CrossNet: A Low-Latency MLaaS Framework for Privacy-Preserving Neural Network Inference on Resource-Limited Devices

Yu Lin, Tianling Zhang, *Student Member, IEEE* Yunlong Mao, *Member, IEEE* and Sheng Zhong, *Fellow, IEEE*

***Abstract*—With the development of cryptographic tools such as Fully Homomorphic Encryption (FHE) and secure Multiparty Computation (MPC), privacy-preserving Machine Learning as a Service (MLaaS) has gained attractiveness for its security when it comes to utilizing cross-domain data. However, cryptographic tools are characterized by huge overhead, which results in the MLaaS quality being unbearably degraded, especially for latency-sensitive MLaaS applications. In this paper, we focus on the problem of low-latency inference associated with MLaaS and propose CrossNet, a Privacy-preserving Neural Network Inference (PPNI) framework based on FHE, for applications with limited client-side computational and communication resources. CrossNet performs model transformations on neural networks so that they can be evaluated in an FHE-friendly manner. Model transformation introduces limited interactions between client and server, thus restricting inference latency. In addition, CrossNet includes a series of layer constructions where elaborate encoding forms and computational orders are designed to further reduce the overhead of transformed layers. CrossNet outperforms the existing FHE-based frameworks by 4x efficiency and reduces nearly 30% inference latency on ResNet-50 in a resource-limited setting.**

***Index Terms*—secure inference, privacy preservation, machine learning, homomorphic encryption, neural network**

I. Introduction

Machine Learning as a Service (MLaaS) has gained immense popularity with intelligent service providers because of its adaptability, inference accuracy, and general superiority to other traditional analytic methods. However, privacy concerns are often raised when MLaaS is used for privacy-aware applications. For example, hospitals and financial institutions are vulnerable to data breach risks when using third-party MLaaS. Fortunately, privacy-preserving MLaaS has been made possible due to the development of Fully Homomorphic Encryption (FHE) [1], [2] and secure Multiparty Computation (MPC) [3], [4]. Among all privacy-preserving MLaaS applications, Privacy-preserving Neural Network Inference (PPNI) [5] is one of the most attractive topics because of its wide range of application scenarios. When a model server wants to provide inference service to clients, PPNI should keep the confidentiality of both its model and client-side private data. To meet the requirements of PPNI, the model server can utilize FHE and/or MPC to evaluate inference results without disclosing any private information. For example, a client can encrypt its data using FHE and upload only the ciphertext, allowing the server to evaluate an encrypted result securely.

There are two prevalent forms of PPNI frameworks, namely *interactive* and *non-interactive* PPNI. The interactive PPNI has MPC as its primary tool and can be further divided into MPC-based and MPC-FHE-based types. In MPC-based PPNI, the client and the server interactively evaluate all computational tasks, including the computation of linear layers and non-linear functions. Unfortunately, due to the overhead associated with MPC, MPC-based PPNI frameworks [4], [6]–[9] produce large computational overhead during the inference process. Recent studies [10], [11] attempt to leverage the benefits of FHE and propose MPC-FHE-based PPNI frameworks, introducing the transformation between MPC secret-sharing and FHE ciphertexts so that the server can evaluate linear layers locally. However, the client is still involved in lots of computational tasks brought by MPC, impeding the application of MPC-FHE-based PPNI for resource-limited clients. Moreover, recent works [12], [13] show that MPC-FHE-based frameworks are vulnerable to model reconstruction attacks performed by malicious clients.

In non-interactive PPNI, the server evaluates the entire model, taking FHE ciphertexts as input, while the client only needs to perform data encryption and decryption. Non-interactive PPNI is more suitable for real-world applications for two reasons: It requires fewer client-side resources and offloads most of the inference computation to the server. On the other hand, under the security guarantees of FHE, non-interactive PPNI is more secure since no intermediate results transmission between the server and the client is needed. However, due to the huge overhead of FHE operations, recent studies [14]–[16] show that non-interactive PPNI suffers from more significant inference latency when compared with interactive PPNI. Moreover, due to the limitation of the multiplicative depth in FHE, non-interactive PPNI has to evaluate a heavy FHE operation called bootstrapping when implementing large-scale neural networks, leading to an extremely high cost for a single inference request [16].

Motivated by the abovementioned latency issue, we propose CrossNet in this paper, an FHE-based PPNI framework that reduces the inference latency by adequately balancing the workload for resource-limited clients. CrossNet absorbs the advantages of interactive and non-interactive PPNI frameworks to improve the inference process while guaranteeing the privacy of both the client and the server. Briefly, CrossNet introduces

Yu Lin, Tianling Zhang, Yunlong Mao, and Sheng Zhong are with the State Key Laboratory for Novel Software Technology, Nanjing University, China, 210023. Part of this work was done by Yu Lin when he was a graduate student at Nanjing University. Now he is with ByteDance Ltd..

Yunlong Mao (maoyl@nju.edu.cn) and Sheng Zhong (zhongsheng@nju.edu.cn) are co-corresponding authors.

a few interactive rounds between the server and the client. As a result, the client will be involved in a lightweight inference preparation process, while the server will finish most of the computational tasks. Our insight is that the interactive preparation process will significantly reduce the computational overhead of FHE-based inference. Therefore, CrossNet transforms a neural network model into its interactive form where the client is required to perform only a small amount of FHE operations when invoking the interactive process. To further trade off the client-side overhead and inference latency, we propose an algorithm to efficiently search for a near-optimal strategy for model transformation. In addition, we integrate a series of homomorphic layers into CrossNet to improve layer-wise efficiency. Compared with existing homomorphic layers proposed in previous studies, our designs further reduce the number of FHE operations by adjusting operational orders according to layer parameters. Note that although these homomorphic layers are designed to suit the application scenario of CrossNet, they can also be independently applied to other FHE-based or MPC-FHE-based frameworks.

Remark. Just like the non-interactive PPNI, CrossNet adopts FHE as the primary cryptographic tool, but it introduces interactions between the client and the server. For fairness, we compare CrossNet with both interactive and non-interactive PPNI under mobile application settings. The experimental results show that, compared with the state-of-the-art (SOTA) non-interactive framework, which takes over 600s for a single inference on a 4-layer neural network, CrossNet reaches 94× improvement by reducing the inference latency to 6.7s. Besides, CrossNet improves up to nearly 4× efficiency on a 10-layer network compared with Cheetah, the SOTA MPC-FHE-based framework. The insensitivity to client-side resources also grants CrossNet the advantage of being more stable in different scenarios. In general, this work makes the following contributions:

- With the objective of improving the quality of PPNI service for scenarios with limited client-side resources, we propose CrossNet to achieve a better balance between inference efficiency and client-side overhead. We design a critical component called *switch layer* to invoke an interactive process so that FHE operations can be evaluated more efficiently among all network layers. An efficient algorithm called *NetSearch* is proposed to fit any network into CrossNet by properly placing a given number of switching layers.
- We propose a series of homomorphic layers to further improve the layer-wise efficiency in CrossNet. Compared with existing homomorphic layers, our constructions of homomorphic layers, e.g. fully connected and convolutional layers, involve fewer FHE operations by handling FHE operations according to layer parameters.
- We provide security analysis, ensuring that the system remains secure against the activities of malicious clients. This is also proof that CrossNet guarantees the security of private data and that the entire model is safe.

II. Related Work

MPC and FHE are the two most widely adopted cryptographic tools in PPNI. Generally, interactive PPNI can further be divided into MPC-based and MPC-FHE-based types, while non-interactive PPNI is only based on FHE. Recent works improve PPNI latency by designing elaborate computational paradigms based on FHE and MPC. We present a summary of the representative frameworks in Table I, in which the column "optimization" refers to the specific problem or module that the framework is attempting to optimize.

MPC-based. Yao's garbled circuits (GC) [3] and secret sharing are the two basic MPC techniques to realize PPNI. DeepSecure [6] employs GC-optimized methods for common arithmetic computations along with low-overhead pre-processing techniques, to safeguard the data privacy from a semi-honest adversary without sacrificing the model accuracy. ABY [4] and ABY2.0 [17] are general mixed-protocol MPC frameworks that enhance computational sub-routines through the conversion and incorporation of arithmetic sharing, boolean sharing, and GC. This allows participants to select the suitable sharing method for varying computation processes, thereby enhancing computational efficiency. ABY2.0 builds upon the foundation of ABY and introduces new sharing approaches. It includes a pre-processing phase to reduce computational and communication overhead in the online process.

Some frameworks adopt a 3-party scenario to improve efficiency. SecureNN [8] and ABY^3 [7] leverage Beaver's triples [37] to accelerate matrix multiplication and reduce the communication overhead by modifying the procedure in which participants collaborate to compute with the secret sharing components. In CrypTFlow [9], a complete system is proposed to convert TensorFlow inference codes into 2-party or 3-party forms, with the assistance of a designed compiler and MPC protocols. Some work [20], [21] attempt to accelerate inference by leveraging GPU for MPC operations and significantly reduce the computing time of MPC protocols. Recently, with the widespread usage of ChatGPT, secure inference on large transformer models is becoming attractive. CipherGPT [22] and PUMA [23] are proposed to construct a secure inference framework on transformer models. They build efficient modules such as GeLU, embedding, layer normalization, and self-attention based on MPC, which are essential components in GPT-like models. MPC is a computationally efficient technique for PPNI, but it still cannot be directly adopted into resource-limited scenarios since MPC requires lots of interactive operations, leading to significant bandwidth costs and client-side overhead.

Apart from the overhead issue, some previous works have focused on achieving a stronger security guarantee in PPNI. BLAZE [18] employs an input-independent pre-processing phase and 3-party online protocols. The framework ensures fairness, which implies that the adversary obtains the output if and only if honest parties do. Furthermore, [19] proposes SWIFT, a 3 or 4-party protocol that provides robustness (guaranteed output delivery, GOD).

MPC-FHE-based. This type leverages both MPC and FHE during its inference process. GAZELLE [24] and GALA [25]

TABLE I: Summary of representative PPNI frameworks. Security: ○- semi-honest, ◐- malicious(abort), ◕- malicious(fairness), ●- malicious(robustness, GOD). Protocols: HE - Homomorphic Encryption, GC - Garbled Circuit, OT - Oblivious Transfer, SS - Secret Sharing

Base	Framework	N-parties	Security	interaction	protocols	optimization
MPC	DeepSecure [6]	2	○	Yes	GC	pre-processing
	ABY2.0 [17]	2	○	Yes	GC+OT+SS	pre-processing + primitives + communication
	SecureNN [8]	3	◐	Yes	SS	primitives + communication
	ABY^3 [7]	3	◐	Yes	GC+SS	primitives + communication
	CrypTFlow [9]	2/3	◐	Yes	SS	primitives(DReLU)
	BLAZE [18]	3	◕	Yes	SS	pre-processing + primitives + security
	SWIFT [19]	3/4	●	Yes	SS	pre-processing + primitives + security
	CRYPTGPU [20]	3	○	Yes	SS	hardware acceleration
	Piranha [21]	-	-	Yes	SS	hardware acceleration
	CipherGPT [22]	2	○	Yes	OT+SS	pre-processing + primitives
	PUMA [23]	3	○	Yes	SS	pre-processing + primitives
MPC-FHE	GAZELLE [24]	2	○	Yes	HE+GC+SS	primitives + homomorphic encryption
	GALA [25]	2	○	Yes	HE+SS	primitives + homomorphic encryption
	CryptFlow2 [26]	2	○	Yes	HE+OT+SS	primitives + communication
	Cheetah [10]	2	○	Yes	HE+OT+SS	primitives + communication
	MUSE [12]	2	◐(clients only)	Yes	HE+GC+SS	primitives + security
	SIMC [27]	2	◐(clients only)	Yes	HE+GC+OT+SS	pre-processing + primitives + security
	FANNG-MPC [28]	2	◐	Yes	HE+GC+SS	pre-processing + protocols + engineering
FHE	CryptoNets [14]	2	○	No	HE	-
	LoLa [29]	2	○	No	HE	message representations
	EVA [30]	2	○	No	HE	homomorphic encryption(circuits)
	CHET [31]	2	○	No	HE	homomorphic encryption(circuits)
	Falcon [15]	2	○	No	HE	primitives + homomorphic encryption
	[32]	2	○	No	HE	homomorphic convolution layers
	[33]	2	○	No	HE	homomorphic encryption(bootstrapping)
	[16]	2	○	No	HE	homomorphic encryption(bootstrapping), function approximation
	BTS [34]	2	○	No	HE	homomorphic encryption(hardware acceleration)
	[35]	2	○	No	HE	homomorphic encryption(hardware acceleration)
	PP-Stream [36]	-	○	Yes	HE	primitives + distributed system

introduce elaborate data packing methods and optimized HE-operations to evaluate linear layers. Compared with the previous work [9], CrypTFlow2 [26] has been extended to enable 2-party DNN inference, which contains new protocols to speed up the evaluation of non-linear functions, resulting in enhanced performance. Cheetah [10] simultaneously designs efficient FHE-based protocols for linear layers and primitives for non-linear functions to improve inference efficiency. However, recent studies show that MPC-FHE-based PPNI is vulnerable to the model reconstruction attack performed by a malicious client [12], [13]. MUSE [12], and SIMC [27] are proposed to solve the security risks under a malicious-client setting. Additional authentication protocols have been incorporated into the frameworks, which enable the server to promptly identify the malicious behavior of the client. Nevertheless, they are only capable of tolerating a malicious client, rather than any party of two. FANNG-MPC [28] is a 2-party framework exclusively supporting PPNI in a dishonest majority setting with active security. Based on the SCALE-MAMBA [38], FANNG-MPC further extends the functionality by separating pre-processing from the online phase and introducing a dealer model, which has been demonstrated to significantly enhance the efficiency of the online inference. A further distinguishing feature of FANNG-MPC is that it delves more deeply into system-level details and demonstrates remarkable efficiency in a practical engineering context.

