FLEXBNN: Fast Private Binary Neural Network Inference With Flexible Bit-Width

Ye Dong, Xiaojun Chen, Xiangfu Song, and Kaiyun Li

***Abstract*— Advancements in deep learning enable neural network (NN) inference to be a service, but service providers and clients want to keep their inputs secret for privacy protection. *Private Inference* is the task of evaluating NN without leaking private inputs. Existing secure multiparty computation (MPC)-based solutions mainly focus on fixed bit-width methodology, such as 32 and 64 bits. Binary Neural Network (BNN) is efficient when evaluated in MPC and has achieved reasonable accuracy for commonly used datasets, but prior private BNN inference solutions, which focus on *Boolean Circuits*, are still costly in communication and run-time. In this paper, we introduce FLEXBNN, a fast private BNN inference framework using three-party computation (3PC) in *Arithmetic Circuits* against semi-honest adversaries with honest-majority. In FLEXBNN, we propose to employ flexible and small bit-width equipped with a seamless bit-width conversion method and design several specific optimizations towards the basic operations: i) We propose bit-width determination methods for Matrix Multiplication and Sign-based Activation function. ii) We integrate Batch Normalization and Max-Pooling into the Sign-based Activation function for better efficiency. iii) More importantly, we achieve seamless bit-width conversion within the Sign-based Activation function with no additional cost. Extensive experiments illustrate that FLEXBNN outperforms state-of-the-art solutions in communication, run-time, and scalability. On average, FLEXBNN is 11× faster than XONN (USENIX Security'19) in LAN, 46× (resp. 9.3×) faster than QUOTIENT (ACM CCS'19) in LAN (resp. WAN), 10× faster than BANNERS (ACM IH&MMSec'21) in LAN, and 1.1-2.9× (resp. 1.5-2.7×) faster than FALCON (semi-honest, PoPETs'21) in LAN (resp. WAN), and improves the respective communication by 500×, 127×, and 1.3-1.5× compared to XONN, BANNERS, and FALCON.**

***Index Terms*— Secure multiparty computation, privacy-preserving, deep learning, binary neural network.**

I. INTRODUCTION

DEEP Learning (DL) can be widely used in many applications such as medical diagnosis [1], credit risk assessment [2], and facial recognition [3]. Given the superior performance, DL-as-a-Service (DLaaS) has been a prevalent business model. In DLaaS, a service provider provides a trained neural network (NN), and a user calls the well-defined API for data analysis. However, such inference services require the client to reveal its inputs to the service providers, which causes privacy concerns. An alternative solution is to let the service provider reveal the NN to the user, this not only incurs commercial losses but also may violate laws [4], [5].

Private Inference can address this challenge. A private inference protocol ensures that the only information available for clients is the inference result, and nothing more is revealed to either party. Secure multiparty computation (MPC) techniques such as homomorphic encryption (HE) [6], [7], garbled circuits (GC) [8], and secret sharing [9] are used to design such protocols. MPC-based private inference protocols can provide high privacy protection, but the challenge is how to obtain privacy with satisfying efficiency. Note that different MPC tools offer their own characteristics and trade-offs.

In Table I, we summarize the recent private machine learning frameworks. HE-based schemes [10], [11] are communication efficient but computation expensive. GC-based works [12], [13] only require a constant round of interactions regardless of the depth of circuits but have a high communication overhead and are expensive for arithmetic operations, *i.e.*, addition and multiplication. Secret sharing-based approaches [14], [15], [16], [17], [18], [19], [20] provide efficient arithmetic operations and supports non-linear functions using much less communication, yet usually require interactions in proportion to the number of multiplication, which may incur significant run-time when evaluating high-depth circuits. Besides, some works combine two or more MPC tools [21], [22], [23], [24], [25], [26], [27], [28], [29], [30], [31], [32], [33], [34] to improve efficiency. To support efficient private inference, one should choose MPC tools and construct efficient protocols depending on specific inference tasks.

From another aspect, recent advances in machine learning provide another possibility for efficiency improvement. Many new methods aimed to maximize training and inference efficiency over the plaintext domain. Among them, the binarized technique provides an efficient solution. Binary neural network (BNN) is one of the neural networks whose parameters and activation units are restricted to take binary values (±1) [35], [36] for better efficiency, *e.g.*, computation and storage. Several methods have been proposed to improve BNN's accuracy. As shown in Table II, although BNN introduces ≈ 12% accuracy loss for AlexNet on Tiny ImageNet, which is a

Manuscript received 15 March 2022; revised 12 September 2022 and 13 January 2023; accepted 26 March 2023. Date of publication 6 April 2023; date of current version 21 April 2023. This work was supported in part by the National Key Research and Development Program of China under Grant 2020YFB1006100 and in part by the Strategic Priority Research Program of the Chinese Academy of Sciences under Grant XDC02040400. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Nele Mentens. *(Corresponding author: Xiaojun Chen.)*

Ye Dong, Xiaojun Chen, and Kaiyun Li are with the Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100045, China, and also with the School of Cyber Security, University of Chinese Academy of Sciences, Beijing 101408, China (e-mail: dongye@iie.ac.cn; chenxiaojun@iie.ac.cn; likaiyun@iie.ac.cn).

Xiangfu Song is with the School of Computing, National University of Singapore, Singapore 119077 (e-mail: songxf@comp.nus.edu.sg).

This article has supplementary downloadable material available at https://doi.org/10.1109/TIFS.2023.3265342, provided by the authors.

Digital Object Identifier 10.1109/TIFS.2023.3265342

1556-6021 © 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: NANJING UNIVERSITY OF SCIENCE AND TECHNOLOGY. Downloaded on December 27,2025 at 15:28:00 UTC from IEEE Xplore. Restrictions apply.

TABLE I

COMPARISON OF VARIOUS PRIVATE MACHINE LEARNING FRAMEWORKS. SEMI-H. IS SHORT FOR SEMI-HONEST AND MALI. DENOTES MALICIOUS. H-MAJ. INDICATES HONEST-MAJORITY AND D-MAJ. IS SHORT FOR DISHONEST-MAJORITY. ● INDICATES THE FRAMEWORK SUPPORTS A FEATURE, AND ∘ INDICATES NOT SUPPORTED FEATURE. ◐ DENOTES QUOTIENT EXPLOITS TERNARIZED NN FOR BETTER EFFICIENCY.ABORT, FAIRNESS, AND GOD DENOTE THE FRAMEWORKS ACHIEVE SECURITY WITH ABORT, FAIRNESS, AND GUARANTEED OUTPUT DELIVERY, RESPECTIVELY

	Framework	Techniques			Threat Model		Corruption		Capability		Bit-width	BNN
		SS	GC	HE	Semi-H.	Mali.	H-Maj.	D-Maj.	Inference	Training	Conversion	Optimization
2 PC	CryptoNets [10]	○	○	●	●	○	○	●	●	○	○	○
	SecureML [21]	●	●	●	●	○	○	●	●	●	○	○
	CryptoDL [11]	○	○	●	●	○	○	●	●	○	○	○
	MiniONN [22]	●	●	●	●	○	○	●	●	○	○	○
	GAZELLE [23]	●	●	●	●	○	○	●	●	○	○	○
	XONN [12]	●	●	○	●	○	○	●	●	○	○	●
	QUOTIENT [13]	●	●	○	●	○	○	●	●	●	○	◐
	DELPHI [24]	●	●	●	●	○	○	●	●	○	○	○
	CryptFlow2 [25]	●	○	●	●	○	○	●	●	○	○	○
	SIRNN [26]	●	○	○	●	○	○	●	●	○	●	○
	Cheetah [27]	●	○	●	●	○	○	●	●	○	○	○
	GForce [42]	●	○	●	●	○	○	●	●	○	○	○
3 PC	Chameleon [28]	●	●	○	●	○	●	○	●	○	○	○
	ABY^3 [29]	●	●	○	●	Abort	●	○	●	●	○	○
	SecureNN [14]	●	○	○	●	○	●	○	●	●	○	○
	BLAZE [30]	●	●	○	●	Fairness	●	○	●	●	○	○
	SWIFT [16]	●	○	○	●	GOD	●	○	●	●	○	○
	CRYPTGPU [43]	●	○	○	●	○	●	○	●	●	○	○
	BANNERS [17][2]	●	○	○	●	○	●	○	●	○	●	●
	FALCON [15]	●	○	○	●	Abort	●	○	●	●	○	○
	FLEXBNN (Ours)	●	○	○	●	○	●	○	●	○	●	●
4 PC	PrivPy [18]	●	○	○	●	○	●	○	●	●	○	○
	FLASH [19]	●	○	○	●	GOD	●	○	●	●	○	○
	Trident [31]	●	●	○	●	Abort	●	○	●	●	○	○
	Fantastic Four [20]	●	○	○	●	GOD	●	○	●	●	○	○
	Tetrad [32]	●	●	○	●	GOD	●	○	●	●	○	○
N PC	SecureQ8 [33]	●	○	●	●	Abort	●	●	●	○	○	○
	MOTION [34]	●	●	○	●	○	○	●	●	○	○	○

challenging dataset in machine learning, it still performs well with reasonable accuracy for commonly used datasets such as MNIST and CIFAR-10. Moreover, BNN is designed with a simplified model structure, thus it enjoys high computation and storage efficiency. So many BNN-based applications have been recently deployed for edge-devices setting [37], [38], [39], where practical efficiency is more preferred than high inference accuracy. Therefore, an efficient private BNN inference protocol is highly desired as many applications in the constraint-resource setting can enjoy data utility without comprising privacy.

TABLE II

TOP-1 ACCURACY OF BNN [36] AND FLOATING NN

Accuracy (%)	Datasets	BNN-Acc.	Floating-Acc
LeNet	MNIST	99.23	99.34
VGG16	CIFAR10	87.78	92.29
AlexNet	Tiny ImageNet	44.87	57.10

A. Challenges

Existing works such as XONN [12], QUOTIENT [13], and BANNERS [17] in Table I have exploited BNN to improve the efficiency of private inference, but these works are still costly in communication and run-time:

1) XONN and QUOTIENT exploit Oblivious Transfer (OT) [40], [41] and GC to achieve private BNN inference. However, due to the usage of GC and OT, they incur ≈ 357× more communication and 9× more run-time than FALCON (without BNN), one of the most efficient 3-party secret sharing-based frameworks for *fixed-point* encoded floating NN.[1] Besides, XONN and QUOTIENT incurs ≈ 500× more communication and 9.3-46× more run-time than our FLEXBNN.

2) Realizing private BNN inference using secret sharing is a promising solution, but naively combining them together gives little benefit or even incurs more costs than secret sharing-based methods: i) Naively evaluating BNN on existing secret sharing-based approaches [14], [15], [29] gains no benefit, because this naive approach still needs a uniform and large ring for the overall inference to encode floating-point inputs. ii) BANNERS leverages repeated secret sharing used in FALCON to perform BNN evaluation, following the same paradigm as XONN. Although BANNERS supports flexible bit-width for different BNN layers, it still utilizes explicit boolean to arithmetic conversion for intermediate element-wise XNOR results without further optimization. Such a drawback incurs huge overhead, and the efficiency of BANNERS is even worse than FALCON [17]. Note that FALCON is designed for complex fixed-point encoded floating NN, this result proves that simply combining repeated secret sharing with BNN cannot naturally obtain efficiency improvement. We will elaborate on more details and efficiency analysis in §III-E.

[1] Encoding floats as integers in large rings with enough fractional bits.

[2] BANNERS failed to verify the correctness of multiplication via the redundancy of replicated secret sharing. We have checked this with the authors.

Authorized licensed use limited to: NANJING UNIVERSITY OF SCIENCE AND TECHNOLOGY. Downloaded on December 27,2025 at 15:28:00 UTC from IEEE Xplore. Restrictions apply.

TABLE III

THEORETICAL ONLINE COMMUNICATION COMPLEXITY (COMM.) AND ROUNDS. COMM. IS IN BITS WHERE r IS THE LOGARITHM OF RING $\mathbb{Z}_R$. $n \times d$ AND $d \times |B|$ ARE THE SIZE OF MATRICES, $\rho = \lceil \log_2(2d+1) \rceil$, k IS THE SIZE OF VECTOR. ℓ AND m DENOTE THE INPUT AND OUTPUT BIT-WIDTH OF Activation.† INDICATES THE ADDITIONAL COST OVER ActivationAND ‡ INDICATES MatMul WITH TRUNCATION

Protocol	Paramters	FALCON		BANNERS		FLEXBNN	
		Rounds	Comm.	Rounds	Comm.	Rounds	Comm.
MatMul	$n \times d, d \times \|B\|, r$	1	$\left(\frac{4}{3}\right) n\|B\|r^{\ddagger}$	1 2	$n\|B\|r$ (BF) $2nd\|B\|\rho$ (BB)	1	$n\|B\|r$ (BF) $n\|B\|\rho$ (BB)
Activation	k, r, ℓ, m	$5 + \log_2 r$	$4kr$	$4 + \log_2 \ell$	$3k\ell + km$	$4 + \log_2 \ell$	$3k\ell + km$
BatchNorm	k, r	1	kr	$\approx 0^{\dagger}$	$\approx 0^{\dagger}$	$\approx 0^{\dagger}$	$\approx 0^{\dagger}$
MP	w, h, r	$(wh-1)(7+\log_2 r)$	$5r + wh$	$\log_2(wh)^{\dagger}$	$wh - 1^{\dagger}$	$\log_2(wh)^{\dagger}$	$wh - 1^{\dagger}$

Therefore, how to design specialized secret sharing techniques by exploiting the properties of BNN to support communication-efficient arithmetic operations, along with minimizing the bit-width and sharing conversion, is challenging and the key to achieving fast private BNN inference.

