0** | 2.22 |
| | Cheetah | 23.1 | **0** | 23.1 | 0.13 | **0** | **0.13** |
| | Skybolt | **0.4** | 1.9 | **2.4** | **0.05** | 0.1 | 0.15 |
| N2 | CrypTFlow2 | 33.5 | **0** | 33.5 | 2.94 | **0** | 2.94 |
| | Cheetah | 18.6 | **0** | 18.6 | 0.17 | **0** | **0.17** |
| | Skybolt | **0.5** | 2.2 | **2.6** | **0.06** | 0.16 | 0.22 |
| N3 | CrypTFlow2 | 119.6 | **0** | 119.6 | 9.66 | **0** | 9.66 |
| | Cheetah | 97.5 | **0** | 97.5 | 0.59 | **0** | **0.59** |
| | Skybolt | **1.2** | 9.1 | **10.3** | **0.17** | 0.54 | 0.71 |
| N4 | CrypTFlow2 | 21.2 | **0** | 21.2 | 1.74 | **0** | 1.74 |
| | Cheetah | 9.4 | **0** | 9.4 | 0.1 | **0** | **0.1** |
| | Skybolt | **0.4** | 1.2 | **1.6** | **0.02** | 0.1 | 0.12 |
| N5 | CrypTFlow2 | 197.9 | **0** | 197.9 | 16.18 | **0** | 16.18 |
| | Cheetah | 165.7 | **0** | 165.7 | 0.99 | **0** | **0.99** |
| | Skybolt | **2.3** | 19.4 | **21.7** | **0.26** | 0.94 | 1.2 |
| N6 | CrypTFlow2 | 22.3 | **0** | 22.3 | 1.74 | **0** | 1.74 |
| | Cheetah | 9.1 | **0** | 9.1 | 0.1 | **0** | **0.1** |
| | Skybolt | **0.3** | 1.3 | **1.6** | **0.02** | 0.1 | 0.12 |

This assessment encompasses classical CIFAR and ImageNet datasets.

### B. Microbenchmarks

*1) Linear Layer Functions:* To demonstrate the efficacy of GDPA, we evaluate the HE matrix multiplication approaches of the naive method, Cheetah, HELiKs, and Skybolt on three different matrix multiplications, comparing their runtime and communication costs. The practical runtime and communication costs of these methods are presented in Fig. 5. Skybolt outperforms the state-of-the-art approach HELiKs by a factor of $2.7-3.1$ in terms of speed. Moreover, the communication overhead of Skybolt is $3.1-3.9\times$ smaller than that of HELiKs. This performance improvement is attributed to the innovative GDPA, which avoids both PRotand extraction operations, and makes better use of ciphertext space.

*2) Nonlinear Layer Functions:* To compare the PCmpprotocol with other counterparts, we evaluated their performance in computing the activation and max-pooling layers of SqueezeNet, ResNet50, and DenseNet121, denoted as N1 to N6. Since HELiKs does not offer enhancements for nonlinear calculations, our comparison focuses on PCmpagainst state-of-the-art nonlinear protocols, CrypTFlow2 and Cheetah.



(a) Online runtime.

(b) Total runtime.

(c) Online communication costs.

(d) Total communication costs.

Fig. 6. Runtime and communication cost comparison on popular CNNs.

Table IV illustrates the runtime and communication costs of these protocols. In the online latency-sensitive phase, Skybolt exhibits a remarkable 23.6 – 82× speedup over Cheetah, concomitant with a 2.6 – 5× reduction in communication overhead, indicating significantly lower inference latency in Skybolt. In terms of total runtime, Skybolt maintains a 5.6 – 9.8× acceleration over Cheetah. These improvements are attributed to our online phase exclusively involving lightweight addition and multiplication of ASS, allowing us to accelerate by GPU parallelization.

## C. End-to-End Inference Evaluation

Prior works have applied their source code to specific CNNs for CIFAR or ImageNet tasks. For comparison, we subject Skybolt to CNNs and datasets that align with existing works. The experimental framework encompasses ResNet18, ResNet50, ResNet101, and ResNet152 applied to the CIFAR-10 classification task, denoted as C1 to C4. Additionally, SqueezeNet, ResNet50, and DenseNet121 are employed for the ImageNet classification task, identified as I1 to I3.

*1) Runtime and Communication Costs:* The results in Fig. 6 illustrate the online and total runtime and communication costs. The online runtime of Skybolt achieves a 5.4 – 7.3× reduction compared to HELiKs, indicating a significantly lower inference latency. Concerning the total runtime, Skybolt remains 1.6 – 2.2× faster than HELiKs. Moreover, our online and total communication overhead are 20.1 – 39.6× and 10.3 – 12.8× smaller than HELiKs. This performance enhancement is ascribed to our novel packing algorithm and the lightweight online phase that only involves ASS and supports GPU acceleration.

Remarkably, compared to these PI approaches, Skybolt provides additional protection to model parameters through a model-sharing mechanism, performing twice the privacy-preserving linear calculations. Despite this increased computational load, Skybolt still exhibits better performance. The performance improvement will be more pronounced when GDPA and PCmpare applied in the same scenario as HELiKs.