FHE-based. CryptoNets [14] is the first HE-based framework designed for high-throughput privacy-preserved inference on the neural network. Later LoLa [29] improves inference latency by using multiple message representations and optimizing FHE-based constructions of network layers. EVA [30], and CHET [31] optimize the circuits and orders of FHE operations, which significantly reduce the computational overhead of the server. In Falcon [15], efficient homomorphic Discrete Fourier Transform (DFT) and inverse DFT algorithms are designed to speed up dot product operations. Nevertheless, these frameworks still require over 100s to evaluate a single inference on a neural network with 2 convolutional layers. To reduce the computational overhead, Ehud Aharoni et al. propose an efficient encoding method of FHE ciphertexts for dot product and matrix multiplication [39]. And [32] generalizes the homomorphic convolution operation to pack as many convolution outputs as possible into each output ciphertext. Some studies [16], [33] apply FHE-based PPNI to neural networks with arbitrary depth by leveraging bootstrapping operations of FHE but also introduce additional overhead. To reduce the computational overhead of FHE-based PPNI, some work [34], [35] propose hardware accelerated methods for FHE operations. Recently, PP-Stream [36] maps privacy-preserving neural network inference into a distributed stream processing system, whose load-balanced resource allocation reduces the inference latency.

III. Background

A. RNS-CKKS

RNS-CKKS (CKKS for short) [40] is a leveled FHE supporting users to perform fixed-point arithmetic operations

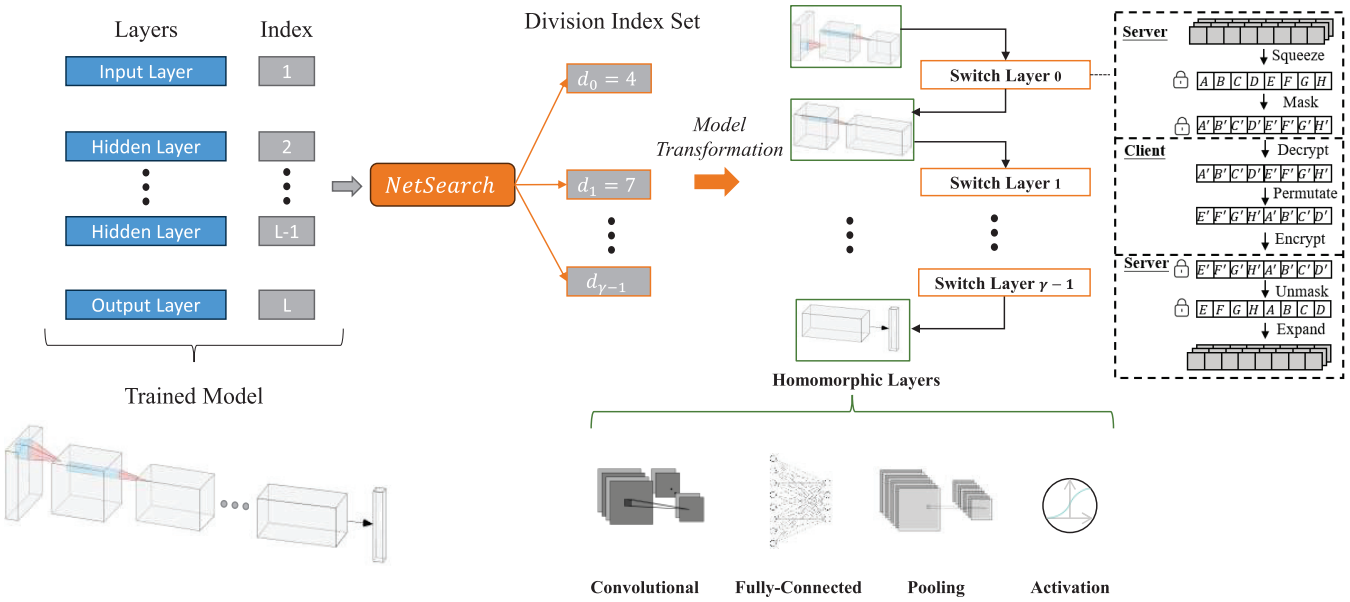


Fig. 1: Overview of CrossNet

directly on the ciphertext. Users can encode $M = \frac{N}{2}$ fixed-point numbers into one N-th cyclotomic polynomial, which can be further encrypted to a ciphertext supporting single instruction multiple data (SIMD) operations. Each fixed-point number is represented as a slot of the ciphertext, and CKKS supports the following slot-wise operations: 1) *AddPT*: add a ciphertext with a plain polynomial; 2) *AddCT*: add two ciphertexts; 3) *MulPT*: multiply a ciphertext with a plain polynomial; 4) *MulCT*: multiply two ciphertexts; 5) *Rot*: rotate slots in ciphertext with given steps.

The polynomial coefficients of CKKS are represented modulo $Q = \prod_{j=1}^{R} Q_j$ with a preset homomorphic level R. Fixed-point numbers are encoded with a given scaling factor Δ and can be encrypted into an r-level ($r \leq R$) ciphertext with modulo $\prod_{j=1}^{r} Q_j$. A low-level ciphertext is more efficient for computation and communication, and an r-level ($r > 1$) ciphertext can be rescaled to $r-1$ level (with a reducing scale $\Delta' = \frac{\Delta}{Q_r}$ if necessary). Since the largest scale of an r-level ciphertext is $\prod_{j=1}^{r} Q_j$, and both *MulPT* and *MulCT* increase the scales of output ciphertexts, only a limited number of multiplication operations are supported for CKKS.

B. Encoding Representation

When using CKKS to evaluate neural network layers, it is necessary to encode input data properly since CKKS supports SIMD operations. Specifically, as 3-dimension data is characterized by channel, height, and width in convolutional neural networks (CNN), there exist the following typical representations for data packing:

1) *Stacked*: The input data is flattened into a one-dimension vector, and several copies of the vector are packed into one ciphertext.
2) *Dense-SIMD*: This representation packs multiple channels into one ciphertext, and each channel is flattened into one dimension. A fixed number of zeros will be filled between every two channels.
3) *Conv-SIMD*: This representation is suitable for the evaluation of convolutional layers. It packs data values corresponding to kernel weights at the same positions into a ciphertext with a fixed number of padded zeros between every two channels.

To evaluate network layers with input data structured in any of these representations, layer weights should also be packed to plain polynomials of CKKS consistently so that arithmetical operations will be performed on the corresponding data and weights.

IV. CrossNet

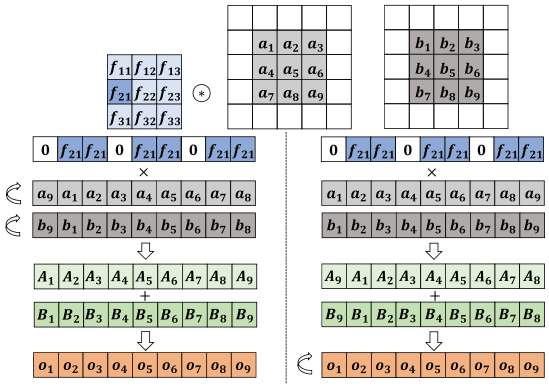
A. Overview

By analyzing existing FHE-based frameworks, we focus on two critical parts to improve the inference process. On the one side, we try to reduce the number of homomorphic operations involved in the inference process by elaborately designing a series of homomorphic layers. With these homomorphic layers, layer evaluation can be applied to ciphertexts efficiently. On the other side, we consider promoting the speed of each single homomorphic operation. Noting that the overhead of homomorphic operations is related to the multiplicative depth during the inference process, our insight is to transform a neural network model into some small segments so that each segment can be evaluated under a smaller multiplicative depth.

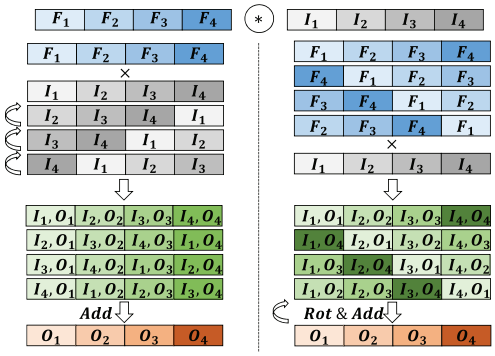
The architecture of CrossNet is shown in Figure 1. Given a trained model, CrossNet first transforms it into a divided form with several segments. We propose an efficient algorithm called *NetSearch* to find a proper transformed structure. After determining the transformed model, the server can evaluate the inference process in ciphertext by replacing each normal layer with its corresponding homomorphic layer. Whenever

completing the evaluation of a model segment, the server, and the client will interactively evaluate a switch layer so that homomorphic levels of ciphertexts can be refreshed. In this section, we first introduce our designs of homomorphic layers, such as convolutional and fully-connected layers. We also explain the structures of switch layers and their evaluating procedure. In the next section, we will introduce the *NetSearch* algorithm and show how to properly transform a neural network into its CrossNet form.

Adaptive homomorphic layers. In this section, we propose a concept named adaptive homomorphic layers for FHE-based interactive PPNI. In adaptive homomorphic layers, each type of layer can be computed in both interactive and non-interactive forms. We design non-interactive (called *basic layers*) and interactive (called *switch layers*) forms for convolutional (Conv), fully connected (FC), and pooling layers. In basic layers, we still let the server evaluate layers locally. Our designs of basic layers fully utilize the SIMD feature of CKKS to reduce the number of FHE operations. In switch layers, the server and the client invoke an interactive process to refresh homomorphic levels while forwarding the computational process. The proposed switch layers guarantee both client-side and server-side privacy. Additionally, by considering the relationship of layer parameters, we optimize data encoding forms and computational routines so that the client will not be involved in too many computational tasks.



(a) Kernel input (left) and output (right) rotations



(b) Channel input (left) and output (right) rotations

Fig. 2: Kernel and channel rotation modes. Every single value of the kernel and input data are noted as f, a, b. Each kernel matrix, input matrix, and output matrix are marked by F, I, and O.

B. Convolutional Layer

In convolutional layers, we use $\mathcal{K}$ to mark the kernel weight/height. These input and output channel numbers are noted by $\mathcal{C}_I$ and $\mathcal{C}_O$. Each ciphertext contains the data of $\mathcal{C}$ channels.

1) Basic-Conv: We propose an optimal-group encoding (OGE) method for Basic-Conv to reduce the number of FHE operations. Briefly, the convolutional process can be accomplished by shifting the weight kernels and computing dot products. To evaluate convolution on ciphertexts, we are able to utilize homomorphic rotation, addition, and multiplication to substitute for the shifting and dot product operations. OGE leverages the observation that the total number of involved homomorphic operations is related to the operation order. For example, if we aggregate some partial results before performing homomorphic rotations, there will be fewer ciphertexts to be rotated, leading to less overhead of rotation. Therefore, considering the relationship between channel numbers and kernel sizes, we propose that OGE automatically select the optimal encoding form of layer weights and evaluate the order of homomorphic operations.