B. Our Techniques

To address the above challenge, we propose FLEXBNN, a fast secret sharing-based private BNN inference framework with flexible bit-width. FLEXBNN is designed utilizing 2-out-of-3 replicated secret sharing and secure against semi-honest adversaries in honest-majority. Though 3PC replicated secret sharing and BNN are inspired by existing works, our main technical innovations lie in designing specialized efficient protocols by exploiting the properties of BNN. Specifically, FLEXBNN is more efficient than prior private BNN inference works from the following intuitions/techniques.

First, FLEXBNN is much more communication-efficient than GC- and OT-based private BNN inference methods XONN [12] and QUOTIENT [13]. Note that secret sharing-based MPC protocols usually require much less communication but increased rounds. However, we observe that the depth of circuits in BNN is rather low since the weights and activations are in ±1. Such property makes secret sharing-based techniques be a better match [44]. Throughout this paper, we highly rely on this observation to design efficient protocol components in FLEXBNN.

Second, FLEXBNN is more efficient in communication and run-time than BANNERS [17]. Benefiting from the properties of BNN, we design FLEXBNN using flexible and smaller rings (*e.g.*, $\mathbb{Z}_{2^8}$ and $\mathbb{Z}_{2^{16}}$) for different layers in arithmetic circuits, instead of boolean circuits [17], to improve the communication while avoiding overflow. Besides, different from prior works [26], [45] which introduce explicit protocols for bit-width conversion with additional costs, we achieve seamless bit-width conversion between different BNN layers implicitly within Sign-based Activation at almost no additional cost.

Third, we design several concrete optimizations towards the basic operations in BNN, including MatMul, Sign-based Activation, Batch Normalization (BatchNorm), and Max-Pooling (MP). These optimizations can significantly improve the concrete efficiency. As FALCON is more efficient than BANNERS, we compare FLEXBNN with it in *batch inference* and *microbenchmarks* to present improvements extensively.

Table III show that our online communication rounds and complexity are more efficient (or comparable) than BANNERS and FALCON. Our main contributions are as follows:

- **Private BNN Inference Framework:** We introduce FLEXBNN, a fast framework which provides efficient private BNN inference using 3PC against semi-honest adversaries in honest-majority setting. At a high level, we propose to employ flexible and small bit-width for the secure evaluation of BNN to improve efficiency.
- **Optimized BNN Operators:** We design several specific optimizations towards MatMul, Sign-based Activation, BatchNorm and MP in BNN. Concretely, i) we propose the bit-width determination methods for the secure evaluation of MatMul and Sign-based Activation function, ii) and integrate BatchNorm and MP into Sign-based Activation function for better efficiency.
- **Seamless Bit-width Conversion:** Flexible bit-width technique enables different layers of BNN to employ small and different bit-widths. Instead of naively utilizing existing bit-width conversion solutions [17], [26], [45], we propose our specialized method via private Sign-based Activation function to support seamless bit-width conversion. Compared with the naive solutions, our method takes full advantage of the property that BNN's activation units are binarized and incurs almost no additional cost than the private Sign-based Activation function.
- **Evaluations:** We implement a prototype of FLEXBNN and perform extensive evaluations on various neural networks and datasets: i) For single inference, FLEXBNN is 11× faster than XONN in LAN, 46× (resp. 9.3×) faster than QUOTIENT in LAN (resp. WAN), 10× faster than BANNERS in LAN on average, 1.1×-2.9× (resp. 1.5×-2.7×) faster than FALCON (semi-honest) in LAN (resp. WAN), and decreases the respective communication by 500×, 127×, and 1.3-1.5× compared to XONN, BANNERS, and FALCON. ii) FLEXBNN is more scalable than FALCON for batch inference. iii) For microbenchmarks, we improve the efficiency by 1.1×-2.6× over FALCON. iv) Experiments demonstrate that FLEXBNN is computation-bound (resp. communication-bound) for large NN (*i.e.*, VGG16 and AlexNet) in LAN (resp. WAN) and is communication-bound for small NN (*i.e.*, LeNet) in both LAN and WAN.

C. Organization

We present the preliminaries in §II. Then, we give the detailed construction of FLEXBNN in §III. Next, we report the

Authorized licensed use limited to: NANJING UNIVERSITY OF SCIENCE AND TECHNOLOGY. Downloaded on December 27,2025 at 15:28:00 UTC from IEEE Xplore. Restrictions apply.

TABLE IV
NOTATION TABLE

P_i	party i in 3PC
$\mathbf{X}$	uppercase bold letter denotes matrix
$\mathbf{x}$	lowercase bold letter denotes vector
x	lowercase letter denotes scalar
$[\![x]\!]^L$	3PC Arithmetic replicated secret sharing in $\mathbb{Z}_L$
$[\![x]\!]^2$	3PC Boolean replicated secret sharing in $\mathbb{Z}_2$
$[\![x]\!]_i^L$, $[\![x]\!]_i^2$	P_i's share of $[\![x]\!]^L$, $[\![x]\!]^2$
$\mathbb{Z}_L$	discrete ring modulo L with $L = 2^\ell$
$\mathcal{F}_f$	the ideal functionality for $f(\cdot)$
$\mathsf{MSB}(x)$	most significant bit of x
$\oplus$, $\wedge$	bit-wise XOR and AND

prototype and experimental results in §IV. Finally, we discuss related works in §V and conclude this work in §VI.

II. BACKGROUND & PRELIMINARIES

In this section, we give the background and preliminaries. The main notations are illustrated in Table IV, and $[\![\cdot]\!]$ can be applied on matrix and vector element-wise.

A. Neural Network

The computational flow of NN is composed of multiple linear and non-linear layers. Each layer receives input and processes it to produce an output that serves as input to the next layer. Below we describe their functionalities.

1) FC & CONV: Fully-Connected (FC) and Convolution (CONV) layers are two main linear layers in NN:

- **FC**: A FC layer in a NN is exactly a matrix multiplication (MatMul). Given an input vector $\mathbf{x} \in \mathbb{R}^{d\times 1}$, a FC layer generates the output $\mathbf{y} \in \mathbb{R}^{n\times 1}$ as $\mathbf{y} = \mathbf{Wx} + \mathbf{b}$, where $\mathbf{W} \in \mathbb{R}^{n\times d}$ is the weight matrix and $\mathbf{b} \in \mathbb{R}^{n\times 1}$ is the bias term. More generally, NN often takes a batch of images as inputs $\mathbf{X}^{d\times|B|}$ ($|B|$ is the batchsize), thus the FC layer can be implemented using MatMul as $\mathbf{Y} = \mathbf{WX} + \mathbf{B}$.
- **CONV**: The CONV layer computes the dot product of a small weight matrix (filter) and the neighborhood of an element in the input. The process repeats by sliding each filter with stride, and the size of the filter is called filter size. Additionally, CONV can be expressed as MatMul. For a generalized exposition, please refer to [14].

Hence, FC and CONV can be implemented using MatMul.

2) Activation, Pooling, & BatchNorm: NN uses activation functions (*e.g.*, ReLU) to model non-linear relationships between the input and output. Sometimes, the pooling and batch normalization ($\mathsf{BatchNorm}$) are also applied.

- **Activation**: The activation functions are applied to the input element-wise and generate an output of the same dimension. One of the most popular activation functions is ReLU function. Given input x, $\mathsf{ReLU}(x) = \mathsf{max}(0, x)$, which outputs x if $x \geq 0$ and 0 otherwise;
- **Pooling**: A Pooling operation arranges inputs into several windows and aggregates inputs within window. Pooling usually calculates the average or maximum value for each window (Average-Pooling or Max-Pooling (MP));
- **BatchNorm**: A $\mathsf{BatchNorm}$ layer is typically applied to shift its input x to amenable ranges. During inference, the $\mathsf{BatchNorm}$ parameters γ ($\gamma > 0$) and β are fixed, $\mathsf{BatchNorm}$ normalizes x as $\gamma \cdot x + \beta$.

3) Binary Neural Network: BNN is one special type of neural network whose linear layers' parameters and activations units are restricted to binary values (*i.e.*, ±1) [12], [36]. Commonly, BNN contains the following components:

- **FC & CONV**: These layers are composed of binary parameters: Values of $\mathbf{W}$, $\mathbf{B}$, and filters are in $\{-1, +1\}$;
- **Sign**(x): $\mathsf{ReLU}(x)$ is replaced by $\mathsf{Sign}(x)$, which outputs $+1$ if $x \geq 0$, and -1 otherwise;
- **MP**: BNN only uses MP, it takes the outputs of activation functions as inputs, and generates binary outputs;
- **BatchNorm**: Since $\gamma > 0$, $\mathsf{BatchNorm}$ can be reduced to addition and computed together with $\mathsf{Sign}(\cdot)$ as $\mathsf{Sign}(x + \left(\frac{\beta}{\gamma}\right))$.

As a result, BNN drastically reduces memory size and accesses and replaces most of the arithmetic operations with binary operations, which leads to a significant improvement in efficiency, and has been used in real applications [37], [39].

B. Three-Party Secure Computation

We introduce the sharing semantics of the 3PC replicated secret sharing technique [46] and its basic primitives.

1) Sharing Semantics: A secret value $x \in \mathbb{Z}_L$ ($L = 2^\ell$) is shared by three random values $x_0, x_1, x_2 \in \mathbb{Z}_L$ with $x = x_0 + x_1 + x_2 \pmod{L}$. These shares are distributed as the pairs $\{[\![x]\!]_0^L = (x_0, x_1), [\![x]\!]_1^L = (x_1, x_2), [\![x]\!]_2^L = (x_2, x_0)\}$, where party P_i holds the $[\![x]\!]_i^L = (x_i, x_{i+1})$. Moreover, in the case of $\ell > 1$ (e.g., $\ell = 32$) which support arithmetic operations such as $+$, $-$, and $\cdot$, we refer to this sharing as *Arithmetic Sharing* using notation $[\![x]\!]^L$ for a modulus L. And in case of $\ell = 1$, $+$ and $\cdot$ are respectively replaced by bit-wise $\oplus$ and $\wedge$. We refer to this as *Boolean Sharing* with notation $[\![x]\!]^2$.

2) Addition & Multiplication: Adding shared values and multiplying shared values with public constants are locally. Multiplication of shared values requires communication.

- **Addition:** Let λ, μ, ν be public constants, and $[\![x]\!]^L$ and $[\![y]\!]^L$ be two secret-shared values. Then $[\![\lambda x + \mu y + \nu]\!]^L$ can be computed as $(\lambda x_0 + \mu y_0 + \nu, \lambda x_1 + \mu y_1, \lambda x_2 + \mu y_2)$ where P_i can compute its pair locally. When $(\lambda = 1, \mu = 1, \nu = 0)$, we get $[\![x + y]\!]^L$.
- **Multiplication:** To multiply two shared values $[\![x]\!]^L$ and $[\![y]\!]^L$ together, the parties firstly compute $z_0 = x_0 y_0 + x_1 y_0 + x_0 y_1$, $z_1 = x_1 y_1 + x_1 y_2 + x_2 y_1$, and $z_2 = x_2 y_2 + x_2 y_0 + x_0 y_2$. Parties then perform *resharing* by letting P_i sends $z_i' = \alpha_i + z_i$ to P_{i-1}, where $\alpha_0 + \alpha_1 + \alpha_2 = 0$ and P_i can generate α_i in offline phase as [15], [29], [46]. Finally, $\{(z_0', z_1'), (z_1', z_2'), (z_2', z_0')\}$ form the 2-out-of-3 replicated secret sharing of $[\![x \cdot y]\!]^L$.

Addition and multiplication can be easily generalized to vector product, MatMul, and CONV [15], [29].

3) MSB Extraction Π_{MSBE}: A key step in computing $\mathsf{ReLU}(x)$ (or $\mathsf{Sign}(x)$) is $\mathcal{F}_{\mathsf{MSBE}}$, which extracts the most significant bit $\mathsf{MSB}(x)$ as shown in Fig 1. There are two main approaches: GC- and secret sharing-based methods. We employ 3PC replicated secret sharing-based approach Π_{MSBE} proposed by FALCON [15] for better efficiency.

Authorized licensed use limited to: NANJING UNIVERSITY OF SCIENCE AND TECHNOLOGY. Downloaded on December 27,2025 at 15:28:00 UTC from IEEE Xplore. Restrictions apply.

Functionality $\mathcal{F}_{\mathsf{MSBE}}$

Parameters: Modular $L = 2^\ell$ for ring $\mathbb{Z}_L$.
Inputs: Receive $[\![x]\!]_i^L$ from P_i for $i \in \{0, 1, 2\}$.
Outputs: Compute the following

- Reconstruct x and compute $\mathsf{MSB}(x)$.
- Share $\mathsf{MSB}(x)$ over $\mathbb{Z}_2$ as $[\![\mathsf{MSB}(x)]\!]^2$, and send $[\![\mathsf{MSB}(x)]\!]_i^2$ to P_i.

Fig. 1. Most significant bit extraction functionality.

Functionality $\mathcal{F}_{\mathsf{daBits}}$

Inputs: Receive (DABITS-GEN) from all parties, ring $\mathbb{Z}_L$.
Outputs: Compute the following

- Sample a random bit $c \overset{\$}{\leftarrow} \{0, 1\}$.
- Share c over $\mathbb{Z}_2$ and $\mathbb{Z}_L$ as $[\![c]\!]^2$ and $[\![c]\!]^L$.
- Send $([\![c]\!]_i^2, [\![c]\!]_i^L)$ to P_i.

