United We Stand: Accelerating Privacy-Preserving Neural Inference by Conjunctive Optimization with Interleaved Nexus

Qiao Zhang[1], **Tao Xiang**[1], **Chunsheng Xin**[2], **Hongyi Wu**[3]

[1]College of Computer Science, Chongqing University, Chongqing, 400044, China
[2]Department of Electrical and Computer Engineering, Old Dominion University, Norfolk, VA, 23529, USA
[3]Department of Electrical and Computer Engineering, The University of Arizona, Tucson, AZ, 85721, USA
qiaozhang@cqu.edu.cn, txiang@cqu.edu.cn, cxin@odu.edu, mhwu@arizona.edu

Abstract

Privacy-preserving Machine Learning as a Service (MLaaS) enables the powerful cloud server to run its well-trained neural model upon the input from resource-limited client, with both of server's model parameters and client's input data protected. While computation efficiency is critical for the practical implementation of privacy-preserving MLaaS and it is inspiring to witness recent advances towards efficiency improvement, there still exists a significant performance gap to real-world applications. In general, state-of-the-art frameworks perform function-wise efficiency optimization based on specific cryptographic primitives. Although it is logical, such independent optimization for each function makes noticeable amount of expensive operations unremovable and misses the opportunity to further accelerate the performance by jointly considering privacy-preserving computation among adjacent functions. As such, we propose COIN: Conjunctive Optimization with Interleaved Nexus, which remodels mainstream computation for each function to conjunctive counterpart for composite function, with a series of united optimization strategies. Specifically, COIN jointly computes a pair of consecutive nonlinear-linear functions in the neural model by reconstructing the intermediates throughout the whole procedure, which not only eliminates the most expensive crypto operations without invoking extra encryption enabler, but also makes the online crypto complexity independent of filter size. Experimentally, COIN demonstrates 11.2× to 29.6× speedup over various function dimensions from modern networks, and 6.4× to 12× speedup on the total computation time when applied in networks with model input from small-scale CIFAR10 to large-scale ImageNet.

Introduction

Machine Learning as a Service (MLaaS) establishes a kind of service pattern where the client $\mathcal{C}$ uploads her input to a cloud server $\mathcal{S}$ which runs its well-trained neural network and returns inference output to $\mathcal{C}$. Such mode relives the awkward requirement, to the clients, for substantial training data and powerful computational resources to train and apply DL models for their specific needs (Najafabadi et al. 2015). However, data privacy arises as a critical challenge in the practical implementation of MLaaS, as $\mathcal{C}$ needs to transmit her private data to $\mathcal{S}$. On the one hand, clients naturally wish to obtain the model output without any other parties know their input. In fact, regulations have been in place to prohibit the disclosure of private data in various domains, e.g., the Health Insurance Portability and Accountability Act (HIPAA) (Assistance 2003) for medical information and the General Data Protection Regulation (GDPR) (Goddard 2017) for business records. On the other hand, $\mathcal{S}$ seeks to maintain the confidentiality of model parameters to safeguard its intellectual property, and only provide the model output in response to the client's inference query.

Privacy-preserving MLaaS offers a promising solution to reconcile MLaaS with data protection. It integrates cryptographic primitives (e.g., Homomorphic Encryption (HE) (Brakerski 2012; Fan and Vercauteren 2012; Cheon et al. 2017) and Multi-Party Computation techniques (MPC) such as Garbled Circuits (GC) (Bellare, Hoang, and Rogaway 2012; Yao 1986) and Oblivious Transfer (OT) (Brassard, Crépeau, and Robert 1986)) into function computation of neural networks such that the server is oblivious to the client's private input while the client gains no access to the server's proprietary model parameters beyond necessary inference output, e.g., the predicted class (Gilad-Bachrach et al. 2016). However, computation efficiency acts as a fundamental challenge to make privacy-preserving MLaaS practical, since large-size circuits and/or function approximations are needed to apply cryptographic primitives to compute all functions, which may result in much high computation complexity and/or degraded prediction accuracy.

A series of recent works have made encouraging progress towards improving system efficiency of privacy-preserving MLaaS (Demmler, Schneider, and Zohner 2015; Liu et al. 2017; Mohassel and Zhang 2017; Juvekar, Vaikuntanathan, and Chandrakasan 2018; Riazi et al. 2018; Rouhani, Riazi, and Koushanfar 2018; Mohassel and Rindal 2018; Riazi et al. 2019; Mishra et al. 2020; Rathee et al. 2020; Boemer et al. 2020; Zhang, Xin, and Wu 2021; Patra et al. 2021; Tan et al. 2021; Hussain et al. 2021; Ng et al. 2021; Huang et al. 2022). Among them, the mixed-primitive frameworks which utilize HE to compute linear functions (e.g., convolution and fully connection) while adopt MPC for nonlinear functions (e.g., ReLU) have demonstrated additional efficiency advantages (Liu et al. 2017; Juvekar, Vaikuntanathan, and Chandrakasan 2018; Rathee et al. 2020; Huang et al. 2022) and it is noteworthy that the inference speed has been improved

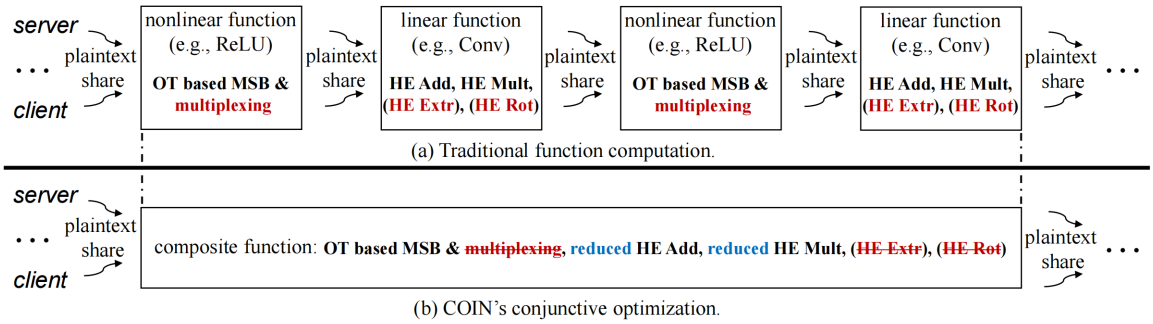

Figure 1: Comparison of basic computation logic between mainstream approaches and COIN.

by several orders of magnitude increase from early HE-based frameworks like CryptoNets (Gilad-Bachrach et al. 2016) to recent mixed-primitive approaches such as CrypTFlow2 (Rathee et al. 2020) and Cheetah (Huang et al. 2022). However, there is still a performance gap to real-world applications and this work aims to further accelerate the computation efficiency of mixed-primitive schemes.

At a high level, the core logic in state-of-the-art privacy-preserving frameworks, particularly the mixed-primitive frameworks, revolves around the fundamental concept of *function-wise optimization*: First, the output of a function is computed based on specific cryptographic primitives, and is shared between $\mathcal{C}$ and $\mathcal{S}$. Second, the respective shares held by $\mathcal{C}$ and $\mathcal{S}$ serve as the input of next function. Since a neural network consists of a stack of linear and nonlinear functions, this function-wise optimization is sequentially applied on functions from beginning to the end. Figure 1(a) shows the basic diagram of such process.