OGE first divides a kernel of size $\mathcal{K} \times \mathcal{K}$ to multiple 1×1 kernels to perform unit convolution. Then it automatically determines the best order of homomorphic multiplications and rotations to minimize the rotation number. Specifically, as shown in Figure 2, each Dense-SIMD ciphertext can be rotated according to $\mathcal{K}$, $\mathcal{C}_I$ and $\mathcal{C}_O$ when computing the dot products of convolution. For the kernel input rotation, ciphertexts are first rotated along kernel weights before being multiplied with unit kernel weights. Therefore, the results of multiplications can be added to obtain the results of each unit convolution. On the contrary, ciphertexts are directly forwarded to multiplication in the feature output rotation, then rotations will be performed after homomorphic additions. Similarly, considering the order of channel-wise multiplications and rotations, we can also construct the channel input and output rotations. Note that although we explain the orders of kernel-wise and channel-wise rotations separately, they should be taken into account simultaneously when performing convolutions. Therefore, there exist four types of rotation modes: Kernel Input Channel Input (KICI), Kernel Input Channel Output (KICO), Kernel Output Channel Input (KOCI), and Kernel Output Channel Output (KOCO). OGE automatically chooses the best mode to reduce the overhead. To the best of our knowledge, GALA [25] has proposed the most efficient Conv. Thus, we compare OGE with it in Table II.

TABLE II: Comparison of Conv on *Rot* counts under $\mathcal{C} = 4, \mathcal{K} = 3$ and various $\mathcal{C}_I, \mathcal{C}_O$.

$\mathcal{C}_I$@$\mathcal{C}_O$	GALA [25]	OGE
8@128	112	70
8@64	64	64
64@8	134	64
128@8	262	70

2) Switch-Conv: To reduce the client-side overhead, we have designed Switch-Conv with a similar structure as SqueezeNet [41]. Switch-Conv contains a squeeze module and an expansion module. CrossNet replaces the original convolutional layer with its corresponding Switch-Conv in the training process. Generally, we consider that a convolutional layer is characterized by a kernel size $\mathcal{K}$, an input channel number $\mathcal{C}_I$ and an output channel number $\mathcal{C}_O$. It can be replaced by a switch layer consisting of the following squeeze module and expansion module. The squeeze module uses kernels of size $\mathcal{K}^{[s]} = 1$ to generate $\mathcal{C}_O^{[s]} = \frac{\mathcal{C}_O}{\sigma}$ channels with a preset squeeze rate σ. The expansion module contains two types of kernel sizes $\mathcal{K}^{[e0]} = 1$ and $\mathcal{K}^{[e1]} = \mathcal{K}$ to output $\mathcal{C}_O$ channels.

Let $\mathscr{F}_{OGE}(pk, \phi, \cdot)$ represent the functionality of evaluating ciphertext for a convolutional layer *phi* using OGE with the key *pk* of CKKS. In Figure 3, we present the formal description of the Switch-Conv protocol $\pi_{switch-conv}$. Briefly, the server will initially forward the data to the squeeze module to reduce the data size. Subsequently, it obfuscates the ciphertexts with random masks and transmits them to the client. The client will decrypt and reorder the data in plaintext. Then, the client encrypts the data with Conv-SIMD representations before sending it back to the server. Finally, the server is capable of unmasking the received ciphertext and forwarding it to the expansion module.

Input The public input is the public key *pk* of CKKS. The server input encrypted intermediate data $[\![I]\!]$, the squeeze module ϕ^s and the expansion module ϕ^e of a switch layer. The client input a secret key *sk*.
Output The server output the encrypted inference result $[\![O]\!]$ of the switch layer.

Setup The server and the client agree on the OGE mode of ϕ^e.
Evaluation The server and the client interactively run the following steps:

1) The server evaluates the squeeze module as $[\![S]\!] = \mathscr{F}_{OGE}(pk, \phi^s, [\![I]\!])$.
2) The server generates a set of random data R and homomorphically add them to the ciphertexts as $[\![U]\!] = AddPT(pk, [\![S]\!], R)$. Then the server sends $[\![U]\!]$ to the client.
3) The client decrypts $U = Dec(sk, [\![U]\!])$. It reorders V according to the OGE mode of ϕ^e to get V.
4) The client encrypts $[\![V]\!] = Enc(pk, V)$ and sends $[\![V]\!]$ to the server.
5) The server reorders R according to the OGE mode of ϕ^e to get R'. It computes $[\![E]\!] = AddPT(pk, [\![V]\!], R')$.
6) Finally, the server forward $[\![E]\!]$ to the expansion module to get $[\![O]\!] = \mathscr{F}_{OGE}(pk, \phi^e, [\![E]\!])$.

Fig. 3: Switch-Conv Protocol $\pi_{switch-conv}$

When interactively evaluating Switch-Conv, an appropriate mode of OGE will selected according to the network bandwidth and the computational power of the client. It seems that the KICI mode is better for expansion modules since the client can reorder data efficiently in plaintext, thus allowing the server to offload more rotation tasks to the client. However, more input rotations also increase the number of intermediate ciphertexts, as well as the client-side encryption and communication overhead. Therefore, KICO and KOCI are more suitable for real applications. To conclude, we present Table III showing the overhead of Basic-Conv and Switch-Conv in different OGE models. Note that since the squeeze model of Switch-Conv is computed as Basic-Conv, there is no need to record it.

TABLE III: Operation counts of Switch-Conv expansion module and Basic-Conv under various OGC modes

	Switch-Conv expansion module		Basic-Conv
	Rot	*Encrypt*	*Rot*
KICI	0	$\mathcal{K}^2\mathcal{C}\frac{\mathcal{C}_I}{\mathcal{C}}$	$\frac{\mathcal{C}_I}{\mathcal{C}} \cdot (\mathcal{C}\mathcal{K}^2 - 1)$
KICO	$\frac{\mathcal{C}_O}{\mathcal{C}} \cdot (\mathcal{C} - 1)$	$\mathcal{K}^2\frac{\mathcal{C}_I}{\mathcal{C}}$	$\frac{\mathcal{C}_I}{\mathcal{C}} \cdot (\mathcal{K}^2 - 1) + \frac{\mathcal{C}_O}{\mathcal{C}} \cdot (\mathcal{C} - 1)$
KOCI	$\frac{\mathcal{C}_O}{\mathcal{C}} \cdot (\mathcal{K}^2 - 1)$	$\mathcal{C}\frac{\mathcal{C}_I}{\mathcal{C}}$	$\frac{\mathcal{C}_I}{\mathcal{C}} \cdot (\mathcal{C} - 1) + \frac{\mathcal{C}_O}{\mathcal{C}} \cdot (\mathcal{K}^2 - 1)$
KOCO	$\mathcal{C}\frac{\mathcal{C}_O}{\mathcal{C}} \cdot (\mathcal{K}^2 - 1) + \frac{\mathcal{C}_O}{\mathcal{C}} \cdot (\mathcal{C} - 1)$	$\frac{\mathcal{C}_I}{\mathcal{C}}$	$\mathcal{C}\frac{\mathcal{C}_O}{\mathcal{C}} \cdot (\mathcal{K}^2 - 1) + \frac{\mathcal{C}_O}{\mathcal{C}} \cdot (\mathcal{C} - 1)$

3) Multi-stride Convolution: There are mainly two methods to implement multi-stride convolution based on FHE. In [16], Lee et al. propose multiplexed parallel convolution to support multiple strides. However, their method results in the gap between two valid values from one channel being increased by stride, thereby affecting the encoding representations of output ciphertexts. Therefore, it brings additional computational overhead to subsequent convolutional layers. In CrossNet, we intend to implement multi-stride convolution by inserting an additional pooling layer after the convolution layer. In practice, we found that the latency of the pooling layer is far less than that of the convolution layer($< 20\%$) and causes no significant increment of global consumption in our framework.

4) Optimization: To further exploit the client's ability to reorder data in plaintext, the server can offload pooling operations to the client if there exists a pooling layer adjacent to Switch-Conv. Note that CKKS only supports linear arithmetic operations, so neural networks are required to use average pooling instead of max-pooling. Besides, due to the exchangeability of linear arithmetic operations, any pooling layer prior to a Switch-Conv can be offloaded to the client. Formally, for any kernel weight matrix $\boldsymbol{W}$ and bias $\boldsymbol{b}$, there exist $\boldsymbol{W}'$ and $\boldsymbol{b}'$ so that the following equation holds:

$$f(\boldsymbol{W} \cdot Avg(\boldsymbol{X}) + \boldsymbol{b}) = f(Avg(\boldsymbol{W}' \cdot \boldsymbol{X} + \boldsymbol{b}')), \tag{1}$$

where $Avg(\cdot)$ is a 2-dimension average pooling function and $f(\cdot)$ is the activation function of the squeeze module. Furthermore, since the functionality of the squeeze module mainly focuses on feature compression, we can also exchange the average-pooling layer and the whole squeeze module (including its activation function), namely $Avg(f(\boldsymbol{W}' \cdot \boldsymbol{X} + \boldsymbol{b}'))$, to train a new neural network with negligible loss of accuracy.

C. Fully-connected Layer

We use $\mathcal{I}$ and $\mathcal{O}$ to note the numbers of input and output features in FC.

1) Basic-FC: We use the hybrid packing method proposed in GAZELLE [24] to pack the weight matrix for Basic-FC. Briefly, hybrid packing packs a weight matrix $\mathbf{W} = \{w_{i,j}\}_{\mathcal{O}\times\mathcal{I}}$ to $\mathbf{V} = \{\mathbf{v}_i\}_{\mathcal{O}}$, where $\mathbf{v}_i = \{w_{k\%\mathcal{O},(i+k)\%\mathcal{I}} | 0 \le k < \mathcal{I}\}$ is encoded to a CKKS polynomial. Therefore, the same slot position of each polynomial corresponds to the same output.

However, original hybrid packing is designed for MPC-FHE-based frameworks, and it cannot be directly adopted into CrossNet due to the different encoding representations. Specifically, the client in MPC-FHE-based frameworks can always reorder the data in plaintext before evaluating FC. As for FHE-based frameworks such as CrossNet, it is necessary to consider the relationship between $\mathcal{I}$, $\mathcal{O}$, and M when adapting the hybrid packing in FC. For example, we cannot rotate ciphertexts cyclically to support hybrid packing when $M = 8$ and $\mathcal{I} = 6$. Therefore, in order to adapt hybrid packing into FHE-based frameworks, we have to introduce a refined hybrid packing.

According to the input feature number $\mathcal{I}$, the refined hybrid packing can be categorized into full-input, majority-input, and minority-input types. The *full-input* means that the input feature number is the same as the slot number, enabling circular rotation of input ciphertexts. The *majority-input* of $\frac{M}{2} < \mathcal{I} < M$ can be regarded as full-input by increasing the input feature number to the same as M. For *minority-input* of $\mathcal{I} \le \frac{M}{2}$, each input ciphertext should be appended with its copy behind the last valid data.

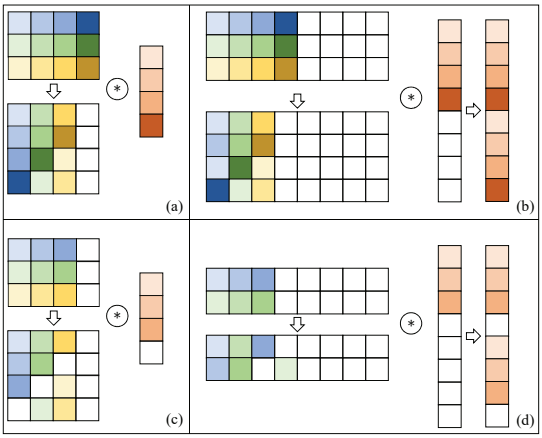


Fig. 4: Refined hybrid packing: (a) full-input and output padding, (b) minority-input and output padding, (c) majority-input and output padding, (d) minority-input and input padding.

In the case where the input feature number is not a multiple of the output feature number, we adopt different padding strategies to further reduce the rotation numbers. For the minority input, we adopt an *input padding* strategy to increase the input feature number when there exists a padded number $\mathcal{I}'$ satisfying $\mathcal{I}'\%\mathcal{O} = 0$. The role of the input padding is to pad extra columns to the weight matrix so that it introduces little additional overhead. For other situations, we use an *output padding* strategy to pad extra rows of the weight matrix so that the input feature number will not exceed the slot number. Figure 4 shows an example of hybrid packing types and padding strategies under different parameters.

