GALA: Greedy ComputAtion for Linear Algebra in Privacy-Preserved Neural Networks

Qiao Zhang, Chunsheng Xin, and Hongyi Wu
Old Dominion University, Norfolk, VA 23529, USA
Email: qzhan002@odu.edu, cxin@odu.edu, h1wu@odu.edu

Abstract—Machine Learning as a Service (MLaaS) is enabling a wide range of smart applications on end devices. However, privacy still remains a fundamental challenge. The schemes that exploit Homomorphic Encryption (HE)-based linear computations and Garbled Circuit (GC)-based nonlinear computations have demonstrated superior performance to enable privacy-preserved MLaaS. Nevertheless, there is still a significant gap in the computation speed. Our investigation has found that the HE-based linear computation dominates the total computation time for state-of-the-art deep neural networks. Furthermore, the most time-consuming component of the HE-based linear computation is a series of Permutation (Perm) operations that are imperative for dot product and convolution in privacy-preserved MLaaS. This work focuses on a deep optimization of the HE-based linear computations to minimize the Perm operations, thus substantially reducing the overall computation time. To this end, we propose *GALA: Greedy computAtion for Linear Algebra* in privacy-preserved neural networks, which views the HE-based linear computation as a series of Homomorphic Add, Mult and Perm operations and chooses the least expensive operation in each linear computation step to reduce the overall cost. GALA makes the following contributions: (1) It introduces a row-wise weight matrix encoding and combines the share generation that is needed for the GC-based nonlinear computation, to reduce the Perm operations for the dot product; (2) It designs a first-Add-second-Perm approach (named *kernel grouping*) to reduce Perm operations for convolution. As such, GALA efficiently reduces the cost for the HE-based linear computation, which is a critical building block in almost all of the recent frameworks for privacy-preserved neural networks, including GAZELLE (Usenix Security'18), DELPHI (Usenix Security'20), and CrypTFlow2 (CCS'20). With its deep optimization of the HE-based linear computation, GALA can be a plug-and-play module integrated into these systems to further boost their efficiency. Our experiments show that it achieves a significant speedup up to $700\times$ for the dot product and $14\times$ for the convolution computation under different data dimensions. Meanwhile, GALA demonstrates an encouraging runtime boost by $2.5\times$, $2.7\times$, $3.2\times$, $8.3\times$, $7.7\times$, and $7.5\times$ over GAZELLE and $6.5\times$, $6\times$, $5.7\times$, $4.5\times$, $4.2\times$, and $4.1\times$ over CrypTFlow2, on AlexNet, VGG, ResNet-18, ResNet-50, ResNet-101, and ResNet-152, respectively.

I. INTRODUCTION

Deep Learning (DL) is becoming prevalent and pervasive, e.g., for pattern recognition [42], medical diagnosis [22], speech recognition [20] and credit-risk assessment [24]. In particular, Convolutional Neural Network (CNN) has demonstrated superior performance in computer vision such as image classification [40], [62] and facial recognition [58]. Since designing and training a deep neural network model requires intensive resource and DL talents, cloud providers begin to offer Machine Learning as a Service (MLaaS) [70], where a proprietary DL model is trained and hosted on a cloud. Clients can utilize the service by simply sending queries (inference) to the cloud and receiving results through a web portal. While this emerging cloud service is embraced as an important tool for efficiency and productivity, the interaction between clients and cloud servers leads to new vulnerabilities. This work focuses on the development of privacy-preserved and computationally efficient MLaaS.

Although communication can be readily secured from end to end, privacy still remains a fundamental challenge. On the one hand, the clients must submit their data to the cloud for inference, but they want the data privacy well protected, preventing curious cloud provider or attacker with access to the cloud from mining valuable information. In many domains such as health care [49] and finance [63], data are extremely sensitive. For example, when patients transmit their physiological data to the server for medical diagnosis, they do not want anyone (including the cloud provider) to see it. Regulations such as Health Insurance Portability and Accountability Act (HIPAA) [8] and the recent General Data Protection Regulation (GDPR) in Europe [25] have been in place to impose restrictions on sharing sensitive user information. On the other hand, cloud providers do not want users to be able to extract their proprietary model that has been trained with significant resource and efforts [66]. Furthermore, the trained model contains private information about the training data set and can be exploited by malicious users [61], [64], [69]. To this end, there is an urgent need to develop effective and efficient schemes to ensure that, in MLaaS, a cloud server does not have access to users' data and a user cannot learn the server's model.

A series of efforts have been made to enable privacy-preserved MLaaS, by leveraging cryptographic techniques as summarized below. The first is the *Homomorphic Encryption (HE)-Based Approaches*. For example, in CryptoNets [27], Faster CryptoNets [19] and CryptoDL [34], the client encrypts data using HE and sends the encrypted data to the server. The server performs polynomial computations (e.g., addition and multiplication) over encrypted data to calculate an encrypted inference result. The client finally obtains the inference outcome after decryption. E2DM [37] adopts a more efficient HE (i.e., packed HE [14]) which packs multiple messages into one

ciphertext and thus improves the computation efficiency. The second approaches is based on *Garbled Circuit (GC)* [72]. DeepSecure [57] and XONN [55] binarize the computations in neural networks and employ GC to obliviously obtain the prediction without leaking sensitive client data. The third approach exploits *Secret Sharing (SS)*. SS is used in [68] and [67] to split the client data into shares. The server only owns one share of the data. The computations are completed by interactive share exchanges. In addition, Differential Privacy (DP) [60], [7], [53] and Secure Enclave (SE) [45], [51], [10], [75] are also explored to protect data security and privacy in neural networks. In order to deal with different properties of linearity (weighted sum and convolution functions) and non-linearity (activation and pooling functions) in neural network computations, several efforts have been made to orchestrate multiple cryptographic techniques to achieve better performance [74], [43], [38], [48], [56], [44], [76], [18], [73], [47], [71], [16], [12], [41], [54], [46]. Among them, the schemes with HE-based linear computations and GC-based nonlinear computations (called the HE-GC neural network framework hereafter) demonstrate superior performance [43], [38], [44], [46]. Specifically, the GAZELLE framework [38] represents the state-of-the-art design for the HE-based linear computation and achieves a speedup of three orders of magnitude than the classic CryptoNets inference system [27].

Despite the rapid improvement, there is still a significant gap in computation speed, rendering the existing schemes infeasible for practical applications. For example, the time constraints in many real-time applications (such as speech recognition) are within a few seconds [2], [4]. In contrast, our benchmark has shown that GAZELLE takes 43 seconds and 659 seconds to run the well-known deep neural networks ResNet-18 and ResNet-152 [32] on an Intel i7-8700 3.2GHz CPU (see detailed experimental settings in Sec. IV), which renders it impractical in real-world applications.

This performance gap motivates us to further improve the efficiency of the HE-GC neural network frameworks. In deep neural network, both the fully-connected and convolutional layers are based on the linear computation, while the activation functions perform nonlinear computation. The former dominates the total computation time in state-of-the-art deep neural networks. For example, the runtime of the nonlinear computation in GAZELLE is merely 2.3%, 1.8%, 1.7%, 1.5%, 1.6%, and 2%, respectively, on AlexNet [40], VGG [62], ResNet-18 [32], ResNet-50 [32], ResNet-101 [32], and ResNet-152 [32]. The nonlinear cost in the original plaintext models is even lower (averaged 1.7%). This indicates a great potential to speed up the overall system through optimizing linear computations. Although a few recent approaches, e.g., DELPHI [46] and CrypTFlow2 [54], perform better than GAZELLE in terms of the overall system runtime, they all inherit the HE-based linear computation in GAZELLE. This work contributes a solid optimization on the HE-based linear computation (i.e., dot product and convolution), which can be integrated into those systems (including GAZELLE, DELPHI and CrypTFlow2) to further improve their overall system performance. The HE-based computation consists of three basic operations: Homomorphic Addition (Add), Multiplication (Mult), and Permutation (Perm). Our investigation has shown that the most time-consuming part of the HE-based computation is a series of Perm operations that are impera-

tive to enable dot product and convolution. Our experiments show that Perm is 56 times slower than Add and 34 times slower than Mult. As shown in Table I, in the dot product by multiplying a 2×2048 matrix with a length-2048 vector, the cost in GAZELLE is dominated by Perm, which takes about *98% of the computation time*. This observation motivates the proposed linear optimization, which aims to minimize the Perm operations, thus substantially reducing the overall computation time. With less Perm operations, the proposed approach demonstrates $10\times$ speedup in the above matrix-vector computation.

TABLE I.   COST OF MATRIX-VECTOR MULTIPLICATION (TIME IN MILLIONSECOND).

Method	Total (ms)	Perm		Mult		Add	
		#	time	#	time	#	time
GAZELLE	2	11	1.96	2	0.01	11	0.037
Proposed	0.2	1	0.17	2	0.01	1	0.003

This significant speedup lies in a simple and efficient idea to choose the least expensive operation in each linear computation step to reduce the overall cost. We name the proposed approach *GALA: Greedy computAtion for Linear Algebra* in privacy-preserved neural networks. We view the HE-based linear computation as a series of Homomorphic Add, Mult and Perm operations. The two inputs are the encrypted vector (or channels) from the client and the plaintext weight matrix (or kernel) from the server. The output is the encrypted dot product (or convolution). The objective in each step is to choose the most efficient operations in the descending priorities of Add, Mult and Perm. To this end, we (1) design a row-wise weight matrix encoding with combined share generation[1] (i.e., a row-encoding-share-RaS (Rotated and Sum) approach) to reduce the number of Perm operations in dot product by $\log_2 \frac{n}{n_o}$ where n is the number of slots in a ciphertext and n_o is the output dimension of dot product, and (2) propose a first-Add-second-Perm approach (named *kernel grouping*) to reduce the number of Perm operations of convolution by a factor of $\frac{c_i}{c_n}$ where c_i and c_n are respectively the number of channels in input data and the number of channels that can be packed in a ciphertext. n is always greater than and can be up to 8192 times of n_o depending on the dimension of dataset [5] and HE implementation [59].

At the same time, $\frac{c_i}{c_n}$ is at least one and can be up to 256 for state-of-the-art neural network architectures such as ResNets [32] where the large channels, i.e., 1024 and 2048, and small kernels size, i.e., 1×1 and 3×3, are adopted. The larger input data from users will result in smaller c_n, which accordingly contributes to higher speedup especially in the state-of-the-art CNNs. As such, GALA efficiently boosts the performance of HE-based linear computation, which is a critical building block in almost all of the recent frameworks for privacy-preserved neural networks, e.g., GAZELLE, DELPHI, and CrypTFlow2. Furthermore, GALA's deep optimization of the HE-based linear computation can be integrated as a plug-and-play module into these systems to further improve their overall efficiency. For example, GALA can serve as a computing module in the privacy-preserved DL platforms, MP2ML [12] and CrypTFlow [41], which are compatible with

---

[1]The resultant linear output will be shared between server and client as the input of GC-based nonlinear computation.

the user-friendly TensorFlow [6] DL framework. Our experiments show that GALA achieves a significant speedup up to $700\times$ for the dot product and $14\times$ for the convolution computation under various data dimensions. Meanwhile, GALA demonstrates an encouraging runtime boost by $2.5\times$, $2.7\times$, $3.2\times$, $8.3\times$, $7.7\times$, and $7.5\times$ over GAZELLE and $6.5\times$, $6\times$, $5.7\times$, $4.5\times$, $4.2\times$, and $4.1\times$ over CrypTFlow2, on AlexNet, VGG, ResNet-18, ResNet-50, ResNet-101, and ResNet-152, respectively. More details are given in Sec. IV.

The rest of the paper is organized as follows. Sec. II introduces the primitives that GALA relies on. Sec. III describes the design details of GALA. The experimental results are illustrated and discussed in Sec. IV. Finally, Sec. V concludes the work.

II. PRELIMINARIES

In this section, we introduce the overall system architecture and threat model, as well as cryptographic tools used in GALA.

A. System Model

We consider an MLaaS system shown in Fig. 1. The client owns private data. The server is in the cloud and has a well-trained deep learning model to provide the inference service based on the received client's data. For example, a doctor sends an encrypted medical image (such as a chest X-ray) to the server, which runs the neural network model and returns the encrypted prediction to the doctor. The prediction is then decrypted into a plaintext result to assist diagnosis and health care planning.