Key observations. While the function-wise optimization is logical and is also followed by most of privacy-preserving frameworks, there is a need to reconsider the computational cost associated with independently computing each function output, particularly the overhead involved in HE-based computation for each linear function. For instance, our preliminary experiments show that one HE-based convolution in VGG-19 (Simonyan and Zisserman 2014) with input size 14×14@512, kernel size 3×3@512, padding size 1×1, and stride size 1×1 takes about 39 seconds under CrypTFlow2 framework on the Intel Core i7-11370H@3.30GHz CPU. In contrast, computing the OT-based nonlinear ReLU with the same input size, which is the previous function to the convolution, only takes about 0.9 seconds, representing only around 2.3% of the time taken by the HE-based convolution. As a neural network consists of a stack of linear and nonlinear functions, this disparity highlights HE operations as the main obstacle in achieving efficient model computation for practical privacy-preserving MLaaS. Since rotation is the most expensive HE operation among the three basic ones (addition, multiplication, and rotation) in HE-based computation for linear functions, minimizing the number of involved rotations becomes crucial to improve the overall computational efficiency (Juvekar, Vaikuntanathan, and Chandrakasan 2018; Rathee et al. 2020; Zhang, Xin, and Wu 2021; Huang et al. 2022).

However, function-wise optimization makes most of expensive HE operations unremovable, which limits further efficiency improvement of privacy-preserving MLaaS. For example, CrypTFlow2 (Rathee et al. 2020) needs rotations with numbers proportional to filter size (e.g., output channels and filter width) to compute the encrypted convolution output. Cheetah (Huang et al. 2022) and Zhang (Zhang et al. 2022) eliminate the rotations by either introducing other HE operation namely Extraction (Extr) or involving intensive ciphertext communication at running time, with all complexities proportional to output channels. As the filter size is large in practical models such as 3×3@512 in VGG (Simonyan and Zisserman 2014), we are motivated to surpass the limitation of computational complexity for function-wise optimization by jointly considering the computing process of composite functions in a neural model. Specifically, we focus on the composite function $f_{\mathsf{c}}f_{\mathsf{r}}f_{\mathsf{c}}f_{\mathsf{r}}(x)$ which includes a pair of consecutive ReLU $f_{\mathsf{r}}(\cdot)$ and convolution function $f_{\mathsf{c}}(\cdot)$, and forms a basic combination in widely-applied networks (Krizhevsky, Sutskever, and Hinton 2012; Simonyan and Zisserman 2014; He et al. 2016).

Instead of separately minimizing the MPC-based cost for getting $y = f_{\mathsf{r}}(x)$ (or $\gamma = f_{\mathsf{r}}(z)$) with respect to x (or z), and the HE cost for computing $z = f_{\mathsf{c}}(y)$ (or $f_{\mathsf{c}}(\gamma)$) with respect to y (or γ), which is the mainstream logic among state-of-the-art frameworks, we propose a conjunctive optimization of composite function $f_{\mathsf{c}}f_{\mathsf{r}}f_{\mathsf{c}}f_{\mathsf{r}}(x)$ with respect to input x. By linking two adjacent linear functions with their nonlinear input as illustrated in Figure 1(b), we aim to reduce the number of costly and unremovable operations within the whole procedure, which is not possible under function-wise computation in mainstream works. To make such optimization truly advantageous, we need to determine the right terms for each party to calculate, ensuring that the overall computation cost for $f_{\mathsf{c}}f_{\mathsf{r}}f_{\mathsf{c}}f_{\mathsf{r}}(x)$ is significantly reduced.

Given the input-independent pregeneration of one share of each function output, we design an online-offline assignment of unfolded terms in $z = f_{\mathsf{c}}f_{\mathsf{r}}(x)$ and subsequent $f_{\mathsf{c}}f_{\mathsf{r}}(z)$. With the client's pregenerated share for $f_{\mathsf{c}}f_{\mathsf{r}}(x)$ and the server's pregenerated share for $f_{\mathsf{c}}f_{\mathsf{r}}(z)$, we construct a set of specific ciphertext and masked intermediates in an input-independent offline phase, and make the data-dependent online computation not only escape part of MPC-based computation, such as multiplexing, but also involve

Func.s	Schemes	Computation Cost for HE Operations		Communication Cost	
		# Rot	# Mult (Add)	# Ciphertexts	# Round
$2\times f_{\mathsf{c}}f_{\mathsf{r}}(x)$	DELPHI	$O(C_iC_oH_iW_i(f_h)^2)$	$O(C_iC_oH_iW_i(f_h)^2)$	$O(C_iH_iW_i + C_oH_oW_o)$	$2+\mathsf{rd}_{\{f'_{\mathsf{r}}(x),\mathsf{Mux}(x)\}}$
	CrypTFlow2	$O(C_iC_oH_iW_i(f_h)^2)$	$O(C_iC_oH_iW_i(f_h)^2)$	$O(C_iH_iW_i + C_oH_oW_o)$	$2+\mathsf{rd}_{\{f'_{\mathsf{r}}(x),\mathsf{Mux}(x)\}}$
	Cheetah	0	$O(C_iC_oH_iW_i)$	$O(C_iH_iW_i + C_oH_oW_o)$	$2+\mathsf{rd}_{\{f'_{\mathsf{r}}(x),\mathsf{Mux}(x)\}}$
	COIN	0	$O(C_iH_iW_i)$	$O(C_iH_iW_i)$	$2+\mathsf{rd}_{\{f'_{\mathsf{r}}(x)\}}$
$2\times f_{\mathsf{w}}f_{\mathsf{r}}(x)$	HElib	$O(n_i)$	$O(n_i)$	$O(n_i)$	$2+\mathsf{rd}_{\{f'_{\mathsf{r}}(x),\mathsf{Mux}(x)\}}$
	GAZELLE	$O(n_in_o - \log n_o)$	$O(n_in_o)$	$O(n_in_o)$	$2+\mathsf{rd}_{\{f'_{\mathsf{r}}(x),\mathsf{Mux}(x)\}}$
	Cheetah	0	$O(n_in_o)$	$O(n_in_o)$	$2+\mathsf{rd}_{\{f'_{\mathsf{r}}(x),\mathsf{Mux}(x)\}}$
	COIN	0	$O(n_i)$	$O(n_i)$	$2+\mathsf{rd}_{\{f'_{\mathsf{r}}(x)\}}$

Table 1: Running-time complexities of COIN to compute composite functions compared with DELPHI (Mishra et al. 2020), CrypTFlow2 (Rathee et al. 2020), Cheetah (Huang et al. 2022), HElib (Halevi and Shoup 2014), and GAZELLE (Juvekar, Vaikuntanathan, and Chandrakasan 2018). Here the fully connection (FC) is denoted as $f_{\mathsf{w}}(\cdot)$. The input of $f_{\mathsf{c}}(\cdot)$ is with size $C_i \times H_i \times W_i$. C_o and f_h are the number and the size of filters, respectively. H_o and W_o are the height and width of convolution output. The FC inputs an n_i-sized vector and outputs the n_o-sized one. rd is the number of communication rounds for target functions. $f'_{\mathsf{r}}(x)$ and $\mathsf{Mux}(x)$ are the derivative of ReLU, and the multiplexing of getting x, respectively. The communication cost to compute $f'_{\mathsf{r}}(x)$ is excluded, and Cheetah needs HE Extr operations in $O(C_oH_oW_o)$ for $f_{\mathsf{c}}f_{\mathsf{r}}(x)$ and in $O(n_o)$ for $f_{\mathsf{w}}f_{\mathsf{r}}(x)$.

filter-independent number of HE additions and multiplications. Such joint computation with interleaved share pregeneration, which we call *COIN: Conjunctive Optimization with Interleaved Nexus*, makes it possible to surpass the complexity limitation in function-wise frameworks, resulting in *elimination of all HE rotations, reduced communication rounds, and light involvement of HE addition and multiplication, without other HE operations such as Extraction*. For example, computing adjacent ReLU and convolution $f_{\mathsf{c}}f_{\mathsf{r}}f_{\mathsf{c}}f_{\mathsf{r}}(x)$ with COIN under aforementioned sizes shows a speedup about 30 times compared to mainstream approaches. Such efficiency advantage makes COIN promising to be applied in AI-based clinical diagnosis, which has been deployed by Ant Group, where a small clinic could obtain a timely and more precise diagnostic analysis result from central hospitals that possess more comprehensive DL models.