2) Switch-FC: We propose weight-group packing (WGP) for Switch-FC to reduce the number of *Rot* when $\mathcal{I}$ and $\mathcal{O}$ grow larger. Firstly, input ciphertexts are forwarded to a switching process so that the client can convert them into stacked representations. To compute matrix multiplication of size $\mathcal{O}\times\mathcal{I}$, a grouping factor $\xi \in [1, \mathcal{O}]$ are picked to divide the weight matrix into ξ groups of size $\mathcal{I}\cdot\frac{\mathcal{O}}{\xi}$. The weights inside each group are packed with the hybrid packing method and the packed weights of $\frac{M}{\mathcal{I}}$ groups can be stacked into one CKKS polynomial. The server only needs to perform $\frac{\mathcal{O}}{\xi} - 1$ rotations on the stacked ciphertexts and transfer ξ ciphertexts to the client. At last, the remained addition and reorder operations can be offloaded to the client in plaintext. In Switch-FC, the server should perform $\frac{\mathcal{O}}{\xi} - 1$ *Rot* and $\frac{\mathcal{O}}{\xi}$ *MulPT*, while $\frac{\xi\mathcal{I}}{M}$ ciphertexts should be transmitted to the client for decryption. Therefore, we can trade off the client-side and the server-side overhead by adjusting ξ.

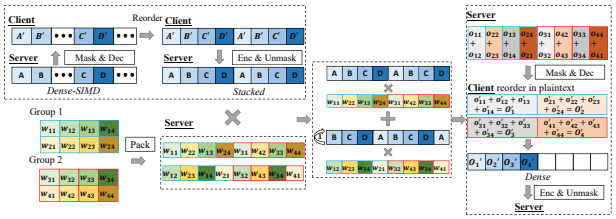


Fig. 5: Matrix multiplication with weight-group packing under $M = 8, \mathcal{I} = 4, \mathcal{O} = 4, \xi = 2$

We have also compared Switch-FC with GALA. The operation counts of Switch-FC and GALA are shown in Table IV, where we set $\xi = min(\sqrt{\frac{M\mathcal{O}}{\mathcal{I}}}, \mathcal{O})$ to balance the numbers of *Rot* and *Dec* in Switch-FC. Switch-FC increases the number of *Dec* on the client side, but the reduction of server-side overhead will significantly improve efficiency when $\mathcal{I}$ and $\mathcal{O}$ grow large. For example, 4095 *Rot* should be performed in GALA under $\mathcal{O} = \mathcal{I} = 4096$, while Switch-FC only costs 64 *Rot* and 64 *Dec*. Note that *Dec* introduces less overhead than *Rot* [40], Switch-FC is more efficient than GALA even though *Dec* should be performed on the client side.

TABLE IV: Comparison of Switch-FC and GALA

	GALA [25]	Switch-FC
MulPT	$\frac{\mathcal{O}\mathcal{I}}{M}$	$\frac{\mathcal{O}}{\xi}$
Rot	$\frac{\mathcal{O}\mathcal{I}}{M} - 1$	$\frac{\mathcal{O}}{\xi} - 1$
Dec (Client)	1	$\frac{\xi\mathcal{I}}{M}$

D. Pooling Layer

1) Basic-Pooling: Basic-Pooling leverages homomorphic rotations to achieve average pooling with a preset kernel size $\mathcal{K}$. Due to the high overhead of rotation, we make use of the advantage of Dense-SIMD representation to reorder multiple channels of data each time. Specifically, we first rotate the input ciphertext and add the rotated results to obtain each pooled column value. Then we add and mask the pooled column values

to separate different pooled rows. Finally, each pooled row is rotated to compose the final pooled ciphertext. An example of average pooling is shown in Figure 6. Note that this type of pooling layer will not affect $\mathcal{C}$. Since $\mathcal{C}$ is essential to the overhead of the subsequent layers, we can also reorder the pooled ciphertext again to change $\mathcal{C}$ if necessary.

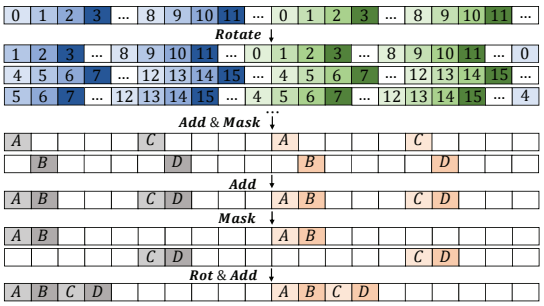


Fig. 6: Average pooling under $\mathcal{C}=2, \mathcal{F}=4, \mathcal{K}=2$.

2) Switch-Pooling: Switch-Pooling can be performed by trivially offloading pooling operations to the client in masked plaintext. Therefore, the server will only need to add and subtract ciphertexts with random masks and send them to the client.

We conclude the operation counts of Basic-Pooling and Switch-Pooling in Table V. Note that in Switch-Pooling, the client will pool and reorder data. Therefore, the channel number in each ciphertext $\mathcal{C}$ will be changed to $\mathcal{C}'$. Intuitively, it holds that $\mathcal{C}' = \mathcal{K}^2\mathcal{C}$.

TABLE V: Operation counts of pooling layers

	Basic-Pooling	Switch-Pooling
MulPT	$2\lceil \frac{\mathcal{C}_I}{\mathcal{C}} \rceil \frac{\mathcal{F}}{\mathcal{K}}$	0
Rot	$\lceil \frac{\mathcal{C}_I}{\mathcal{C}} \rceil \frac{\mathcal{F}}{\mathcal{K}} (2^{\mathcal{K}}+1)$	0
Enc	0	$\lceil \frac{\mathcal{C}_I}{\mathcal{C}'} \rceil$
Dec	0	$\lceil \frac{\mathcal{C}_I}{\mathcal{C}} \rceil$

E. Activation Function

Recent studies [42], [43] propose some ways to evaluate non-linear functions with CKKS, but they suffer from huge overhead. For example, Pegasus [42] spends over 1s to compute the rectified linear unit (ReLU) activation function for one slot. Previous FHE-based PPNI directly uses square activation function [14], [15], [29]. However, because of the loss of accuracy associated with this method, it is only suitable for simple neural networks [44]. Instead of approximating activation functions with polynomials [45], [46], we adopt 2-degree learnable polynomials [47] as activation functions combined with batch normalization to improve inference accuracy. To reduce the overhead and level consumption, we merge the parameters of batch normalization and learnable polynomials:

$$f(x) = \frac{a}{\sqrt{v_2+\varepsilon}}x^2 + \frac{b}{\sqrt{v_1+\varepsilon}}x + c - \frac{a\overline{x^2}}{\sqrt{v_2+\varepsilon}} - \frac{b\overline{x}}{\sqrt{v_1+\varepsilon}}, \quad (2)$$

where a,b,c are trained learnable parameters. v_1, v_2 are trained variances of x, x^2. $\overline{x}, \overline{x^2}$ are trained means of x, x^2. ε is a small value to avoid division zero.

V. Model Transformation

After replacing γ original layers with their corresponding switch layers, a neural network will be divided into $\gamma+1$ segments so that inference results can be computed by iteratively evaluating these segments. To minimize the inference latency, we further focus on the optimal placement of the switch layers for any neural network. The placement of γ switch layers can be modeled by a division index set $\mathcal{D}_\gamma$, which contains the indexes of switch layers. Basically, there are two strategies (*overheard measure* and *transformed model simulation*) to determine $\mathcal{D}_\gamma$ for a given γ and a model $\mathcal{M}$.

A. Overhead Measure

Notations. We use $\mathcal{V}$ to mark the set of FHE operations. For each FHE operation $v \in \mathcal{V}$, we use $n_v[k]$ to note the number of operation v involved in k-th layer. The homomorphic level at k-th layer is noted by r_k. For r_k-th homomorphic level, we note the runtime cost of v as $h_v[r_k]$.

The objective of using the overhead measure strategy is to count the computational and communication overhead based on the statistics of FHE operations and theoretical computation. We can iterate over all combinations of division indexes to search for the best $\mathcal{D}_\gamma$. In each iteration, the number of FHE operations and the total size of transferred ciphertexts are counted according to our designs of homomorphic layers. Therefore, the computational cost can be formalized by:

$$T_{\mathcal{D}_\gamma} = \sum_{k \notin \mathcal{D}_\gamma}^{L} \sum_{v \in \mathcal{V}} n_v[k] \cdot h_v[r_k] + \sum_{k \in \mathcal{D}_\gamma} \sum_{v \in \mathcal{V}} n'_v[k] \cdot h_v[1]. \quad (3)$$

In CrossNet, computational tasks take the majority of the total runtime cost due to the large overhead of FHE operations. Therefore, the best $\mathcal{D}_\gamma$ can be obtained by estimating the overhead under all combinations of switch layers to find one with minimal computational cost.

B. Transformed Model Simulation

However, it is not efficient when the number of convolutional layers and γ grows large. For example, we should compile and test $\binom{32}{6} = 906192$ transformed models for ResNet-34 under $\gamma = 6$. Therefore, we propose an algorithm to efficiently output a transformed model that is as close to the globally optimized one as possible. We define the following division gain of a division index $d \in \mathcal{D}_\gamma$:

$$G_d = \frac{T_{\mathcal{M}[l_1:l_2]} - T_{\mathcal{M}[l_1:d]} - T_{\mathcal{M}[d:l_2]}}{T_{\mathcal{M}[1:L]}}, \quad (4)$$

where $\mathcal{M}[1:L]$ is the original neural network and $\mathcal{M}[l_1:l_2]$ is the segment containing d-th layer, namely $l_1 < d < l_2$. $T_{\mathcal{M}[l_1:l_2]}$, representing the runtime cost of $\mathcal{M}[l_1:l_2]$. It is straightforward that a larger division gain indicates that more runtime cost is being saved. Since the division gain is distributed from 0

Algorithm 1: *NetSearch* Algorithm

Input : The number of switch layers γ, a model $\mathcal{M} = \{\phi_i\}_{1 \le i \le L}$, a candidate number t, a gain threshold α
Output : A division index set $\mathcal{D}_\gamma$.

Set $\mathcal{D}_\gamma = \{\}$
$T_{\mathcal{M}[1:L]} \leftarrow$ Compile and test $\mathcal{M}[1:L]$
for $i = 1$ **to** γ **do**
  Set $List \leftarrow Divide(\mathcal{M}, \mathcal{D}_\gamma)$, $\mathcal{D}' = \{\}, \mathcal{G} = \{\}$
  for $\mathcal{M}[l_1 : l_2] \in List$ **do** `// random strategy`
    $T_{\mathcal{M}[l_1:l_2]} \leftarrow$ Compile and test $\mathcal{M}[l_1 : l_2]$
    for $k = l_1 + 1$ **to** $l_2 - 1$ **do**
      if $\rho(\phi_k) ==$ "*Conv*" **then**
        $T_{\mathcal{M}[l_1:k]}, T_{\mathcal{M}[k:l_2]} \leftarrow$ Compile and test $\mathcal{M}[l_1 : k], \mathcal{M}[k : l_2]$
        Evaluate G_k by Eq. 4
        $\mathcal{D}' = \mathcal{D}' \bigcup \{d'\}$, $\mathcal{G} = \mathcal{G} \bigcup \{G_k\}$
  $d_i \leftarrow PickIndex(\mathcal{D}', \mathcal{G}, t, \alpha)$, $\mathcal{D}_\gamma \leftarrow \mathcal{D}_\gamma \bigcup \{d_i\}$
  for $d \in BackOrder(\mathcal{D}_\gamma, d_i)$ **do** `// backward strategy`
    $\mathcal{D}_\gamma = \mathcal{D}_\gamma / d$, $l_1 \leftarrow \beta_-(d, \mathcal{D}_\gamma)$, $l_2 \leftarrow \beta_+(d, \mathcal{D}_\gamma)$
    Set $\mathcal{D}' = \{\}, \mathcal{G} = \{\}$
    for $k = l_1 + 1$ **to** $l_2 - 1$ **do**
      if $\rho(\phi_k) ==$ "*Conv*" **then**
        $T_{\mathcal{M}[l_1:k]}, T_{\mathcal{M}[k:l_2]} \leftarrow$ Compile and test $\mathcal{M}[l_1 : k], \mathcal{M}[k : l_2]$
        Evaluate G_k by Eq. 4
        $\mathcal{D}' = \mathcal{D}' \bigcup \{d'\}$, $\mathcal{G} = \mathcal{G} \bigcup \{G_k\}$
    $d' \leftarrow MaxIndex(\mathcal{D}', \mathcal{G})$, $\mathcal{D}_\gamma = \mathcal{D}_\gamma \bigcup d'$
return $\mathcal{D}_\gamma$

to 1, we can use an appropriate threshold α to filter division indexes.