While various deep learning techniques can be employed to enable MLaaS, we focus on the Convolutional Neural Network (CNN), which has achieved wide success and demonstrated superior performance in computer vision such as image classification [40], [62] and face recognition [58]. A CNN consists of a stack of layers to learn a complex relation among the input data, e.g., the relations between pixels of an input image. It operates on a sequence of linear and nonlinear transformations to infer a result, e.g., whether an input medical image indicates that the patient has tuberculosis. The linear transformations are in two typical forms: *dot product* (i.e., matrix-vector multiplication) and *convolution*. The nonlinear transformations leverage *activations* such as the Rectified Linear Unit (*ReLu*) to approximate complex functions [35] and *pooling* (e.g., max pooling and mean pooling) for dimensionality reduction. CNN repeats the linear and nonlinear transformations recursively to reduce the high-dimensional input data to a low-dimensional feature vector for classification at the *fully connected layer*.
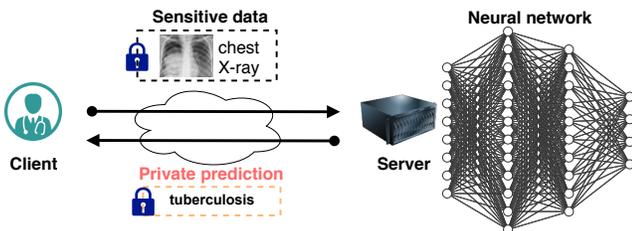
Without losing generality, we use image classification as an example in the following discussion, aiming to provide a lucid understanding of the CNN architecture as illustrated in Fig. 2.

Convolution. The input to a convolutional layer has the dimension of $u_w \times u_h \times c_i$, where u_w and u_h are the width and height of the input feature map and c_i is the number of the feature maps (or channels). For the first layer, the feature maps are simply the input image(s). Hereafter, we use the subscripts i and o to denote the input and output, respectively. The input is convolved with c_o groups of kernels. The size of each group of kernels is $k_w \times k_h \times c_i$, in which k_w and k_h are the width and height of the kernel. The number of channels of each kernel group must match with the input, i.e., c_i. The convolution will produce the feature output, with a size of $w_o \times h_o \times c_o$. Specifically, the process of convolution can be visualized as placing the kernel at different locations of the input data. At each location, a sum of element-wise product is computed between the kernel and corresponding data values within the kernel window, as shown in Fig. 2.

Dot Product. The last convolutional layer is typically connected with the fully-connected layer, which computes the weighted sum, i.e., a dot product between the weight matrix w of size $n_o \times n_i$ and a flatten feature vector of size $n_i \times 1$. The output is a vector with the size of $n_o \times 1$. Each element of the output vector is calculated as a sum of element-wise product between one row of weight matrix and the flatten feature vector, as shown in Fig. 2.

Activation. Nonlinear activation is applied to convolutional and weighted-sum outputs in an elementwise manner, as shown in Fig. 2. The commonly used activation functions include *ReLu*, $f(x) = \max\{0, x\}$; *sigmoid*, $f(x) = \frac{1}{1+e^{-x}}$; and *tanh*, $f(x) = \frac{e^{2x}-1}{e^{2x}+1}$. The last layer uses the *softmax* function $f(x) = \frac{e^x}{\sum_i e^{x(i)}}$ to normalize the output into a probability vector.

Pooling. Pooling conducts downsampling to reduce dimensionality. In this work, we consider *Mean pooling*, which is implemented in CryptoNets and also commonly adopted in state-of-the-art CNNs. It splits a feature map into regions and averages the regional elements. Compared to max pooling (another pooling function which selects the maximum value

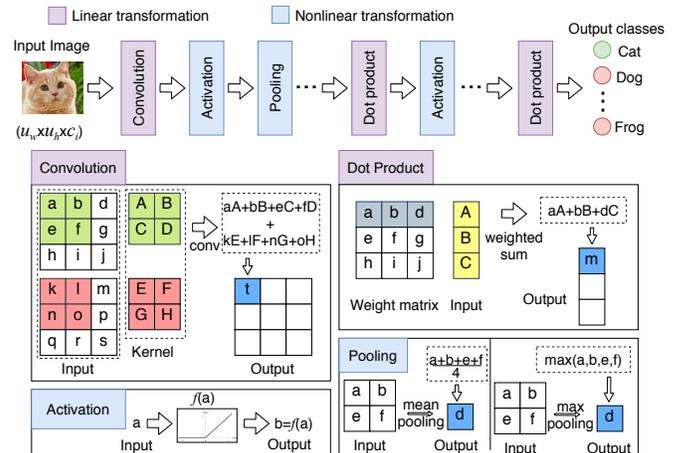Fig. 1. An overview of the MLaaS system.

Fig. 2. An overview of CNN model.

in each region), authors in [77] have claimed that while the max and mean pooling functions are rather similar, the use of mean pooling encourages the network to identify the complete extent of the object, which builds a generic localizable deep representation that exposes the implicit attention of CNNs on an image. In HE-GC neural network frameworks, the mean pooling is easily conducted on the shares of both client and server, without extra cost [44], [38].

In this work, we mainly focus on privacy-preserved linear optimization (i.e., convolution and dot product). The privacy-preserved nonlinear optimizations (especially activations) are based on GC as introduced in other HE-GC approaches such as GAZELLE [38].

B. Threat Model

Similar to GAZELLE [38] and other previous works, namely the SecureML [48], MiniONN [44], DeepSecure [57] and XONN [55], we adopt the semi-honest model, in which both parties try to learn additional information from the message received (assuming they have a bounded computational capability). That is, the client $\mathcal{C}$ and server $\mathcal{S}$ will follow the protocol, but $\mathcal{C}$ wants to learn model parameters and $\mathcal{S}$ attempts to learn the client's data. Note that, many applications are built on well-known deep network structures such as AlexNet [40], VGG-16/19 [62] and ResNet-50 [32]. Hence we do not intend to protect the structure (number of layers, kernel size, etc), but focus on the protection of model parameters. In the case that the implemented structure is proprietary and has to be protected, the server can introduce redundant layers and kernels to hide the real structure at a computational expense [44], [38]. Hence, the overarching goal is to make the server oblivious of the private data from the client, and prevent the client from learning model parameters of the server. GAZELLE has demonstrated the security of HE-GC neural network framework according to the cryptographic standard of ideal/real security [29], [28], [30]. The same security framework is adopted in this work.

Note that, while the client can use the server's prediction service as a blackbox oracle to extract the model [66], [69], or even infer the training set [26], [50], [61], GALA does not aim to protect against the black-box attack. Instead, it focuses on protecting the input data and the model parameters during the inference process, which stays in line with the threat model of GAZELLE [38], SecureML [48], DELPHI [46], CrytoFlow2 [54], etc., the output of neural network model is returned to the client which decrypts the result and gets the plaintext prediction.

C. Cryptographic Tools

The proposed privacy-preserved deep neural network framework, i.e., GALA, employs three fundamental cryptographic tools as outlined below.

(1) Packed Homomorphic Encryption. Homomorphic Encryption (HE) is a cryptographic primitive that supports meaningful computations on encrypted data without the decryption key, which has found increasing applications in data communications, storage and computations [65]. Traditional HE operates on individual ciphertext [52], while the *packed homomorphic encryption* (PHE) enables packing of multiple values into a single ciphertext and performs component-wise homomorphic computation in a Single Instruction Multiple Data (SIMD) manner [14] to take advantage of parallelism. Among various PHE techniques, our work builds on the Brakerski-Fan-Vercauteren (BFV) scheme [23], which involves four parameters[2]: 1) ciphertext modulus q, 2) plaintext modulus p, 3) the number of ciphertext slots n, and 4) a Gaussian noise with a standard deviation σ. The secure computation involves two parties, i.e., the client $\mathcal{C}$ and server $\mathcal{S}$.

In PHE, the encryption algorithm encrypts a plaintext message vector x from $\mathbb{Z}^n$ into a ciphertext $[x]$ with n slots. We denote $[x]_{\mathcal{C}}$ and $[x]_{\mathcal{S}}$ as the ciphertext encrypted by client $\mathcal{C}$ and server $\mathcal{S}$, respectively. The decryption algorithm returns the plaintext vector x from the ciphertext $[x]$. Computation can be performed on the ciphertext. In a general sense, an evaluation algorithm inputs several ciphertext $[x_1], [x_2], \cdots$, and outputs a ciphertext $[x'] = f([x_1], [x_2], \cdots)$. The function f is constructed by homomorphic addition (Add), Multiplication (Mult) and permutation (Perm). Specifically, Add($[x],[y]$) outputs a ciphertext $[x+y]$ which encrypts the elementwise sum of x and y. Mult($[x],s$) outputs a ciphertext $[x \odot s]$ which encrypts the elementwise multiplication of x and plaintext s. It is worth pointing out that GALA is designed to require scalar multiplication between a ciphertext and a plaintext only, but not the much more expensive multiplication between two ciphertext. Hereafter, we use ScMult to denote the scalar multiplication involved in GALA. Perm($[x]$) permutes the n elements in $[x]$ into another ciphertext $[x_\pi]$, where $x_\pi = (x(\pi_0), x(\pi_1), \cdots)$ and π_i is a permutation of $\{0, 1, \cdots, n - 1\}$. Additionally, the computation cost for a series of Perm operations on the same ciphertext can be optimized by first conducting one Perm Decomposition (DecPerm) on the ciphertext and then doing the corresponding series of Hoisted Perm (HstPerm) operations [38]. Since only one DecPerm is involved, it can amortize the total permutation time.

The run-time of Add and ScMult is significantly lower than that of Perm. From our experiments, a Perm operation is 56 times slower than an Add operation and 34 times slower than a ScMult operation. This observation motivates the proposed linear optimization, which aims to minimize the number of Perm operations, thus substantially reducing the overall computation time.

Meanwhile, PHE introduces noise in the ciphertext which theoretically hides the original message [38], [13]. Assume the noise of $[x]$ and $[y]$ are η_0 and η_1, then the noise after the Add operation is approximately $\eta_0 + \eta_1$. The noise after a ScMult operation is $\eta_{mult}\eta_0$ where η_{mult} is the *multiplicative noise growth of the SIMD scalar multiplication operation* [38]. The noise after a Perm operation is $\eta_0 + \eta_{rot}$ where η_{rot} is the *additive noise growth of a permutation operation* [38]. Roughly, we have $\eta_{rot} > \eta_{mult} \gg \eta_0 \gg 1$. If the noise goes beyond a certain level, the decryption would fail. Thus it is also important to have a good noise management over the ciphertext. We will show in Sec. III-C that GALA has a better noise control than GAZELLE, which further guarantees the overall success for the linear computations.

(2) Secret Sharing. In the secret sharing protocol, a value is shared between two parties, such that combining the two

---

[2]The readers are referred to [38] for more detail.

secrets yields the true value [56]. In order to additively share a secret m, a random number, r, is selected and two shares are created as $\langle m \rangle_0 = r$ and $\langle m \rangle_1 = m - r$. Here, m can be either plaintext or ciphertext. A party that wants to share a secret sends one of the shares to the other party. To reconstruct the secret, one needs to only add two shares $m = \langle m \rangle_0 + \langle m \rangle_1$.

While the overall idea of secret sharing (SS) is straightforward, creative designs are often required to enable its effective application in practice. Specifically, in the HE-GC neural network framework, the linear result from the dot product or convolution is encrypted at the server side and needs to be shared with the client to enable the following GC-based nonlinear computation. Assume m is the resulted ciphertext of a linear computation at the server, GAZELLE then generates the share $\langle m \rangle_0 = r$ and sends $\langle m \rangle_1 = m - r$ to the client. The two shares act as the input of the GC-based nonlinear computation. Here the computation of m involves a series of Perm operations, which is time-consuming. Instead of directly generating the share $\langle m \rangle_0 = r$ for m, we develop a *share-RaS (Rotate and Sum) computing* for dot product which lets the server generate an indirect share r' for the incomplete m, m', while the true r is easy to be derived from r' and the true $\langle m \rangle_1 = m - r$ is easy to be derived from $m' - r'$. The computation of m' eliminates a large number of Perm operations thus reducing the computation complexity. Specifically, Our result shows that the proposed *share-RaS computing* demonstrates $19\times$ speedup for the dot product by multiplying a 16×128 matrix with a length-128 vector (the detailed benchmarks are shown in Sec. IV).

(3) Oblivious Transfer. In the 1-out-of-k Oblivious Transfer (OT) [15], denoted as $\binom{k}{1}$-OT$_\ell$, the sender's inputs are the k strings, $m_0, m_1, \cdots, m_{k-1} \in \{0,1\}^\ell$, and the receiver's input is a value $i \in \{0, 1, \cdots, k-1\}$. At the end of the OT execution, the receiver obtains m_i from the functionality and the sender receives no output. Here, the OT protocol guarantees that 1) the receiver learns nothing about $m_{j,j \neq i}$, and 2) the sender learns nothing about i. An advancement in the practicality of OT protocols is the OT extension [36], which is further optimized such as in [39]. A special type of OT extension is the correlated OT extension (COT) [9]. Particularly, the 1-out-of-2 COT, denoted as $\binom{2}{1}$-COT$_\ell$, can be used for linear computation[3]. In $\binom{2}{1}$-COT$_\ell$, the sender's two inputs to each OT are not independent. Instead, the two inputs to each OT instance are a random value s_0 and a value $s_1 = f(s_0)$ for a correlation function f of the sender's choice. The receiver obtains either s_0 or s_1 as output depending on b.