Our contributions. Overall, the contributions of this work are summarized as follows.

- We initiate the comprehensive investigation of conjunctive optimization for efficient privacy-preserving MLaaS and introduce our approach called COIN, which challenges the conventional function-wise mode by remodeling the computation process from input to output of the same function, to the conjunctive counterpart that goes through consecutive functions in a neural model.
- The complexity advantages of COIN's optimization for composite functions are highlighted in Table 1. One notable benefit is that COIN eliminates the need for additional cryptographic operations while removing rotations. Besides, COIN ensures that the running-time cryptographic complexity is independent of filter size, which contributes to the overall efficiency and effectiveness to boost the performance of computing the whole model.
- Through extensive experiments, COIN outperforms the function-wise computation typically employed in state-of-the-art works. Specifically, COIN exhibits a speedup from 11.2× to 29.6× across various function dimensions in modern networks. Furthermore, COIN achieves notable speed gains of 6.4× to 12× when integrated into networks with datasets from small-scale CIFAR10 to large-scale ImageNet.

The rest of the paper is organized as follows. First, we introduce system setup and primitives that are adopted in COIN. Then, the design of COIN to compute $f_{\mathsf{c}}f_{\mathsf{r}}f_{\mathsf{c}}f_{\mathsf{r}}(x)$ and the strategies that best adapt COIN to neural networks are elaborated. Next, the experimental results are illustrated and discussed. Finally, we conclude the paper.

Preliminaries

Notations. $[\![i]\!]$ is the set of integers $\{0, 1, \ldots, i-1\}$ for nonzero integer i. $\lceil\cdot\rceil$ and $\lfloor\cdot\rfloor$ are the ceiling function and flooring function, respectively. $r \xleftarrow{\$} \mathcal{D}$ randomly samples a component r from a set $\mathcal{D}$. The logical XOR is $\oplus$. $\mathbb{Z}_p = \mathbb{Z} \cap [-\lfloor p/2 \rfloor, \lfloor p/2 \rfloor]$ for $p > 2$, and $\mathbb{Z}_2 = \{0, 1\}$. Furthermore, $+, -$, and $\boxtimes$ are element-wise addition, subtraction, and multiplication in either ciphertext or plaintext domain, depending on whether ciphertext is involved or not.

System Model

We consider the context of cryptographic inference as shown in Figure 2 where $\mathcal{C}$ holds a private input while $\mathcal{S}$ owns the network with proprietary model parameters. The server processes $\mathcal{C}$'s private input through a sequence of linear and nonlinear functions to finally classify the input into one of the potential classes. After the inference, $\mathcal{C}$ learns the network architecture such as the types and dimensions of involved functions, and the network output, while $\mathcal{S}$ learns nothing. Such learnt information is commonly assumed in

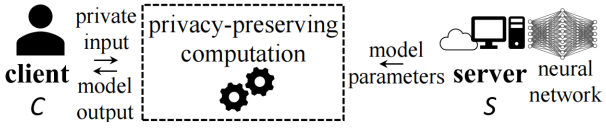

Figure 2: Overview of Cryptographic Inference.

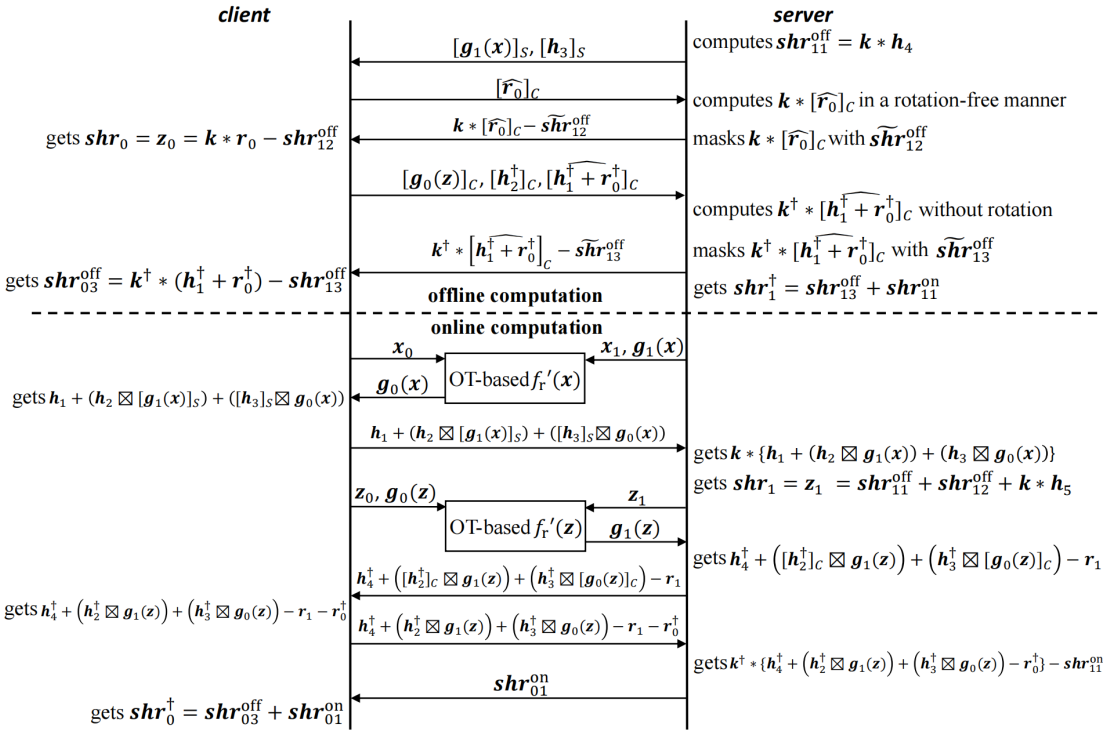

Figure 3: Overview of COIN to compute composite function $f_c f_r f_c f_r(x)$.

the state-of-the-art frameworks, e.g., MiniONN (Liu et al. 2017) and Cheetah (Huang et al. 2022). Furthermore, we target at the functions widely-applied in Convolutional Neural Network (CNN). As for the linear functions, we focus on four types namely Convolution (Conv), Fully Connection (FC), Batch Normalization (BN), and Mean Pooling (MeanPool). As for the nonlinear functions, we mainly deal with ReLU and Max Pooling (MaxPool).

Threat Model and Security

COIN is secure against a semi-honest adversary based on the simulation paradigm (Lindell 2016). Specifically, a computationally bounded adversary corrupts either $\mathcal{C}$ or $\mathcal{S}$ at the beginning of the protocol while it follows the protocol specification honestly. Security is modeled by defining two interactions: a real-world interaction where $\mathcal{C}$ and $\mathcal{S}$ execute protocol in presence of an adversary and environment, and an ideal-world interaction where both parties send their inputs to a trusted third party that computes target functionality faithfully. Security requires that for every adversary in real world, there is a simulator, in the ideal world, which makes no environment be able to distinguish between real-world and ideal-world interactions.

