SecBNN: Efficient Secure Inference on Binary Neural Networks

Hanxiao Chen, *Student Member, IEEE*, Hongwei Li, *Fellow, IEEE*, Meng Hao, *Student Member, IEEE*, Jia Hu, *Graduate Student Member, IEEE*, Guowen Xu, *Member, IEEE*, Xilin Zhang, *Member, IEEE*, and Tianwei Zhang, *Member, IEEE*

***Abstract*— This work studies secure inference on Binary Neural Networks (BNNs), which have binary weights and activations as a desirable feature. Although previous works have developed secure methodologies for BNNs, they still have performance limitations and significant gaps in efficiency when applied in practice. We present SecBNN, an efficient secure two-party inference framework on BNNs. SecBNN exploits appropriate underlying primitives and contributes efficient protocols for the non-linear and linear layers of BNNs. Specifically, for non-linear layers, we introduce a secure sign protocol with an innovative adder logic and customized evaluation algorithms. For linear layers, we propose a new binary matrix multiplication protocol, where a divide-and-conquer strategy is provided to recursively break down the matrix multiplication problem into multiple sub-problems. Building on top of these efficient ingredients, we implement and evaluate SecBNN over two real-world datasets and various model architectures under LAN and WAN. Experimental results show that SecBNN substantially improves the communication and computation performance of existing secure BNN inference works by up to 29× and 14×, respectively.**

***Index Terms*— Binary neural networks, private inference, secure two-party computation.**

I. Introduction

DEEP learning (DL) [1] has been widely adopted in real-life applications. However, it is challenging to deploy large-weight DL models in computation- and energy-constrained devices (e.g., smartphones, wearables, and edge IoT devices). For example, AlexNet [2] contains 60 million parameters and 650,000 neurons, which is computationally expensive and consumes extensive memory resources for inference. Fortunately, a game-changing technique, *Binary Neural Network* (BNN) [3], has been introduced as one of the most efficient solutions for such devices. BNN is an extreme case of parameter quantization, where weights and activations are restricted to ± 1. Such binary characteristics have led to remarkable progress in applying BNNs on resource-limited platforms, since they can save significant amounts of storage, computation, and energy consumption [4], [5]. With the increasing trend of lightweight and practical neural networks, more and more research efforts are devoted to BNNs in academia [6], [7], [8], [9], [10]. Meanwhile, due to their superior performance, BNNs have been practically employed in various applications like person re-identification [11], robotics [12], wearable devices [13], and self-driving car [14].

Similar to other DL techniques, BNNs suffer from privacy issues when applied in remote inference services with the client-server architecture [15], [16]. In particular, a server hosting a BNN model provides inference APIs to the clients who can send their queries to the server and receive the corresponding analysis results. In this process, the queries may be highly sensitive and the server can easily compromise the privacy of clients (i.e., revealing the queries). An attractive option to tackle this issue is the *secure inference* based on secure two-party computation (2PC) techniques [17], [18], [19], [20], [21], [22], [23], [24], [25], which can provide provable security guarantees. At a high level, secure inference considers a real-world scenario, where the *server* holds the proprietary weights w of a model M, and the *client* pays to learn the prediction $M(w, x)$ on a privacy-sensitive sample x. 2PC protocols are applied to guarantee that the server learns nothing about x, while the client learns nothing about w beyond $M(w, x)$ and what can be deduced from the result.