III. System Description

In this section, we introduce the proposed system, GALA, for streamlining the linear computations (i.e., matrix-vector multiplication and convolution) in privacy-preserved neural network models. The HE-based linear computation consists of three basic operations: Homomorphic Addition (Add), Multiplication (Mult), and Permutation (Perm). Our investigation has shown that the linear computation dominates the total computation cost and the most time-consuming part of HE-based linear computation is a series of Perm operations that are

---

imperative to enable dot product and convolution. GALA aims to minimize the Perm operations, thus substantially reducing the overall computation time. We view the HE-based linear computation as a series of Add, Mult and Perm. The two inputs to linear computation are the encrypted vector (or channels) from the client and the plaintext weight matrix (or kernel) from the server. The output is the encrypted dot product (or convolution). The objective in each step is to choose the most efficient operations in the descending priorities of Add, Mult and Perm. Therefore, the overhead for the HE-based linear computation can be efficiently reduced by GALA. The recent privacy-preserved neural network frameworks can integrate GALA as a plug-and-play module to further boost their efficiency. We also analyze the (better) noise management and (guaranteed) system security of GALA.

A. Row-encoding-share-RaS Matrix-Vector Multiplication

We first focus on matrix-vector multiplication (dot product) which multiplies a plaintext matrix at the server with an encrypted vector from the client. We first discuss a naive method followed by the mechanism employed in the state-of-the-art framework (i.e., GAZELLE [38]), and then introduce the proposed optimization of GALA that significantly improves the efficiency in matrix-vector multiplication.

For a lucid presentation of the proposed GALA and comparison with the state-of-the-art framework, we adopt the same system model used in [38]. More specifically, we consider a Fully Connected (FC) layer with n_i inputs and n_o outputs. The number of slots in one ciphertext is n. We also adopt the assumptions used in [38]: n, n_i and n_o are powers of two, and n_o and n_i are smaller than n. If they are larger than n, the original $n_o \times n_i$ matrix can be split into $n \times n$ sized blocks that are processed independently.

1) Naive Method: The naive calculation for matrix-vector multiplication is shown in Figure 3, where w is the $n_o \times n_i$ plaintext matrix on the server and $[x]_{\mathcal{C}}$ is the HE-encrypted vector provided by the client. The server encodes each row of w into a separate plaintext vector (see step (a) in Figure
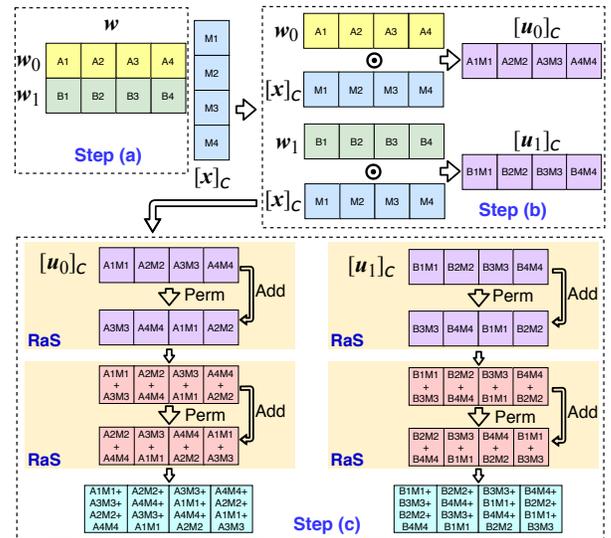

Fig. 3. Naive matrix-vector multiplication.

---

[3]We refer readers to [11], [21], [48], [54] for more details.

3). The length of each encoded vector is n (including padded 0's if necessary). We denote these encoded plaintext vectors as $\boldsymbol{w}_0, \boldsymbol{w}_1, \cdots, \boldsymbol{w}_{(n_o-1)}$. For example, the yellow and green rows in step (a) of Figure 3 are $\boldsymbol{w}_0$ and $\boldsymbol{w}_1$, respectively.

The server intends to compute the dot product between $\boldsymbol{w}$ and $[\boldsymbol{x}]_{\mathcal{C}}$. To this end, it first uses ScMult to compute the elementwise multiplication between $\boldsymbol{w}_i$ and the encrypted input vector $[\boldsymbol{x}]_{\mathcal{C}}$ to get $[\boldsymbol{u}_i]_{\mathcal{C}} = [\boldsymbol{w}_i \odot \boldsymbol{x}]_{\mathcal{C}}$ (see step (b) in Figure 3). The sum of all elements in $\boldsymbol{u}_i$ will be the i-th element of the desired dot product between $\boldsymbol{w}$ and $[\boldsymbol{x}]_{\mathcal{C}}$. However, as discussed in Sec. II-C, it is not straightforward to obtain the sum under the packed HE. A *rotate-and-sum* (RaS) calculation must be used here, as illustrated in step (c) of Figure 3. Specifically, the entries in $[\boldsymbol{u}_i]_{\mathcal{C}}$ are first rotated through Perm by $\frac{n_i}{2}$ positions such that the first $\frac{n_i}{2}$ entries of the rotated $[\boldsymbol{u}_i]_{\mathcal{C}}$ are actually the second $\frac{n_i}{2}$ entries of the original $[\boldsymbol{u}_i]_{\mathcal{C}}$. Then the server uses Add to conduct elementwise addition between the rotated $[\boldsymbol{u}_i]_{\mathcal{C}}$ and the original $[\boldsymbol{u}_i]_{\mathcal{C}}$, which results in a ciphertext whose first $\frac{n_i}{2}$ entries contain the elementwise sum of the first and second $\frac{n_i}{2}$ entries of $\boldsymbol{u}_i$. The server conducts this RaS process for $\log_2 n_i$ iterations. Each iteration acts on the resulted ciphertext from the previous iteration, and rotates by half of the previous positions, as shown in Step (c) of Figure 3. Finally, the server gets a ciphertext where the first entry is the i-th element in $\boldsymbol{wx}$. By applying this procedure on each of the n_o rows (i.e., $\boldsymbol{w}_0, \boldsymbol{w}_1, \cdots, \boldsymbol{w}_{(n_o-1)}$), the server obtains n_o ciphertext. Altogether, the first entries of those ciphertext correspond to $\boldsymbol{wx}$.

We now analyze the complexity of the above linear computation process, in terms of the number of operations and output ciphertext. We consider the process starting from the server's reception of $[\boldsymbol{x}]_{\mathcal{C}}$ (i.e., the encrypted input data from the client) until it obtains the to-be-shared ciphertext[4] (i.e., the n_o ciphertext after RaS). There are a total of n_o scalar multiplications (ScMult) operations, $n_o \log_2 n_i$ Perm operations and $n_o \log_2 n_i$ Add operations. It yields n_o output ciphertext, each of which contains one element of the linear result $\boldsymbol{wx}$. This inefficient use of the ciphertext space results in a low efficiency for linear computations.

2) Hybrid Calculation (GAZELLE): In order to fully utilize the n slots in a ciphertext and further reduce the complexity, the state-of-the-art scheme is to combine the diagonal encoding [31] and RaS, by leveraging the fact that n_o is usually much smaller than n_i in FC layers. This hybrid method shows that the number of expensive Perm operations is a function of n_o rather than n_i, thus accelerating the computation of FC layers [38]. The basic idea of the hybrid method is shown in Figure 4.

Specifically, the server encodes $\boldsymbol{w}$ into n_o plaintext vectors through a diagonal manner. For example, in step (a) of Figure 4, the first plaintext vector $\boldsymbol{w}_0$ consists of the yellow elements of matrix $\boldsymbol{w}$, (A1, B2, A3, B4), and the second plaintext vector $\boldsymbol{w}_1$ consists of the green elements (A2, B3, A4, B1). Note that the $\boldsymbol{w}_0$ in this method is different from the $\boldsymbol{w}_0$ in the naive method of Figure 3. So is $\boldsymbol{w}_1$.
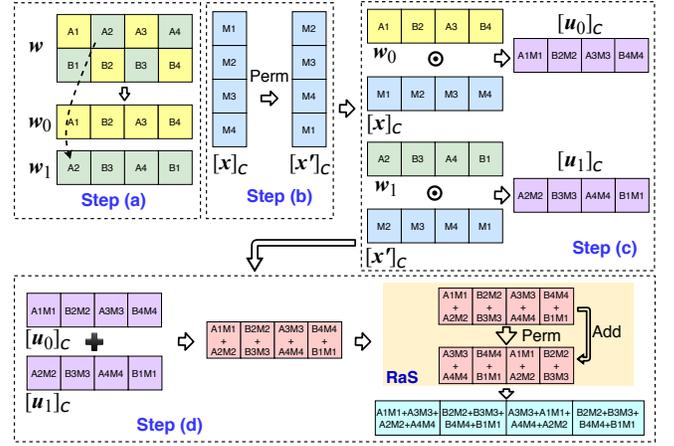
Fig. 4. Hybrid matrix-vector multiplication.

The server then rotates $[\boldsymbol{x}]_{\mathcal{C}}$ by i positions, shown in step (b), and uses ScMult to perform elementwise multiplication with $\boldsymbol{w}_i$. For example, in step (c) of Figure 4, $\boldsymbol{w}_0$ is multiplied with the encrypted data $[\boldsymbol{x}]_{\mathcal{C}}$ and $\boldsymbol{w}_1$ is multiplied with the input that is rotated by one position (i.e., $[\boldsymbol{x}']_{\mathcal{C}}$). As a result, the server gets n_o multiplied ciphertext, $\{[\boldsymbol{u}_i]_{\mathcal{C}}\}$. The entries in each of $\{[\boldsymbol{u}_i]_{\mathcal{C}}\}$ are partial sums of the elements in the matrix-vector multiplication $\boldsymbol{wx}$. For example, as shown in step (c) of Figure 4, the server obtains two multiplied ciphertext (i.e., $[\boldsymbol{u}_0]_{\mathcal{C}}$ and $[\boldsymbol{u}_1]_{\mathcal{C}}$) whose elements are partial sums of the first and second elements of $\boldsymbol{wx}$ (i.e., (A1M1 + A2M2 + A3M3 + A4M4) and (B1M1 + B2M2 + B3M3 + B4M4)). Then the server sums them up elementwise, to form another ciphertext, which is the vector in the middle of step (d) in Figure 4. At this point, similar to the naive method, the server proceeds with $\log_2 \frac{n_i}{n_o}$ RaS iterations and finally obtains a single ciphertext whose first n_o entries are the corresponding n_o elements of $\boldsymbol{wx}$ (see the first two elements of the vector after RaS in step (d)).

Furthermore, as the number of slots n in a ciphertext is always larger than the dimension of the input vector, n_i, the computation cost is further reduced by packing copies of input $\boldsymbol{x}$ as much as possible to form $[\boldsymbol{x}_{\text{pack}}]_{\mathcal{C}}$. Thus $[\boldsymbol{x}_{\text{pack}}]_{\mathcal{C}}$ has $\frac{n}{n_i}$ copies of $\boldsymbol{x}$ and the server is able to multiply $\frac{n}{n_i}$ encoded vectors with $[\boldsymbol{x}_{\text{pack}}]_{\mathcal{C}}$ by one ScMult operation. Therefore the server gets $\frac{n_i n_o}{n}$ rather than n_o multiplied ciphertext. The resulted single ciphertext now has $\frac{n}{n_o}$ rather than $\frac{n_i}{n_o}$ blocks. The server then applies $\log_2 \frac{n}{n_o}$ RaS iterations to get the final ciphertext, whose first n_o entries are the n_o elements of $\boldsymbol{wx}$.

The hybrid method requires $\frac{n_i n_o}{n}$ scalar multiplications (ScMult), $\frac{n_i n_o}{n} - 1$ HstPerm rotations for $[\boldsymbol{x}_{\text{pack}}]_{\mathcal{C}}$, $\log_2 \frac{n}{n_o}$ Perm rotations, and $\frac{n_i n_o}{n} + \log_2 \frac{n}{n_o} - 1$ additions (Add). There is only one output ciphertext, which efficiently improves the slot utilization compared to the naive method.