To avoid deterministic local optimal results, we propose Algorithm 1 with *random strategy* and *backward strategy* to search division indexes. The random strategy uses the function $PickIndex(\mathcal{D}', \mathcal{G}, t, \alpha)$ to randomly pick a division index $d \in \mathcal{D}'$ so that G_d is in the top t of $\mathcal{G}$ and $max(\{\mathcal{G}\}) - G_d \le \alpha$. The backward strategy means that all the selected indexes should be adjusted after picking a new index. Specifically, we use a function $BackOrder(\mathcal{D}_\gamma, d_i)$ to compose a set with increasing order of the distance between d_i and $d \in \mathcal{D}_\gamma / d_i$. For example, the function outputs $\{8,3,2,11\}$ when $\mathcal{D}_\gamma = \{2,3,6,8,11\}$ and $d_i = 6$. Therefore, we can replace each previous index d with a new max-gain index d' using the function *MaxIndex*. We omit the construction of these auxiliary functions since they are not hard to design. The time complexity of Algorithm 1 is $O(\gamma L)$, which is efficient for dividing real-world network models. Although the algorithm is expected to output a nearly optimal division set, we can still execute it multiple times to obtain a globally optimal set due to the randomness of the algorithm.

VI. Analysis

A. Security Analysis

Notations. We use X and $\mathcal{M}$ to represent the client's private data and the server's trained model. We use *sk* and *pk* to denote the secret and public keys generated by the client. We mark an encrypted data with $[\![\cdot]\!]$. $\mathcal{D}_\gamma$ is the division index set for γ switch layers. For each $d_i \in \mathcal{D}_\gamma$, ϕ_{d_i} is the i-th switch layer. We use $[\![U_{d_i}]\!]$ and $[\![V_{d_i}]\!]$ to represent the encrypted outputs of the squeeze module and the encrypted inputs of the expansion module at the i-th switch layer.

Taking the client's private data X and the server's trained model $\mathcal{M}$ as inputs, we model the following ideal PPNI functionality $\mathscr{F}_{PPNI}$.

Definition 1. *Functionality $\mathscr{F}_{PPNI}$.*

- *Input: The server inputs a trained model $\mathcal{M}$ and the client inputs private data X to the $\mathscr{F}_{PPNI}$.*
- *Computation: Upon receiving $\mathcal{M}$ and X, the $\mathscr{F}_{PPNI}$ evaluates the model inference procedure to obtain $Y = \mathcal{M}(X)$.*
- *Output: The $\mathscr{F}_{PPNI}$ outputs Y to the client and nothing to the server.*

1) Semi-honest Security: We take into consideration the security against the following potentially corrupt parties:

- A **corrupted model server** $\mathcal{A}_1$. This server should not obtain any private information regarding the client's private data. Hence, there should exist a Probabilistic Polynomial-time (PPT) simulator, ensuring that $\mathcal{A}_1$ is unable to distinguish whether the incoming messages in its view are generated by the simulator or from the actual PPNI protocol.
- A **corrupted client** $\mathcal{A}_2$. This client should not gain any private information about the server's private model. Thus, there should be a PPT simulator such that $\mathcal{A}_2$ cannot distinguish whether the incoming messages in its view are generated by a PPT simulator or from the real PPNI protocol.

Formally, we define the semi-honest security of the PPNI protocol π. Let $View_c^\pi(X, \mathcal{M})$ and $View_s^\pi(X, \mathcal{M})$ represent the client-side and server-side views respectively after the execution of the CrossNet protocol π using the private input data X and the trained model $\mathcal{M}$. Let $\mathscr{F}_{PPNI}(X, \mathcal{M})$ denote the ideal output of the client in the ideal functionality of PPNI. The CrossNet protocol π is considered semi-honest if there exist PPT simulators $\mathcal{S}_1$ and $\mathcal{S}_2$ that satisfy the following equations for all X and $\mathcal{M}$:

$$View_s^\pi(X, \mathcal{M}) \stackrel{c}{\equiv} \mathcal{S}_1(\mathcal{M}, \mathcal{D}_\gamma),$$
$$(View_c^\pi(X, \mathcal{M}), Out_c^\pi(X, \mathcal{M})) \stackrel{c}{\equiv} \Big(\mathcal{S}_2(X, \mathcal{D}_\gamma, 1^\lambda), \mathscr{F}_{PPNI}(X, \mathcal{M})\Big).$$

Theorem 1. *The CrossNet protocol π securely realizes $\mathscr{F}_{PPNI}$ in the presence of one semi-honest adversaries $\mathcal{A}_1$ or $\mathcal{A}_2$.*

Proof. We prove the security by defining simulators for both the corrupted server and client so that the joint distribution of the simulators' outputs and the functionality $\mathscr{F}_{PPNI}$ outputs are distinguishable from the views of the corrupted parties.

Security against the corrupted server $\mathcal{A}_1$. We construct the PPT simulator $\mathcal{S}_1$ for the corrupted server to work as follows:

1) $\mathcal{S}_1$ randomly generate a set of numbers R and encrypt them as $[\![R]\!] = Enc(pk, R)$. The R has the same size as the user data to simulate the encrypted data sent to the server.
2) At each switch layer ϕ_{d_i} ($d_i \in \mathcal{D}_\gamma$), $\mathcal{S}_1$ generates another set of numbers R_i and encrypt as $[\![R_i]\!] = Enc(pk, R_i)$. Then it outputs $[\![R_i]\!]$ to the server as the simulates of the expansion-module inputs $[\![V_{d_i}]\!]$.
3) Finally, $\mathcal{S}_1$ outputs nothing.

Due to the indistinguishability under the chosen-plaintext attack (IND-CPA) of CKKS, the distributions of randomly generated $[\![R]\!]$ and $[\![R_i]\!]$ are statistically close to the distribution of

the real inputs $[\![X]\!]$ and $[\![V_{d_i}]\!]$. Therefore, we conclude that $View_s^{\pi}(X,\mathcal{M}) \stackrel{c}{\equiv} \mathcal{S}_1(\mathcal{M},\mathcal{D}_\gamma)$.

Security against the corrupted client $\mathcal{A}_2$. We prove that CrossNet is secure against a corrupted client by constructing the following simulator $\mathcal{S}_2$. $\mathcal{S}_2$ extracts the client's private data X and submit it to the ideal functionality $\mathscr{F}_{PPNI}$. During the process of CrossNet protocol π, it generates a set of random numbers R_i' with the same size as U_{d_i} at the switch layer $\phi_{d_i}, d_i \in \mathcal{D}_\gamma$. $\mathcal{S}_2$ outputs $[\![R_i']\!] = Enc(pk, R_i')$ to simulate the intermediate results $[\![U_{d_i}]\!]$ received by the client. Since $[\![U_{d_i}]\!]$ is randomized with one-time padding, the distributions of the decrypted U_{d_i} and R_i' are indistinguishable. Finally, $\mathcal{S}_2$ will output the inference result to the client as in the ideal functionality. □

Note that compared with MPC-based or MPC-FHE-based frameworks, CrossNet is more secure in protecting the security of model structures since the server is not required to interact with the client at every layer. However, the client still needs to know the structures of switch layers. We consider that they are acceptable since most private model structures are adapted from public neural networks, such as ResNet [48] and LLaMA [49].

2) Security against the malicious client: We will explore the security against the malicious client from two aspects. On the one hand, we discuss the information leakage caught by the inference results. Remark that in any PPNI protocol, clients are always able to obtain the inference results of the model. Consequently, it is reasonable for us to assume that the server can prevent the client from inferring model information based on the inference results through engineering restrictions (such as frequency access control, etc.). On the other hand, we will prove that the additional intermediate data exposed to the client in CrossNet does not increase the client's advantage in reconstructing the server's model.

Information leakage via inference results. Due to the fact that the security of the model is closely related to the numbers of parameters, layers and activation functions of the model, we represent the model complexity as $Cx(\mathcal{M})$ for model $\mathcal{M}$. We present the following complex model assumption (CMA) to describe a model with sufficient complexity that could hardly be reconstructed in real-world applications.

Assumption 1. *(**Complex Model Assumption**) When the model complexity satisfies $Cx(\mathcal{M}) \geq \alpha$, the information leakage via inference results is negligible against a malicious client $\mathcal{A}_c$ in a PPNI protocol Π_{PPNI} if $\mathcal{A}_c$ can only observe the final inference result $Y = \mathcal{M}(X)$ for any input data X.*

CMA is necessary for any PPNI protocol that desires to guarantee the model security against malicious clients. We do not further discuss the lower bound of α because it is related to both the model structure and application scenarios.

Information leakage via intermediate data. Without considering the information leakage brought by inference results, we define the following indistinguishability game between a malicious client $\mathcal{A}_c$ and a challenger $\mathcal{C}$ to capture the information leakage via intermediate data during a PPNI protocol Π_{PPNI}.

- **Setup** $\mathcal{C}$ prepares a trained model $\mathcal{M}$. $\mathcal{C}$ and $\mathcal{A}$ agrees on the necessary public parameters and keys.
- **Phase 1** $\mathcal{A}$ is allowed to send inference requests to $\mathcal{C}$ with any input data in polynomial times. It can receive intermediate data during the protocol execution except inference results.
- **Challenge** $\mathcal{A}$ generates two data X^0 and X^1 to request inference service. $\mathcal{C}$ selects a random bit b and takes X^b as input to invoke the PPNI protocol Π_{PPNI} except that $\mathcal{C}$ will not return the inference result to $\mathcal{A}$.
- **Phase 2** Phase 1 is repeated adaptively.
- **Guess** The adversary $\mathcal{A}$ outputs its guess b'.

The adversary's advantage to win the game is defined as $Pr[b' = b] - \frac{1}{2}$.

Definition 2. *The information leakage via intermediate data in a PPNI protocol Π_{PPNI} is negligible against a malicious client if no PPT adversary $\mathcal{A}_c$ has the advantage at least*

$$Adv_{\mathcal{A}_c}^{\Pi_{PPNI}}(1^\lambda) = \left| Pr[b' = b] - \frac{1}{2} \right| > \varepsilon(\lambda).$$

Theorem 2. *The information leakage via intermediate data of the CrossNet protocol π is negligible under the IND-CPA of one time pad.*

Proof. We reduce the security of CrossNet protocol to the IND-CPA of one time pad as described in the following security game:

- **Setup** $\mathcal{C}$ initializes a trained model $\mathcal{M}$ and a division set $\mathcal{D}_\gamma$. $\mathcal{C}$ publishes $\mathcal{D}_\gamma$. $\mathcal{A}_c$ generates a pair of CKKS keys and gives the public key pk to $\mathcal{C}$.
- **Phase 1** $\mathcal{A}_c$ picks a set of input data $\{X_k\}$ to request inference requests with them. For each input data X_k, $\mathcal{A}_c$ will invoke CrossNet protocol with $\mathcal{C}$. At each switch layer ϕ_{d_i} where $d_i \in D_\gamma$, $\mathcal{A}_c$ obtains random data R_{d_i} instead of the squeeze-module output U_{d_i}.
- **Challenge** $\mathcal{A}_c$ generates two data X^0, X^1 and gives to $\mathcal{C}$. $\mathcal{C}$ randomly picks $b \stackrel{\$}{\leftarrow} \{0,1\}$ to invoke the CrossNet protocol with X^b. Instead of receiving $\{U_{d_i}^b | d_i \in \mathcal{D}_\gamma\}$, $\mathcal{A}_c$ received a set of random data $\{R_{d_i}^b | d_i \in \mathcal{D}_\gamma\}$.
- **Phase 2** Phase 1 is repeated adaptively.
- **Guess** The simulator $\mathcal{C}$ outputs the guess of $\mathcal{A}_c$.