TABLE V
ACCURACY COMPARISON

| Benchmarks | C1 | C2 | C3 | C4 | I1 | I2 | I3 |
|---|---|---|---|---|---|---|---|
| Plaintext | 98.65% | 98.97% | 99.24% | 99.37% | 55.47% | 76.11% | 74.07% |
| Skybolt | 98.59% | 98.94% | 99.19% | 99.35% | 55.40% | 76.05% | 74.03% |

*2) Inference Accuracy:* Skybolt upholds inference accuracy by faithfully completing CNN inference without approximation. Any potential accuracy reduction is ascribed to the replacement of floating-point numbers with fixed-point numbers, which is an essential step in PI schemes. Prior research [11], [12], [20] have proven that the impact of this inevitable replacement on accuracy is negligible. The results in Table V demonstrate that the accuracy loss of Skybolt is negligible.

## VIII. Conclusion

In this paper, we propose Skybolt, a single-server PI framework for outsourcing CNN inference, which is even more efficient than HELiKs. Skybolt is founded on 2PC protocols that ensure both client and model owner privacy, faithfully implementing CNN inference without approximation. For the linear calculation protocol, we develop a homomorphic matrix encryption scheme that supports matrix multiplication with reduced computation and communication costs. For the nonlinear calculation protocol, we successfully construct a lightweight online phase that only involves elementary operations of secret shares, in contrast to existing protocols entailing heavy GC or OT. Evaluated on well-established models, Skybolt reduces inference latency by 5.4 – 7.3×, along with a 20.1 – 39.6× decrease in communication cost in comparison to HELiKs.

## References

[1] C. Wu, X.-J. Wu, T. Xu, Z. Shen, and J. Kittler, "Motion complement and temporal multifocusing for skeleton-based action recognition," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 34, no. 1, pp. 34–45, Jan. 2024.

[2] S. Zhao, T. Xu, X.-J. Wu, and J. Kittler, "Distillation, ensemble and selection for building a better and faster Siamese based tracker," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 34, no. 1, pp. 182–194, Jan. 2024.

[3] B. Xu, H. Liang, W. Gong, R. Liang, and P. Chen, "A visual representation-guided framework with global affinity for weakly supervised salient object detection," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 34, no. 1, pp. 248–259, Jan. 2024.

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

[5] M. Li, S. S. M. Chow, S. Hu, Y. Yan, C. Shen, and Q. Wang, "Optimizing privacy-preserving outsourced convolutional neural network predictions," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 3, pp. 1592–1604, May/Jun. 2022.

[6] *Sygic Official Website*. Accessed: Apr. 5, 2025. [Online]. Available: https://www.sygic.com

[7] X. Yang, J. Chen, K. He, H. Bai, C. Wu, and R. Du, "Efficient privacy-preserving inference outsourcing for convolutional neural networks," *IEEE Trans. Inf. Forensics Security*, vol. 18, pp. 4815–4829, 2023.

[8] X. Yang et al., "Fregata: Fast private inference with unified secure two-party protocols," *IEEE Trans. Inf. Forensics Security*, vol. 19, pp. 8472–8484, 2024.

[9] *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*, Gen. Data Protection Regulation (GDPR), Brussels, Belgium, 2012.

[10] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "Delphi: A cryptographic inference service for neural networks," in *Proc. USENIX Secur. Symp.*, 2020, pp. 2505–2522.

[11] D. Rathee et al., "CrypTFlow2: Practical 2-Party secure inference," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2020, pp. 325–342.

[12] 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.

[13] S. Balla and F. Koushanfar, "HELiKs: HE linear algebra kernels for secure inference," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2023, pp. 2306–2320.

[14] Y. Liu, Y. Yang, Z. Ma, X. Liu, Z. Wang, and S. Ma, "PE-HEALTH: Enabling fully encrypted CNN for health monitor with optimized communication," in *Proc. IEEE/ACM 28th Int. Symp. Quality Service (IWQoS)*, Jun. 2020, pp. 1–10.

[15] H. Huang, Q. Wang, X. Gong, and T. Wang, "Orion: Online backdoor sample detection via evolution deviance," in *Proc. 32nd Int. Joint Conf. Artif. Intell.*, Macao, China, Aug. 2023, pp. 864–874.

[16] 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.

[17] 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. 395–412.

[18] A. Davidson, G. Pestana, and S. Celi, "FrodoPIR: Simple, scalable, single-server private information retrieval," in *Proc. Priv. Enhancing Technol.*, vol. 2023, Jan. 2023, pp. 365–383.

[19] M. Zhou, W.-K. Lin, Y. Tselekounis, and E. Shi, "Optimal single-server private information retrieval," in *Proc. EUROCRYPT*. Cham, Switzerland: Springer, Jan. 2023, pp. 395–425.

[20] Z. Huang, W. Lu, C. Hong, and J. Ding, "Cheetah: Lean and fast secure two-party deep neural network inference," in *Proc. USENIX Secur. Symp.*, 2022, pp. 1–19.