3) Row-encoding-share-RaS Multiplication (GALA): The proposed GALA framework is motivated by two observations on the hybrid method. First, the hybrid method essentially strikes a tradeoff between Perm and HstPerm operations, where the number of Perms (which is the most expensive HE operation) is proportional to the number of slots in a ciphertext.
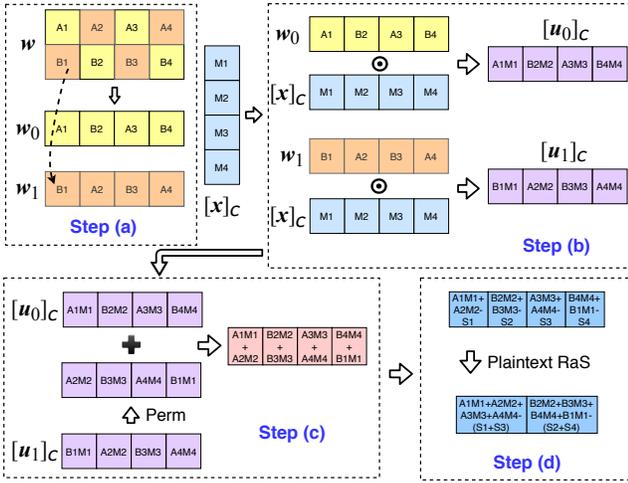
Fig. 5. Row-encoding-share-RaS multiplication.

This is not desired as we prefer a large n to pack more data for efficient SIMD HE. GALA aims to make the number of Perm operations disproportional to the number of slots and eliminate all HstPerm operations on the input ciphertext.

The second observation is the $\log_2 \frac{n}{n_o}$ RaS operations. We discover that this is actually unnecessary. Specifically, the unique feature in the HE-GC neural network framework is that the resultant single ciphertext from linear computing is shared between the client and server, to be the input for the nonlinear computing in the next phase. As the shares are in plaintext, we propose to transfer the final $\log_2 \frac{n}{n_o}$ RaS operations in the HE domain to $\log_2 \frac{n}{n_o}$ RaS operations in plaintext. This significantly reduces expensive Perm operations. For example, multiplying a 16×128 matrix with a length-128 vector by our proposed scheme shows about $19\times$ speedup compared with the hybrid method [38] on a commodity machine (see detailed benchmarks in Sec. IV).

Figure 5 illustrates GALA's matrix-vector calculation. The server first conducts the row-wise weight matrix encoding which encodes w into n_o plaintext vectors in a diagonal manner, as shown in step (a) in Figure 5. Compared with the hybrid method, the row-wise weight matrix encoding of GALA enables the server to directly multiply w_i and $[x]_{\mathcal{C}}$, eliminating the Perm operations on $[x]_{\mathcal{C}}$ in step (b). Furthermore, the encoding also benefits the noise management in the resultant to-be-shared ciphertext as to be analyzed in Sec. III-C.

As a result, the server gets n_o multiplied ciphertext, $\{[u_i]_{\mathcal{C}}\}$, such that the first entry of $[u_i]_{\mathcal{C}}$ is a partial sum of the i-th element of the matrix-vector multiplication wx. For example, in step (b) of Figure 5, the first element A1M1 in $[u_0]_{\mathcal{C}}$ is a partial sum of the first element of wx (i.e., A1M1 + A2M2 + A3M3 + A4M4), and the first element in $[u_1]_{\mathcal{C}}$ is a partial sum of the 2nd element of wx (i.e., B1M1 + B2M2 + B3M3 + B4M4). Then, the server conducts rotations on each $[u_i]_{\mathcal{C}}$, with totally $(n_o - 1)$ Perm operations excluding the trivial rotation by zero, to make the first entry of $[u_i]_{\mathcal{C}}$ to be a partial sum of the first element of wx. Next, the server adds all of the rotated $[u_i]_{\mathcal{C}}$ to obtain a single ciphertext whose entries are repeatedly a partial sum of the elements of wx. For example, in step (c) of Figure 5, $[u_1]_{\mathcal{C}}$

is rotated by one position, and then added with $[u_0]_{\mathcal{C}}$ to get one ciphertext, whose entries are the partial sum of the first and second elements of wx.

Till now, the natural next step is to conduct $\log_2 \frac{n_i}{n_o}$ RaS iterations to get a final ciphertext whose first n_o entries are the n_o elements of wx, i.e., the approach used by the hybrid method [44], [38]. With GALA, we propose to eliminate the $\log_2 \frac{n_i}{n_o}$ time-consuming RaS iterations by integrating it with the generation of shares for the GC-based nonlinear computing.

As introduced in the hybrid method [44], [38], in order to do the GC based nonlinear computing, the encrypted linear output is shared as follows: (1) the server generates a random vector; (2) the server subtracts the random vector from the ciphertext (the encrypted linear output); (3) the subtracted ciphertext is sent to the client, which subsequently decrypts it and obtains its share.

Here we let the server encode a similar random vector and subtract it from the ciphertext obtained in step (c) of Figure 5. The subtracted ciphertext is sent to the client, which decrypts ciphertext, and then applies $\log_2 \frac{n_i}{n_o}$ RaS iterations on the plaintext, as illustrated in step (d) of Figure 5. Similarly, the server gets its share by $\log_2 \frac{n_i}{n_o}$ plaintext RaS iterations on its encoded random vector. Hence, in GALA, the server replaces the ciphertext RaS operations by much faster plaintext RaS operations. This significantly improves the computation efficiency.

Furthermore, in order to make use of all slots in a ciphertext, the client packs $\frac{n}{n_i}$ input x to form a packed vector $[x_{\text{pack}}]_{\mathcal{C}}$. Then the server multiplies $\frac{n}{n_i}$ encoded weight vectors with $[x_{\text{pack}}]_{\mathcal{C}}$ by one ScMult operation. As a result, the server obtains $\frac{n_i n_o}{n}$ multiplied ciphertext, which are respectively rotated to enable the elementwise sum, finally producing a single ciphertext that has $\frac{n}{n_o}$ to-be-accumulated blocks. Without any further HE RaS iterations, the server then starts to encode the random vector for the share generation. The only extra computation is the plaintext RaS iteration(s) at both the client and server, which is much faster compared to the ones in HE domain.

As a result, GALA needs $\frac{n_i n_o}{n}$ ScMult operations, $(\frac{n_i n_o}{n} - 1)$ Perm operations, and $(\frac{n_i n_o}{n} - 1)$ Add operations. It yields one output ciphertext, and makes efficient utilization of ciphertext slots. Table II compares the complexity among the naive method, the hybrid method (i.e., GAZELLE) and the proposed row-encoding-share-RaS matrix-vector multiplication (GALA). We can see that the proposed method completely eliminates the HstPerm operations and significantly reduces the Perm operations.

TABLE II. COMPLEXITY COMPARISON OF THREE METHODS.

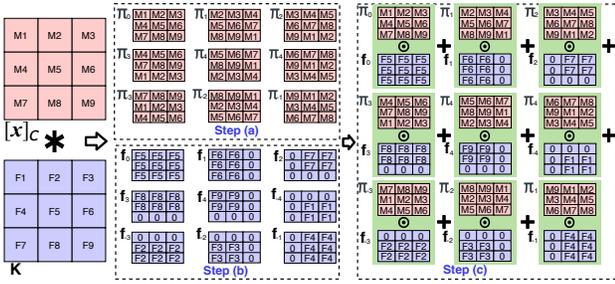| Method	# Perm	# HstPerm	# ScMult	# Add
Naive	$n_o \log_2 n_i$	0	n_o	$n_o \log_2 n_i$
GAZELLE	$\log_2 \frac{n}{n_o}$	$\frac{n_i n_o}{n} - 1$	$\frac{n_i n_o}{n}$	$\log_2 \frac{n}{n_o} + \frac{n_i n_o}{n} - 1$
GALA	$\frac{n_i n_o}{n} - 1$	0	$\frac{n_i n_o}{n}$	$\frac{n_i n_o}{n} - 1$

Fig. 6. SISO convolution.

B. Kernel Grouping Based Convolution

In this subsection, we introduce GALA's optimization for convolution. Similar to the discussion on the matrix-vector multiplication, we first begin with the basic convolution for the Single Input Single Output (SISO), then go through the state-of-the-art scheme for the Multiple Input Multiple Output (MIMO) (i.e., the GAZELLE framework [38]). Finally we elaborate GALA's first-Add-second-Perm (*kernel grouping*) scheme that achieves more efficient convolution computation. We assume the server has c_o plaintext kernels with a size of $k_w \times k_h \times c_i$ and the client sends to the server the encrypted data in the size of $u_w \times u_h$ with c_i channels. The server needs to homomorphically convolve the encrypted data from the client with its plaintext kernels to produce the encrypted output.

1) Basic SISO convolution: SISO is a special case of MIMO where $c_i = c_o = 1$. In this case, the encrypted data from the client has a size of $u_w \times u_h$ with one channel (i.e., a 2D image) and there is only one kernel with size $k_w \times k_h$ (i.e., a 2D filter) at the server. The SISO convolution is illustrated by an example in Figure 6 where $[\boldsymbol{x}]_{\mathcal{C}}$ is the encrypted data from the client and K is the plaintext kernel at the server. The process of convolution can be visualized as placing the kernel K at different locations of the input data $[\boldsymbol{x}]_{\mathcal{C}}$. At each location, a sum of an element-wise product between the kernel and corresponding data values within the kernel window is computed. For example, in Figure 6, the first value of the convolution between $[\boldsymbol{x}]_{\mathcal{C}}$ and kernel K is (M1F5 + M2F6 + M4F8 + M5F9). It is obtained by first placing the center of K, i.e., F5, at M1 and then calculating the element-wise product between K and the part of $[\boldsymbol{x}]_{\mathcal{C}}$ that is within K's kernel window (i.e., M1, M2, M4 and M5). The final result is the sum of the element-wise product. The rest of convolution values are calculated similarly by placing F5 at M2 to M9.

We now elaborate the convolution by an example when F5 is placed at M5 (i.e., the central element of $[\boldsymbol{x}]_{\mathcal{C}}$). In this example, the kernel size is $k_w k_h = 9$. The convolution is derived by summing the element-wise product between the 9 values in K and the corresponding 9 values around M5. This can be achieved by rotating $[\boldsymbol{x}]_{\mathcal{C}}$ in a raster scan fashion [38]. Specifically, $[\boldsymbol{x}]_{\mathcal{C}}$ is converted to a vector by concatenating all rows. Then, it is rotated by $(k_w k_h - 1)$ rounds, with half of them in the forward direction and the other half in the backward direction. We denote π_j as the rotation by j positions, where a positive sign of j indicates the forward direction and negative the backward direction, as shown in step (a) of Figure 6.

The convolution is obtained by (1) forming the kernel
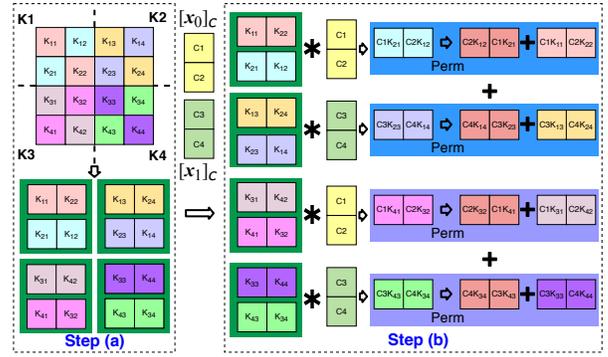

Fig. 7. MIMO convolution.

coefficients according to the partial sum at the corresponding location as shown in step (b) of Figure 6, (2) scaling the 9 rotated π_j with the corresponding kernel coefficients, and (3) summing up all scaled π_j (see step (c)).

The rotation for $[\boldsymbol{x}]_{\mathcal{C}}$ is completed by HstPerm[5]. The scaling is done by ScMult and the summation is achieved by Add. Therefore, the SISO convolution requires a total of $(k_w k_h - 1)$ HstPerm operations (excluding the trivial rotation by zero), $k_w k_h$ ScMult operations and $(k_w k_h - 1)$ Add operations. The output is one ciphertext[6] which contains the convolution result.

2) Output Rotation based MIMO convolution (GAZELLE): We now consider the more general case, i.e., MIMO, where c_i or c_o is not one. The naive approach is to directly apply SISO convolution by first encrypting the c_i input channels into c_i ciphertext, $\{[\boldsymbol{x}_i]_{\mathcal{C}}\}$. Each of the c_o kernels includes c_i filters. Each $[\boldsymbol{x}_i]_{\mathcal{C}}$ is convolved with one of the c_i filters by SISO and the final convolution is obtained by summing up all of the c_i SISO convolutions. As a result, the naive approach requires $c_i(k_w k_h - 1)$ HstPerm operations (for c_i input channels), $c_i c_o k_w k_h$ ScMult operations and $c_o(c_i k_w k_h - 1)$ Add operations. There are c_o output ciphertext.

Given the number of slots n in a ciphertext is usually larger than the channel size $u_w u_h$, the ciphertext utilization (i.e., the meaningful slots that output desired results) in the c_o output ciphertext is low.