Fig. 2. The daBits functionality.

TABLE V
COMPARISON OF MOST SIGNIFICANT BIT EXTRACTION. RSS INDICATES REPLICATED SECRET SHARING

Protocol	Threat Model	Corruption	Setting	Multiplication
edaBits [47]	Mali.	D-Maj.	N PC	Beaver-triples
Rabbit [48]	Mali.	D-Maj.	N PC	Beaver-triples
Π_{MSBE} [15]	Semi-H.	H-Maj.	3 PC	RSS.

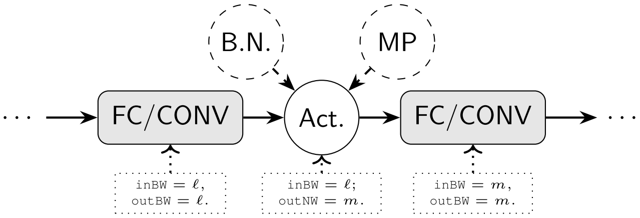


Fig. 3. A high-level view of FLEXBNNN, different FC/CONV might use different bit-widths (BW) and bit-width conversion is achieved in Activation (Act. for short).

Recall x_i is share of x modulo L, considering $x = x_0 + x_1 + x_2 \pmod{L}$ as a boolean circuit [15], [29], we have

$$\mathsf{MSB}(x) = \mathsf{MSB}(x_0) \oplus \mathsf{MSB}(x_1) \oplus \mathsf{MSB}(x_2) \oplus c, \quad (1)$$

where c is the carry bit from the previous index. A key insight here is that $\mathsf{MSB}(x_i)$s can be computed locally, and c is the WA_3 function computed on x_i's modulo $\frac{L}{2}$ (ignoring $\mathsf{MSB}(x_i)$), which is synonymous with computing WA_3 on $2x_i$'s modulo L, where $\mathsf{WA}_3(a_0, a_1, a_2, L)$ is defined as

$$\mathsf{WA}_3(a_0, a_1, a_2, L) = \begin{cases} 1, & L \leq \sum_{i=0}^{2} a_i < 2L \\ 0, & \text{otherwise}, \end{cases} \quad (2)$$

for given a_0, a_1, a_2, and L. Letting $a_i = 2x_i$, we have

$$\mathsf{MSB}(x) = \mathsf{MSB}(x_0) \oplus \mathsf{MSB}(x_1) \oplus \mathsf{MSB}(x_2) \oplus \mathsf{WA}_3(2x_0, 2x_1, 2x_2, L). \quad (3)$$

Note that the secret sharing-based method typically requires more communication rounds when evaluating a high-depth circuit, whereas the GC-based approach only needs constant rounds but much more communication. However, BNN is equipped with a much smaller bit-width, resulting in a low-depth circuit. Therefore, the secret-sharing based approach, rather than the GC-based one, is a better match for FLEXBNN.

Besides, edaBits [47] and Rabbit [48] can also extract MSB. As shown in Table V, the edaBits-based method focuses on resisting malicious adversaries in dishonest-majority with N parties and needs Beaver triples [49] for multiplication. Rabbit improves the efficiency of [47] but follows the same paradigm. Π_{MSBE} of [15] is inspired by edaBits, but specialized for 3PC and more efficient benefiting from the fast multiplication of replicated secret sharing. Improving Rabbit [48] in 3PC is a meaningful future work, We exploit Π_{MSBE} in this work.

4) Doubly-Authenticated Bits (daBits): A daBit is defined as a pair $([\![c]\!]^2, [\![c]\!]^L)$, where $c \in \{0, 1\}$ is a random bit. We make use of daBits to convert one single bit from the 3PC replicated Boolean Sharing to the Arithmetic world in this work, and the ideal functionality for generating daBits $\mathcal{F}_{\mathsf{daBits}}$ is illustrated in Fig. 2. Classic daBits can be generated in the offline phase, and for the detailed protocol please refer to [50].

5) Fixed-Point Representation: In FLEXBNN, we encode the floating-point inputs $x \in \mathbb{R}$ and binary weights w as fixed-point integers in a discrete ring [15], [29] for better efficiency:

- x is encoded as $\mathsf{EN}(x, f, R) = \lfloor 2^f \cdot x \rfloor \pmod{R}$, where $R = 2^r$, $r = 32$, and $f = 13$. Therefore, $[0, \frac{R}{2} - 1)$ represents $x \in \mathbb{R}^+$, and $[\frac{R}{2}, R)$ is for negative values;
- A binary value w is encoded similarly in a much smaller ring (*i.e.*, $\mathbb{Z}_L$), but with no fractional bit.

The encoding can be extended in vectorization [15], [29].

III. FLEXBNN CONSTRUCTION

We first give our intuition of flexible bit-width in §III-A. In §III-B, we show a high-level overview of the system and threat model. In §III-C and III-D, we propose our private protocols for BNN operators. And in §III-E, we present the workflow of seamless bit-width conversion.

A. Intuition of FLEXBNN

Recall the parameters and activation units of BNN are binarized. Utilizing these properties, our high-level intuitions of FLEXBNN are as follows:

- **FC** & **CONV**: Given the sizes of **FC** & **CONV**, we can determine their ranges. Therefore, we use flexible and smaller bit-widths (dotted arrows in Fig. 3).
- **Sign-based Activation**: Sign-based Activation extracts the Boolean-shared most significant bit via Π_{MSBE} and converts the boolean shares to arithmetic shares. Benefiting from the flexible bit-width, we can run Π_{MSBE} on the smaller bit-width. Furthermore, we integrate MP and BatchNorm into Activation (dashed arrows in Fig. 3).

Authorized licensed use limited to: NANJING UNIVERSITY OF SCIENCE AND TECHNOLOGY. Downloaded on December 27,2025 at 15:28:00 UTC from IEEE Xplore. Restrictions apply.

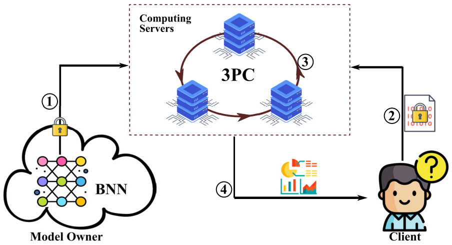


Fig. 4. Workflow of FLEXBNN. ① Model owner secret-shares BNN $\mathcal{M}$ among computing servers. ② Client secret-shares data $\mathbf{X}$. ③ Computing servers compute $\mathbf{y} = \mathsf{Eval}(\mathcal{M}, \mathbf{X})$ secretly. ④ Computing servers return $\mathbf{y}$ to client.

With the flexible bit-width technique, different FC (or CONV) layers of BNN are likely to be equipped with different bit-widths. Naively integrating prior bit-width conversion protocols [26], [45] will introduce additional costs. We achieve seamless bit-width conversion by exploiting the properties of BNN as illustrated in Fig. 3: In BNN, Activation function is applied between FC (or CONV) layers, we set the input (resp. output) bit-width of Activation as ℓ (resp. m), which is the bit-width of prior (resp. following) FC or CONV layers (as the dotted arrow for Act. in Fig. 3).

B. Overview

1) System Model: In Fig. 4, we employ three non-colluding servers for private inference. The BNN model and data are secret-shared among the servers. We define three entities in FLEXBNN: Model Owner, Client, and Computing Servers:

- **Model Owner:** In the offline phase, the model owner secret-shares BNN $\mathcal{M}$ among servers using 3PC RSS. More importantly, the model owner defines the bit-width for each layer of $\mathcal{M}$ and reveals it to servers.
- **Client:** In the online inference, the client shares its input $\mathbf{X}$ among the servers. In the end, the client receives the shares of inference result $\mathbf{y}$ and reconstructs it.
- **Computing Servers:** The computing servers are responsible for executing the private BNN inference protocols to jointly compute $\mathbf{y} = \mathsf{Eval}(\mathcal{M}, \mathbf{X})$ securely. All immediate values are in secret-shared fashion. After execution, the servers return the shares of $\mathbf{y}$ to the client.

The model owner and client can be parties in 3PC. In Fig. 4, we show them separately for ease of workflow illustration.

2) Threat Model: We design FLEXBNN against semi-honest adversaries in honest-majority. Namely, we assume that each of the three computing servers follows the protocol, but may *individually* try to learn information about other servers' inputs. Formally, we consider the standard simulation-based security definition in the presence of semi-honest adversaries:

Definition 1 (Semi-Honest Security): Let Π *be a 3PC protocol running in real-world and* $\mathcal{F} : (\{0,1\}^n)^3 \rightarrow (\{0,1\}^m)^3$ *be the ideal randomized functionality. We say* Π *securely computes* $\mathcal{F}$ *in presence of a single semi-honest corruption if for every corrupted party* P_i *(*$i \in \{0, 1, 2\}$*) and input* $\mathbf{x} \in (\{0,1\}^n)^3$*, there exists an efficient simulator* $\mathcal{S}$ *such that:*

$$\{\mathsf{view}_{i,\Pi}(\mathbf{x}), \mathsf{output}_{\Pi}(\mathbf{x})\} \overset{c}{\approx} \{\mathcal{S}(i, x_i, \mathcal{F}_i(\mathbf{x})), \mathcal{F}(\mathbf{x})\},$$

where $\mathsf{view}_{i,\Pi}(\mathbf{x})$ *is the view of* P_i *in the execution of* Π *on* $\mathbf{x}$, $\mathsf{output}_{\Pi}(\mathbf{x})$ *is the output of all parties, and* $\mathcal{F}_i(\mathbf{x})$ *denotes the* i*th output of* $\mathcal{F}(\mathbf{x})$.

3) MPC in Hybrid Model: Throughout this paper, we rely on some necessary ideal functionalities to design protocols. Also, we formalize some specialized ideal functionalities and securely realize them by designing protocols. Whenever using these components to construct protocols, we directly use the ideal functionalities for simplicity [51], [52].

C. Private Binary MatMul

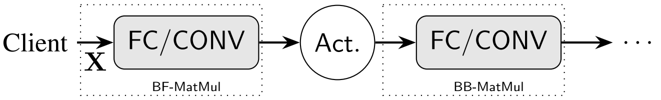


Fig. 5. Illustration of BF-MatMul and BB-MatMul in BNN.

As we discussed, we use MatMul to realize FC and CONV layers. Due to the good property of BNN, we can implement MatMul operations with two kinds of simplified circuits:

- BF-MatMul: Only the weight matrix is Binarized, and input is full of the Fixed-point representation of floating-point numbers (input layer);
- BB-MatMul: The weights and input matrices are both full of Binary values (± 1) (hidden linear layers).

We describe their constructions as follows.

1) BF-MatMul: FLEXBNN takes floating-point matrix $\mathbf{X} \in \mathbb{R}^{d \times |B|}$ as input. To multiply binary weight matrix $\mathbf{W} \in \{-1, 1\}^{n \times d}$ with $\mathbf{X}$ securely, a naive solution is transforming $\mathbf{W}$ and $\mathbf{X}$ as big integers meanwhile keeping enough f fractional bits, multiplying the transformed ones, and truncating the product to keep f fractional bits. This method requires 1 round and $\left(\frac{4}{3}\right) n|B|r$ bits of communication for each party on average [29]. However, It turns out that we can compute $\mathbf{WX}$ more efficiently as $\mathbf{W}$ only consists of binary integers. This means we only need to scale $\mathbf{X} \leftarrow \lfloor 2^f \cdot \mathbf{X} \rfloor \mod R$ in $\mathbb{Z}_R$ to keep enough fractional part, meanwhile maintaining $\mathbf{W}$ without scaling. In this way, we can compute $\mathbf{WX}$ without truncation as $\mathbf{Y}$ still maintains f fractional bits. As a result, we also need 1 round but $n|B|r$ bits of communication per party. Besides, we need to encode $\mathbf{B}$ with scaled by 2^f as well to compute $\mathbf{WX} + \mathbf{B}$.

2) BB-MatMul: To multiply two binary matrices $\mathbf{W} \in \{-1, +1\}^{n \times d}$ and $\mathbf{X} \in \{-1, +1\}^{d \times |B|}$ as Fig. 5, the naive solution is letting the parties secret-share $\mathbf{W}$ and $\mathbf{X}$ in large ring $\mathbb{Z}_R$ without scaling and compute MatMul in arithmetic circuits with $n|B|r$ bits of communication. XONN [12] and BANNERS [17] evaluate $\mathbf{WX}$ as XNORPOPCOUNT in boolean circuits by GC and Boolean Sharing, respectively. But they incur a blow-up in communication since they cannot exploit the *sum-then-resharing* technique of arithmetic circuits [29].

Differently, BB-MatMul computes $\mathbf{WX}$ by using a smaller ring for encoding and achieves binary MatMul in arithmetic

Authorized licensed use limited to: NANJING UNIVERSITY OF SCIENCE AND TECHNOLOGY. Downloaded on December 27,2025 at 15:28:00 UTC from IEEE Xplore. Restrictions apply.

Functionality $\mathcal{F}_{\mathsf{B2S}}$

Inputs: Receive $[\![b]\!]^2_i$ from P_i, ring $\mathbb{Z}_M$.
Outputs: Compute the following

- Reconstruct b and compute $y = 1 - 2b$.
- Share y over $\mathbb{Z}_M$ as $[\![y]\!]^M$ and send $[\![y]\!]^M_i$ to P_i.