Cryptographic Primitives

Homomorphic Encryption (HE). HE is a primitive that supports linear operations over encrypted vectors without decryption, and the encrypted output matches the corresponding operations on plaintext (Juvekar, Vaikuntanathan, and Chandrakasan 2018). The correctness of HE is firstly guaranteed by a decryption process such that $\boldsymbol{x} = D(sk, [\boldsymbol{x}])$ where sk is the secret key, $\boldsymbol{x} = (x_0, x_1, \ldots, x_{N-1}) \in \mathbb{Z}_p^N$, and the ciphertext $[\boldsymbol{x}] = E(pk, \boldsymbol{x})$ where pk is the public key. Such process is done by either the client or the server when attached with subscript $\mathcal{C}$ or $\mathcal{S}$. Second, HE is able to securely evaluate an arithmetic circuit consisting of addition and multiplication gates based on the following homomorphic operations: (1) Addition (Add, $+$): $D(sk, [\boldsymbol{u}] + [\boldsymbol{v}]) = \boldsymbol{u} + \boldsymbol{v}$ and $D(sk, [\boldsymbol{u}] + \boldsymbol{v}) = \boldsymbol{u} + \boldsymbol{v}$. (2) Subtraction (Sub, $-$): $D(sk, [\boldsymbol{u}] - [\boldsymbol{v}]) = \boldsymbol{u} - \boldsymbol{v}$ and $D(sk, [\boldsymbol{u}] - \boldsymbol{v}) = \boldsymbol{u} - \boldsymbol{v}$. (3) Multiplication (Mult, $\boxtimes$): $D(sk, [\boldsymbol{u}] \boxtimes \boldsymbol{v}) = \boldsymbol{u} \boxtimes \boldsymbol{v}$. (4) Rotation (Rot,$R([\boldsymbol{u}], l)$): $D(sk, R([\boldsymbol{u}], l)) = \boldsymbol{u}_\ell$. Here $\boldsymbol{u}_\ell = (u_\ell, \ldots, u_{N-1}, u_0, \ldots, u_{\ell-1})$ and $\ell \in [\![N]\!]$, $\boldsymbol{u} = (u_0, \ldots, u_{N-1})$, $\boldsymbol{v} = (v_0, \ldots, v_{N-1}) \in \mathbb{Z}_p^N$, and a rotation by $(-\ell)$ is the same as a rotation by $(N - \ell)$.

Additive Secret Sharing (ASS). ASS generates shares of input $x \in \mathbb{Z}_p$ as shr_0 and shr_1. The shares are randomly sampled in $\mathbb{Z}_p$ such that $shr_0 + shr_1 = x \bmod p$. In this work, ASS is adopted to form the shares of intermediates in computing composite functions.

Oblivious Transfer (OT). OT involves a sender that owns multiple messages and a receiver that has the message index. The receiver can get the desired message at the end of the protocol without knowing sender's other messages, and the sender neither knows which message the receiver obtains. OT is utilized to realize a highly efficient computation for derivative of ReLU (Rathee et al. 2020) and COIN adopts such module to facilitate its joint computation.

System Description

Overview

Figure 3 illustrates the overview of COIN to compute composite function $f_{\mathsf{c}}f_{\mathsf{r}}f_{\mathsf{c}}f_{\mathsf{r}}(\boldsymbol{x})$. By considering the pregeneration property of one share of each function, we specifically associate the unfolded terms in $f_{\mathsf{c}}f_{\mathsf{r}}f_{\mathsf{c}}f_{\mathsf{r}}(\boldsymbol{x})$ such that large portion of the computation is offloaded into an input-independent offline phase while the overall overhead is comparable to SOTA frameworks. Particularly, the whole procedure is divided into two phases namely online computation and offline computation, and the offline computation mainly involves in a Rot-free sharing to prepare intermediates based on which the online computation enables each party to finally get the share of $f_{\mathsf{c}}f_{\mathsf{r}}f_{\mathsf{c}}f_{\mathsf{r}}(\boldsymbol{x})$ in a much more efficient way with reduced communication rounds and filter-independent crypto complexity. Moreover, since a neural model also includes other functions than $f_{\mathsf{c}}f_{\mathsf{r}}f_{\mathsf{c}}f_{\mathsf{r}}(\boldsymbol{x})$, a series of structure adaptations are introduced to maximize COIN's computational advantages.

Conjunctive Optimization for $f_{\mathbf{c}}f_{\mathbf{r}}f_{\mathbf{c}}f_{\mathbf{r}}(\boldsymbol{x})$

Recall that we aim to optimize $f_{\mathsf{c}}f_{\mathsf{r}}f_{\mathsf{c}}f_{\mathsf{r}}(\boldsymbol{x})$ such that the efficiency is much improved than function-wise computation of mainstream frameworks. We achieve this goal by first investigating the unfolded expression of $\boldsymbol{z} = f_{\mathsf{c}}f_{\mathsf{r}}(\boldsymbol{x})$, and then bringing $f_{\mathsf{c}}f_{\mathsf{r}}(\boldsymbol{z})$ into consideration. Specifically, $f_{\mathsf{c}}f_{\mathsf{r}}(\boldsymbol{x})$ is expressed as:

$$\begin{aligned} &f_{\mathsf{c}}f_{\mathsf{r}}(\boldsymbol{x}) = \boldsymbol{k} * f_{\mathsf{r}}(\boldsymbol{x}) = \boldsymbol{k} * \{f'_{\mathsf{r}}(\boldsymbol{x}) \boxtimes \boldsymbol{x})\} \\ &= \boldsymbol{k} * \{\boldsymbol{h}_1 + \boldsymbol{r}_0 + \boldsymbol{h}_2 \boxtimes \boldsymbol{g}_1(\boldsymbol{x}) + \boldsymbol{h}_3 \boxtimes \boldsymbol{g}_0(\boldsymbol{x})\} + \boldsymbol{k} * \boldsymbol{h}_4 \\ &= \boldsymbol{k} * \boldsymbol{h}_5 + \boldsymbol{k} * \boldsymbol{h}_4 + \boldsymbol{k} * \boldsymbol{r}_0 \end{aligned} \tag{1}$$

where

$$\begin{cases} \boldsymbol{h}_1 = \boldsymbol{x}_0 \boxtimes \boldsymbol{g}_0(\boldsymbol{x}) - \boldsymbol{r}_0 \\ \boldsymbol{h}_2 = \boldsymbol{x}_0 \boxtimes \{1 - 2 \boxtimes \boldsymbol{g}_0(\boldsymbol{x})\} \\ \boldsymbol{h}_3 = \boldsymbol{x}_1 \boxtimes \{1 - 2 \boxtimes \boldsymbol{g}_1(\boldsymbol{x})\} \\ \boldsymbol{h}_4 = \boldsymbol{x}_1 \boxtimes \boldsymbol{g}_1(\boldsymbol{x}) \\ \boldsymbol{h}_5 = \boldsymbol{h}_1 + \boldsymbol{h}_2 \boxtimes \boldsymbol{g}_1(\boldsymbol{x}) + \boldsymbol{h}_3 \boxtimes \boldsymbol{g}_0(\boldsymbol{x}), \end{cases} \tag{2}$$