In order to improve the ciphertext utilization and computation efficiency for MIMO convolution, the state-of-the-art method (i.e., the output rotation [38]) first packs c_n channels of input data into one ciphertext, which results in $\frac{c_i}{c_n}$ input ciphertext (see Figure 7 where the four input channels form two ciphertext, each of which includes two channels). Meanwhile, the c_o kernels are viewed as a $c_o \times c_i$ kernel block and each row of the block includes c_i 2D filters for one kernel. Then the MIMO convolution is viewed as a matrix-vector multiplication where the element-wise multiplication is replaced by convolution. As each ciphertext holds c_n channels, the kernel block is divided into $\frac{c_o c_i}{c_n^2}$ blocks (see step (a) in Figure 7, where the kernel block is divided into K1 to K4).

Next, each divided block is diagonally encoded into c_n vectors such that the first filters in all vectors are in the first

---

[5]With a common DecPerm operation.

[6]We assume the input size $u_w u_h$ is smaller that the ciphertext size n.
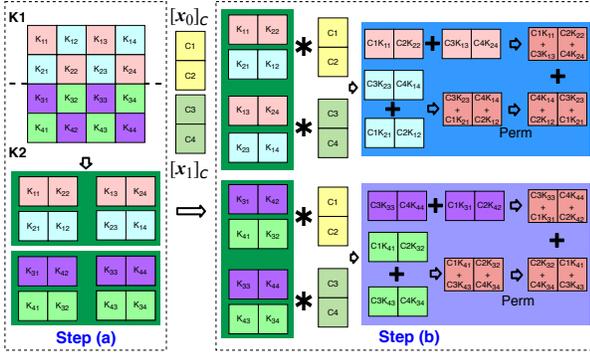
Fig. 8.   Kernel grouping based MIMO convolution.

column of the kernel block (see the four groups of vectors in step (a) of Figure 7). In this way, each input ciphertext can directly convolve with the vectors in each divided block by SISO, and the convolution for each divided block is obtained by rotating the c_n convolved vectors to the same kernel order as the diagonal one and summing them up (see step (b)).

Finally, the convolution for c_n kernels is calculated by adding the convolution of $\frac{c_i}{c_n}$ blocks associated with the same kernels as illustrated in step (b) of Figure 7.

Clearly, there are $\frac{c_o}{c_n}$ output ciphertext, as expected. For each of the $\frac{c_o c_i}{c_n^2}$ blocks, there are total c_n SISO-like convolutions, requiring $c_n k_w k_h$ ScMult operations, $(c_n - 1)$ Perm operations and $(c_n k_w k_h - 1)$ Add operations. Next, there are $\frac{c_i}{c_n}$ block convolutions which are associated with the same kernel order. Thus they are added up to obtain the final convolution result. Meanwhile, the rotation group for each input ciphertext is reused to convolve with different kernel blocks. Thus there are total $\frac{c_i(k_w k_h - 1)}{c_n}$ HstPerm operations with $\frac{c_i}{c_n}$ common DecPerm operations. In all, the MIMO convolution needs a total of $\frac{c_i c_o}{c_n^2}(c_n - 1)$ Perm, $\frac{c_i}{c_n}(k_w k_h - 1)$ HstPerm, $k_w k_h \frac{c_i c_o}{c_n}$ ScMult and $\frac{c_o}{c_n}(c_i k_w k_h - 1)$ Add operations.

3) Kernel Grouping Based MIMO convolution (GALA): One key observation on the above MIMO convolution is that, each of the $\frac{c_o c_i}{c_n^2}$ blocks needs $(c_n - 1)$ expensive Perm operations in order to get the convolution for that block. However, we actually do not need to get the convolution for each block. As our goal is to get the convolution for each kernel, the blocks that are associated with the same kernel are combined in our proposed first-Add-second-Perm approach (*kernel grouping*) to reduce the Perm cost. Specifically, in step (a) of Figure 8, the whole kernel block is divided into two blocks K1 and K2 such that each block is the combination of $\frac{c_i}{c_n}$ c_n-by-c_n divided blocks, which correspond to the same kernels (i.e., the first and second kernel in K1 and the third and fourth kernel in K2).

For each newly formed block, all of the vectors are first convolved with the corresponding input ciphertext by SISO-like convolution. Then the convolved vectors that are associated with the same kernel order are first added together (see the addition of convolved vectors before rotation in step (b) of Figure 8). Finally, these added vectors are rotated to the same kernel order and summed up to obtain the convolution result (see the rotation and final addition for each block in step (b)

of Figure 8).

This kernel grouping calculation results in $(c_n - 1)$ Perm operations for each of $\frac{c_o}{c_n}$ newly formed blocks, which reduces the Perm complexity by a factor of $\frac{c_i}{c_n}$ compared with GAZELLE's MIMO convolution. This reduction is nontrivial especially for the state-of-the-art neural networks such as ResNets [32], where $\frac{c_i}{c_n}$ can be 256. This is because these neural networks contain a large number of large-size feature maps in order to capture the complex input features [62], [40], [32].

Similar to the output rotation based MIMO convolution discussed above, there are $\frac{c_o}{c_n}$ output ciphertext in the proposed scheme. For each of the $\frac{c_o}{c_n}$ newly formed blocks, there are c_i SISO-like convolutions. Then for each of the c_n kernel orders, there are $\frac{c_i}{c_n}$ convolutions to be summed up, which results in c_n added convolutions. These added convolutions are further rotated to the same kernel order and summed up to get the final convolution. Therefore, the proposed MIMO convolution requires a total of $\frac{c_o}{c_n}(c_n - 1)$ Perm, $\frac{c_i}{c_n}(k_w k_h - 1)$ HstPerm, $k_w k_h \frac{c_i c_o}{c_n}$ ScMult, and $\frac{c_o}{c_n}(c_i k_w k_h - 1)$ Add operations.

Table III compares the overall complexity for convolution computations. GALA's kernel grouping approach reduces the expensive Perm operations by a factor of $\frac{c_i}{c_o}$ without increasing other operations compared with the output rotation based MIMO convolution (i.e., the GAZELLE framework). The reduction in Perm operations leads to a significant speedup. Specifically, GALA shows about $14\times$ speedup compared with GAZELLE in the convolution between input data with a size of 16×16 with 2048 channels, and 512 kernels with a size of $1\times1@2048$ on a commodity machine (see detailed benchmarks in Sec. IV).

TABLE III.   COMPLEXITY COMPARISON OF CONVOLUTION.

Method	# Perm	# HstPerm[♯]	# ScMult	# Add
GAZELLE	$\frac{c_i c_o (c_n - 1)}{c_n^2}$	$\frac{c_i (k_w k_h - 1)}{c_n}$	$\frac{c_i c_o k_w k_h}{c_n}$	$\frac{c_o (c_i k_w k_h - 1)}{c_n}$
GALA	$\frac{c_o (c_n - 1)}{c_n}$	$\frac{c_i (k_w k_h - 1)}{c_n}$	$\frac{c_i c_o k_w k_h}{c_n}$	$\frac{c_o (c_i k_w k_h - 1)}{c_n}$

[♯]Rotations of the input with $\frac{c_i}{c_n}$ common DecPerm operations.

C. Noise Management

The packed HE (e.g., the BFV scheme) introduces noise in the ciphertext which theoretically hides the original message [38], [13]. However, the noise management is critical to the correct decryption of ciphertext after a series of HE operations. We will show that GALA has better noise management compared with GAZELLE.

Based on the computation complexity of matrix-vector multiplication and convolution, along with the noise change for HE operations as described in Sec. II-C, Table IV shows the noise growth of different schemes. As for the matrix-vector multiplication, GALA has a lower noise growth while keeping the number of output ciphertext as small as one[7]. As for the convolution computation, GALA reduces the noise

---

[7]Note that the noise in Table IV is calculated by assuming $(\frac{n_i n_o}{n} - 1) \geq 0$. The noise of GALA is still lower than that of GAZELLE when $(\frac{n_i n_o}{n} - 1) < 0$ as it means one ciphertext can hold data with size $n_o \times n_i$, which only involves one ScMult operation in GALA, and GAZELLE needs to subsequently conduct a series of RaS operations.

term associated with rotation by a factor of $\frac{c_i}{c_n}$ compared to GAZELLE. This is nontrivial especially for state-of-the-art neural networks such as ResNets [32], where $\frac{c_i}{c_n}$ can be 256. The number of output ciphertext is also maintained as small as $\frac{c_o}{c_n}$. Overall, GALA features a lower noise growth and lower computation complexity compared with GAZELLE.

TABLE IV. COMPARISON OF NOISE MANAGEMENT.

Matrix-vector Multiplication		
Method	Noise after computation	# Cipher
Naive	$n_i \eta_0 \eta_{mult} + (n_i - 1)\eta_{rot}$	n_o
GAZELLE	$n_i \eta_0 \eta_{mult} + [\frac{n_i n_o - n}{n_o}\eta_{mult} + \frac{n - n_o}{n_o}]\eta_{rot}$	1
GALA	$\frac{n_i n_o}{n}\eta_0 \eta_{mult} + (\frac{n_i n_o}{n} - 1)\eta_{rot}$	1
Convolution Computation		
Method	Noise after computation	# Cipher
GAZELLE	$c_i \eta_\Delta + \frac{c_i}{c_n}(c_n - 1)\eta_{rot}$	$\frac{c_o}{c_n}$
GALA	$c_i \eta_\Delta + (c_n - 1)\eta_{rot}$	$\frac{c_o}{c_n}$

$\eta_\Delta = k_w k_h \eta_{mult}\eta_0 + (k_w k_h - 1)\eta_{rot}\eta_{mult}$

D. System Security

GALA is based on the same security framework as GAZELLE [38]. The security of linear computation in GALA is fully protected by the security of HE (e.g., the BFV scheme [13], [23]). The nonlinear computation (which is not the focus of this paper) is protected by Garbled Circuits (GC) [72] or its alternatives. The security of GC-based nonlinear computation has been proven in TASTY [33] and MP2ML [12].

IV. EVALUATION

We conduct the experiments in both LAN and WAN settings. The LAN setting is implemented on a Gigabit Ethernet in our lab between two workstations as the client and server, respectively. Both machines run Ubuntu, and have an Intel i7-8700 3.2GHz CPU with 12 threads and 16 GB RAM. The WAN setting is based on a connection between a local PC and an Amazon AWS server with an average bandwidth 200Mbps and round-trip time around 13ms. We have downloaded the codes released by GAZELLE[8], DELPHI[9] and CrypTFlow2[10], and run all experiments on the same hardware devices and network settings. We conduct a series of experiments under various neural network architectures. In each experiment, we first run the baseline algorithm (i.e., GAZELLE, DELPHI or CrypTFlow2) to obtain the baseline total runtime (including online runtime and offline runtime), and then replace the linear computation of the baseline algorithm by GALA to get a new total runtime, which is then used to compute the speedup.

While the codes for GAZELLE, DELPHI and CrypTFlow2 are implemented in different ways (for example, GAZELLE is based on its crypto platform while DELPHI and CrypTFlow2 are based on the Microsoft SEAL library), we focus on the speedup of GALA on top of each of them. We also set the cryptographic parameters in line with GAZELLE: 1) Parameters for both HE and GC schemes are selected for a 128-bit security level. 2) A plaintext modulus p of 20 bits is enough to store all the intermediate values in the network

computation. 3) The ciphertext modulus q is chosen to be a 60-bit pseudo-Mersenne prime that is slightly smaller than the native machine word on a 64-bit machine to enable lazy modular reductions. 4) The selection of the number of slots is the smallest power of two that allows for a 128-bit security which in our case is $n = 2048$. We refer readers to [38] for more details about the parameter selection.

A. Microbenchmarks

In this section, we benchmark and compare the runtime of GALA's linear optimization (i.e., matrix-vector multiplication and convolution computation) with state-of-the-art approaches. We claim the same communication cost and inference accuracy with GAZELLE and achieve improved computation efficiency.

1) Matrix-Vector Multiplication: Table V compares the computation complexity of GALA's matrix-vector optimization with GAZELLE and two other optimization schemes (i.e., a diagonal method (Diagonal) [31] and an extended method (Extended) [17]). We can see that GALA largely reduces the expensive Perm operation to zero in our cases (including the HstPerm) while GAZELLE needs up to 11 Perm and Extended [17] needs up to 520 Perm (including HstPerm). On the other hand, GALA also maintains a light overhead for HE multiplication/addition, i.e., only one multiplication, compared with other three optimizations, e.g., Diagonal [31] and Extended [17] involve up to 2048 multiplication/addtion.

The runtime results for matrix-vector multiplication are summarized in Table VI, which includes the original runtime of GAZELLE, DELPHI and CrypTFlow2, and the speedup of GALA on top of each. We take the share-RaS calculation cost (see the plaintext computing for final share at the client in step (d) of Figure 5) as part of the runtime cost of GALA for fair comparison. Meanwhile, as multiple copies are packed in one ciphertext, the HstPerm operation includes a common DecPerm to enable hoist optimization for rotation (see the details in [38]). As can be seen from Table VI, GALA's optimization gains a large speedup due to the row-encoding-share-RaS module, which reduces the costly Perm, Mult, and Add operations for a series of RaS calculation. Specifically, GALA achieves the speedup of $1795\times$, $208\times$ and $57\times$ over the Diagonal [31] under different matrix dimensions in the LAN setting. This benefit stems from the fact that the computation complexity of the Diagonal is related to the input dimension n_i, which is always large in the state-of-the-art neural networks such as AlexNet [40], VGG [62] and ResNet [32]. For a similar reason, GALA significantly outperforms the Extended method [17].