Two lines of research work can achieve secure BNN inference. Unfortunately, they suffer from noteworthy efficiency issues. In particular, (1) the advanced general 2PC-based systems [23], [24], [25] can be directly applied to the BNN inference task. However, they are not designed for the binary properties of BNNs, and thus exhibit large performance bottlenecks. (2) Researchers also design new solutions dedicated to BNN inference. As the first attempt, Riazi et al. [15] propose a BNN private inference framework XONN, which purely exploits Yao,s Garbled Circuit (GC) techniques [26] to implement both linear and non-linear layers. It provides a modular approach for integrating 2PC protocols with BNNs, achieving a constant-round secure inference scheme. However, as shown in recent studies [23], [25], GC has expensive communication costs due to transmitting garbled tables, and high computation overhead introduced by invoking encryption operations per

Received 10 April 2024; revised 14 August 2024; accepted 6 September 2024. Date of current version 13 November 2024. This work was supported in part by the National Key Research and Development Program of China under Grant 2022YFB3103500, in part by the National Natural Science Foundation of China under Grant 62020106013, in part by Sichuan Science and Technology Program under Grant 2023ZYD0142, in part by Chengdu Science and Technology Program under Grant 2023-XT00-00002-GX, in part by the Fundamental Research Funds for Chinese Central Universities under Grant ZYGX2020ZB027 and Grant Y030232063003002, and in part by the Post-Doctoral Innovation Talents Support Program under Grant BX20240053. The associate editor coordinating the review of this article and approving it for publication was Dr. Paolo Gasti. *(Corresponding author: Hongwei Li.)*

Hanxiao Chen, Hongwei Li, Meng Hao, Jia Hu, Guowen Xu, and Xilin Zhang are with the School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 610056, China (e-mail: chenhanxiao.chx@gmail.com; hongweili@uestc.edu.cn; menghao303@gmail.com; jiahu@std.uestc.edu.cn; guowen.xu@foxmail.com; xilinzhang@uestc.edu.cn).

Tianwei Zhang is with the College of Computing and Data Science, Nanyang Technological University, Singapore 639798 (e-mail: tianwei.zhang@ntu.edu.sg).

Digital Object Identifier 10.1109/TIFS.2024.3484936

gate. Thus, it becomes the principal performance bottleneck. Recently, Samragh et al. [16] designed an optimized variant (denoted as XONN+ in this paper), where the non-linear protocols are still constructed using expensive GC, but the evaluation of linear functions has been optimized utilizing Oblivious Transfer (OT) [27]. Nevertheless, the communication overhead of OT-based linear layers is still heavy.

This paper presents SecBNN, an efficient secure two-party inference framework tailored for BNNs. The core novelty of SecBNN lies in its ability to exploit the binary characteristics of BNNs to design cryptographic protocols specifically crafted for each layer within the model. It provides the following new insights. (1) **A new sign protocol for non-linear layers**. Sign is an important activation function in BNNs, to binarize the output of linear layers. We propose a new adder logic and effective evaluation algorithms to evaluate this function. This design minimizes the required communication rounds and AND gates. (2) **An efficient binary matrix multiplication protocol for hidden linear layers**. In secure BNN inference, matrix multiplication occupies the major overhead. To solve this issue, we first propose a new protocol based on correlated OT (COT) [28], and further design a divide-and-conquer strategy to recursively break down the matrix multiplication problem into multiple sub-problems over a smaller size, each of which can be evaluated more efficiently. (3) **A new binary-integer multiplication protocol for the first linear layer**. This protocol is customized for the first linear layer of BNNs, where the weights are binary but the inference samples are integers. We implement it with a new COT-based method for better communication overhead.

To evaluate the designed protocols, we apply SecBNN to representative benchmarks (e.g., MNIST, and CIFAR10). Compared to secure BNN inference works [15], [16], SecBNN significantly reduces the communication (i.e., $3.33 \sim 29.62\times$) and computation (i.e., $3.20 \sim 14.29\times$) cost. In sum, we make the following contributions.

- We provide a novel paradigm with a new adder logic and efficient protocols for evaluating the sign function in non-linear layers of BNNs.
- We design an efficient binary matrix multiplication protocol for linear layers with a new divide-and-conquer strategy and customized constructions.
- The experimental results show that SecBNN achieves up to $29\times$ and $14\times$ communication and computation improvements over prior works, respectively.