kernel $\boldsymbol{k} \in \mathbb{Z}_p^{C_o \times C_i \times f_h \times f_h}$. $\boldsymbol{x}, \boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{r}_0, \boldsymbol{h}_1, \boldsymbol{h}_2, \boldsymbol{h}_3, \boldsymbol{h}_4 \in \mathbb{Z}_p^{C_i \times H_i \times W_i}$. $\boldsymbol{x}_0$ and $\boldsymbol{x}_1$ are shares at $\mathcal{C}$ and $\mathcal{S}$ satisfying $\boldsymbol{x}_0 + \boldsymbol{x}_1 = \boldsymbol{x}$. $\boldsymbol{r}_0$ is generated by $\mathcal{C}$. $\boldsymbol{g}_0(\boldsymbol{x})$ and $\boldsymbol{g}_1(\boldsymbol{x})$ are the respective Boolean shares of client and server obtained from the OT-based module for getting derivative of ReLU $f'_{\mathsf{r}}(\boldsymbol{x})$ (Rathee et al. 2020) where $\boldsymbol{g}_0(\boldsymbol{x}) \oplus \boldsymbol{g}_1(\boldsymbol{x}) = f'_{\mathsf{r}}(\boldsymbol{x}) \in \mathbb{Z}_2^{C_i \times H_i \times W_i}$. Here we unfold $f_{\mathsf{c}}f_{\mathsf{r}}(\boldsymbol{x})$ by putting together the variables that belong to either $\mathcal{C}$ or $\mathcal{S}$.

Since $\boldsymbol{k}$ is owned by $\mathcal{S}$, $\boldsymbol{r}_0$ and one of the shares of both $\boldsymbol{x}$ and $f'_{\mathsf{r}}(\boldsymbol{x})$ could be pregenerated, we first utilize the pregeneration of $\boldsymbol{x}_1, \boldsymbol{g}_1(\boldsymbol{x})$ and $\boldsymbol{r}_0$ to construct necessary intermediates at offline phase to enable much more efficient online computation. Specifically, $\mathcal{S}$ obtains plaintext $\boldsymbol{k} * \boldsymbol{h}_4$ in Eq. (1). It also encrypts $\boldsymbol{g}_1(\boldsymbol{x})$ and $\boldsymbol{h}_3$, and sends them to the client. Furthermore, a rotation-free computation, with only HE addition and multiplication, is conducted to obtain shares of $\boldsymbol{k} * \boldsymbol{r}_0$ at both parties. Such elimination of Rot is inspired from the relationships among Conv, dot product

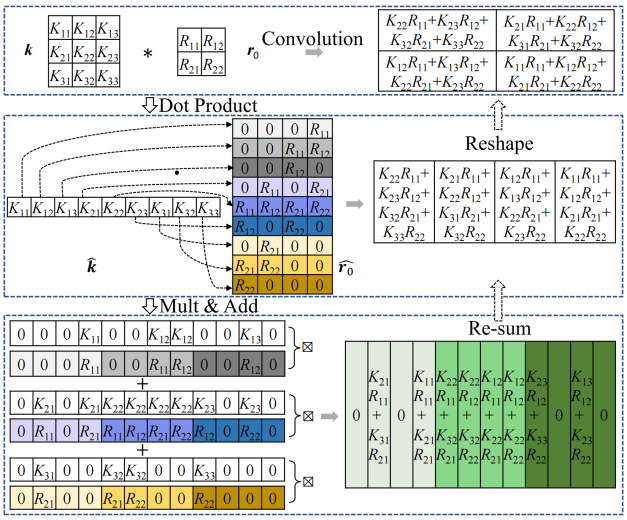

Figure 4: Relations among Conv, FC, and HE operations.

namely FC, and HE operations, as illustrated in Figure 4. As each number of convolution output is the sum of kernel values scaled by input elements that are within the kernel window, the overall convolution is equivalent to the dot product between a flattened kernel matrix and a re-organized input matrix (Jia et al. 2014). Therefore, $\mathcal{C}$ encrypts multiple rows of the re-organized $\boldsymbol{r}_0$ namely $\widehat{\boldsymbol{r}_0}$ which enables $\mathcal{S}$ to perform HE Mult, Add, and ciphertext sharing to finally share the $\boldsymbol{k} * \boldsymbol{r}_0$ at both parties. Given the constructed shares and the $\mathcal{S}$-encrypted $\boldsymbol{g}_1(\boldsymbol{x})$ and $\boldsymbol{h}_3$ formed at offline phase, the online phase involves an OT-based module for getting share of $f'_{\mathsf{r}}(\boldsymbol{x})$ at the client namely $\boldsymbol{g}_0(\boldsymbol{x})$. Then $\mathcal{C}$ is able to obtain encrypted $\boldsymbol{h}_5$ based on Eq. (2), which is later decrypted by the server to obtain its share of $\boldsymbol{z} = f_{\mathsf{c}}f_{\mathsf{r}}(\boldsymbol{x})$ namely $\boldsymbol{z}_1$. While the share of $\boldsymbol{z} = f_{\mathsf{c}}f_{\mathsf{r}}(\boldsymbol{x})$ at $\mathcal{C}$ namely $\boldsymbol{z}_0$ comes from its share of $\boldsymbol{k} * \boldsymbol{r}_0$ at offline phase.

To efficiently enable subsequent computation for $f_{\mathsf{c}}f_{\mathsf{r}}(\boldsymbol{z})$ based on known $\boldsymbol{z}_0$ at $\mathcal{C}$ and online-formed $\boldsymbol{z}_1$ at $\mathcal{S}$, we let the client pregenerate Boolean share of $f'_{\mathsf{r}}(\boldsymbol{z})$ namely $\boldsymbol{g}_0(\boldsymbol{z})$ and rewriting Eq. (1) as

$$\begin{aligned} &f_{\mathsf{c}}f_{\mathsf{r}}(\boldsymbol{z}) = \boldsymbol{k}^\dagger * f_{\mathsf{r}}(\boldsymbol{z}) = \boldsymbol{k}^\dagger * \{f'_{\mathsf{r}}(\boldsymbol{z}) \boxtimes \boldsymbol{z})\} \\ &= \boldsymbol{k}^\dagger * \{\boldsymbol{h}_4^\dagger + \boldsymbol{h}_2^\dagger \boxtimes \boldsymbol{g}_1(\boldsymbol{z}) + \boldsymbol{h}_3^\dagger \boxtimes \boldsymbol{g}_0(\boldsymbol{z}) - \boldsymbol{r}_0^\dagger\} + \\ &\boldsymbol{k}^\dagger * (\boldsymbol{h}_1^\dagger + \boldsymbol{r}_0^\dagger) = \boldsymbol{k}^\dagger * (\boldsymbol{h}_5^\dagger - \boldsymbol{r}_0^\dagger) + \boldsymbol{k}^\dagger * (\boldsymbol{h}_1^\dagger + \boldsymbol{r}_0^\dagger) \end{aligned} \tag{3}$$