Meanwhile, GALA has a speedup of $59\times$, $13\times$ and $19\times$ over GAZELLE under different matrix dimensions in the LAN setting. This computation gain comes from the HstPerm-free scheme (i.e., row-encoding) and elimination of RaS computation (i.e., share-RaS scheme) compared to GAZELLE, which is particularly effective for large $\frac{n_i}{n_o}$ ratio and large ciphertext slots (see the superior performance for the neural network with a dimension of 1×2048). These features suit well to current convolutional neural networks which have tens of thousands of values to be fed into the fully connected layers [62], [32].

Compared with DELPHI and CrypTFlow2, GALA achieves a speedup for weight matrix multiplication up to

---

TABLE V.   COMPUTATION COMPLEXITY OF MATRIX-VECTOR MULTIPLICATION.

Dimension ($n_o \times n_i$): 1×2048				
Metric	Diagonal[31]	GAZELLE	Extended[17]	GALA
# Perm	0	11	0	0
# HstPerm♮	2047	0	2047	0
# ScMult	2048	1	2048	1
# Add	2047	11	2047	0
Dimension ($n_o \times n_i$): 2×1024				
Metric	Diagonal[31]	GAZELLE	Extended[17]	GALA
# Perm	0	10	9	0
# HstPerm♮	1023	0	511	0
# ScMult	1024	1	512	1
# Add	1023	10	520	0
Dimension ($n_o \times n_i$): 16×128				
Metric	Diagonal[31]	GAZELLE	Extended[17]	GALA
# Perm	0	7	4	0
# HstPerm♮	127	0	7	0
# ScMult	128	1	8	1
# Add	127	7	11	0

♮Rotations of the input with a common DecPerm

TABLE VI.   RUNTIME COST OF MATRIX-VECTOR MULTIPLICATION.

Dimension ($n_o \times n_i$): 1×2048					
Approach	Comm. (MB)	LAN Time	LAN Speedup	WAN Time	WAN Speedup
Diagonal[31]	0.03	57	**1795×**	75	**4×**
Extended[17]	0.03	57.5	**1796×**	77	**4×**
GAZELLE[38]	0.03	1.9	**59×**	19.3	**1×**
DELPHI[46]	0.14	28	**700×**	59.5	**3.2×**
CrypTFlow2[54]	0.13	28	**700×**	46.2	**2.5×**
Dimension ($n_o \times n_i$): 2×1024					
Diagonal[31]	0.03	28	**208×**	47	**2.5×**
Extended[17]	0.03	16	**116×**	36	**1.9×**
GAZELLE[38]	0.03	1.8	**13×**	19	**1×**
DELPHI[46]	0.13	26.5	**176×**	57.8	**3.1×**
CrypTFlow2[54]	0.13	26.5	**176×**	44.7	**2.4×**
Dimension ($n_o \times n_i$): 16×128					
Diagonal[31]	0.03	3.7	**57×**	21	**1×**
Extended[17]	0.03	1	**16×**	20.4	**1×**
GAZELLE[38]	0.03	1.2	**19×**	21	**1×**
DELPHI[46]	0.13	20.5	**292×**	51.7	**2.8×**
CrypTFlow2[54]	0.13	20.5	**292×**	38.7	**2.1×**

TABLE VII.   COMPUTATION COMPLEXITY OF CONVOLUTION.

Input [†]	Kernel [‡]	Metric	GAZELLE[38]	GALA
16×16@128	1×1@128	# DecPerm	1792	112
		# HstPerm	1792	112
		# ScMult	2048	2048
		# Add	2032	2032
16×16@2048	1×1@512	# DecPerm	114944	2048
		# HstPerm	114688	1792
		# ScMult	131072	131072
		# Add	130944	130944
16×16@128	3×3@128	# DecPerm	1808	128
		# HstPerm	1920	240
		# ScMult	18432	18432
		# Add	18416	18416
16×16@2048	5×5@64	# DecPerm	14592	312
		# HstPerm	20480	6200
		# ScMult	409600	409600
		# Add	409592	409592

[†]Dim. is in the form of $u_w \times u_h @ c_i$

[‡]Dim. is in the form of $k_w \times k_h @ c_o$ with c_i channels per kernel

TABLE VIII.   RUNTIME COST OF CONVOLUTION.

Dimension (Input Dim.[†], Kernel Dim.[‡]): 16×16@128, 1×1@128					
Approach	Comm. (MB)	LAN Time	LAN Speedup	WAN Time	WAN Speedup
GAZELLE	0.5	321	**9×**	408	**3.2×**
DELPHI	2.1	391	**3.1×**	502	**2.3×**
CrypTFlow2	2	389	**3.1×**	482	**2.2×**
Dimension (Input Dim.[†], Kernel Dim.[‡]): 16×16@2048 , 1×1@512					
GAZELLE	8	20583.5	**14×**	21784	**8.7×**
DELPHI	31	17939	**4.4×**	19205	**3.7×**
CrypTFlow2	29	17928	**4.4×**	19101	**3.6×**
Dimension (Input Dim.[†], Kernel Dim.[‡]): 16×16@128, 3×3@128					
GAZELLE	0.5	457	**2.6×**	547	**2.1×**
DELPHI	2	2563.6	**5.8×**	2671	**5×**
CrypTFlow2	1.9	2559	**5.8×**	2648	**5×**
Dimension (Input Dim.[†], Kernel Dim.[‡]): 16×16@2048, 5×5@64					
GAZELLE	8	5875.2	**1.7×**	7073	**1.5×**
DELPHI	31	56499	**7.4×**	57765	**6.3×**
CrypTFlow2	29	56409	**7.4×**	57582	**6.5×**

[†]Dim. is in the form of $u_w \times u_h @ c_i$

[‡]Dim. is in the form of $k_w \times k_h @ c_o$ with c_i channels per kernel

700× in the LAN setting. This is largely due to GALA's deep optimization for HE computation. We also notice that GALA's speedup slows down in WAN which is due to the communication rounds needed for conversions between HE and GC. Therefore it leads to significant round time in total compared with the light HE computation overhead. For example, the round-trip time is around 13 milliseconds while the GALA's optimized HE cost is within one millisecond.

2) Convolution Computation: We benchmark and compare the computation complexity and runtime of GALA with GAZELLE, DELPHI and CrypTFlow2 for convolution calculation. The details are illustrated in Table VII and VIII. As for the computation complexity, we compare GALA with GAZELLE whose privacy-preserved convolution calculation over HE is one of the most optimized methods in current literature. While introducing no extra HE multiplication/addition, GALA reduces the most expensive Perm, i.e., DecPerm and HstPerm, by up to 59× for input size of 16×16@2048 with kernel size of 1×1@512. This block with large channels and small kernel size is featured in state-of-the-art neural networks such as ResNets [32], which makes GALA suitable to boost

the modern networks.

As for runtime comparison shown in Table VIII, GALA demonstrates 9×, 14× and 2.6× speedup over GAZELLE with different input and kernel dimensions in LAN setting. As analyzed in Sec. III-B, due to the fundamental complexity reduction by GALA's kernel grouping approach, GALA reduces the expensive Perm operation by a factor of $\frac{c_i}{c_n}$. As we mention above, the large speedup is achieved under large input channels and small kernel size, the proposed approach fits very well with the state-of-the-art networks such as ResNets [32], where the feature maps are always with large channels (which results in large c_i while c_n is fixed) and small kernels (that are usually 1×1, 3×3 and 5×5 at most, which benefit small HE multiplication/addition). Meanwhile, the speedup over DELPHI and CrypTFlow2 is up to 7.4× in the LAN setting. On the other hand, the speedup of GALA in the WAN setting is also decent, up to 8.7×, 6.3× and 6.5× for GAZELLE, DELPHI and CrypTFlow2, respectively. This is because the computation cost of convolution increases accordingly with regard to the communication cost, compared with the case of matrix-vector multiplication.

TABLE IX. COMPUTATION COMPLEXITY OF STATE-OF-THE-ART NEURAL NETWORK MODELS.

Net. Frameworks	Metric	GAZELLE[38]	GALA
MLP	# Perm	70	55
	# ScMult	56	56
	# Add	70	55
AlexNet	# Perm	40399	1157
	# DecPerm	143	142
	# HstPerm	1493	1492
	# ScMult	481298	481298
	# Add	481096	481089
VGG	# Perm	66055	2115
	# DecPerm	161	160
	# HstPerm	1283	1280
	# ScMult	663556	663556
	# Add	663370	663363
ResNet-18	# Perm	180375	5921
	# DecPerm	483	482
	# HstPerm	3467	3464
	# ScMult	1399363	1399363
	# Add	1398778	1398771
ResNet-50	# Perm	1464119	30615
	# DecPerm	2819	2818
	# HstPerm	3863	3848
	# ScMult	2935408	2935408
	# Add	2931734	2931727
ResNet-101	# Perm	2560823	64887
	# DecPerm	6083	6082
	# HstPerm	8215	8200
	# ScMult	5302896	5302896
	# Add	5294326	5294319
ResNet-152	# Perm	3463991	95127
	# DecPerm	8963	8962
	# HstPerm	12055	12040
	# ScMult	7252592	7252592
	# Add	7239894	7239887

TABLE X. RUNTIME COST OF CLASSIC MODEL.

		Network Model: MLP			
Approach	Comm. (MB)	LAN (ms)		WAN (ms)	
		Time	Speedup	Time	Speedup
SecureML	0.21	31.9	2.6×	79.3	1.5×
MiniONN	4.4	14.1	1×	227.6	1×
GAZELLE	0.21	15	1×	84.9	1×
DELPHI	84	204.5	3.1×	3658.3	1×
CrypTFlow2	12.4	246	2.3×	780.6	1.2×

TABLE XI. RUNTIME COST OF STATE-OF-THE-ART MODELS.

		Network Model: AlexNet			
Approach	Comm. (MB)	LAN (ms)		WAN (ms)	
		Time	Speedup	Time	Speedup
GAZELLE	17.45	11,019.2	2.5×	13,669.6	1.9×
DELPHI	617	90,090.1	2.9×	114,955	2×
CrypTFlow2	116.6	69,133.6	6.5×	73,876.8	4.8×
OT-based CrypTFlow2	2,108	226,431.7	21×	310,985.6	20×
		Network Model: VGG			
GAZELLE	22.8	18,067.4	2.7×	21,566.2	2×
DELPHI	718.5	123,198.4	2.9×	152,176.4	1.5×
CrypTFlow2	150	97,038.9	6×	103,169.1	4.6×
OT-based CrypTFlow2	5,063.7	340,342.9	21×	543,242	24×
		Network Model: ResNet-18			
GAZELLE	54	42,748.3	3.2×	51,032.7	2.3×
DELPHI	2,033.9	250,618.4	2.6×	332,524.2	1.9×
CrypTFlow2	354	190,684.7	5.7×	205,146.8	4.3×
OT-based CrypTFlow2	6,292.1	650,989.7	19.5×	903,492.6	19×
		Network Model: ResNet-50			
GAZELLE	297.1	276,886.8	8.3×	321,600.2	4×
DELPHI	10,489	746,568.8	1.7×	1167,566.8	1.4×
CrypTFlow2	1,831	425,454.4	4.5×	499,429.6	2.9×
OT-based CrypTFlow2	13,104	1364,463.2	14.4×	3307,902.6	19×
		Network Model: ResNet-101			
GAZELLE	603.1	486,745.2	7.7×	577,454.9	3.7×
DELPHI	22,199.4	1411,383.8	1.7×	2302,091.8	1.3×
CrypTFlow2	3,582.8	777,057.4	4.2×	921,735.6	2.8×
OT-based CrypTFlow2	23,857	2467,606.1	13.3×	6006,071.4	18.2×
		Network Model: ResNet-152			
GAZELLE	873.1	659,833.7	7.5×	786,587	3.6×
DELPHI	29,433	1975,798.9	1.6×	3157,176.8	1.3×
CrypTFlow2	5,141	1065,103.4	4.1×	1272,772.6	2.7×
OT-based CrypTFlow2	32,804	3379,188.7	13×	8245,124.5	17.5×

B. Performance with Classic Networks

In this section, we benchmark the GALA performance on a 4-layer Multi Layer Perceptron (MLP)[11] which is also adopted in other privacy preserving frameworks including GAZELLE, SecureML [48] and MiniONN [44] as a baseline network, as well as state-of-the-art neural network models including AlexNet [40], VGG [62], ResNet-18 [32], ResNet-50 [32], ResNet-101 [32], and ResNet-152 [32]. We use MNIST dataset [3] for the MLP and CIFAR-10 dataset [1] for state-of-the-art networks.