Fig. 6. Bit to sign functionality.

Functionality $\mathcal{F}_{\mathsf{BA}}$

Inputs: Receive $[\![x]\!]^L_i$ from P_i, ring $\mathbb{Z}_L$ and $\mathbb{Z}_M$.
Outputs: Compute the following

- Reconstruct x and compute $b = \mathsf{MSB}(x)$.
- Compute $y = 1 - 2b$.
- Share y over $\mathbb{Z}_M$ as $[\![y]\!]^M$ and send $[\![y]\!]^M_i$ to P_i.

Fig. 7. Binary activation functionality.

circuits. Recall $\mathbf{W}$ and $\mathbf{X}$ are binarized, and parameters $(n, d, |B|)$ are static and public in advance, the range of $\mathbf{WX}$ must be in $[-d, d]$. Therefore, we can use a small modular with $\lceil\log_2(2d+1)\rceil$ bits. Taking $\mathbf{B}$ into consideration, our protocol only needs a modular M of $\lceil\log_2(2d+2)\rceil$ bits for encoding. Since $(2d+2) \ll R$ in practice, we reduce the communication by $\left(\frac{r}{\lceil\log_2(2d+2)\rceil}\right)\times$ over the naive method using $\mathbb{Z}_R$.

D. Private Binary Activation, MP, and BatchNorm

Protocol Π_{BA} achieves Sign-based $\mathsf{Activation}$ securely by utilizing Π_{MSBE} (c.f. II-B.3) and Π_{B2S}. We first present Π_{B2S} in §III-D.1, and then construct Π_{BA} in §III-D.2. Next, We show how to compute binary MP and $\mathsf{BatchNorm}$ together with Π_{BA} with small additional overhead in §III-D.3 and §III-D.4.

1) Boolean to Sign Conversion Protocol Π_{B2S}: As shown in Fig. 6, functionality $\mathcal{F}_{\mathsf{B2S}}$ converts a boolean-shared bit $[\![b]\!]^2$ to its corresponding arithmetic form $[\![y]\!]^M$ of m bits (*i.e.*, $M = 2^m$), where $y = +1$ if $b = 0$, and $y = -1$ otherwise. We design protocol Π_{B2S} for $\mathcal{F}_{\mathsf{B2S}}$ utilizing daBits [50].

In the offline phase, we generate $([\![c]\!]^2, [\![c]\!]^M)$ for a random secret bit c using $\mathcal{F}_{\mathsf{daBits}}$ specified in Fig. 2. In the online phrase, the parties reveal $e = b \oplus c$ and compute:

$$[\![y]\!]^M = [\![y^+]\!]^M + (e + (1-2e)\cdot[\![c]\!]^M)\cdot([\![y^-]\!]^M - [\![y^+]\!]^M), \tag{4}$$

where y^+ denotes $+1$ and y^- denote -1. The above protocol requires $1 + m$ bits of online communication in 2 rounds.

Given it is always $y^+ = +1$ and $y^- = -1$ in BNN, equation (4) can be simplified as $[\![y]\!]^M = 1 - 2e - 2(1 - 2e)\cdot[\![c]\!]^M$. As a result, Π_{B2S} only requires 1 bit of online communication in 1 round, and we analyze its correctness in Appendix VI-A.

a) Security: We have Theorem 1 to capture the security of Π_{B2S}. The proof can be found in Appendix VI-B.

Theorem 1: Π_{B2S} securely realizes $\mathcal{F}_{\mathsf{B2S}}$ in the presence of one semi-honest party in ($\mathcal{F}_{\mathsf{daBits}}$)-hybrid model.

Algorithm 1 Boolean to Sign Conversion Π_{B2S}

```
Input: $[\![b]\!]^2$, the 2-out-of-3 boolean shares of bit $b$.
Output: $[\![y]\!]^M = \mathsf{B2S}([\![b]\!]^2)$, the 2-out-of-3 arithmetic
    shares of $y = 1 - 2b$ in ring $\mathbb{Z}_M$ with $M = 2^m$.
The parties call functionality $\mathcal{F}_{\mathsf{Pre}}$ to receive daBits ($[\![c]\!]^2$,
    $[\![c]\!]^M$) for a random bit $c$.
The parties compute $[\![e]\!]^2 = [\![b]\!]^2 \oplus [\![c]\!]^2$ and reveal $e$.
The parties compute $[\![y]\!]^M = 1 - 2e - 2(1-2e)\cdot[\![c]\!]^M$.
return $[\![y]\!]^M$.
```

Functionality $\mathcal{F}_{\mathsf{BAM}}$

Inputs: Receive $\{[\![x_1]\!]^L_i, [\![x_2]\!]^L_i, \ldots, [\![x_{w\times h}]\!]^L_i\}$ from P_i for each MP window, window size $w \times h$, ring $\mathbb{Z}_L$ and $\mathbb{Z}_M$.
Outputs: Compute the following

- Reconstruct $\{x_1, x_2, \ldots, x_{w\times h}\}$.
- Compute $y_j = \mathsf{Sign}(x_j)$ for $j = 1, 2, \ldots, w \times h$.
- Compute the maximum $z = \mathsf{Max}(y_1, y_2, \ldots, y_{w\times h})$.
- Share y over $\mathbb{Z}_M$ as $[\![z]\!]^M$ and send $[\![z]\!]^M_i$ to P_i.

Fig. 8. Binary Activation+MP functionality.

2) Binary Activation Protocol Π_{BA}: As shown in Fig. 7, funcationality $\mathcal{F}_{\mathsf{BA}}$ takes x as input and outputs $y \in \{-1, +1\}$. This can be implemented by firstly extracting the most significant bit $[\![b]\!]^2 = \mathcal{F}_{\mathsf{MSBE}}([\![x]\!]^L)$, then computing $[\![y]\!]^M$ using protocol Π_{B2S}. The full protocol is described in Algorithm 2.

Compared with $\mathsf{ReLU}(x) = (1-b)\cdot x$, protocol Π_{BA} does not require the multiplication between $(1-b)$ and x, which saves 1 round and m bits of communication. More importantly, we can achieve seamless share conversion from $\mathbb{Z}_L$ to $\mathbb{Z}_M$ without any additional cost based on Π_{BA}, and the workflow of conversion is in §III-E. Π_{BA} can be generalized to vectors and matrices trivially, and we omit them for brevity.

Algorithm 2 Binary Activation Π_{BA}

```
Input: $[\![x]\!]^L$, the 2-out-of-3 arithmetic shares of $x$ in ring
    $\mathbb{Z}_L$ with $L = 2^\ell$.
Output: $[\![y]\!]^M = \mathsf{Sign}([\![x]\!]^L)$, the 2-out-of-3 arithmetic
    shares of $\mathsf{Sign}([\![x]\!]^L)$ in ring $\mathbb{Z}_M$ with $M = 2^m$.
The parties run protocol $\Pi_{\mathsf{MSBE}}$ on $[\![x]\!]^L$ to obtain the
    2-out-of-3 boolean shares $[\![b]\!]^2 = \mathcal{F}_{\mathsf{MSBE}}([\![x]\!]^L)$.
The parties call functionality $\mathcal{F}_{\mathsf{B2S}}$ over $[\![b]\!]^2$ to obtain
    $[\![y]\!]^M = \mathcal{F}_{\mathsf{B2S}}([\![b^2]\!])$.
return $[\![y]\!]^M$.
```

a) Security: We have Theorem 2 to capture the security of Π_{BA}. The proof can be found in Appendix VI-C.

Theorem 2: Π_{BA} securely realizes $\mathcal{F}_{\mathsf{BA}}$ in the presence of one semi-honest party in ($\mathcal{F}_{\mathsf{MSBE}}, \mathcal{F}_{\mathsf{B2S}}$)-hybrid model.

3) Binary Activation+MP Protocol Π_{BAM}: As illsustrated in funcationlity $\mathcal{F}_{\mathsf{BAM}}$ in Fig. 8, BNN performs MP after Sign-based $\mathsf{Activation}$ and MP outputs a maximal value within each MP window. To compute MP securely, a straightforward approach is performing secure comparison over the activated values; the parties need to perform $w \times h - 1$ comparisons with

Authorized licensed use limited to: NANJING UNIVERSITY OF SCIENCE AND TECHNOLOGY. Downloaded on December 27,2025 at 15:28:00 UTC from IEEE Xplore. Restrictions apply.

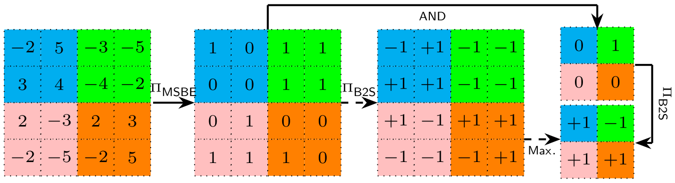


Fig. 9. Illustration of Sign-based $\mathsf{Activation}$+MP, where the window size is $w = 2$ and $h = 2$.

Functionality $\mathcal{F}_{\mathsf{BNAM}}$

Inputs: Receive $\{[\![x_1]\!]_i^L, [\![x_2]\!]_i^L, \ldots, [\![x_{w\times h}]\!]_i^L\}$ for each MP window and $[\![\frac{\beta}{\gamma}]\!]_i^L$ from P_i, window size $w \times d$, ring $\mathbb{Z}_L$ and $\mathbb{Z}_M$.

Outputs: Compute the following

- Reconstruct $\{x_1, x_2, \ldots, x_{w\times h}\}$ and $\left(\frac{\beta}{\gamma}\right)$.
- Compute $y_j = \mathsf{Sign}(x_j + \left(\frac{\beta}{\gamma}\right))$ for $j = 1, 2, \ldots, w\times h$.
- Compute the maximum $z = \mathsf{Max}(y_1, y_2, \ldots, y_{w\times h})$.
- Share z over $\mathbb{Z}_M$ as $[\![z]\!]^M$ and send $[\![z]\!]_i^M$ to P_i.

Fig. 10. Binary Activation+MP functionality.

a window of size $w \times h$, plus an additional secure 1-out-of-2 multiplexer operation [14], [15] after each comparison.

We provide an alternative and more efficient method for the task. In short, the parties can compute Binary $\mathsf{Activation}$ and MP together, and we show how to reduce overhead by exploiting some specialized properties of BNN. Specifically, instead of computing Binary $\mathsf{Activation}$ first and then secure comparisons as discussed earlier (*i.e.*, following the dashed arrow in Fig. 9), the parties first perform logical AND between all MSBs of values in the window, and then convert the result to an arithmetic sharing of ± 1 via Π_{B2S} (*i.e.*, following the solid arrow in Fig. 9). We call the resulting protocol Π_{BAM}.

Now the parties only need to perform n cascaded AND operations first; this can be efficiently evaluated with $w \times h - 1$ bits of communication in $\log_2(w \times h)$ rounds [15], [29]. Then the parties convert the AND result, in each window, by a single invocation of Π_{B2S} with 1 bit of communication in 1 round. Compared with the naive approach, the saving in the communication of our method is significant.

a) Security: Theorem 3 captures the security of Π_{BAM}. The proof can be found in Appendix VI-D.

Theorem 3: Π_{BAM} securely realizes $\mathcal{F}_{\mathsf{BAM}}$ in the presence of one semi-honest party in ($\mathcal{F}_{\mathsf{MSBE}}, \mathcal{F}_{\mathsf{B2S}}$)-hybrid model.

4) Binary BatchNorm+Activation+MP Protocol Π_{BNAM}: The ideal functionality $\mathcal{F}_{\mathsf{BNAM}}$ is defined as shown in Fig. 10. We first show solution to $\mathsf{BatchNorm} + \mathsf{Activation}$, and then construct Π_{BNAM}. In BNN, Sign-based activation function and $\mathsf{BatchNorm}$ can be combined as $y = \mathsf{Sign}(x + \left(\frac{\beta}{\gamma}\right))$. Therefore, y can be computed by running protocol Π_{BA} on $x + \left(\frac{\beta}{\gamma}\right)$. Note γ and β are available for model owner, thus it can compute $\left(\frac{\beta}{\gamma}\right)$ in plaintext and secret share $\left(\frac{\beta}{\gamma}\right)$ in setup.

One issue is that x, which is output by the prior linear layer, can have different fractional bits depending on concrete computation tasks in BNN. To maintain correctness, we must encode $\left(\frac{\beta}{\gamma}\right)$ with the same fractional bits as x. Correspondingly, there are two cases illustrated in Fig. 11:

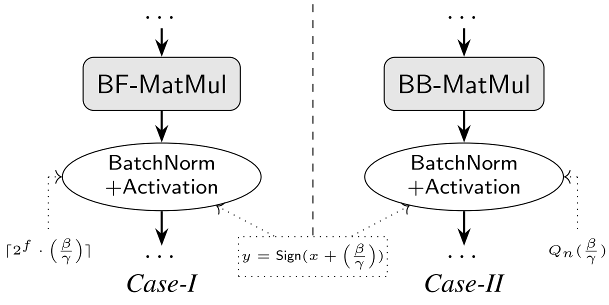

BF-MatMul
BatchNorm +Activation
BB-MatMul
BatchNorm +Activation
$\lceil 2^f \cdot \left(\frac{\beta}{\gamma}\right) \rceil$
$y = \mathsf{Sign}(x + \left(\frac{\beta}{\gamma}\right))$
$Q_n(\frac{\beta}{\gamma})$
Case-I
Case-II

Fig. 11. Encodings of $\mathsf{BatchNorm}$ parameter $\left(\frac{\beta}{\gamma}\right)$ in BNN.