where

$$\begin{cases} \boldsymbol{h}_1^\dagger = \boldsymbol{z}_0 \boxtimes \boldsymbol{g}_0(\boldsymbol{z}) \\ \boldsymbol{h}_2^\dagger = \boldsymbol{z}_0 \boxtimes \{1 - 2 \boxtimes \boldsymbol{g}_0(\boldsymbol{z})\} \\ \boldsymbol{h}_3^\dagger = \boldsymbol{z}_1 \boxtimes \{1 - 2 \boxtimes \boldsymbol{g}_1(\boldsymbol{z})\} \\ \boldsymbol{h}_4^\dagger = \boldsymbol{z}_1 \boxtimes \boldsymbol{g}_1(\boldsymbol{z}) \\ \boldsymbol{h}_5^\dagger = \boldsymbol{h}_4^\dagger + \boldsymbol{h}_2^\dagger \boxtimes \boldsymbol{g}_1(\boldsymbol{z}) + \boldsymbol{h}_3^\dagger \boxtimes \boldsymbol{g}_0(\boldsymbol{z}) \end{cases} \tag{4}$$

kernel $\boldsymbol{k}^\dagger \in \mathbb{Z}_p^{C_o^\dagger \times C_o \times f_h^\dagger \times f_h^\dagger}$. $\boldsymbol{z}, \boldsymbol{z}_0, \boldsymbol{z}_1, \boldsymbol{h}_1^\dagger, \boldsymbol{h}_2^\dagger, \boldsymbol{h}_3^\dagger, \boldsymbol{h}_4^\dagger, \boldsymbol{r}_0^\dagger \in \mathbb{Z}_p^{C_o \times H_o \times W_o}$, and $\boldsymbol{r}_0^\dagger$ is randomly generated by $\mathcal{C}$. $\boldsymbol{g}_0(\boldsymbol{z})$ and $\boldsymbol{g}_1(\boldsymbol{z})$ are the respective Boolean shares of client and server and $\boldsymbol{g}_0(\boldsymbol{z}) \oplus \boldsymbol{g}_1(\boldsymbol{z}) = f'_{\mathsf{r}}(\boldsymbol{z}) \in \mathbb{Z}_2^{C_o \times H_o \times W_o}$.

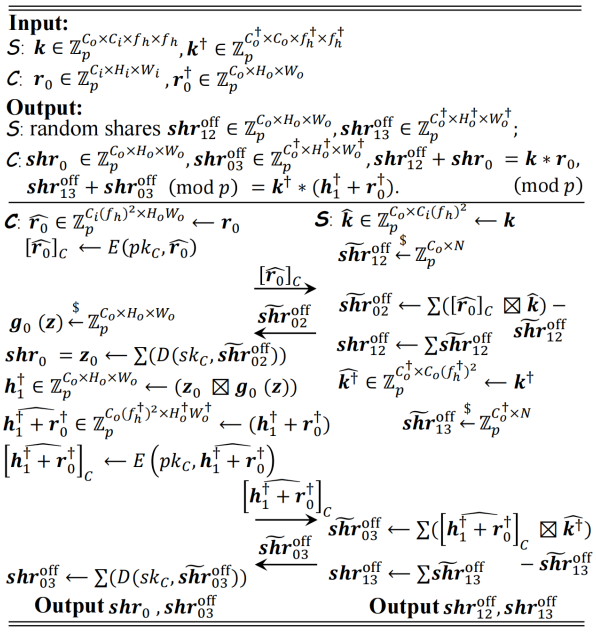

Figure 5: Offline sharing of $\boldsymbol{k} * \boldsymbol{r}_0$ and $\boldsymbol{k}^\dagger * (\boldsymbol{h}_1^\dagger + \boldsymbol{r}_0^\dagger)$.

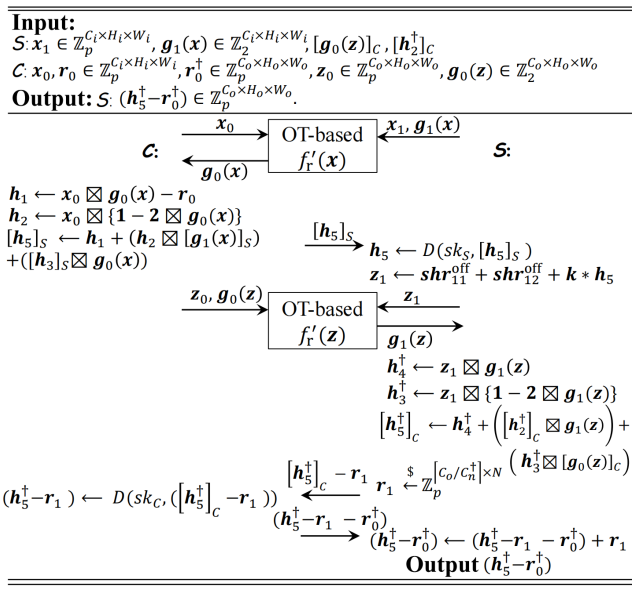

Figure 6: Rot-free computation for $(\boldsymbol{h}_5^\dagger - \boldsymbol{r}_0^\dagger)$ at online phase.

Given $\boldsymbol{k}^\dagger$ at $\mathcal{S}$, and the obtained $\boldsymbol{g}_0(\boldsymbol{z})$, $\boldsymbol{z}_0$, and $\boldsymbol{r}_0^\dagger$, we treat $(\boldsymbol{h}_1^\dagger + \boldsymbol{r}_0^\dagger)$ as $\boldsymbol{r}_0$ in Eq. (1) and $\boldsymbol{k}^\dagger * (\boldsymbol{h}_1^\dagger + \boldsymbol{r}_0^\dagger)$ is thus shared in a way similar to the sharing of $\boldsymbol{k} * \boldsymbol{r}_0$ at offline phase. Figure 5 details the offline sharing as a whole. On the other hand, $(\boldsymbol{h}_5^\dagger - \boldsymbol{r}_0^\dagger)$ acts similarly as $\boldsymbol{h}_5$ in Eq. (5) which enables $\mathcal{S}$ to obtain plaintext $(\boldsymbol{h}_5^\dagger - \boldsymbol{r}_0^\dagger)$, and the concrete process is described in Figure 6. Then, $\mathcal{S}$ directly gets $\boldsymbol{k}^\dagger * (\boldsymbol{h}_5^\dagger - \boldsymbol{r}_0^\dagger)$ in plaintext, and shares the result with $\mathcal{C}$. After that, $\mathcal{S}$ and $\mathcal{C}$ locally obtain their respective shares of $f_\mathrm{c} f_\mathrm{r} f_\mathrm{c} f_\mathrm{r}(\boldsymbol{x})$ namely $\boldsymbol{shr}_1^\dagger$ and $\boldsymbol{shr}_0^\dagger$, which serve as the input of subsequent computation.

Additionally, we deal with composite function consisting of ReLU and FC, $f_\mathrm{w} f_\mathrm{r} f_\mathrm{w} f_\mathrm{r}(\boldsymbol{x})$, in a way similar to that of computing $f_\mathrm{c} f_\mathrm{r} f_\mathrm{c} f_\mathrm{r}(\boldsymbol{x})$ where $\boldsymbol{k}$ and convolution " $*$ " are replaced by weight matrix $\boldsymbol{w}$ and dot product " $\cdot$ ", respectively.