The replacement in the above security game holds if $\mathcal{A}_c$ cannot distinguish one-time padded data from random data with non-negligible advantage. □

3) Security against the malicious server: CrossNet has the capability to ensure data security against a malicious server because in the simulated view of the server, only the CKKS ciphertexts are received. As we described in the proof of Theorem 1, the view of the server in the CrossNet protocol is exactly the same as the ideal functionality $\mathscr{F}_{PPNI}$ under the context of the IND-CPA of CKKS. Therefore, a malicious server is not able to guess the input data or obtain more information by actively tempering the intermediate results during the inference process.

B. Division Number Selection

Although adding switch layers leads to a lower computational overhead on the server side, it also increases the client-side overhead. Therefore, we introduce a quality gain to measure the improvement of each switch layer. For brevity, we use the function $\mathbb{T}_{\mathcal{M}}(\widetilde{\mathcal{D}}_\gamma)$ to represent the total computation cost for a model $\mathcal{M}$ under the index set $\widetilde{\mathcal{D}}_\gamma$. The quality gain is defined as:

$$Q_{\gamma\text{-}\gamma'} = \frac{\mathbb{T}_{\mathcal{M}}(\widetilde{\mathcal{D}}_\gamma) - \mathbb{T}_{\mathcal{M}}(\widetilde{\mathcal{D}}_{\gamma'})}{\mathbb{T}_{\mathcal{M}}(\widetilde{\mathcal{D}}_\gamma)}. \tag{5}$$

Algorithm 2: *DivNum* Algorithm

Input : A quality lower bound $\widetilde{Q}$, a network model $\mathcal{M} = \{\phi_i\}_{1 \le i \le L}$

Output : A division number γ_o and the corresponding index set $\mathcal{D}_{\gamma_o}$.

Set $\gamma_{max} \leftarrow MaxDivNum(\mathcal{M})$, $\mathcal{D}_\gamma = \{\}$
for $\gamma = 1$ *to* γ_{max} **do**
  Set $\gamma' = \gamma + 1$
  Search $\mathcal{D}_\gamma, \mathcal{D}_{\gamma'}$ by Algorithm 1
  Evaluate $Q_{\gamma\text{-}\gamma'}$ with $\mathcal{D}_\gamma, \mathcal{D}_{\gamma'}$ by Eq. (5)
  if $Q_{\gamma\text{-}\gamma'} \ge Q$ **then**
    Set $\gamma_o = \gamma, \mathcal{D}_{\gamma_o} = \mathcal{D}$
Return $\gamma_o, \mathcal{D}_{\gamma_o}$

The quality gain indicates the improvement of the inference latency by increasing the number of switch layers from γ to γ'. The optimization problem of quality gain is different from the optimization of division gain since it is simultaneously related to the division number γ and the division index set $\widetilde{\mathcal{D}}_\gamma$. A division set with optimal quality gain does not always guarantee the optimal division gain. Therefore, to search for a proper division number with a quality lower bound, we increase the division number γ until its division set of optimal division gain is smaller than the lower bound. As shown in Algorithm 2, we set $\gamma' = \gamma + 1$ to measure the adjacent quality gain brought by one additional switch layer. Note that the function *MaxDivNum* counts the number of convolution layers in $\mathcal{M}[2 : L-1]$ as the maximum division number of $\mathcal{M}$. We record the quality gains of VGG-16, ResNet-18, and ResNet-34 in Table VI when increasing γ from 1 to 6.

TABLE VI: Adjacent quality gain of VGG-16, ResNet-18 and ResNet-34

(γ, γ')	VGG-16	ResNet-18	ResNet-34
(1, 2)	0.396	0.472	0.407
(2, 3)	0.407	0.515	0.296
(3, 4)	0.323	0.202	0.285
(4, 5)	0.244	0.122	0.263
(5, 6)	0.126	0.132	0.224

VII. Experiment

A. Experiment Setting

We conduct experiments on both symmetric and asymmetric settings. In the symmetric setting, we test with a powerful client using the same machine configuration as the server. In the asymmetric setting, we limit the client-side resources by using a lower-performance device or limiting the CPU percentage. Besides, we use SEAL 3.7 [50] to implement CrossNet. All experiments use the default 128-bit security level. We use OpenMP to support multi-threading. We implement our framework with C++ and compile our codes with G++ 7.5.0.

Except where specifically stated, we configure all experiments in a WAN setting (nearly 100Mbps and 60ms delay) and compare CrossNet with FHE-based and MPC-FHE-based frameworks under 1-thread and 4-thread settings, respectively, according to previous studies.

Symmetric setting. The server and client have the same configuration. The experiments of simple networks(Network-A, B, C, D, as shown below) are tested on a machine with 2 Intel(R) Xeon(R) Silver 4110 CPU @ 2.10GHz and 32GB RAM. For the experiments of deep networks(ResNet, VGG16), they are tested on another machine with 40 Intel(R) Xeon(R) Gold 5115 CPU @ 2.40GHz and 512GB RAM for higher speed.

Asymmetric setting. Due to the compatibility of avx512 instructions, we first compare CrossNet with other PPNI frameworks by limiting the CPU usage on a machine with Intel(R) Xeon(R) Gold 5115 CPU @ 2.40GHz and 512GB RAM. We also applied an Nvidia Jetson NX module with a 6-core 1.9GHz NVIDIA Carmel Arm®v8.2 CPU and 8GB RAM and a Raspberry Pi 3B with a 4-core 1.2GHz Broadcom BCM2837 CPU and 1GB RAM as two computationally weak client devices to evaluate CrossNet in a asymmetrical client-server setting.

Remark. For simplicity, we mark fully connected, convolutional, average-pooling, and switch layers with *FC*, *Conv*, *Avg-pooling* and *SW*, respectively. We use $[s], [e0], [e1]$ to mark the parameters corresponding to the squeeze and expansion parts of the switch layer. We replace non-linear activation functions and max-pooling layers of original models with 2-degree learnable polynomial functions and average pooling layers, respectively. We set $\sigma = 8$ for all switch layers.

B. Parameter Setting and Network Structure

To trade off the inference efficiency and model accuracy, we adopt the following parameters according to the level consumption of divided model segments.

1) *Param-A* ($R \le 3$): $N = 2^{13}$, 50 bits for each level, 40 bits for data encryption, 15 bits for convolution and linear weights, 35 bits for 1st order weight of activation function and 15 bits for 2nd order weight.
2) *Param-B* ($R \le 6$): $N = 2^{14}$, 60 bits for each level, 40 bits for data encryption, 20 bits for convolution weight, 15 bits for linear weights, 40 bits for 1st order weight of activation function and 20 bits for 2nd order weight.
3) *Param-C* ($R > 6$): $N = 2^{15}$, the same scales as Param-B.

Following previous FHE-based frameworks, we focus on MNIST [51] and CIFAR-10 [52] datasets and adopt four classical models to compare with previous studies. Network-A [29] with (1-Conv, 2-FC) and Network-B [53] with (2-Conv,

2-FC) are designed for MNIST. We apply Network-A into CrossNet under $\gamma = 0$ with the following layers:

1) Conv: $\mathcal{K} = 5, \mathcal{C}_I = 1, \mathcal{C}_O = 5$, stride $(2,2)$, pad $(2,2)$, 2-degree learnable activation function.
2) Fast-FC: $\mathcal{I} = 980, \mathcal{C}_O = 100$, 2-degree learnable activation function.
3) Basic-FC: $\mathcal{I} = 100, \mathcal{C}_O = 10$.

Using our *NetSearch* algorithm, we divide Network-B under $\gamma = 1$ into the following form:

1) Conv: $\mathcal{K} = 5, \mathcal{C}_I = 1, \mathcal{C}_O = 16$, stride $(1,1)$, pad $(2,2)$, 2-degree learnable activation function.
2) Avg-pooling: $\mathcal{K} = 2$.
3) SW: Squeeze module of $\mathcal{K}^{[s]} = 1, \mathcal{C}_I^{[s]} = 16, \mathcal{C}_O^{[s]} = 2$, expansion module of $\mathcal{K}^{[e0]} = 1, \mathcal{K}^{[e1]} = 5, \mathcal{C}_I^{[e0]} = \mathcal{C}_I^{[e1]} = 2, \mathcal{C}_O^{[e0]} = \mathcal{C}_O^{[e1]} = 8$, stride $(1,1)$, and 2-degree learnable activation function for each module.
4) Avg-pooling: $\mathcal{K} = 2$.
5) Fast-FC: $\mathcal{I} = 1024, \mathcal{C}_O = 100$, 2-degree learnable activation function.
6) Basic-FC: $\mathcal{I} = 100, \mathcal{C}_O = 10$.

Network-C [29] with (3-Conv, 1-FC) and Network-D [53] with (7-Conv, 1-FC) are designed for CIFAR-10. They are divided under $\gamma = 2$ as shown in Figure 7 and 8.

1) Conv: $\mathcal{K} = 3, \mathcal{C}_I = 3, \mathcal{C}_O = 128$, stride $(1,1)$, pad $(1,1)$, 2-degree learnable activation function.
2) Avg-pooling: $\mathcal{K} = 2$.
3) SW: Squeeze module of $\mathcal{K}^{[s]} = 1, \mathcal{C}_I^{[s]} = 128, \mathcal{C}_O^{[s]} = 10$, expansion module of $\mathcal{K}^{[e0]} = 1, \mathcal{K}^{[e1]} = 3, \mathcal{C}_I^{[e0]} = \mathcal{C}_I^{[e1]} = 10, \mathcal{C}_O^{[e0]} = 32, \mathcal{C}_O^{[e1]} = 48$, stride $(1,1)$, and 2-degree learnable activation function for each module.
4) Avg-pooling: $\mathcal{K} = 2$.
5) SW: Squeeze module of $\mathcal{K}^{[s]} = 1, \mathcal{C}_I^{[s]} = 80, \mathcal{C}_O^{[s]} = 20$, expansion module of $\mathcal{K}^{[e0]} = 1, \mathcal{K}^{[e1]} = 3, \mathcal{C}_I^{[e0]} = \mathcal{C}_I^{[e1]} = 20, \mathcal{C}_O^{[e0]} = \mathcal{C}_O^{[e1]} = 80$, stride $(1,1)$, and 2-degree learnable activation function for each module.
6) Avg-pooling: $\mathcal{K} = 2$.
7) FC: $\mathcal{I} = 2560, \mathcal{C}_O = 10$.

Fig. 7: Network-C under $\gamma = 2$

1) Conv: $\mathcal{K} = 3, \mathcal{C}_I = 3, \mathcal{C}_O = 64$, stride $(1,1)$, pad $(1,1)$, 2-degree learnable activation function.
2) Conv: $\mathcal{K} = 3, \mathcal{C}_I = 64, \mathcal{C}_O = 64$, stride $(1,1)$, pad $(1,1)$, 2-degree learnable activation function.
3) Avg-pooling: $\mathcal{K} = 2$.
4) SW: Squeeze module of $\mathcal{K}^{[s]} = 1, \mathcal{C}_I^{[s]} = 64, \mathcal{C}_O^{[s]} = 8$, expansion module of $\mathcal{K}^{[e0]} = 1, \mathcal{K}^{[e1]} = 3, \mathcal{C}_I^{[e0]} = \mathcal{C}_I^{[e1]} = 8, \mathcal{C}_O^{[e0]} = \mathcal{C}_O^{[e1]} = 32$, stride $(1,1)$, and 2-degree learnable activation function for each module.
5) Conv: $\mathcal{K} = 3, \mathcal{C}_I = 64, \mathcal{C}_O = 64$, stride $(1,1)$, pad $(1,1)$, 2-degree learnable activation function.
6) Avg-pooling: $\mathcal{K} = 2$.
7) SW: Squeeze module of $\mathcal{K}^{[s]} = 1, \mathcal{C}_I^{[s]} = 64, \mathcal{C}_O^{[s]} = 8$, expansion module of $\mathcal{K}^{[e0]} = 1, \mathcal{K}^{[e1]} = 3, \mathcal{C}_I^{[e0]} = \mathcal{C}_I^{[e1]} = 8, \mathcal{C}_O^{[e0]} = \mathcal{C}_O^{[e1]} = 32$, stride $(1,1)$, and 2-degree learnable activation function for each module.
8) Conv: $\mathcal{K} = 1, \mathcal{C}_I = 64, \mathcal{C}_O = 64$, stride $(1,1)$, pad $(1,1)$, 2-degree learnable activation function.
9) Conv: $\mathcal{K} = 1, \mathcal{C}_I = 64, \mathcal{C}_O = 16$, stride $(1,1)$, pad $(1,1)$, 2-degree learnable activation function.
10) FC: $\mathcal{I} = 1024, \mathcal{C}_O = 10$.