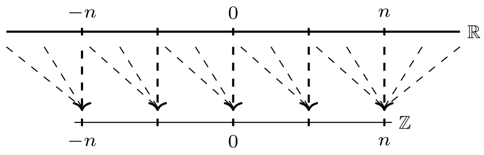


Fig. 12. Illstration of $Q_n(\frac{\beta}{\gamma})$. The continuous interval is mapped to the discrete interval.

- *Case-I*: When x is computed by $\mathsf{BF\text{-}MatMul}$, we encode x with f fractional bits. We hence encode $\left(\frac{\beta}{\gamma}\right)$ as $\lceil 2^f \cdot \left(\frac{\beta}{\gamma}\right) \rceil$, and secret-share $\lceil 2^f \cdot \left(\frac{\beta}{\gamma}\right) \rceil$ in the large ring $\mathbb{Z}_R$.
- *Case-II*: x is computed by $\mathsf{BB\text{-}MatMul}$. In this case, x is with 0 fractional bits. Assume x lies in $[-n, n]$, then we encode $\left(\frac{\beta}{\gamma}\right)$ by rounding up and saturating as $Q_n(\frac{\beta}{\gamma}) = \mathsf{min}(\mathsf{max}(\lceil \frac{\beta}{\gamma} \rceil, -n), n)$, where $Q_n(\frac{\beta}{\gamma})$ maps $\left(\frac{\beta}{\gamma}\right)$ to be the rounding-up integers within $[-n, n]$ as shown in Fig. 12. Therefore, $x + Q_n(\frac{\beta}{\gamma})$ is in $[-2n, 2n]$, and we need $\ell' = \lceil \log_2(4n + 1) \rceil$ bits for encoding to avoid overflow. For better overall efficiency, we use $\mathbb{Z}_{2^{\ell'}}$ to encode the prior linear layer as well.

$\mathsf{BatchNorm} + \mathsf{Activation}$ can be computed together as above. When taking MP into consideration, protocol Π_{BNAM} is constructed as follows: we compute AND on the $\mathsf{MSB}(x + \left(\frac{\beta}{\gamma}\right))$ of each window in the Boolean world, and then convert the Boolean results into Arithmetic shares using protocol Π_{B2S}.

a) Security: Theorem 4 shows the security of Π_{BNAM}. The proof can be found in Appendix VI-E.

Theorem 4: Π_{BNAM} securely realizes $\mathcal{F}_{\mathsf{BNAM}}$ in presence of one semi-honest party in ($\mathcal{F}_{\mathsf{MSBE}}, \mathcal{F}_{\mathsf{Mult}}, \mathcal{F}_{\mathsf{B2S}}$)-hybrid model.

E. Workflow of Seamless Bit-Width Conversion

In Table VI, we show our configurations (*i.e.*, protocol and bit-width) of NN-A and compare FLEXBNN with BANNERS.

BANNERS also supports flexible bit-width, but it incurs 30-191× more communication and 6-14× more run-time than FALCON (without BNN, c.f. §IV-B). The additional costs mainly come from *binary vector product* in MatMul: Given two binary vectors $\mathbf{w}$ and $\mathbf{x}$ of size d, ❶ BANNERS first computes XNOR operation in boolean circuits, ❷ and then

Authorized licensed use limited to: NANJING UNIVERSITY OF SCIENCE AND TECHNOLOGY. Downloaded on December 27,2025 at 15:28:00 UTC from IEEE Xplore. Restrictions apply.

TABLE VI

NN-A and FlexBNN Configurations. `inBW` and `outBW` Denote Input and Output Bit-Widths. B2A Indicates Boolean to Arithmetic Conversion. To Take Full Advantages of Integer Datatypes, We Use Rings $\mathbb{Z}_{2^8}$, $\mathbb{Z}_{2^{16}}$, and $\mathbb{Z}_{2^{32}}$ for Arithmetic Shares

Layer	Architecture	FlexBNN	Banners
FC	`input` 28×28, `output 128.`	BF-MatMul, `inBW=outBW=32.`	BF-MatMul, `inBW=outBW=32.`
Activation	`input 128, output 128.`	Π_{BA}, `inBW=32, outBW=16.`	Π_{MSBE}, `inBW=32, outBW=1.`
FC	`input 128, output 128.`	BB-MatMul, `inBW=outBW=16.`	XNOR, `inBW=outBW=1;` B2A, `inBW=1,outBW=16;` Popcount, `inBW=outBW=16.`
Activation	`input 128, output 128.`	Π_{BA}, `inBW=16, outBW=16.`	Π_{MSBE}, `inBW=16, outBW=1.`
FC	`input 128, output 10.`	BB-MatMul, `inBW=outBW=16.`	XNOR, `inBW=outBW=1;` B2A, `inBW=1,outBW=16;` Popcount, `inBW=outBW=16.`

explicitly converts the boolean shares of the intermediate vector (of size d) to arithmetic shares in $\mathbb{Z}_{2^\tau}$ for flexible bit-width, where $\tau = \lceil \log_2(2d+1) \rceil$. ❸ Finally, the parties compute summation locally. This requires $\approx 2d\tau$ bits communication, which is linearly dependent on d. Falcon only needs r bits for integer vector product (independent of d) by exploiting *sum-then-rehsharing* [29] of arithmetic circuits. Even though $r > \tau$ (*i.e.*, $r = 32$ and $\tau = 16$), Falcon is more efficient than Banners ($2d\tau > r$) since it is commonly $d = 128$ to 4096 in NN. In terms of the overall inference, the improvement benefiting from smaller bit-width is compromised by the additional costs of binary vector product.

In contrast, suppose our Π_{BA} is with ℓ-bit inputs x and m-bit outputs y as Fig. 3, seamless bit-width conversion works as follows: i) parties compute $[\![b]\!]^2 = \mathcal{F}_{\mathsf{MSBE}}([\![x]\!]^L)$ as step 1 of Algorithm 2. ii) In step 2 of Algorithm 2, parties convert $[\![b]\!]^2$ to the corresponding arithmetic shares $[\![y]\!]^M$ of m-bit. Therefore, we can convert the ℓ-bit inputs to m-bit outputs at no additional cost than Π_{BA}. Even though FlexBNN needs Π_{B2S} for Activation (not required in Banners), our total communication is much more efficient than Banners.

We propose the above seamless bit-width conversion mainly inspired by the special property that BNN's activation units are binarized. In Π_{BAM} and Π_{BNAM}, we can achieve seamless bit-width conversion similarly as shown in Fig. 9.

IV. System Implementation & Evaluation

We compare FlexBNN to various 2PC and 3PC frameworks. All frameworks are secure against semi-honest adversaries, and 3PC methods work in honest-majority. FlexBNN focuses on security against semi-honest adversaries in 3PC honest-majority. Note that we compare FlexBNN with semi-honest Falcon for fair comparisons in this work.

Experimental Setup: We implement FlexBNN on top of Falcon in C++ and run our experiments on Intel(R) Xeon(R) CPU E5-2650 v3@ 2.30GHz servers with 64GB RAM. Our evaluation uses similar or poorer hardware compared to prior works. For LAN, our bandwidth is about 625MBps and round trip time (rtt) is about 0.2ms. For WAN, our bandwidth and rtt are about 40MBps and 70ms, respectively.

Optimizations: All offline computations are executed as Falcon and we focus on online efficiency and do not take offline costs into account. We omit Softmax or ReLU of the output layer since it has no impact on results. All experiments are executed 10 times and we record the average as the results.

TABLE VII

Accuracy of Frameworks for NN and Datasets

Accuracy (%)	A	B	C	LeNet	VGG16	AlexNet
SecureML	93.4	-	-	-	-	-
MiniONN	97.6	98.9	99.0	-	-	-
XONN	97.6	98.6	99.0	-	-	-
Quotient	96.0	-	-	-	-	-
Chameleon	-	99.0	-	-	-	-
SecureNN	93.4	-	99.1	99.1	-	-
Banners	97.1	97.2	98.5	-	-	-
Falcon	97.4	97.8	98.6	99.1	90.2	55.7
FlexBNN	96.1	97.5	98.3	98.2	87.0	44.5

Neural Networks: For MNIST [53], we run FlexBNN on NN-A, B, C, and LeNet for fair comparisons with SecureML [21], MiniONN [22], XONN [12], Quotient [13], Chameleon [28], SecureNN [14], and Banners [17]. Besides, we evaluate FlexBNN on VGG16 (on CIFAR-10) [54] and AlexNet (on Tiny ImageNet) [55] as Falcon [15].

A. Accuracy

As shown in table VII, 1) For NN-A, -B, -C, and LeNet on MNIST and VGG16 on CIFAR-10, we achieve comparable accuracy as prior works: i) For NNs on MNIST, the accuracy loss of FlexBNN is ≤1.5%. ii) For VGG16 on CIFAR-10, FlexBNN introduces a accuracy loss of ≈3% compared with Falcon. 2) For AlexNet on the more challenging dataset Tiny ImageNet, FlexBNN introduces non-negligible accuracy loss, which is ≈11% compared with Falcon. As a consequence, FlexBNN is better for the commonly used datasets (*e.g.*, MNIST and CIFAR-10) since the degradation can be acceptable in many applications.

B. Single Inference

We measure the costs for a single query and show our improvements as follows.

1) Improvements on Small NN: Compared with existing works, FlexBNN improves communication and run-time significantly as Table VIII shows. On average, we reduce the communication by 1537× over MiniONN, 500× over XONN, 300× over Chameleon, 127× over Banners, 126× over SecureNN, and 1.3× over Falcon. In terms of run-time, we are 488× faster than SecureML, 177× faster than

Authorized licensed use limited to: NANJING UNIVERSITY OF SCIENCE AND TECHNOLOGY. Downloaded on December 27,2025 at 15:28:00 UTC from IEEE Xplore. Restrictions apply.

TABLE VIII

EXPERIMENTAL RESULTS OF SECUREML, MINIONN, XONN, QUOTIENT, CHAMELEON, BANNERS, SECURENN, FALCON, AND FLEXBNN FOR SMALL NETWORKS ON MNIST DATASET. COMM. IS REPORTED IN MB AND RUN-TIME IS IN SECONDS

Framework			A		B		C		LeNet	
			Comm.	Run-time	Comm.	Run-time	Comm.	Run-time	Comm.	Run-time
2 PC	SecureML	LAN	-	4.88	-	-	-	-	-	-
	MiniONN	LAN	15.8	1.04	47.6	1.28	657.5	9.32	-	-
	XONN	LAN	4.29	0.13	38.3	0.16	32.1	0.15	-	-
	QUOTIENT	LAN	-	0.46	-	-	-	-	-	-
	QUOTIENT	WAN	-	8.8	-	-	-	-	-	-
3 PC	Chameleon	LAN	-	-	12.9	2.7	-	-	-	-
	SecureNN	LAN	2.1	0.043	4.05	0.076	8.86	0.13	18.94	0.23
	BANNERS	LAN	2.30	0.14	2.54	0.12	15.36	0.20	-	-
	FALCON	LAN	0.012	0.043	0.049	0.033	0.51	0.09	0.74	0.08
	FLEXBNN	LAN	**0.008**	**0.01**	**0.043**	**0.01**	**0.43**	**0.031**	**0.61**	**0.074**
	SecureNN	WAN	2.1	2.43	4.05	3.06	8.86	3.93	18.94	4.08
	FALCON	WAN	0.012	1.50	0.049	1.20	0.51	3.53	0.74	3.06
	FLEXBNN	WAN	**0.008**	**0.95**	**0.043**	**1.047**	**0.43**	**2.01**	**0.61**	**2.07**

TABLE IX

EXPERIMENTAL RESULTS OF FALCON AND FLEXBNN FOR VGG16 ON CIFAR-10 AND ALEXNET ON TINY IMAGENET

Framework	LAN/WAN	VGG16		AlexNet	
		Comm.	Run-time	Comm.	Run-time
FALCON	LAN	13.52	2.05	19.22	1.78
FLEXBNN	LAN	**7.92**	**1.52**	**13.66**	**1.24**
FALCON	WAN	13.52	15.36	19.22	27.51
FLEXBNN	WAN	**7.92**	**8.83**	**13.66**	**5.02**

TABLE X

RUN-TIME FOR A BATCH OF $|B| = 1$ AND 128 IMAGES

Neural Networks	Run-time, LAN		Run-time, WAN	
	$\|B\| = 1$	$\|B\| = 128$	$\|B\| = 1$	$\|B\| = 128$
LeNet	0.077	1.83	2.38	9.06
VGG16	1.53	62.14	9.18	146.35
AlexNet	1.25	139.33	5.32	282.18

MiniONN, 11× faster than XONN, 46× faster than QUOTIENT in LAN when compared to 2PC-based protocols. And when compared to 3PC-based schemes, we are 290× faster than Chameleon, 10× faster than BANNERS, 5.2× faster than SecureNN, and 2.9× faster than FALCON in LAN. And in WAN, we achieve respective 9.3×, 2.2×, and 1.5× improvements over QUOTIENT, SecureNN, and FALCON.

2) Improvements on Large NN: As illustrated in Table IX, FLEXBNN also outperforms FALCON in both communication and run-time when evaluating the large NN. Concretely, we reduce communication by 1.5× and achieve 1.1× and 2.7× faster in respective LAN and WAN on average.