Table IX shows computation complexity of the proposed GALA compared with GAZELLE. We can see that GALA reduces GAZELLE's Perm by 34×, 31×, 30×, 47×, 39×, and 36× for AlexNet, VGG, ResNet-18, ResNet-50, ResNet-101, and ResNet-152, respectively. The fundamental base for this speedup lies in GALA's deep optimization for HE-based linear computation. We also notice that GALA achieves limited

---

[11]The network structure is 784-128-128-10.

reduction of Perm in MLP (from 70 to 55). This is due to the small ratio between the number of slots in the ciphertext and output dimension in each layer, i.e, $\frac{n}{n_o}$, which limits the performance gain. The limited gain is also observed in Table X which shows the system speedup of GALA over GAZELLE, CrypTFlow2, DELPHI, SecureML and MiniONN. Specifically, GALA boosts CrypTFlow2 by 2.3× in the LAN setting. SecureML also gains 2.6× in the LAN setting. Meanwhile, GALA's performance is similar to GAZELLE and MiniONN. The is due to the relatively small network size and noticeable communication overhead (i.e., the large round time in total compared with computation cost). Nevertheless, none of the competing schemes achieves a better performance than GALA.

It is worth pointing out that the MLP network is not widely adopted in practical scenarios. On the other hand, as the state-of-the-art deep neural networks utilize large channels and small-size kernels to capture data features while the size of feature maps is large, GALA is especially effective for accelerating such large state-of-the-art network models.

Table XI shows the runtime of GAZELLE, DELPHI and CrypTFlow2, and the speedup of GALA on top of each. By reducing HE operations, especially Perm operations, GALA achieves noticeable boost over the GAZELLE, DELPHI and CrypTFlow2 frameworks. Specifically, the results show that GALA boosts GAZELLE by 2.5× (from 11s to 4.3s), 2.7× (from 18s to 6.5s), 3.2× (from 43s to 13s), 8.3× (from 276s to 33s), 7.7× (from 486s to 62s), and 7.5× (from 659s to 87s) in LAN setting, on AlexNet, VGG, ResNet-18, ResNet-

TABLE XII.   PERCENTAGES OF LINEAR COMPUTATION IN
STATE-OF-THE-ART NEURAL NETWORK MODELS.

Networks	GAZELLE	DELPHI	CrypTFlow2	Plaintext
AlexNet	97.7	76.9	98.7	98.5
VGG	98.2	77.9	98.8	98.1
ResNet-18	98.3	75.1	98.6	98.9
ResNet-50	98.5	55.2	96.8	97.9
ResNet-101	98.4	53.2	96.5	98.3
ResNet-152	98	52	96.4	98.4

TABLE XIII.   ACCURACY WITH FLOATING AND FIXED POINT IN
STATE-OF-THE-ART NEURAL NETWORK MODELS. TOP-1 ACCURACY: ONLY
THE PREDICTION WITH THE HIGHEST PROBABILITY IS A TRUE LABEL;
TOP-5 ACCURACY: ANY ONE OF THE FIVE MODEL PREDICTIONS WITH
HIGHER PROBABILITY IS A TRUE LABEL.

Network Models	Floating-point		Fix-point	
	Top1	Top5	Top1	Top5
AlexNet	78.89%	97.32%	78.43%	97.26%
VGG	92.09%	99.72%	92.05%	99.68%
ResNet-18	93.33%	99.82%	93.21%	99.81%
ResNet-50	93.86%	99.85%	93.86%	99.84%
ResNet-101	94.16%	99.79%	94.12%	99.79%
ResNet-152	94.23%	99.81%	94.15%	99.79%

50, ResNet-101, and ResNet-152, respectively.

CrypTFlow2 (CCS'20) is the latest framework for privacy preserved neural networks. It optimizes the nonlinear operations of DELPHI, and adopts a similar HE scheme of DELPHI for linear operations. GALA is an efficient plug-and-play module to optimize the linear operations of CrypTFlow2. As shown in the Tables VI and VIII, GALA's optimization of linear operations can further boost CrypTFlow2 by $700\times$ and $7.4\times$ for matrix-vector multiplication and convolution in the LAN setting, respectively. This speedup stems from GALA's streamlined HE calculation compared with the one of CrypTFlow2. Slow-down is observed in the WAN setting, but CrypTFlow2 can still gain up to $6.5\times$ speedup for convolution due to the computation-intensive nature for large input channels with small kernel dimensions featured in state-of-the-art network models. As for the overall system speedup, GALA can boost CrypTFlow2 by $6.5\times$, $6\times$, $5.7\times$, $4.5\times$, $4.2\times$, and $4.1\times$ in LAN, and by $4.8\times$, $4.6\times$, $4.3\times$, $2.9\times$, $2.8\times$, and $2.7\times$ in WAN, based on the aforementioned network architectures.

It might appear anti-intuitive that while CrypTFlow2 is a more recent system than DELPHI, the speedup of GALA over DELPHI is smaller than its speedup over CrypTFlow2. This is because CrypTFlow2 has optimized the nonlinear part of DELPHI, significantly reducing its runtime. As a result, the runtime of linear operations in CrypTFlow2 accounts for a very high percentage as illustrated in Table XII. Hence CrypTFlow2 can benefit more from GALA's optimization of linear computation, resulting in a higher speedup in terms of the overall runtime. It is worth pointing out that the ability to accelerate CrypTFlow2 is highly desirable since it is the latest privacy-preserving framework. Meanwhile, we also show GALA's speedup on top of the OT-based CrypTFlow2 which relies on OT to complete the linear computation. As significant communication cost, including round cost, is involved in OT, the overhead of linear computation, especially in the WAN setting, increases compared with HE-based CrypTFlow2, which results in greater speedup achieved by GALA.

Next we examine the runtime breakdown of different layers for those six state-of-the-art networks as shown in Fig. 9, which allows detailed observation. Note that the layer indexing here is slightly different from the original plaintext model for the sake of HE operations, e.g., the nonlinear activation or pooling following a convolution operation is counted as a separate layer. The x-axis of each subfigure in Fig. 9 shows the layer index of a sequence of linear (convolution or matrix-vector multiplication) and nonlinear (activation or pooling) layers that constitute each network model. The y-axis plots the accumulated running time (milliseconds) up to a layer, and the speedup of GALA over GAZELLE in each layer.

For example, Fig. 9 (a) illustrates the result for AlexNet.

The most time-consuming computation in GAZELLE is in layer "6", "8" and "10", which are all convolution computations. This is evidenced by the large jump of runtime from these layers to the next layer. GALA decreases the time for these linear computations by nearly $3\times$. Meanwhile, the nonlinear layers (activation/pooling) have a speedup of 1, as GALA has the same computation cost as GAZELLE in those layers. Since the nonlinear computation contributes to only a small portion of the total cost, it does not significantly affect the overall performance gain of GALA that focuses on accelerating the linear computation. Note that GALA does not benefit much in the first layer of AlexNet, i.e., the first convolution, as the input has only three channels. However, the speedup for the following more costly convolutions allows GALA to effectively reduce the overall cost. A similar observation can be seen from the result on VGG. As for the four ResNets frameworks, the most significant performance gain stems from the convolution with 1×1 kernels. As ResNets repeat the blocks with multiple 1×1 convolution kernels, GALA effectively accelerates this type of convolution due to its deeply optimized linear computation mechanism (see details in Sec. III-B), thus reducing the overall runtime. Similar trend is observed for DELPHI and CrypTFlow2.

It is also worth mentioning that GALA focuses on optimizing the HE-based linear operations only and can be integrated into a baseline model (such as GAZELLE, CryptFlow2, or DELPHI). The proposed approach does not introduce approximation. Hence it does not result in any accuracy loss compared to the baseline privacy preserved model. Furthermore, compared with the original plaintext model, the only possible accuracy loss in GALA comes from the quantification of floating point numbers to fixed point numbers in the HE operations. Such quantification is indispensable in all HE-based frameworks including CryptFlow2. From our experiments, the model accuracy loss due to quantification is negligible, as shown in Table XIII.

V.   CONCLUSION AND FURTHER DISCUSSIONS

This paper has focused on a deep optimization on the HE-based linear computation in privacy-preserved neural networks. It aims to minimize the Perm operations, thus substantially reducing the overall computation time. To this end, we have proposed *GALA: Greedy computAtion for Linear Algebra*, which views the HE-based linear computation as a series of Homomorphic Add, Mult and Perm operations and chooses the least expensive operation in each linear computation step to reduce the overall cost. GALA has made the following
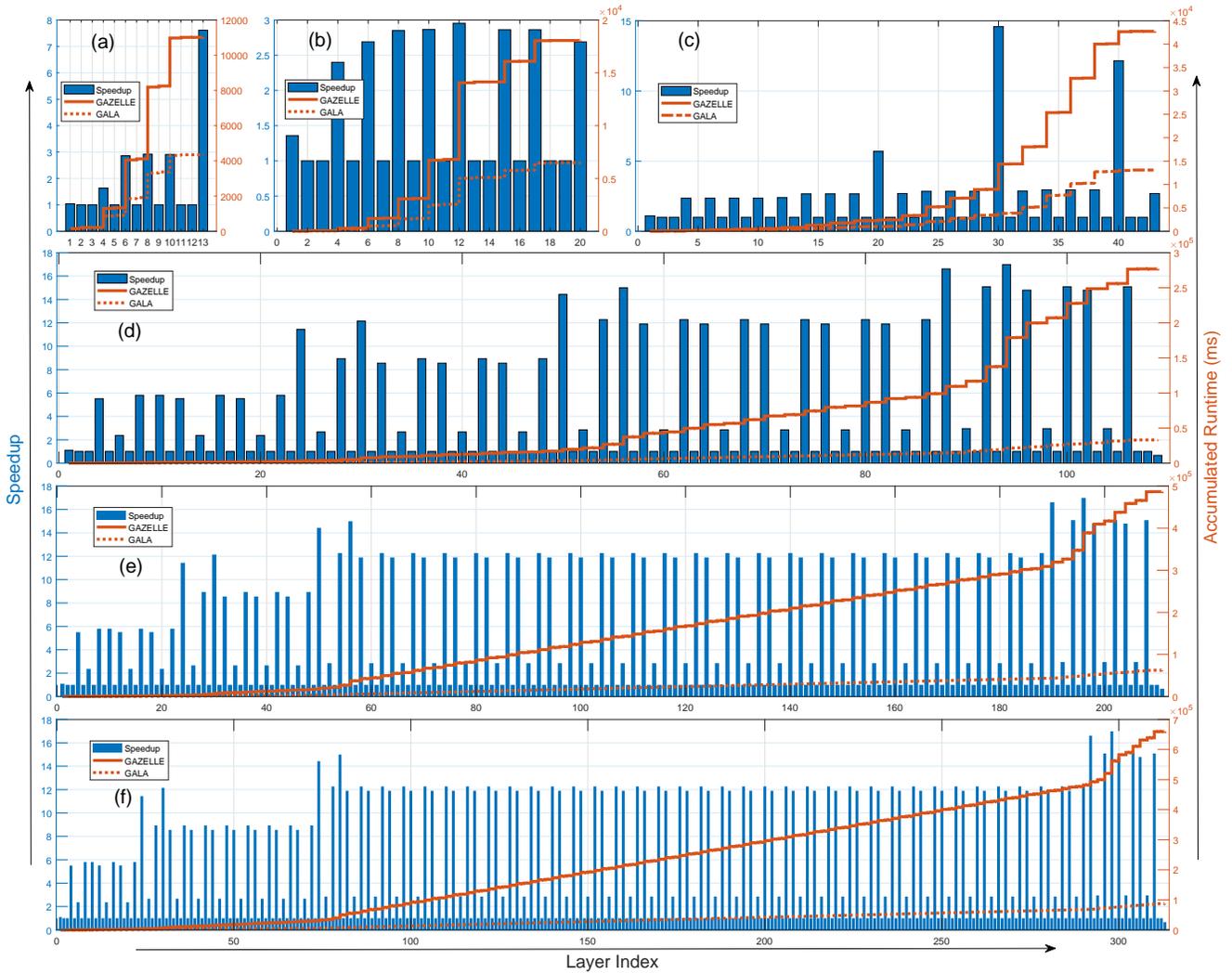
Fig. 9. Layer-wise accumulated runtime and GALA speedup over GAZELLA on different networks: (a) AlexNet; (b) VGG; (c) ResNet-18; (d) ResNet-50; (e) ResNet-101; (f) ResNet-152. The bar with values on the left y-axis indicates speedup, and the curve with values on the right y-axis indicates the accumulated runtime. The layers with speedup of 1 are nonlinear layers.

contributions: (1) It has introduced a row-wise weight matrix encoding with combined share generation (i.e., row-encoding-share-RaS (Rotated and Sum)) to reduce the Perm operations for dot product. (2) It has designed a first-Add-second-Perm approach (named *kernel grouping*) to reduce the Perm operations for convolution. As such, GALA efficiently reduces the cost for the HE-based linear computation, which is a critical building block in almost all of the recent frameworks for privacy-preserved neural networks, including GAZELLE, DELPHI, and CrypTFlow2. With its deep optimization of the HE-based linear computation, GALA can be a plug-and-play module integrated into these systems to further boost their efficiency. Our experiments show that GALA achieves a significant speedup up to $700\times$ for the dot product and $14\times$ for the convolution computation under different data dimensions. Meanwhile, GALA demonstrates an encouraging runtime boost by $2.5\times$, $2.7\times$, $3.2\times$, $8.3\times$, $7.7\times$, and $7.5\times$ over GAZELLE and $6.5\times$, $6\times$, $5.7\times$, $4.5\times$, $4.2\times$, and $4.1\times$ over CrypTFlow2, on AlexNet, VGG, ResNet-18, ResNet-50, ResNet-101, and ResNet-152, respectively.