Network Adaptation for Best Usage

A neural network always contains other functions than $f_\mathrm{c} f_\mathrm{r} f_\mathrm{c} f_\mathrm{r}(\boldsymbol{x})$, e.g., MaxPool, MeanPool, and BN. Therefore, we propose to do function adaptation to maximize the utilization of proposed joint optimization. Firstly, we stretch $f_\mathrm{c} f_\mathrm{r} f_\mathrm{c} f_\mathrm{r}(\boldsymbol{x})$ into composite function including B consecutive blocks of combined function $f_\mathrm{c} f_\mathrm{r}(\cdot)$ as

$$\underbrace{f_\mathrm{c} f_\mathrm{r}}_{B} \cdots \underbrace{f_\mathrm{c} f_\mathrm{r}}_{2} \underbrace{f_\mathrm{c} f_\mathrm{r}}_{1}(\boldsymbol{x}) \tag{5}$$

Here, If $B = 1$, the shares of $\mathcal{S}$ and $\mathcal{C}$ namely $\boldsymbol{shr}_1$ and $\boldsymbol{shr}_0$ serve as the input of subsequent function. If $B = 2$, $\mathcal{S}$ sets its final share as $\{\boldsymbol{shr}_{13}^{\mathrm{off}} + \boldsymbol{k}^\dagger * (\boldsymbol{h}_5^\dagger - \boldsymbol{r}_0^\dagger)\}$ while $\mathcal{C}$ sets the one to $\boldsymbol{shr}_{03}^{\mathrm{off}}$. If $B \geq 3$, the computation for i-th ($i \geq 2$) block works as that for $f_\mathrm{c} f_\mathrm{r}(\boldsymbol{z})$ without the final sharing from $\mathcal{S}$ to $\mathcal{C}$. It makes $\mathcal{S}$ always get its share for previous block online similar to $\boldsymbol{z}_1$ in Figure 3, enabling iterative process in the logic of computing $f_\mathrm{c} f_\mathrm{r}(\boldsymbol{z})$. Similar logic is applied to composite function with hybrid $f_\mathrm{c}(\cdot)$ and $f_\mathrm{w}(\cdot)$.

Secondly, we aim to reassemble other functions in the neural network such that $f_\mathrm{c} f_\mathrm{r}(\boldsymbol{x})$ or $f_\mathrm{w} f_\mathrm{r}(\boldsymbol{x})$ appears as much as possible, enabling most optimization for composite function in form of Eq. (5). As such, we identify three function blocks where ReLU and Conv are separated by other functions and transform them into another composite functions where ReLU and Conv are adjacent. At a high level, function block in ReLU→MaxPool→Conv is equivalently shaped in MaxPool→ReLU→Conv. Function block in ReLU→MeanPool→Conv is transformed into another block with only ReLU and Conv by separating summing and averaging in MeanPool into ReLU and Conv. Finally, function block in Conv→BN is transformed into another Conv by combining the scalars in BN with the process for computing Conv.

Evaluation

We implement COIN based on the open-sourced code from CrypTFlow2 (Rathee et al. 2020) and all experiments are run in LAN with gigabit bandwidth. Each machine possesses a CPU with Intel Core i7-11370H@3.3GHz, and the system memory is 38.9GB. We evaluate COIN over VGG-19 (Simonyan and Zisserman 2014) and ResNet-34 (He et al. 2016) using CIFAR10 (cif 2021) and ImageNet (Deng et al. 2009) datasets.

Online Performance

Microbenchmarks. COIN features with a light-weight online complexity to compute composite function $f_\mathrm{c} f_\mathrm{r} f_\mathrm{c} f_\mathrm{r}(\boldsymbol{x})$. As such we demonstrate the concrete performance over $f_\mathrm{c} f_\mathrm{r}(\boldsymbol{x})$, which is the minimum block in $f_\mathrm{c} f_\mathrm{r} f_\mathrm{c} f_\mathrm{r}(\boldsymbol{x})$.

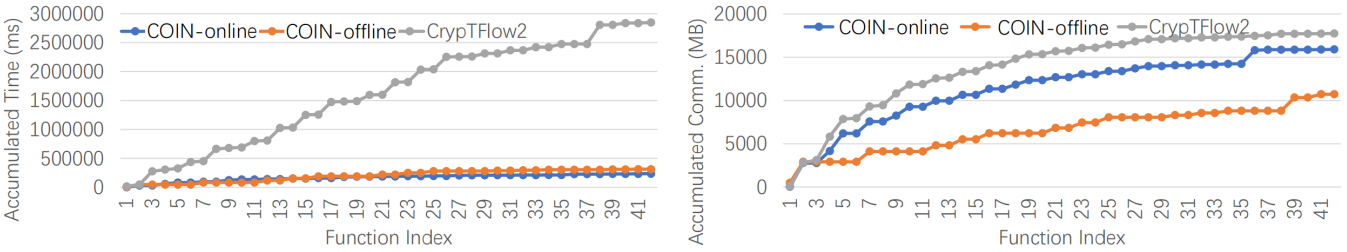

Figure 7: Performance Breakdown over VGG with ImageNet.

H_i, W_i, C_i s, f_h, C_o	Time (s)/Speedup			Comm. (MB)	
	COIN	CF2	GZ	COIN	CF2
14,14,512 1,3,512	1.32	39.2 29.6×	720×	86.8	95.4
28,28,128 1,3,128	1	11.27 11.2×	120×	86.9	95.62
8,8,128 2,3,256	0.27	6.75 24.5×	380×	7.57	9.23

Table 2: Online computation of $f_{\mathsf{c}} f_{\mathsf{r}}$ in various dimensions.

Data	Models	Time (s)/Speedup		Comm. (GB)	
		COIN	CF2	COIN	CF2
CIFAR10	VGG-19	12.5 6.4×	80.8	0.89	0.36
	ResNet-34	16.4 7×	116.6	0.26	0.27
ImageNet	VGG-19	235.8 12×	2848.6	15.5	17.3
	ResNet-34	77.4 8.6×	670.5	5.2	5

Table 3: Online computation of various neural models.

Specifically, the goal is to output shares of $f_{\mathsf{c}} f_{\mathsf{r}}(\boldsymbol{x})$ given the shares of $\boldsymbol{x}$ with size of $H_i \times W_i @ C_i$. As shown in Table 2, COIN shows up to 29.6× speedup and up to 720× speedup on various function sizes from modern networks compared to CrypTFlow2 and GAZELLE (CF2 and GZ in abbreviation), respectively. Meanwhile, COIN escapes multiplexing needed in mainstream works, enabling a lighter communication cost.

Over modern networks. Table 3 shows the performance over different modern networks by plugging in COIN's optimization. The key takeaways are summarized as follows. As for VGG-19 and ResNet-34 with CIFAR10, COIN shows 6.4× and 7× running-time speedup, respectively. On the other hand, COIN transfers more data than CrypTFlow2 especially over VGG-19. The main reason behind is the inclusion of offline process due to the nonpregenerated property of shares of MaxPool, which makes the offline computation for subsequent composite function turn to online phase, disabling our offline-online division and introducing extra communication load at running time. That offline inclusion has less impact on communication cost over ResNet-34 since there are less MaxPools. It implies that COIN is more suitable with less Maxpools in terms of large-size networks with small-scale input. As for VGG-19 and ResNet-34 with ImageNet, COIN shows 12× and 8.6× running-time speedup with reduced communication cost, respectively. The main reason is that the overall efficiency harvest of COIN's model adaptation towards Maxpool is more significant than the cost of corresponding offline inclusion. This suggests that COIN is applicable to large-size networks with large-scale input.

Offline Included Performance Breakdown

In order to concretely understand COIN's adaptability over different function dimensions, including the overhead in offline phase, we further break down the performance of tested networks in function wise. Figure 7 demonstrates the break down over VGG-19 with ImageNet. It shows that COIN outperforms CrypTFlow2 in terms of both time and communication at offline and online phases. The main reason is due to the large size of input makes our network adaptation for the five MaxPools more beneficial to the subsequent conjunctive optimization, with a much smaller input size after each pooling operation. The details over other networks can be similarly analyzed. Overall, COIN is more advantageous than function-wise computation over various CNNs with various input sizes, and similar observations are found in the WAN.