[21] Y. Zhang, R. Zhao, X. Xiao, R. Lan, Z. Liu, and X. Zhang, "HF-TPE: High-fidelity thumbnail{-} preserving encryption," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 32, no. 3, pp. 947–961, Mar. 2022.

[22] Y. Xian, X. Wang, X. Wang, Q. Li, and X. Yan, "Spiral-transform-based fractal sorting matrix for chaotic image encryption," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 8, pp. 3320–3327, Aug. 2022.

[23] P. Liu, X. Wang, and Y. Su, "Image encryption via complementary embedding algorithm and new spatiotemporal chaotic system," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 33, no. 5, pp. 2506–2519, May 2023.

[24] C. Juvekar, V. Vaikuntanathan, and A. P. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *Proc. USENIX Secur. Symp.*, Jan. 2018, pp. 1651–1669.

[25] X. Ding, X. Zhang, N. Ma, J. Han, G. Ding, and J. Sun, "RepVGG: Making VGG-style ConvNets great again," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 13728–13737.

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

[27] 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.

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

[29] N. Yan et al., "Efficient and straggler-resistant homomorphic encryption for heterogeneous federated learning," in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, May 2024, pp. 791–800.

[30] B. Jayaraman and D. Evans, "Evaluating differentially private machine learning in practice," in *Proc. USENIX Secur. Symp.*, Aug. 2019, pp. 1895–1912.

[31] M. Abadi et al., "Deep learning with differential privacy," in *Proc. CCS*, Oct. 2016, pp. 308–318.

[32] X. Liu, Z. Liu, Q. Li, K. Xu, and M. Xu, "Pencil: Private and extensible collaborative learning without the non-colluding assumption," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2024.

[33] G. Xu, G. Li, S. Guo, T. Zhang, and H. Li, "Secure decentralized image classification with multiparty homomorphic encryption," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 33, no. 7, pp. 3185–3198, Jul. 2023.

[34] Z. Ghodsi, N. K. Jha, B. Reagen, and S. Garg, "Circa: Stochastic ReLUs for private deep learning," in *Proc. NeurIPS*, Jan. 2021, pp. 2241–2252.

[35] H. Peng et al., "AutoReP: Automatic ReLU replacement for fast private network inference," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2023, pp. 5155–5165.

[36] 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.

[37] N. Ferguson, B. Schneier, and T. Kohno, *Cryptography Engineering: Design Principles and Practical Applications*. Hoboken, NJ, USA: Wiley, 2010.

[38] M. Bakiri, C. Guyeux, J.-F. Couchot, L. Marangio, and S. Galatolo, "A hardware and secure pseudorandom generator for constrained devices," *IEEE Trans. Ind. Informat.*, vol. 14, no. 8, pp. 3754–3765, Aug. 2018.

[39] P. Kietzmann, T. C. Schmidt, and M. Wählisch, "A guideline on pseudorandom number generation (PRNG) in the IoT," *ACM Comput. Surveys*, vol. 54, no. 6, pp. 1–38, Jul. 2022.

[40] S. Tezuka and P. L'Ecuyer, "Efficient and portable combined Tausworthe random number generators," *ACM Trans. Model. Comput. Simul.*, vol. 1, no. 2, pp. 99–112, Apr. 1991.

[41] (2020). *SEAL*. [Online]. Available: https://github.com/microsoft/SEAL

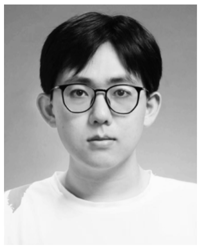
**Xuanang Yang** received the B.E. degree in software engineering from Wuhan University, Hebei, China, in 2019, where he is currently pursuing the Ph.D. degree with the School of Cyber Science and Engineering. His research outcomes have appeared in IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY. His 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 150 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.

**Yuqing Li** (Member, IEEE) received the Ph.D. degree in electronic engineering from Shanghai Jiao Tong University, Shanghai, China, in 2019. From 2019 to 2020, she was a Post-Doctoral Fellow with Hong Kong University of Science and Technology, Hong Kong. From 2020 to 2022, she was a Researcher with the Huawei Hong Kong Research Center, Hong Kong. She is currently an Associate Professor with the School of Cyber Science and Engineering, Wuhan University, Wuhan, China. Her research interests include data privacy and security, distributed machine learning, and edge computing.

**Kun He** (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.

**Xiaojie Huang** received the B.E. degree in information security from Wuhan University, Wuhan, China, in 2023, where he is currently pursuing the M.S. degree with the School of Cyber Science and Engineering. His research interests include privacy-preserving machine learning and secure multi-party computation.



**Zikuan Jiang** received the B.E. degree in cybersecurity from Wuhan University, Wuhan, China, in 2023, 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.

**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 Wuhan University. She has published more than 100 research papers in many international journals and conferences, such as IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, *International Journal of Parallel and Distributed Systems*, USENIX Security, CCS, and INFOCOM. Her current research interests include network security, wireless networks, cloud computing, and mobile 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 research interests include distributed machine learning.