Fig. 8: Network-D under $\gamma = 2$

C. Implementation

We provide an efficient framework to transform any trained neural network, e.g. ResNet, into its CrossNet form and implement CrossNet protocol. The framework contains two parts: a model transformation program and a model evaluation program for secure inference. The model transformation program takes a trained model, a division number, and parameters of CKKS as inputs to find the best-divided model based on the NetSearch algorithm. The transformed model will be serialized to a specific format (e.g., JSON) so that the model evaluation program can load the model into its CrossNet form to provide inference service. We implement the model transformation program with Python and the model evaluation program with C++. In the model transformation program, we pretested the overhead of homomorphic operations under different parameters to estimate the overhead of each transformed model.

D. Evaluation

1) Model Transformation: To find the relation between the overhead and the divided model, we apply the NetSearch algorithm on different network structures and division numbers γ, and the results are presented in Table VII. We run the algorithm 20 times for every condition and record the averaged number of the underlying iteration and the time cost. As the results show, increasing the depth of the network would increase the size of the search space and lead to more time consumption, while the total consumption is almost in proportion to the division number. However, the cost is still acceptable even with a large model like ResNet-50.

TABLE VII: Overhead of NetSearch algorithm

Network	layers	γ	Iteration number	Time cost(s)
VGG16	16	2	22694	0.32
ResNet-18	18	2	29996	0.42
ResNet-20	20	2	27119	0.40
ResNet-50	50	2	204299	2.79
	50	3	326167	4.54
	50	5	531514	7.77
	50	8	771413	11.88

2) Symmetric experiment: **Comparison with FHE-based Frameworks.** As shown in Table VIII and IX, we compare CrossNet with FHE-based frameworks, e.g., CryptoNets [14], LoLa [29] and Falcon [15], on Network-A and Network-C. Since Network-A only contains 3 layers, we directly implement it in CrossNet without model transformation. Although we do not perform any model transformation on Network-A, CrossNet improves both communication and computational overhead since we propose more efficient homomorphic layers as shown in Section IV-B and IV-C. On the one side, these layers involve less homomorphic operations to greatly improve the inference efficiency. On the other side, these layers cost less multiplication depth, so we can select more efficient homomorphic parameters than other FHE-based frameworks.

We transform Network-C under $\gamma = 2$ and bind an activation function to the second Conv for accuracy improvement when implementing it with CrossNet. Compared with other FHE-based frameworks, CrossNet achieves higher accuracy since we use the learnable polynomial function to replace the square

TABLE VIII: Overhead and accuracy on Network-A

Framework	Comm. (MB)	Latency (s)	Accuracy
CryptoNets	368	184.9	98.95%
LoLa	51	9.4	98.95%
Falcon	51	8.0	98.95%
CrossNet	8.4	1.4	98.70%

function as the activation function. In terms of inference latency, CrossNet accelerates the inference process of Network-C by replacing the last two Conv with their corresponding switch layers so that Network-C is transformed into three segments. Since the evaluation of each segment involves less multiplicative depth, we can set a smaller polynomial degree of CKKS so that all homomorphic operations will be evaluated more efficiently within each segment. Therefore, CrossNet reduces nearly 99% latency compared with Falcon on Network-C.

TABLE IX: Overhead and accuracy on Network-C

Framework	Comm. (MB)	Latency (s)	Accuracy
LoLa	210	3379.6	76.5%
Falcon	128.5	634.2	76.5%
CrossNet	11.8	6.7	81.8%

Comparison with MPC-FHE-based Frameworks. As shown in Table X and XI, we also compare CrossNet with MPC-FHE-based frameworks, e.g., CrypTFlow2 (FHE-based linear layers) [26], Cheetah [10] and Delphi [11]. The Network-B and Network-D are transformed under $\gamma = 1$ and $\gamma = 2$, respectively. Besides, in Network-B, we collapse the second average pooling layer and its adjacent fully connected layer since they only contain linear arithmetic operations.

TABLE X: Overhead and accuracy on Network-B

Framework	Comm. (MB)	Latency (s)	Accuracy
CrypTFlow2(FHE)	18.9	9.7	99.28%
Cheetah	2.5	5.2	99.28%
CrossNet	17.4	2.2	99.31%

Compared with FHE-based frameworks, MPC-FHE-based frameworks are more efficient since they do not need to use large FHE parameters to support more multiplication depths. However, CrossNet is still competitive since we can increase the number of switch layers to improve the efficiency of homomorphic operations. For example, CrossNet achieves the lowest latency on Network-B. Although the large composition of the homomorphic level increases the size of the ciphertext, resulting in CrossNet costing more bandwidth than Cheetah, the 2.5× latency improvement is still attractive for real-world applications.

TABLE XI: Overhead and accuracy on Network-D

Framework	Comm. (MB)	Latency (s)	Accuracy
Delphi	148.1	54.4	85.14%
CrypTFlow2(FHE)	328.0	45.4	85.14%
Cheetah	35.8	11.2	85.14%
CrossNet	29.9	10.6	83.7%

The results of Table XI show that Cheetah achieves similar performance to CrossNet. Note that MPC-FHE-based frameworks only consume one multiplication depth on linear layers irrespective of the model being evaluated, making them actually more suitable for large-scale models. However, since MPC-FHE-based frameworks introduce more interactive rounds and client-side overheads, CrossNet gets more practical when the client-side communication and computational resources get worse, as we will discuss in the asymmetric experiments.

Evaluation on large models. We further consider applying CrossNet on some classical neural networks to evaluate its potential for real-world applications. Firstly, we simulate the inference latencies of AlexNet, VGG-16, ResNet-18, and ResNet-34. As shown in Figure 9, although it still costs hundreds of seconds to process these large models, we can discover the potential to improve the inference latency by increasing the division number γ. For example, we can reduce nearly half of the latency by increasing γ from 2 to 4 on AlexNet.

We compare CrossNet with a FHE-based framework proposed by Lee et al. [16] on ResNet-20 under (1-thread, $\gamma = 2$) and with Cheetah on ResNet-50 under (4-thread, $\gamma = 10$) as shown in TableXII and XIII, where the communication overhead of [16] is omitted since it is non-interactive. The results show that CrossNet improves the inference efficiency over 4× compared with [16]. Because of the multiplicative depth limitation of CKKS, we have to choose large homomorphic parameters, leading to longer latency than Cheetah. However, we will show that in the weak-client setting, CrossNet outperforms Cheetah due to less client-side overhead.

TABLE XII: Overhead on ResNet-20

Framework	Server send (MB)	Client send (MB)	Latency (s)
[16]	-	-	4737.2
CrossNet	19.1	204.3	992.6

TABLE XIII: Overhead on ResNet-50

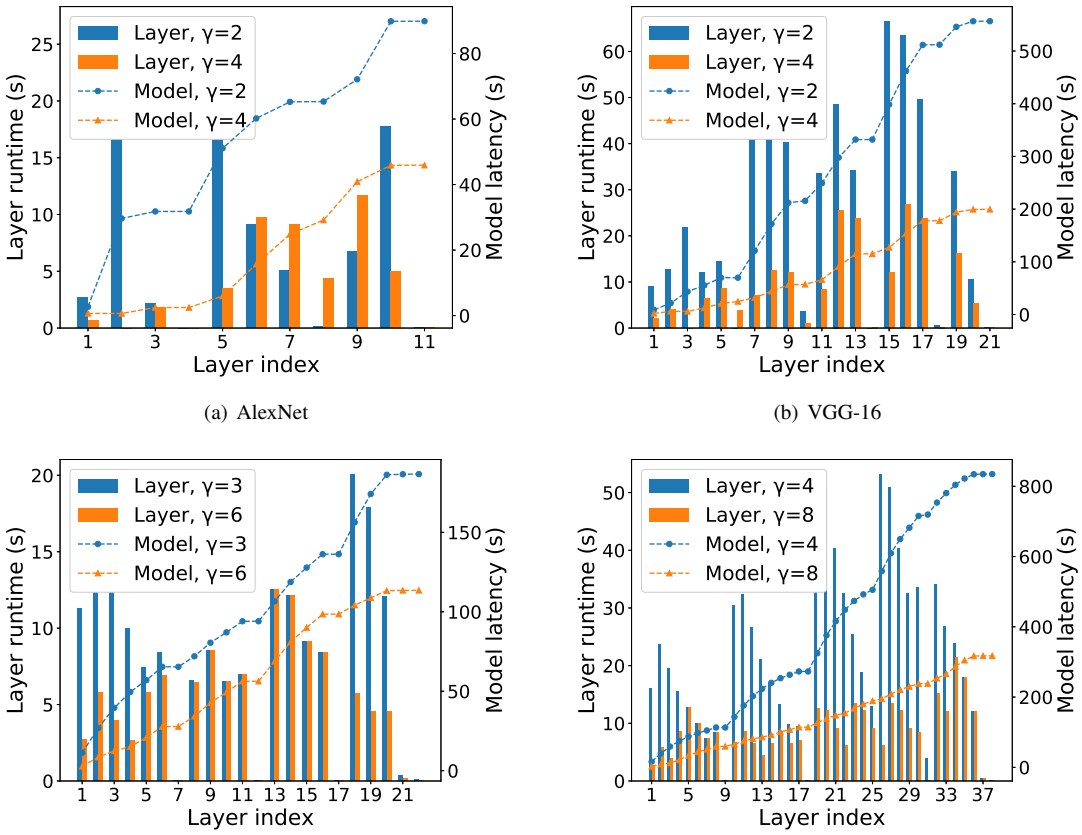
Framework	Server send (MB)	Client send (MB)	Latency (s)
Cheetah	1966.8	835.4	462.2
CrossNet	447.3	1043.6	2056.7

3) Asymmetric experiment: We conduct the following three sets of asymmetric experiments to discover the potential of CrossNet for resource-limited clients.

TABLE XIV: ResNet-50 latency vs. server-side thread number

Thread number	Latency (s) Cheetah	Latency (s) CrossNet
1	1349	6600
2	1101	3620
4	1041	2037
16	-	831
32	-	727

Comparison under various server's settings. Fixing client-side thread number to 1, we compare CrossNet with Cheetah under various server-side thread numbers as shown in Table XIV. We configure the server-side thread number from 1



(a) AlexNet (b) VGG-16

(c) ResNet-18 (d) ResNet-34

Fig. 9: Runtime costs on AlexNet, VGG-16, ResNet-18 and ResNet-34.

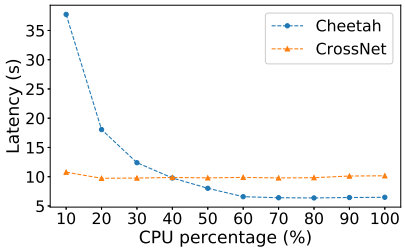
to 32, but the implementation of Cheetah only supports at most 4 threads. The results show that compared with Cheetah, CrossNet can reduce latency by improving server-side resources even if the client has weak computational power. For example, the inference latency of Cheetah is only reduced from 1101s to 1041s by increasing the server's thread number from 2 to 4. But in CrossNet, we can reduce over 40% latency on the same configuration. Furthermore, when setting the server's thread number to 32, CrossNet achieves 727s latency. Therefore, when providing inference service to resource-limited clients, we can still expect to reduce latency by using a more powerful server in CrossNet.

TABLE XV: ResNet-50 latency vs. client-side CPU percentage

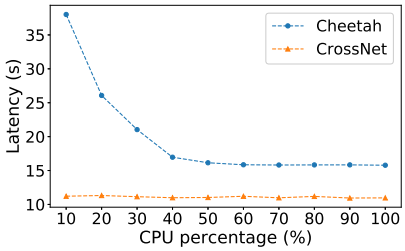
CPU percentage	Latency (s) Cheetah	CrossNet
100%	1041	2037
50%	1630	2045
30%	2204	2062
25%	2580	2067
20%	2954	2102

Comparison under various client's settings. We also compare with Cheetah under different client-side resources to show that CrossNet is more suitable when the client's resources get worse. As shown in Figure 10, we compare with Cheetah on Network-D under different client-side CPU percentages and communication delays. The inference latency of Network-D is nearly unchanged in CrossNet when decreasing the client's CPU percentage from 100% to 10% and increasing the communication delay from 20ms to 500ms. However, Cheetah is more sensitive to the client-side resources as the performance decreases 7x with CPU percentage from 100% to 10% under 20ms delay. We further apply CrossNet and Cheetah to ResNet-50 under the client's various CPU percentages, as shown in Table XV. When fixing the server's thread number to 4, CrossNet becomes competitive if the client's CPU percentage decreases lower than 30%. The results also show that CrossNet is less susceptible to the client's resources since the client only needs to perform encryption, decryption, and some simple reordering operations.