The efficiency improvements mainly come from two aspects: i) Benefiting from BNN's features and our seamless bit-width conversion method, we can employ flexible and small rings to encode different BNN layers instead of using $\mathbb{Z}_{2^{32}}$ or even $\mathbb{Z}_{2^{64}}$ for all layers. This method reduces the bit-width in secure computation and results in depth-efficient circuits, which improves communication and run-time simultaneously. ii) Furthermore, we propose several novel in-depth optimizations for the secure computation of binary non-linear layers. These optimizations embed the complex non-linear operations, *e.g.*, BatchNorm and MP, into the basic Sign-based Activation with a little additional overhead.

C. Batch Inference

We can leverage FLEXBNN to process a batch of images, which amortizes the run-time and improves the scalability.

1) Amortized Run-Time: Table X shows the run-time of FLEXBNN over a batch of 128 images on LeNet, VGG16, and AlexNet in LAN and WAN settings. Concretely, the amortized time for a single image drops from 0.077s to 0.014s (5× improvements) in LAN and from 2.38s to 0.07s (34× improvements) in WAN for LeNet. Similarly, the amortized time of VGG16 is reduced from 1.53s to 0.48s (3× improvement) in LAN and from 9.18s to 1.14s (8× improvement) in WAN, and amortized time of AlexNet is reduced by 1.16× (1.25s to 1.08s) in LAN and 2.4× (5.32s to 2.20s) in WAN.

2) Scalability Improvements: To illustrate our scalability improvements, we further measure the communication costs and run-time of LeNet, VGG16, and AlexNet with different batchsize in Fig. 13. We have the following findings: i) Firstly, the communication costs and run-time (in LAN and WAN) scale linearly with the batchsize $|B|$. This is not unexpected; after all, we require more communication and run-time when processing more images. ii) More importantly, both the communication overhead and run-time of our FLEXBNN are lower than that of FALCON for every given batchsize $|B|$. Moreover, our costs increase slower than that of FALCON for different $|B|$. Therefore, our FLEXBNN is more scalable for processing a large amount of images. For example, when $|B| = 128$, FLEXBNN is far more efficient than FALCON in both run-time and communication for all NN considered in this literature.

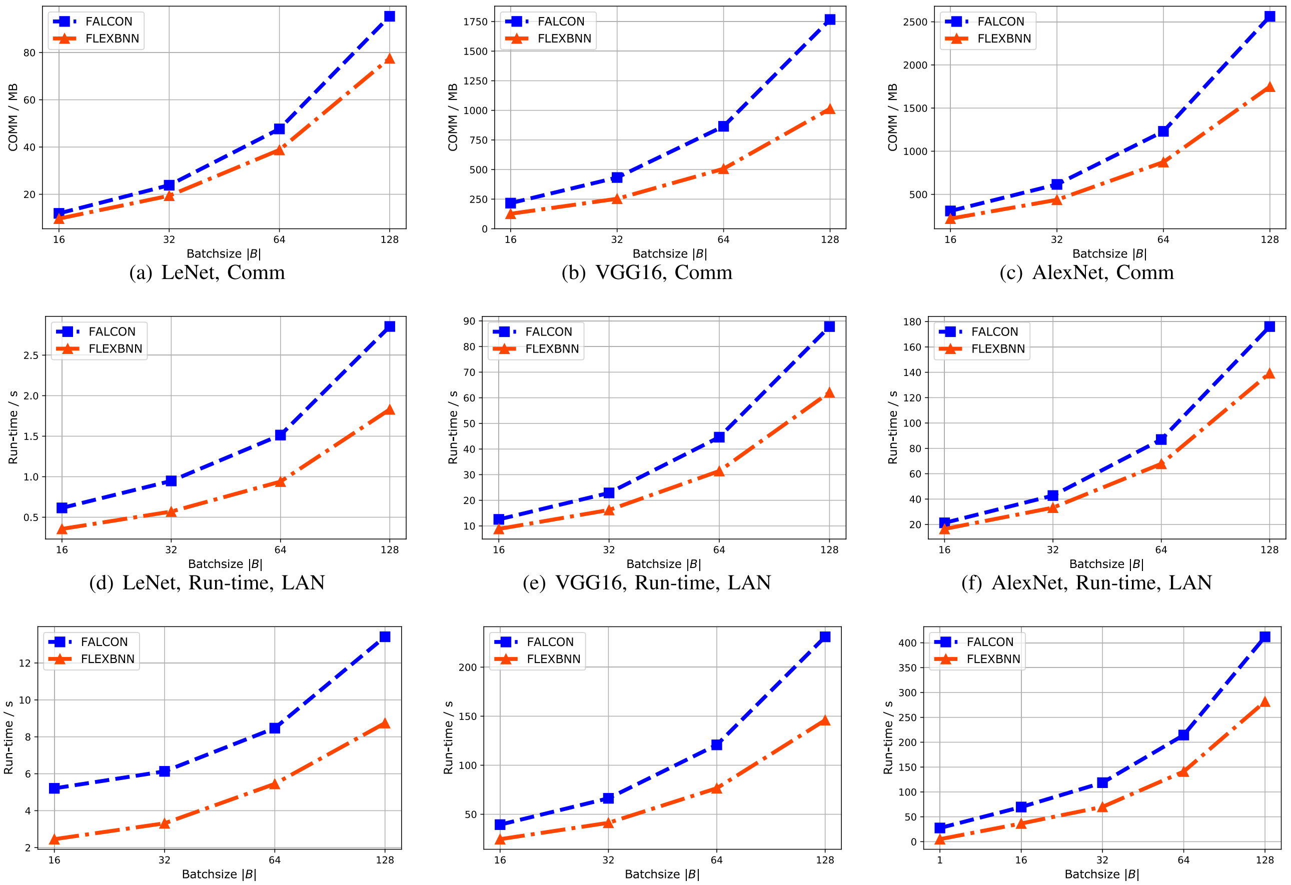
The improvements of amortized run-time in WAN are more significant than in LAN since we can save more communication time in WAN. The scalability improvements mainly arise from the smaller encoding rings and our optimizations.

D. Microbenchmarks

Table XI shows our improvements for basic operators.

1) Improvements on CONV *&* FC*:* For CONV, we reduce the communication overhead by 1.8× and improve the run-time by 1.1× in LAN and 1.3× in WAN on average. And

Authorized licensed use limited to: NANJING UNIVERSITY OF SCIENCE AND TECHNOLOGY. Downloaded on December 27,2025 at 15:28:00 UTC from IEEE Xplore. Restrictions apply.



(a) LeNet, Comm
(b) VGG16, Comm
(c) AlexNet, Comm
(d) LeNet, Run-time, LAN
(e) VGG16, Run-time, LAN
(f) AlexNet, Run-time, LAN
(g) LeNet, Run-time, WAN
(h) VGG16, Run-time, WAN
(i) AlexNet, Run-time, WAN

Fig. 13. Communication and run-time of batch inference for LeNet, VGG16, and AlexNet of FALCON and FLEXBNN.

TABLE XI

MICROBENCHMARKS OF CONV, FC, Activation, Activation + MP, AND BatchNorm + Activation + MP. WE COMPARE OUR EVALUATIONS ON LENET, VGG16, AND ALEXNET WITH A BATCH OF $|B| = 128$ IMAGES OF CORRESPONDING DATASETS TO FALCON

Neural Networks		LeNet			VGG16			AlexNet		
Layers	Framework	Comm	Run-time LAN	Run-time WAN	Comm	Run-time LAN	Run-time WAN	Comm	Run-time LAN	Run-time WAN
CONV	FALCON	10.05	0.42	1.62	188	43.51	60.57	411	112.91	148.14
	FLEXBNN	**6.72**	**0.36**	**1.19**	**87.6**	**41.48**	**48.63**	**256**	**109.39**	**116.48**
FC	FALCON	0.35	0.067	0.84	6.28	3.46	4.25	1.53	1.91	2.19
	FLEXBNN	**0.13**	**0.061**	**0.16**	**2.09**	**2.83**	**3.02**	**1.04**	**1.84**	**1.97**
Activation	FALCON	2.74	0.078	0.84	853	17.15	84.51	11.1	0.26	1.92
	FLEXBNN	**1.31**	**0.031**	**0.335**	**560**	**10.84**	**57.55**	**5.75**	**0.15**	**1.41**
Activation +MP	FALCON	84.79	1.74	10.04	720	14.45	73.19	290	5.77	29.46
	FLEXBNN	**69.41**	**1.38**	**7.08**	**364**	**6.99**	**37.15**	**146**	**2.76**	**13.97**
BatchNorm +Activation+MP	FALCON	-	-	-	-	-	-	1850	48.52	230.27
	FLEXBNN	-	-	-	-	-	-	**1339**	**25.19**	**148.36**

for FC, we reduce the communication overhead by 2.4× and improve the run-time by at most 1.1× and 2.6 in LAN and WAN.

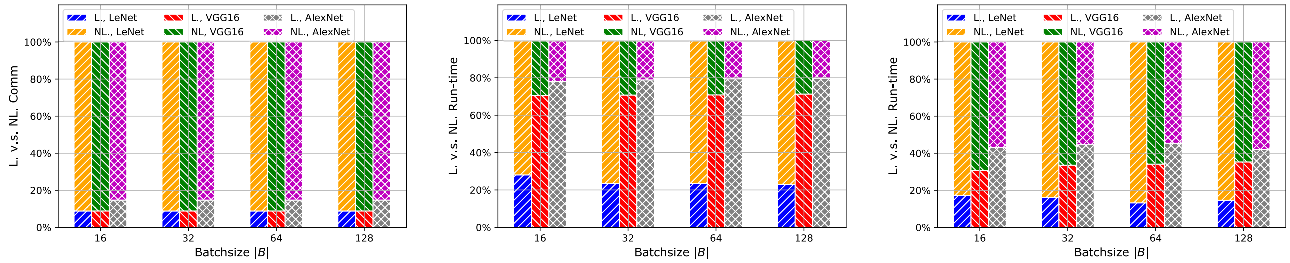
2) Improvements on Activation*,* MP *&* BatchNorm*:* For the individual Acitvation function, we reduce the communication by 1.9× and improve the run-time by 1.8× in LAN and WAN. When combining Activation and MP together, we reduce the communication costs by 1.7× and improve the run-time by 1.8× on average in both LAN and WAN. Finally, when putting BatchNorm, Activation, and MP together, we can reduce the communication costs by 1.4× and improve the run-time by 1.9× in LAN and 1.6× in WAN.

For CONV and FC layers, the improvements mainly come from that we do not require truncation for BF-MatMul and employ small bit-width for BB-MatMul. On the other hand, our circuits for non-linear functions are more depth-efficient than FALCON. And we propose several optimizations.

E. Linear V.S. Non-Linear Overhead

In Fig. 14, we measure linear and non-linear layers' costs to explore our efficiency bottleneck in different settings. For communication costs, the non-linear layers are more expensive. For run-time, we have the following observations:

Authorized licensed use limited to: NANJING UNIVERSITY OF SCIENCE AND TECHNOLOGY. Downloaded on December 27,2025 at 15:28:00 UTC from IEEE Xplore. Restrictions apply.



(a) Communication (b) Run-time, LAN (c) Run-time, WAN

Fig. 14. Linear vs. Non-Linear communication and run-time of private BNN inference using FLEXBNN in LAN and WAN for LeNet, VGG16, and AlexNet, where L is short for linear layers, and NL denotes non-linear layers.

1) Computation-Bound on Large NN in LAN: Fig. 14(b) shows the run-time of linear layers is significantly more costly. Even though the non-linear layers are more communication expensive than linear layers, the communication does not introduce much run-time, which is far less than MatMul. This is because there are enough internet resources. Hence, the costs of MatMul dominate the run-time, which indicates FLEXBNN is computation-bound when evaluating large NN in LAN.

2) Communication-Bound on Large NN in WAN: Fig. 14(c) shows that the run-time required by non-linear layers becomes a significant fraction of total run-time when evaluating FLEXBNN on large NN in WAN. The reason is that we are strictly limited by the internet resources in WAN. As non-linear layers incur much more communication overhead and rounds, they introduce more run-time than linear layers.

3) Communication-Bound on Small NN: For small NN (*i.e.*, LeNet), FLEXBNN is communication-bound. But the non-linear layers in WAN take a greater proportion of total run-time than in LAN. This is consistent with the analysis of large NN.

Through the above observation and analysis, we claim that when evaluating FLEXBNN on large NN in LAN, we should focus on designing faster computation solutions (e.g., GPUs). And when evaluating FLEXBNN in WAN, reducing communication will benefit much more to the overall efficiency.

F. Malicious Security, Capability, & Accuracy

FALCON achieves security with abort against malicious adversaries in 3PC honest-majority by exploiting random multiplication triples. This comes at the cost of generating triples in offline and performing checks in online phase. Strengthening FLEXBNN to resist malicious adversaries is also meaningful. Besides, FLEXBNN focuses on private BNN inference but not training and introduces non-negligible accuracy loss for large-scale datasets (*e.g.*, Tiny ImageNet). Therefore, it is challenging and meaningful to extend FLEXBNN to support BNN training and improve its accuracy on large-scale datasets.

V. RELATED WORK

Given the breadth of MPC-based private NN, we summarize the works in protocol design and quantization technique.

A. Protocol Design

In the earlier stage, the works on private machine learning evaluation mainly focused on traditional machine learning models such as linear regression [56], [57], [58], logistic regression [59], decision trees [60], k-means clustering [61], [62], SVM [63], [64], [65], [66], [67], and statistical algorithms [68], [69], [70], [71].