It is worth pointing out that even with the significant progress toward privacy preserved machine learning in recent years (including this work), there still exists a large performance gap between the plaintext system (generally below a second) and the privacy preserved system (ranging from seconds to hundreds of second). Nevertheless, it is still promising to attain the long-term goal for practical implementation of privacy preserved machine learning. First, the privacy preserved machine learning system is to be deployed on clouds with abundant computation power. Hence, even though it takes significantly more time than the plaintext system on the same local hardware, running it on clouds with parallel computing infrastructure can significantly reduce the gap. Second, the research efforts on the in-depth optimization of the privacy-preserved computation further help to close the runtime gap. Altogether, the combination of advanced algorithms and cloud computation resources may enable the privacy-preserved system to achieve a response time well suited for some practical applications in the near future.

REFERENCES

[1] *CIFAR-10 dataset*. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html

[2] *Developer reference for Alexa interface*. [Online]. Available: https://developer.amazon.com/zh/docs/device-apis/alexa-interface.html

[3] *MNIST dataset*. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[4] *Reference for Google developers*. [Online]. Available: https://developers.google.com/actions/reference/rest/conversation-webhook

[5] *UCI Iris dataset*. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/iris

[6] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "TensorFlow: A system for large-scale machine learning," in *12th USENIX symposium on Operating Systems Design and Implementation*, 2016, pp. 265–283.

[7] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 308–318.

[8] A. Act, "Health insurance portability and accountability act of 1996," *Public Law*, vol. 104, p. 191, 1996.

[9] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner, "More efficient oblivious transfer and extensions for faster secure computation," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*, 2013, pp. 535–548.

[10] S. P. Bayerl, T. Frassetto, P. Jauernig, K. Riedhammer, A.-R. Sadeghi, T. Schneider, E. Stapf, and C. Weinert, "Offline model guard: Secure and private ml on mobile devices," in *Proceedings of the DATE*, 2020.

[11] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *Proceedings of the Annual International Cryptology Conference*. Springer, 1991, pp. 420–432.

[12] F. Boemer, R. Cammarota, D. Demmler, T. Schneider, and H. Yalame, "MP2ML: A mixed-protocol machine learning framework for private inference," in *Proceedings of the 15th International Conference on Availability, Reliability and Security*, 2020, pp. 1–10.

[13] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical GapSVP," in *Proceedings of the Annual Cryptology Conference*. Springer, 2012, pp. 868–886.

[14] Z. Brakerski, C. Gentry, and S. Halevi, "Packed ciphertexts in LWE-based homomorphic encryption," in *International Workshop on Public Key Cryptography*. Springer, 2013, pp. 1–13.

[15] G. Brassard, C. Crépeau, and J.-M. Robert, "All-or-nothing disclosure of secrets," in *Proceedings of the Conference on the Theory and Application of Cryptographic Techniques*. Springer, 1986, pp. 234–238.

[16] N. Chandran, D. Gupta, A. Rastogi, R. Sharma, and S. Tripathi, "EzPC: Programmable, efficient, and scalable secure two-party computation for machine learning," Cryptology ePrint Archive, Report 2017/1109, Tech. Rep., 2017.

[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 *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 395–412.

[18] T. Chen and S. Zhong, "Privacy-preserving backpropagation neural network learning," *IEEE Transactions on Neural Networks*, vol. 20, no. 10, pp. 1554–1564, 2009.

[19] E. Chou, J. Beal, D. Levy, S. Yeung, A. Haque, and L. Fei-Fei, "Faster CryptoNets: Leveraging sparsity for real-world encrypted inference," *arXiv preprint arXiv:1811.09953*, 2018.

[20] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pretrained deep neural networks for large-vocabulary speech recognition," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 1, pp. 30–42, 2012.

[21] D. Demmler, T. Schneider, and M. Zohner, "ABY–A framework for efficient mixed-protocol secure two-party computation," in *Proceedings of the NDSS*, 2015.

[22] R. Fakoor, F. Ladhak, A. Nazi, and M. Huber, "Using deep learning to enhance cancer diagnosis and classification," in *Proceedings of the International Conference on Machine Learning*, vol. 28. ACM New York, USA, 2013.

[23] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *IACR Cryptology ePrint Archive*, vol. 2012, p. 144, 2012.

[24] Q. Fan and J. Yang, "A denoising autoencoder approach for credit risk analysis," in *Proceedings of the 2018 International Conference on Computing and Artificial Intelligence*. ACM, 2018, pp. 62–65.

[25] I. File, "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)," *General Data Protection Regulation*, 2012.

[26] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 1322–1333.

[27] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy," in *Proceedings of the International Conference on Machine Learning*, 2016, pp. 201–210.

[28] O. Goldreich, *Foundations of cryptography: volume 2, basic applications*. Cambridge University Press, 2009.

[29] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game, or a completeness theorem for protocols with honest majority," in *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, 2019, pp. 307–328.

[30] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," *SIAM Journal on Computing*, vol. 18, no. 1, pp. 186–208, 1989.

[31] S. Halevi and V. Shoup, "Algorithms in HElib," in *Proceedings of the Annual Cryptology Conference*. Springer, 2014, pp. 554–571.

[32] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[33] W. Henecka, S. K ögl, A.-R. Sadeghi, T. Schneider, and I. Wehrenberg, "TASTY: Tool for automating secure two-party computations," in *Proceedings of the 17th ACM Conference on Computer and Communications Security*, 2010, pp. 451–462.

[34] E. Hesamifard, H. Takabi, M. Ghasemi, and R. N. Wright, "Privacy-preserving machine learning as a service," in *Proceedings on Privacy Enhancing Technologies*, vol. 2018, no. 3, 2018, pp. 123–142.

[35] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1991.

[36] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank, "Extending oblivious transfers efficiently," in *Proceedings of the Annual International Cryptology Conference*. Springer, 2003, pp. 145–161.

[37] X. Jiang, M. Kim, K. Lauter, and Y. Song, "Secure outsourced matrix computation and application to neural networks," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018, pp. 1209–1222.

[38] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *Proceedings of the 27th USENIX Security Symposium*. USENIX Association, 2018.

[39] V. Kolesnikov and R. Kumaresan, "Improved OT extension for transferring short secrets," in *Proceedings of the Annual Cryptology Conference*. Springer, 2013, pp. 54–70.

[40] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.

[41] N. Kumar, M. Rathee, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "CrypTFlow: Secure tensorflow inference," in *Proceedings of the 2020 IEEE Symposium on Security and Privacy*. IEEE, 2020, pp. 336–353.

[42] H. Li, Z. Lin, X. Shen, J. Brandt, and G. Hua, "A convolutional neural network cascade for face detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 5325–5334.

[43] S. Li, K. Xue, B. Zhu, C. Ding, X. Gao, D. Wei, and T. Wan, "FALCON: A fourier transform based approach for fast and secure convolutional neural network predictions," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 8705–8714.

[44] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via minionn transformations," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 619–631.

[45] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar, "Innovative instructions and software model for isolated execution," in *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, 2013.

[46] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "DELPHI: A cryptographic inference service for neural networks," in *Proceedings of the 29th USENIX Security Symposium*, 2020.

[47] P. Mohassel and P. Rindal, "ABY3: A mixed protocol framework for machine learning," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018, pp. 35–52.

[48] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *Proceedings of the 38th IEEE Symposium on Security and Privacy*. IEEE, 2017, pp. 19–38.

[49] M. Mozaffari-Kermani, S. Sur-Kolay, A. Raghunathan, and N. K. Jha, "Systematic poisoning attacks on and defenses for machine learning in healthcare," *IEEE Journal of Biomedical and Health Informatics*, vol. 19, no. 6, pp. 1893–1905, 2015.

[50] M. Nasr, R. Shokri, and A. Houmansadr, "Comprehensive privacy analysis of deep learning," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2019.

[51] O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. Costa, "Oblivious multi-party machine learning on trusted processors," in *Proceedings of the 25th USENIX Security Symposium*, 2016, pp. 619–636.

[52] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1999, pp. 223–238.

[53] N. Phan, Y. Wang, X. Wu, and D. Dou, "Differential privacy preservation for deep auto-encoders: An application of human behavior prediction," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[54] D. Rathee, M. Rathee, N. Kumar, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "CrypTFlow2: Practical 2-party secure inference," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 325–342.

[55] M. S. Riazi, M. Samragh, H. Chen, K. Laine, K. Lauter, and F. Koushanfar, "XONN: XNOR-based oblivious deep neural network inference," in *Proceedings of the 28th USENIX Security Symposium*, 2019, pp. 1501–1518.

[56] 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 *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*. ACM, 2018, pp. 707–721.

[57] B. D. Rouhani, M. S. Riazi, and F. Koushanfar, "DeepSecure: Scalable provably-secure deep learning," in *Proceedings of the 55th ACM/ESDA/IEEE Design Automation Conference*. IEEE, 2018, pp. 1–6.

[58] F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 815–823.

[59] "Microsoft SEAL (release 3.2)," https://github.com/Microsoft/SEAL, Feb. 2019, microsoft Research, Redmond, WA.

[60] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 1310–1321.

[61] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE, 2017, pp. 3–18.

[62] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proceedings of the International Conference on Learning Representations*, 2015.

[63] S. Sohangir, D. Wang, A. Pomeranets, and T. M. Khoshgoftaar, "Big Data: Deep learning for financial sentiment analysis," *Journal of Big Data*, vol. 5, no. 1, p. 3, 2018.

[64] C. Song, T. Ristenpart, and V. Shmatikov, "Machine learning models that remember too much," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 587–601.

[65] H. Takabi, J. B. Joshi, and G.-J. Ahn, "Security and privacy challenges in cloud computing environments," *IEEE Security & Privacy*, no. 6, pp. 24–31, 2010.

[66] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction APIs," in *Proceedings of the USENIX Security Symposium*, 2016, pp. 601–618.

[67] S. Wagh, D. Gupta, and N. Chandran, "SecureNN: 3-Party secure computation for neural network training," vol. 1, 2019, p. 24.

[68] L. Wan, W. K. Ng, S. Han, and V. Lee, "Privacy-preservation for gradient descent methods," in *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2007, pp. 775–783.

[69] B. Wang and N. Z. Gong, "Stealing hyperparameters in machine learning," in *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE, 2018, pp. 36–52.

[70] 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," in *Proceedings of the VLDB Endowment*, vol. 12, no. 2, 2018, pp. 128–140.

[71] R. Xu, J. B. Joshi, and C. Li, "CryptoNN: Training neural networks over encrypted data," in *Proceedings of the IEEE 39th International Conference on Distributed Computing Systems*. IEEE, 2019, pp. 1199–1209.

[72] A. C.-C. Yao, "How to generate and exchange secrets," in *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*. IEEE, 1986, pp. 162–167.

[73] J. Yuan and S. Yu, "Privacy preserving back-propagation neural network learning made practical with cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 1, pp. 212–221, 2013.

[74] Q. Zhang, C. Wang, H. Wu, C. Xin, and T. V. Phuong, "GELU-Net: A globally encrypted, locally unencrypted deep neural network for privacy-preserved learning." in *Proceedings of the IJCAI*, 2018, pp. 3933–3939.

[75] W. Zheng, A. Dave, J. G. Beekman, R. A. Popa, J. E. Gonzalez, and I. Stoica, "Opaque: An oblivious and encrypted distributed analytics platform," in *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation*, 2017, pp. 283–298.

[76] W. Zheng, R. A. Popa, J. E. Gonzalez, and I. Stoica, "Helen: Maliciously secure coopetitive learning for linear models," in *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE, 2019, pp. 724–738.

[77] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2921–2929.