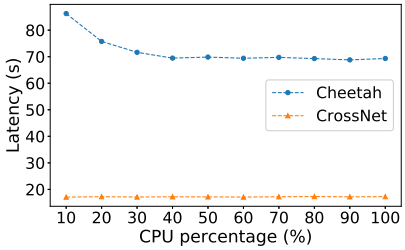
Stability of CrossNet for resource-limited clients. We also compare the client-side and server-side costs on large models to show that CrossNet is stable for various types of weak clients. We first test the time and bandwidth usages under a *powerful client* with the same resource as in the symmetric setting. Then we configure a *weak client* by replacing the client's device with an Nvidia Jetson developer kit or a Raspberry Pi board (we refer them to weak-client A and B in the table, respectively) as mentioned in Section VII-A. The results in Table XVI show that the efficiency of CrossNet is not sensitive to the client-side computational recourses since it is only required to encrypt and decrypt data. For example, the client-side time usage only



(a) 100Mbps, 20ms



(b) 100Mbps, 100ms



(c) 100Mbps, 500ms

Fig. 10: Inference latency vs. client-side CPU percentage under different communication delays.

increases by 4s on ResNet-50 when testing with the Jetson development board. Although the client's time usage increases significantly when testing on the Raspberry Pi, which is an extremely weak device compared with the server, the overall inference efficiency is still stable since the server's time usage is the bottleneck.

TABLE XVI: Client and server overhead under different client settings

		Time usage (s)			
Network	γ	Server	Powerful client	Weak-client A	Weak-client B
ResNet-18	2	409.8	1.1	2.2	37.0
VGG16	3	539.3	1.2	2.4	33.6
ResNet-50	10	1867.5	6.3	10.9	212.0

		Bandwidth usage of sending (MB)	
Network	γ	Server	Powerful/Weak client
ResNet-18	2	19.6	187.7
VGG16	3	13.1	215.3
ResNet-50	10	447.3	1043.6

VIII. Conclusion

In this paper, we introduce CrossNet, a communication-efficient interactive FHE-based framework for low-latency PPNI in resource-limited scenarios. CrossNet adopts a practical system model that considers both the hardware resources on the client side and the quality of the inference service. CrossNet improves the inference efficiency by introducing a model transformation method and well-designed homomorphic layers. The experiment results show that CrossNet is more efficient and stable for the target scenario compared with recent studies.

Acknowledgement

The authors would like to thank the editor and the anonymous reviewers for the time and effort they have kindly put into this paper. Our work has been improved by following the suggestions they have provided. This work was supported in part by the Natural Science Foundation on Frontier Leading Technology Basic Research Project of Jiangsu under Grant BK20222001, in part by the National Natural Science Foundation of China under Grants NSFC-62272222 and NSFC-62272215, and in part by Jiangsu Natural Science Foundation Excellent Youth Project under Grant BK20230080.

References

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

[2] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2017, pp. 409–437.

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

[4] D. Demmler, T. Schneider, and M. Zohner, "Aby-a framework for efficient mixed-protocol secure two-party computation." in *NDSS*, 2015.

[5] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *2017 IEEE symposium on security and privacy (SP)*. IEEE, 2017, pp. 19–38.

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

[7] 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*, 2018, pp. 35–52.

[8] S. Wagh, D. Gupta, and N. Chandran, "Securenn: 3-party secure computation for neural network training." *Proc. Priv. Enhancing Technol.*, vol. 2019, no. 3, pp. 26–49, 2019.

[9] N. Kumar, M. Rathee, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "Cryptflow: Secure tensorflow inference," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 336–353.

[10] Z. Huang, W.-j. Lu, C. Hong, and J. Ding, "Cheetah: Lean and fast secure two-party deep neural network inference," *Cryptology ePrint Archive*, 2022.

[11] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "Delphi: A cryptographic inference service for neural networks," in *29th {USENIX} Security Symposium ({USENIX} Security 20)*, 2020, pp. 2505–2522.

[12] R. Lehmkuhl, P. Mishra, A. Srinivasan, and R. A. Popa, "Muse: Secure inference resilient to malicious clients," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 2201–2218.

[13] S. Chen and J. Fan, "Seek: model extraction attack against hybrid secure inference protocols," *arXiv preprint arXiv:2209.06373*, 2022.

[14] 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 *International conference on machine learning*. PMLR, 2016, pp. 201–210.

[15] Q. Lou, W.-j. Lu, C. Hong, and L. Jiang, "Falcon: fast spectral inference on encrypted data," *Advances in Neural Information Processing Systems*, vol. 33, pp. 2364–2374, 2020.

[16] E. Lee, J.-W. Lee, J. Lee, Y.-S. Kim, Y. Kim, J.-S. No, and W. Choi, "Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions," in *International Conference on Machine Learning*. PMLR, 2022, pp. 12 403–12 422.

[17] A. Patra, T. Schneider, A. Suresh, and H. Yalame, "{ABY2. 0}: Improved {Mixed-Protocol} secure {Two-Party} computation," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 2165–2182.

[18] A. Patra and A. Suresh, "Blaze: blazing fast privacy-preserving machine learning," *arXiv preprint arXiv:2005.09042*, 2020.

[19] N. Koti, M. Pancholi, A. Patra, and A. Suresh, "{SWIFT}: Super-fast and robust {Privacy-Preserving} machine learning," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 2651–2668.

[20] S. Tan, B. Knott, Y. Tian, and D. J. Wu, "Cryptgpu: Fast privacy-preserving machine learning on the gpu," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 1021–1038.

[21] J.-L. Watson, S. Wagh, and R. A. Popa, "Piranha: A {GPU} platform for secure computation," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 827–844.

[22] X. Hou, J. Liu, J. Li, Y. Li, W. jie Lu, C. Hong, and K. Ren, "Ciphergpt: Secure two-party gpt inference," Cryptology ePrint Archive, Paper 2023/1147, 2023, https://eprint.iacr.org/2023/1147. [Online]. Available: https://eprint.iacr.org/2023/1147

[23] Y. Dong, W. jie Lu, Y. Zheng, H. Wu, D. Zhao, J. Tan, Z. Huang, C. Hong, T. Wei, and W. Chen, "Puma: Secure inference of llama-7b in five minutes," 2023.

[24] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "{GAZELLE}: A low latency framework for secure neural network inference," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018, pp. 1651–1669.

[25] Q. Zhang, C. Xin, and H. Wu, "Gala: Greedy computation for linear algebra in privacy-preserved neural networks," *arXiv preprint arXiv:2105.01827*, 2021.

[26] 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.

[27] N. Chandran, D. Gupta, S. L. B. Obbattu, and A. Shah, "{SIMC}:{ML} inference secure against malicious clients at {Semi-Honest} cost," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 1361–1378.

[28] N. Aaraj, A. Aly, T. Güneysu, C. Marcolla, J. Mono, R. Paludo, I. Santos-González, M. Scholz, E. Soria-Vazquez, V. Sucasas *et al.*, "Fanng-mpc: Framework for artificial neural networks and generic mpc," *Cryptology ePrint Archive*, 2023.

[29] A. Brutzkus, R. Gilad-Bachrach, and O. Elisha, "Low latency privacy preserving inference," in *International Conference on Machine Learning*. PMLR, 2019, pp. 812–821.

[30] R. Dathathri, B. Kostova, O. Saarikivi, W. Dai, K. Laine, and M. Musuvathi, "Eva: An encrypted vector arithmetic language and compiler for efficient homomorphic computation," in *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2020, pp. 546–561.

[31] R. Dathathri, O. Saarikivi, H. Chen, K. Laine, K. Lauter, S. Maleki, M. Musuvathi, and T. Mytkowicz, "Chet: an optimizing compiler for fully-homomorphic neural-network inferencing," in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2019, pp. 142–156.

[32] D. Kim and C. Guyot, "Optimized privacy-preserving cnn inference with fully homomorphic encryption," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 2175–2187, 2023.

[33] J.-W. Lee, H. Kang, Y. Lee, W. Choi, J. Eom, M. Deryabin, E. Lee, J. Lee, D. Yoo, Y.-S. Kim *et al.*, "Privacy-preserving machine learning with fully homomorphic encryption for deep neural network," *IEEE Access*, vol. 10, pp. 30 039–30 054, 2022.

[34] S. Kim, J. Kim, M. J. Kim, W. Jung, J. Kim, M. Rhu, and J. H. Ahn, "Bts: An accelerator for bootstrappable fully homomorphic encryption," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 711–725.

[35] W. Jung, S. Kim, J. H. Ahn, J. H. Cheon, and Y. Lee, "Over 100x faster bootstrapping in fully homomorphic encryption through memory-centric optimization with gpus," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 114–148, 2021.

[36] Q. Liu, Q. Huang, X. Chen, S. Wang, W. Wang, S. Han, and P. P. Lee, "Pp-stream: Toward high-performance privacy-preserving neural network inference via distributed stream processing," in *Proceedings of the 40th IEEE International Conference on Data Engineering (ICDE 2024)*, 2024.

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

[38] A. Aly, K. Cong, D. Cozzo, M. Keller, E. Orsini, D. Rotaru, O. Scherer, P. Scholl, N. P. Smart, T. Tanguy *et al.*, "Scale–mamba v1. 14: Documentation," *Documentation. pdf*, 2021.

[39] E. Aharoni, N. Drucker, G. Ezov, H. Shaul, and O. Soceanu, "Complex encoded tile tensors: Accelerating encrypted analytics," *IEEE Security & Privacy*, vol. 20, no. 05, pp. 35–43, 2022.

[40] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "A full rns variant of approximate homomorphic encryption," in *International Conference on Selected Areas in Cryptography*. Springer, 2018, pp. 347–368.

[41] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[42] W.-j. Lu, Z. Huang, C. Hong, Y. Ma, and H. Qu, "Pegasus: Bridging polynomial and non-polynomial evaluations in homomorphic encryption." *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 1606, 2020.

[43] J. H. Cheon, D. Kim, D. Kim, H. H. Lee, and K. Lee, "Numerical method for comparison on homomorphically encrypted numbers," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2019, pp. 415–445.

[44] R. E. Ali, J. So, and A. S. Avestimehr, "On polynomial approximations for privacy-preserving and verifiable relu networks," *arXiv preprint arXiv:2011.05530*, 2020.

[45] A. Kim, Y. Song, M. Kim, K. Lee, and J. H. Cheon, "Logistic regression model training based on the approximate homomorphic encryption," *BMC medical genomics*, vol. 11, no. 4, pp. 23–31, 2018.

[46] E. Hesamifard, H. Takabi, and M. Ghasemi, "Cryptodl: Deep neural networks over encrypted data," *arXiv preprint arXiv:1711.05189*, 2017.

[47] M. Goyal, R. Goyal, and B. Lall, "Learning activation functions: A new paradigm of understanding neural networks. arxiv 2019," *arXiv preprint arXiv:1906.09529*.

[48] 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.

[49] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[50] "Microsoft SEAL (release 3.7)," https://github.com/Microsoft/SEAL, Sep. 2021, microsoft Research, Redmond, WA.

[51] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[52] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[53] 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*, 2017, pp. 619–631.

Yu Lin was a graduate student in the Department of Computer Science at Nanjing University from 2019 to 2022. Now he is a software engineer at ByteDance. His research interests include security, privacy, and deep learning.

Tianling Zhang received the B.S. degree in 2022 and has been a graduate student since then, both in the Department of Computer Science at Nanjing University. His research interests include security, privacy, and federated learning.

Yunlong Mao received B.S. and Ph.D. degrees in computer science from Nanjing University in 2013 and 2018, respectively. He is currently an associate professor with the State Key Laboratory for Novel Software Technology at Nanjing University. His current research interests include security, privacy, machine learning, and blockchain.

Sheng Zhong received the B.S. and M.S. degrees from Nanjing University in 1996 and 1999, respectively, and the Ph.D. degree from Yale University in 2004, all in computer science. He is currently a professor with the State Key Laboratory for Novel Software Technology at Nanjing University. He is interested in security, privacy, and economic incentives.