CryptoNets [10] and CryptoDL [11] used HE to protect data privacy in secure NN inference. Mohassel et al. proposed SecureML [21] on the two-server setting with secret sharing and GC. Meanwhile, Liu et. al. proposed MiniONN [22] to optimize the matrix multiplications protocols. DeepSecure [72] used GC to achieve privacy-preserving DL prediction, and GAZELLE [23] combined techniques from HE and MPC to achieve fast private inference. EPIC [73] supported efficient private image classification based on SVM, MPC, and transfer learning, and was secure against malicious adversaries and much more efficient than GAZELLE. The main novelty of EPIC is that it is built upon transfer learning to minimize the cost of the privacy-preserving part. EzPC [74] was a ABY-based [75] private inference framework, and its follow-up works [25], [26] focused on improving performance.

In terms of 3-party setting, Chameleon [28] used the same technique as in [22] for matrix multiplication, but employed a semi-honest third-party to generate correlated multiplication triplets in offline. SecureNN [14] and Cryptflow [76] constructed novel protocols for non-linear functions with the help of the third-party. What's more, some schemes were proposed based on 3PC replicated secret sharing for better efficiency [15], [29]. FALCON was one of the most efficient methods and can be scalable to large neural networks such as VGG16 and AlexNet. Additionally, works [16], [19], [20], [30], [32], [77] were evaluated in 3PC, or N-party ($N > 3$) with a focus on resisting malicious adversaries in honest-majority and required much more costs. And works like [78], [79], [80] focused on constructing maliciously secure protocols in dishonest majority setting. Besides, some recent works [24], [42], [43], [81], [82] proposed to exploit GPU to improve the efficiency of MPC protocols and private machine learning. And we are willing to combine them with private BNN for further research. More protocols can be referred to [34], [83], [84], [85].

Authorized licensed use limited to: NANJING UNIVERSITY OF SCIENCE AND TECHNOLOGY. Downloaded on December 27,2025 at 15:28:00 UTC from IEEE Xplore. Restrictions apply.

B. Quantization Technique

The above works mainly employed naively quantization method where the floating-point values are encoded as fixed-point integers with enough fractional bits in large rings. Some other works utilized lower bits to represent quantized neural networks [12], [13], [33]. Riazi et al. proposed XONN [12], where the weights and activations are in ±1 and the authors used GC and OT to provide constant round private inference. QUOTIENT [13] was proposed to realize the secure computation of ternarized neural networks by GC and OT. Therefore, they were suffering from the enormous communication costs introduced by GC. BANNERS was similar to FLEXBNN and aimed to evaluate BNN with replicated secret sharing, but it computed binary vector product via XNOR in boolean circuits and required explicit boolean to arithmetic conversion for intermediate results. Thus, BANNERS incurs more costs than ours. More advanced quantized method [33] utilized fixed-point *scale* and lower-bit (*e.g.*, 8-bit) *zero-point* to encode floating-point values for better efficiency than Cryptflow [76]. This kind of quantization, which is supported by existing machine learning frameworks (*e.g.*, TensorFlow and PyTorch), is inherent from the underlying NN and different from BNN. Integrating our flexible bit-width and seamless conversion techniques into [33] will be meaningful future work. In addition, Rathee et al.proposed SECFLOAT [86] to evaluate 2PC protocols directly on floating-point NN for high-precision tasks, but this method incurs much more costs than fixed-point encoded NN. Furthermore, the bit-width conversion plays an important role in private quantized NN approaches. Recently, Delpech de Saint Guilhem et al. proposed to convert an additive sharing over a ring into an additive sharing over the integers [45], and Rathee et al. proposed different rings conversion protocols [26]. These works achieved excellent performance. But naively using their protocols in FLEXBNN will introduce explicit conversion protocol, which gains no benefits and even increase the costs. Thus, we are willing to focus on combining these protocols with BNN's special properties for future work.

VI. CONCLUSION

In this work, we propose FLEXBNN for private BNN inference in 3PC with honest-majority. We employ flexible bit-widths and achieve seamless bit-width conversion. Extensive experiments demonstrate our improvements. In future, we will enhance FLEXBNN to resist malicious adversaries and improve the efficiency by hardware such as GPU.

APPENDIX

A. Correctness of Π_{B2S}

We analyze the correctness of protocol Π_{B2S} as follows. From the definition of functionality $\mathcal{F}_{\mathsf{B2S}}$, we have $y = (1-b)\cdot y^{+} + b\cdot y^{-}$. Since $e = b \oplus c$, we have $b = c \oplus e = c + e - 2ce$. Combining them together, we have $y = y^{+} + (e + (1-2e)\cdot c)\cdot(y^{-} - y^{+})$. As $[\![y]\!]^M$, $[\![y+]\!]^M$, $[\![y^{-}]\!]^M$ and $[\![c]\!]^M$ are of secret-shared fashion in protocol Π_{B2S}, we have equation (4). Furthermore, it can be guaranteed that $y^{+} = +1$ and $y^{-} = -1$ in BNN, we can simplify equation (4) by using local operations.

B. Proof of Theorem 1

Proof: For a corrupted party P_i where $i \in \{0,1,2\}$, we show a polynomial simulator Sim_i who can generate a simulated view that is indistinguishable from the one of real-world execution. Sim_i works as follows:

1) Sim_i invokes the simulator $\mathsf{Sim}_{\mathsf{Pre},i}$ for P_i in protocol Π_{Pre}. Sim_i simply appends the simulated view generated by $\mathsf{Sim}_{\mathsf{Pre},i}$ as a part of its own simulation.
2) Sim_i samples a random value $e \xleftarrow{\$} \mathbb{Z}_2$ and appends e to the simulated view.

Our protocol works in hybrid model. The simulator for Π_{Pre} exists by definition. $[\![c]\!]^2$ and $[\![c]\!]^M$ are random generated by $\mathcal{F}_{\mathsf{Pre}}$. In real-world execution, c is a random secret bit, thus $e = b \oplus c$ is random. Given this, a random bit e sampled by Sim_i suffices for simulation. As regards to local computation, *e.g.*, $[\![y]\!]^M = 1 - 2e - 2(1-2e)\cdot[\![c]\!]^M$, the simulator Sim_i can simulate trivially because there is no protocol message from local computation. Overall, the view generated by Sim_i is indistinguishable from the real-world execution. □

C. Proof of Theorem 2

Proof: The simulator Sim_i for $i \in \{0,1,2\}$ works as follows:

1) Sim_i invokes the simulator $\mathsf{Sim}_{\mathsf{MSBE},i}$ for P_i in protocol Π_{MSBE}. Sim_i simply appends the simulated view generated by $\mathsf{Sim}_{\mathsf{MSBE},i}$ as a part of its own simulation.
2) Sim_i invokes the simulator $\mathsf{Sim}_{\mathsf{B2S},i}$ for P_i in protocol Π_{B2S}. Sim_i simply appends the simulated view generated by $\mathsf{Sim}_{\mathsf{B2S},i}$ as a part of its own simulation.

The view generated by invoking $\mathsf{Sim}_{\mathsf{MSBE},i}$ and $\mathsf{Sim}_{\mathsf{B2S},i}$ is indistinguishable from the real-world execution, using similar arguments as shown previously in proving Theorem 1. □

D. Proof of Theorem 3

Proof: In protocol Π_{BAM}, the parties only needs to perform AND computation within each window followed by calling protocol Π_{B2S} for share conversion. The simulator Sim_i for $i \in \{0,1,2\}$ works as follows:

1) For each AND gate, Sim_i randomly samples a random value $r \xleftarrow{\$} \mathbb{Z}_2$ as the simulated view.
2) Sim_i invokes the simulator $\mathsf{Sim}_{\mathsf{B2S},i}$ for P_i in protocol Π_{B2S}. Sim_i simply appends the simulated view generated by $\mathsf{Sim}_{\mathsf{B2S},i}$ as a part of its own simulation.

In 3PC, each party holds two shares of a secret bit value. To compute AND between two shared x and y, P_i firstly computes a single share $z_i = x_i y_i + x_{i+1} y_i + x_i y_{i+1}$ (3-out-of-3 shares of $x \wedge y$) using local shares. To get 2-out-of-3 shares, P_i uses a PRF F to generate its own share α_i, where $\alpha_0 \oplus \alpha_1 \oplus \alpha_2 = 0$, and P_i sends $z_i' = z_i \oplus \alpha_i$ to P_{i+1}. Therefore, a random value $r \xleftarrow{\$} \mathbb{Z}_2$ suffices for the simulation, otherwise a distinguisher $\mathcal{D}$ can be constructed to distinguish F from a truly random function. The view generated by invoking $\mathsf{Sim}_{\mathsf{B2S},i}$ is indistinguishable from the real-world execution, using the similar argument as previously. Overall, the simulation is computationally indistinguishable from the real-world execution. □

Authorized licensed use limited to: NANJING UNIVERSITY OF SCIENCE AND TECHNOLOGY. Downloaded on December 27,2025 at 15:28:00 UTC from IEEE Xplore. Restrictions apply.

E. Proof of Theorem 4

Proof: Π_{BNAM} evaluates $\mathsf{BatchNorm{+}Activation{+}MP}$ together. In terms of $\mathsf{MSB}(x + \frac{\beta}{\gamma})$, $x + \frac{\beta}{\gamma}$ can be trivially simulated because it only involves local computation. For the view from Π_{MSBE} and Π_{B2S}, Sim_i just uses corresponding simulators for the two protocols to complete simulation. As regards to simulating AND gates, we take the same strategy done in proving Theorem 4. Overall, the simulation is computationally indistinguishable from the real-world execution. □

REFERENCES

[1] G. Litjens et al., "A survey on deep learning in medical image analysis," *Med. Image Anal.*, vol. 42, pp. 60–88, Dec. 2017.

[2] E. Angelini, G. Di Tollo, and A. Roli, "A neural network approach for credit risk evaluation," *Quart. Rev. Econ. Finance*, vol. 48, no. 4, pp. 733–755, Nov. 2008.

[3] N. S. Singh, S. Hariharan, and M. Gupta, "Facial recognition using deep learning," in *Advances in Data Sciences, Security and Applications*. Singapore: Springer, 2020, pp. 375–382.

[4] *The Health Insurance Portability and Accountability Act of 1996 (HIPAA)*. Accessed: 1996. [Online]. Available: https://www.hhs.gov/hipaa/index.html

[5] *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the Protection of Natural Persons With Regard to the Processing of Personal Data and on the Free Movement of Such Data, and Repealing Directive 95/46/EC (GDPR)*. Accessed: 2016. [Online]. Available: https://gdpr-info.eu/

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

[7] H. Chen, K. Laine, and R. Player, "Simple encrypted arithmetic library-SEAL v2.1," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.* Cham, Switzerland: Springer, 2017, pp. 3–18.

[8] A. C.-C. Yao, "How to generate and exchange secrets," in *Proc. 27th Annu. Symp. Found. Comput. Sci. (SFCS)*, Oct. 1986, pp. 162–167.

[9] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979.

[10] 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 *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 201–210.

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

[12] M. S. Riazi, M. Samragh, H. Chen, K. Laine, K. Lauter, and F. Koushanfar, "XONN: XNOR-based oblivious deep neural network inference," in *Proc. 28th USENIX Secur. Symp. (USENIX Secur.)*, 2019, pp. 1501–1518.

[13] N. Agrawal, A. Shahin Shamsabadi, M. J. Kusner, and A. Gascón, "QUOTIENT: Two-party secure neural network training and prediction," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2019, pp. 1231–1247.

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

[15] S. Wagh, S. Tople, F. Benhamouda, E. Kushilevitz, P. Mittal, and T. Rabin, "Falcon: Honest-majority maliciously secure framework for private deep learning," *Proc. Privacy Enhancing Technol.*, vol. 2021, no. 1, pp. 188–208, Jan. 2021.

[16] N. Koti, M. Pancholi, A. Patra, and A. Suresh, "SWIFT: Super-fast and robust privacy-preserving machine learning," in *Proc. 30th USENIX Secur. Symp. (USENIX Secur.)*, 2021, pp. 2651–2668.

[17] A. Ibarrondo, H. Chabanne, and M. Önen, "Banners: Binarized neural networks with replicated secret sharing," in *Proc. ACM Workshop Inf. Hiding Multimedia Secur.*, Jun. 2021, pp. 63–74.

[18] Y. Li and W. Xu, "PrivPy: General and scalable privacy-preserving data mining," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2019, pp. 1299–1307.

[19] M. Byali, H. Chaudhari, A. Patra, and A. Suresh, "FLASH: Fast and robust framework for privacy-preserving machine learning," *Proc. Privacy Enhancing Technol.*, vol. 2020, no. 2, pp. 459–480, Apr. 2020.

[20] A. Dalskov, D. Escudero, and M. Keller, "Fantastic four: Honest-majority four-party secure computation with malicious security," in *Proc. 30th USENIX Secur. Symp. (USENIX Secur.)*, 2021.

[21] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2017, pp. 19–38.

[22] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via MiniONN transformations," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 619–631.

[23] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *Proc. 27th USENIX Secur. Symp. (USENIX Secur.)*, 2018, pp. 1651–1669.

[24] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "Delphi: A cryptographic inference system for neural networks," in *Proc. Workshop Privacy-Preserving Mach. Learn. Pract.*, Nov. 2020, pp. 2505–2522.

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

[26] D. Rathee et al., "SiRnn: A math library for secure RNN inference," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2021, pp. 1003–1020.