Conclusion

In this paper, we have embarked on comprehensive investigation on conjunctive optimization for efficient privacy-preserving MLaaS, and have proposed COIN which features by a computation module for composite function with a series of joint optimization strategies. COIN aims to remodel the computation process from input to output of the same function, to the conjunctive counterpart that goes through consecutive functions in a neural model. Theoretically, COIN not only eliminates the most expensive crypto operations without invoking extra encryption enabler, but also makes the running-time crypto complexity independent of filter size. Experimentally, COIN has demonstrated 11.2× to 29.6× speedup over various function dimensions from modern networks, and 6.4× to 12× speedup on the total computation time when plugged in networks with input from small-scale CIFAR10 to large-scale ImageNet. Additionally, although COIN deals with adjacent ReLU and Conv, it can be readily integrated into other complex architectures such as the feed-forward blocks in transformers, leakyReLU and piecewise linear functions.

Acknowledgments

The research of Q. Zhang and T. Xiang is supported by the National Key R&D Program of China under Grant 2022YFB3103500, the National Natural Science Foundation of China under Grants 62302067, 62072062 and U20A20176, China Postdoctoral Science Foundation under Grant 2023M730407, the Fundamental Research Funds for the Central Universities under Grant 2022CDJXY-020, CCF-AFSG Research Fund under Grant RF20220009, and the Technology Innovation and Application Development Key Project supported by Chongqing Science and Technology Bureau under Grant CSTB2022TIAD-KPX0178.

References

2021. CIFAR10 dataset. https://www.cs.toronto.edu/~kriz/cifar.html. Accessed: 2023-04-03.

Assistance, H. C. 2003. Summary of the HIPAA privacy rule. *Office for Civil Rights*.

Bellare, M.; Hoang, V. T.; and Rogaway, P. 2012. Foundations of garbled circuits. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, 784–796.

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

Brakerski, Z. 2012. Fully homomorphic encryption without modulus switching from classical GapSVP. In *Proceedings of the Annual Cryptology Conference*, 868–886. Springer.

Brassard, G.; Crépeau, C.; and Robert, J.-M. 1986. All-or-nothing disclosure of secrets. In *Proceedings of the Conference on the Theory and Application of Cryptographic Techniques*, 234–238. Springer.

Cheon, J. H.; Kim, A.; Kim, M.; and Song, Y. 2017. Homomorphic encryption for arithmetic of approximate numbers. In *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security*, 409–437. Springer.

Demmler, D.; Schneider, T.; and Zohner, M. 2015. ABY-A framework for efficient mixed-protocol secure two-party computation. In *Proceedings of the NDSS*.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Li, F.-F. 2009. ImageNet: A large-scale hierarchical image database. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, 248–255.

Fan, J.; and Vercauteren, F. 2012. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, 2012: 144.

Gilad-Bachrach, R.; Dowlin, N.; Laine, K.; E. Lauter, K.; Naehrig, M.; and Wernsing, J. 2016. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24*, 201–210.

Goddard, M. 2017. The EU General Data Protection Regulation (GDPR): European regulation that has a global impact. *International Journal of Market Research*, 59(6): 703–705.

Halevi, S.; and Shoup, V. 2014. Algorithms in HElib. In *Proceedings of the CRYPTO*, 554–571.

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

Huang, Z.; Lu, W.-j.; Hong, C.; and Ding, J. 2022. Cheetah: Lean and Fast Secure Two-Party Deep Neural Network Inference. In *Proceedings of the 31th USENIX Security Symposium (USENIX Security 22)*.

Hussain, S. U.; Javaheripi, M.; Samragh, M.; and Koushanfar, F. 2021. COINN: Crypto/ML Codesign for Oblivious Inference via Neural Networks. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 3266–3281.

Jia, Y.; Shelhamer, E.; Donahue, J.; Karayev, S.; Long, J.; Girshick, R.; Guadarrama, S.; and Darrell, T. 2014. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM International Conference on Multimedia*, 675–678.

Juvekar, C.; Vaikuntanathan, V.; and Chandrakasan, A. 2018. GAZELLE: A low latency framework for secure neural network inference. In *Proceedings of the 27th USENIX Security Symposium (USENIX Security 18)*, 1651–1669.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. *Proceedings of the Advances in Neural Information Processing Systems*, 25: 1097–1105.

Lindell, Y. 2016. How To Simulate It - A Tutorial on the Simulation Proof Technique. *Cryptology ePrint Archive, Report 2016/046*.

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

Mishra, P.; Lehmkuhl, R.; Srinivasan, A.; Zheng, W.; and Popa, R. A. 2020. DELPHI: A cryptographic inference service for neural networks. In *Proceedings of the 29th USENIX Security Symposium (USENIX Security 20)*, 2505–2522.

Mohassel, P.; and Rindal, P. 2018. ABY^3: A mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 35–52.

Mohassel, P.; and Zhang, Y. 2017. SecureML: A system for scalable privacy-preserving machine learning. In *Proceedings of the 2017 IEEE Symposium on Security and Privacy (SP)*, 19–38. IEEE.

Najafabadi, M. M.; Villanustre, F.; Khoshgoftaar, T. M.; Seliya, N.; Wald, R.; and Muharemagic, E. 2015. Deep learning applications and challenges in big data analytics. *Journal of Big Data*, 2(1): 1–21.

Ng, L. K. L.; Chow, S. S. M.; Woo, A. P. Y.; Wong, D. P. H.; and Zhao, Y. 2021. Goten: GPU-Outsourcing Trusted Execution of Neural Network Training. In *Proceedings of the AAAI Conference on Artificial Intelligence 2021*.

Patra, A.; Schneider, T.; Suresh, A.; and Yalame, H. 2021. ABY2.0: Improved mixed-protocol secure two-party computation. In *Proceedings of the 30th USENIX Security Symposium (USENIX Security 21)*.

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

Riazi, M. S.; Samragh, M.; Chen, H.; Laine, K.; Lauter, K.; and Koushanfar, F. 2019. XONN: Xnor-based oblivious deep neural network inference. In *Proceedings of the 28th USENIX Security Symposium (USENIX Security 19)*, 1501–1518.

Riazi, M. S.; Weinert, C.; Tkachenko, O.; Songhori, E. M.; Schneider, T.; and Koushanfar, F. 2018. Chameleon: A hybrid secure computation framework for machine learning applications. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, 707–721.

Rouhani, B. D.; Riazi, M. S.; and Koushanfar, F. 2018. Deepsecure: Scalable provably-secure deep learning. In *Proceedings of the 55th Annual Design Automation Conference*, 1–6.

Simonyan, K.; and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Tan, S.; Knott, B.; Tian, Y.; and Wu, D. J. 2021. CRYPTGPU: Fast Privacy-Preserving Machine Learning on the GPU. *arXiv preprint arXiv:2104.10949*.

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

Zhang, Q.; Xiang, T.; Xin, C.; Chen, B.; and Wu, H. 2022. Joint Linear and Nonlinear Computation across Functions for Efficient Privacy-Preserving Neural Network Inference. *arXiv preprint arXiv:2209.01637*.

Zhang, Q.; Xin, C.; and Wu, H. 2021. GALA: Greedy ComputAtion for Linear Algebra in Privacy-Preserved Neural Networks. In *Proceedings of the 2021 ISOC Network and Distributed System Security Symposium*.