[27] Z. Huang, W. J. Lu, C. Hong, and J. Ding, "Cheetah: Lean and fast secure two-party deep neural network inference," in *Proc. 31st USENIX Secur. Symp. (USENIX Secur.)*, 2022, pp. 1–19.

[28] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar, "Chameleon: A hybrid secure computation framework for machine learning applications," in *Proc. Asia Conf. Comput. Commun. Secur.*, May 2018, pp. 707–721.

[29] P. Mohassel and P. Rindal, "ABY^3: A mixed protocol framework for machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 35–52.

[30] A. Patra and A. Suresh, "BLAZE: Blazing fast privacy-preserving machine learning," in *Proc. Symp. Netw. Distrib. Syst. Secur. (NDSS)*, 2020, 1–28.

[31] H. Chaudhari, R. Rachuri, and A. Suresh, "Trident: Efficient 4PC framework for privacy preserving machine learning," in *Proc. Symp. Netw. Distrib. Syst. Secur. (NDSS)*, 2019, pp. 1–26.

[32] N. Koti, A. Patra, R. Rachuri, and A. Suresh, "Tetrad: Actively secure 4PC for secure training and inference," in *Proc. Symp. Netw. Distrib. Syst. Secur. (NDSS)*, 2021, pp. 1–27.

[33] A. Dalskov, D. Escudero, and M. Keller, "Secure evaluation of quantized neural networks," *Proc. Privacy Enhancing Technol.*, vol. 2020, no. 4, pp. 355–375, Oct. 2020.

[34] L. Braun, D. Demmler, T. Schneider, and O. Tkachenko, "MOTION—A framework for mixed-protocol multi-party computation," *ACM Trans. Privacy Secur.*, vol. 25, no. 2, pp. 1–35, May 2022.

[35] W. Tang, G. Hua, and L. Wang, "How to train a compact binary neural network with high accuracy?" in *Proc. 31st AAAI Conf. Artif. Intell.*, 2017, pp. 1–10.

[36] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2016, pp. 525–542.

[37] J. Zhang, Y. Pan, T. Yao, H. Zhao, and T. Mei, "DaBNN: A super fast inference framework for binary neural networks on ARM devices," in *Proc. 27th ACM Int. Conf. Multimedia*, Oct. 2019, pp. 2272–2275.

[38] M. Yayla and J.-J. Chen, "Memory-efficient training of binarized neural networks on the edge," in *Proc. 59th ACM/IEEE Design Autom. Conf.*, Jul. 2022, pp. 661–666.

[39] N. Fasfous, M.-R. Vemparala, A. Frickenstein, L. Frickenstein, M. Badawy, and W. Stechele, "Binarycop: Binary neural network-based covid-19 face-mask wear and positioning predictor on edge devices," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, Jun. 2021, pp. 108–115.

[40] V. Kolesnikov and R. Kumaresan, "Improved OT extension for transferring short secrets," in *Proc. Annu. Cryptol. Conf.* Cham, Switzerland: Springer, 2013, pp. 54–70.

Authorized licensed use limited to: NANJING UNIVERSITY OF SCIENCE AND TECHNOLOGY. Downloaded on December 27,2025 at 15:28:00 UTC from IEEE Xplore. Restrictions apply.

[41] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner, "More efficient oblivious transfer and extensions for faster secure computation," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2013, pp. 535–548.

[42] L. K. Ng and S. S. Chow, "GForce: GPU-friendly oblivious and rapid neural network inference," in *Proc. 30th USENIX Secur. Symp. (USENIX Secur.)*, 2021, pp. 2147–2164.

[43] S. Tan, B. Knott, Y. Tian, and D. J. Wu, "CryptGPU: Fast privacy-preserving machine learning on the GPU," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2021, pp. 1021–1038.

[44] T. Schneider and M. Zohner, "GMW vs. Yao? Efficient secure two-party computation with low depth circuits," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.* Cham, Switzerland: Springer, 2013, pp. 275–292.

[45] C. D. De Saint Guilhem, E. Makri, D. Rotaru, and T. Tanguy, "The return of eratosthenes: Secure generation of RSA moduli using distributed sieving," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2021, pp. 594–609.

[46] T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara, "High-throughput semi-honest secure three-party computation with an honest majority," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2016, pp. 805–817.

[47] D. Escudero, S. Ghosh, M. Keller, R. Rachuri, and P. Scholl, "Improved primitives for mpc over mixed arithmetic-binary circuits," in *Proc. Annu. Int. Cryptol. Conf.* Cham, Switzerland: Springer, 2020, pp. 823–852.

[48] E. Makri, D. Rotaru, F. Vercauteren, and S. Wagh, "Rabbit: Efficient comparison for secure multi-party computation," in *Proc. Int. Conf. Financial Cryptography Data Secur.* Cham, Switzerland: Springer, 2021, pp. 249–270.

[49] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *Proc. Annu. Int. Cryptol. Conf.* Cham, Switzerland: Springer, 1991, pp. 420–432.

[50] D. Rotaru and T. Wood, "Marbled circuits: Mixing arithmetic and Boolean circuits with active security," in *Proc. Int. Conf. Cryptol. India*. Cham, Switzerland: Springer, 2019, pp. 227–249.

[51] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Proc. 42nd IEEE Symp. Found. Comput. Sci.*, Jun. 2001, pp. 136–145.

[52] C. Hazay and Y. Lindell, *Efficient Secure Two-Party Protocols: Techniques and Constructions*. Heidelberg, Germany: Springer, 2010.

[53] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[54] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.

[55] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, vol. 25, Dec. 2012, pp. 1097–1105.

[56] W. Du and M. Atallah, "Privacy-preserving cooperative scientific computations," in *Proc. 14th IEEE Comput. Secur. Found. Workshop*. Washington, DC, USA: IEEE Computer Society, Jun. 2001, pp. 273–274.

[57] W. Du, Y. S. Han, and S. Chen, "Privacy-preserving multivariate statistical analysis: Linear regression and classification," in *Proc. SIAM Int. Conf. Data Mining*, Apr. 2004, pp. 222–233.

[58] A. P. Sanil, A. F. Karr, X. Lin, and J. P. Reiter, "Privacy preserving regression modelling via distributed computation," in *Proc. 10th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2004, pp. 677–682.

[59] A. B. Slavkovic, Y. Nardi, and M. M. Tibbits, "'Secure' logistic regression of horizontally and vertically partitioned distributed databases," in *Proc. 7th IEEE Int. Conf. Data Mining Workshops (ICDMW)*, Oct. 2007, pp. 723–728.

[60] Á. Kiss, M. Naderpour, J. Liu, N. Asokan, and T. Schneider, "SoK: Modular and efficient private decision tree evaluation," *Proc. Privacy Enhancing Technol.*, vol. 2019, no. 2, pp. 187–208, Apr. 2019.

[61] P. Bunn and R. Ostrovsky, "Secure two-party k-means clustering," in *Proc. 14th ACM Conf. Comput. Commun. Secur.*, Oct. 2007, pp. 486–497.

[62] G. Jagannathan and R. N. Wright, "Privacy-preserving distributed k-means clustering over arbitrarily partitioned data," in *Proc. 11th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2005, pp. 593–599.

[63] H. Yu, J. Vaidya, and X. Jiang, "Privacy-preserving SVM classification on vertically partitioned data," in *Proc. Pacific–Asia Conf. Knowl. Discovery Data Mining*. Cham, Switzerland: Springer, 2006, pp. 647–656.

[64] J. Vaidya, H. Yu, and X. Jiang, "Privacy-preserving SVM classification," *Knowl. Inf. Syst.*, vol. 14, no. 2, pp. 161–178, Feb. 2008.

[65] A. Barnett et al., "Image classification using non-linear support vector machines on encrypted data," *IACR Cryptol. ePrint Arch.*, Sep. 2017.

[66] S. Laur, H. Lipmaa, and T. Mielikäinen, "Cryptographically private support vector machines," in *Proc. 12th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2006, pp. 618–624.

[67] Y. Rahulamathavan, R. C.-W. Phan, S. Veluru, K. Cumanan, and M. Rajarajan, "Privacy-preserving multi-class support vector machine for outsourcing the data classification in cloud," *IEEE Trans. Dependable Secure Comput.*, vol. 11, no. 5, pp. 467–479, Sep. 2014.

[68] Y. Lindell and B. Pinkas, "Privacy preserving data mining," in *Proc. Annu. Int. Cryptol. Conf.* Cham, Switzerland: Springer, 2000, pp. 36–54.

[69] L. J. M. Aslett, P. M. Esperança, and C. C. Holmes, "Encrypted statistical machine learning: New privacy preserving methods," 2015, *arXiv:1508.06845*.

[70] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser, "Machine learning classification over encrypted data," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2015, pp. 1–34.

[71] T. Graepel, K. Lauter, and M. Naehrig, "ML confidential: Machine learning on encrypted data," in *Proc. Int. Conf. Inf. Secur. Cryptol.* Cham, Switzerland: Springer, 2012, pp. 1–21.

[72] B. D. Rouhani, M. S. Riazi, and F. Koushanfar, "DeepSecure: Scalable provably-secure deep learning," in *Proc. 55th ACM/ESDA/IEEE Design Autom. Conf. (DAC)*, Jun. 2018, pp. 1–6.

[73] E. Makri, D. Rotaru, N. P. Smart, and F. Vercauteren, "EPIC: Efficient private image classification (or: Learning from the masters)," in *Proc. Cryptographers Track RSA Conf.* Cham, Switzerland: Springer, 2019, pp. 473–492.

[74] N. Chandran, D. Gupta, A. Rastogi, R. Sharma, and S. Tripathi, "EzPC: Programmable and efficient secure two-party computation for machine learning," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P)*, Jun. 2019, pp. 496–511.

[75] D. Demmler, T. Schneider, and M. Zohner, "ABY—A framework for efficient mixed-protocol secure two-party computation," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2015, pp. 1–15.

[76] N. Kumar, M. Rathee, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "CrypTFlow: Secure TensorFlow inference," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2020, pp. 336–353.

[77] A. Dalskov, D. Escudero, and A. Nof, "Fast fully secure multi-party computation over any ring with two-thirds honest majority," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2022, pp. 653–666.

[78] R. Cramer, I. Damgård, D. Escudero, P. Scholl, and C. Xing, "SPD$\mathbb{Z}_{2^k}$: Efficient MPC mod 2^k for dishonest majority," in *Proc. Annu. Int. Cryptol. Conf.* Cham, Switzerland: Springer, 2018, pp. 769–798.

[79] I. Damgård, D. Escudero, T. Frederiksen, M. Keller, P. Scholl, and N. Volgushev, "New primitives for actively-secure MPC over rings with applications to private machine learning," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2019, pp. 1102–1120.

[80] M. Keller, "MP-SPDZ: A versatile framework for multi-party computation," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2020, pp. 1575–1590.

[81] Z. Chen, F. Zhang, A. C. Zhou, J. Zhai, C. Zhang, and X. Du, "ParSecureML: An efficient parallel secure machine learning framework on GPUs," in *Proc. 49th Int. Conf. Parallel Process. (ICPP)*, Aug. 2020, pp. 1–11.

[82] J.-L. Watson, S. Wagh, and R. A. Popa, "Piranha: A GPU platform for secure computation," in *Proc. 31st USENIX Secur. Symp. (USENIX Secur.)*, 2022, pp. 1–19.

[83] M. Barni, C. Orlandi, and A. Piva, "A privacy-preserving protocol for neural-network-based computation," in *Proc. 8th Workshop Multimedia Secur.*, Sep. 2006, pp. 146–151.

Authorized licensed use limited to: NANJING UNIVERSITY OF SCIENCE AND TECHNOLOGY. Downloaded on December 27,2025 at 15:28:00 UTC from IEEE Xplore. Restrictions apply.

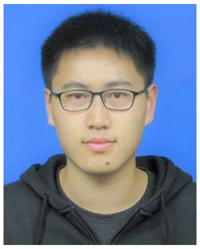
[84] H. Chabanne, A. De Wargny, J. Milgram, C. Morel, and E. Prouff, "Privacy-preserving classification on deep neural network," *IACR Cryptol. ePrint Arch.*, Mar. 2017.

[85] M. Chase, R. Gilad-Bachrach, K. Laine, K. Lauter, and P. Rindal, "Private collaborative neural network learning," *IACR Cryptol. ePrint Arch.*, Aug. 2017. [Online]. Available: https://eprint.iacr.org/2017/762

[86] D. Rathee, A. Bhattacharya, R. Sharma, D. Gupta, N. Chandran, and A. Rastogi, "SecFloat: Accurate floating-point meets secure 2-party computation," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2022, pp. 576–595.

Ye Dong was born in 1995. He received the B.S. degree in computer science and technology from Shandong University in 2018. He is currently pursuing the Ph.D. degree with the School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China. His research interests include practical secure computation and privacy-preserving machine learning.

Xiaojun Chen was born in 1979. He received the Ph.D. degree. He is currently a Professorate Senior Engineer. His research interests include big data, privacy preservation, and data security. He is a member of CCF.



Xiangfu Song received the Ph.D. degree from Shandong University. He is currently a Research Fellow with the National University of Singapore. His research interests include applied cryptography and practical secure computation.

Kaiyun Li was born in 1997. He received the master's degree from the University of Chinese Academy of Sciences. His research interests include cryptography techniques in data mining.

Authorized licensed use limited to: NANJING UNIVERSITY OF SCIENCE AND TECHNOLOGY. Downloaded on December 27,2025 at 15:28:00 UTC from IEEE Xplore. Restrictions apply.