We begin with the technical contributions of SecBNN in Section II, and detail the useful preliminaries in Section III. In Section IV, we provide the threat model and high-level overview of SecBNN. Sections V and VI present our non-linear and linear protocols, respectively. Section VII reports the experimental results. In Section VIII, we discuss the related works, and then conclude this work in Section IX.

II. Technical Contributions

A BNN consists of alternating non-linear and linear layers. The former contains sign activation and maxpooling functions, while the latter can be divided into the first linear layer and hidden linear layers according to different input forms. To achieve secure BNN inference efficiently, SecBNN takes advantage of the binary characteristic of BNNs, i.e., the weights and activations are restricted as ± 1,[1] and provides new insights in designing protocols for each layer.

A. Non-Linear Layers

The non-linear activation function in BNNs can be represented as a sign protocol, where the server and the client hold the secret shares of $x \in \mathbb{Z}_{2^\ell}$, and aim to securely compute $y = \mathsf{sign}(x)$,[2] thereby obtaining the secret shares of y. Given the boolean mapping, the sign function can be reformulated as $y = \mathsf{MSB}(x) \oplus 1$, and hence the problem is converted into securely computing MSB.[3] Besides, the other non-linear layer, i.e., maxpooling, can be evaluated using the simple idea in [15], which causes a slight overhead in our evaluation. Hence, we omit the detail here and illustrate it in Section V.

Three strategies are currently available for securely evaluating MSB. The first one is based on GC, which was employed in prior secure BNN inference works [15], [16]. Nevertheless, as discussed in Section I, GC has heavy communication and computation overhead [23]. The second one is to leverage OT, which was proposed in the state-of-the-art 2PC-based inference system, Cheetah [25]. However, we observe that this solution requires multiple calls to the costly 1-out-of-2^m OT primitive (m is a hyperparameter). Another one is to employ an advanced adder like the parallel prefix adder (PPA) [29], [30]. Such adders consist of two components: input computation and circuit evaluation, both requiring AND and cost-free XOR operations. Unfortunately, directly applying this construction cannot achieve better performance than Cheetah [25], due to the large number of AND gates and communication rounds. As a result, these approaches suffer from notable performance bottlenecks.

In SecBNN, we re-enable the adder-based method for MSB, because it only requires secure evaluation of AND gates, which is more friendly than GC and OT. Compared to the advanced adder, i.e., PPA, our method employs a more streamlined adder circuit and more efficient evaluation protocols. Technically, (1) we design a novel circuit logic that integrates the input computation and circuit evaluation processes together, and reduce the number of AND gates based on a lean tree-based evaluation. Compared to the counterpart based on PPA, our protocol reduces both communication and interaction rounds. (2) For the protocol design, we identify two types of AND gates with different input forms that appear in the circuit. To improve communication performance, we construct customized protocols for each type of AND gates. In contrast, the PPA-based method uses a general protocol for all AND evaluations, without capturing this feature. Overall, our solution offers superior efficiency over the GC-based counterparts [15], [16] and the state-of-the-art OT-based method [25]. Table I shows the theoretical results.

B. Hidden Linear Layers

For the evaluation of hidden linear layers, a binary matrix multiplication protocol is needed, which takes as input two

[1] Similar as prior secure BNN inference works [15], [16], in SecBNN, the binary weights and activations are represented in the boolean form using the mappings: $+1 \rightarrow 1$ and $-1 \rightarrow 0$.

[2] $\mathsf{sign}(x)$ is +1 if $x \geq 0$ and -1 otherwise.

[3] $\mathsf{MSB}(x)$ is the most significant bit of x.

TABLE I

COMPARISON OF COMMUNICATION COST WITH PRIOR WORKS FOR NON-LINEAR PROTOCOLS. FOR THE CONCRETE EXAMPLE, WE USE THE SECURITY PARAMETER $\lambda = 128$

Operation	Protocol	Communication	Round
Sign activation on $\{0,1\}^\ell$	GC [15]	$2\lambda\ell$	2
	GC [16]	$5\lambda\ell$	2
	GMW [31]	$\approx 6\lambda\ell$	$\log\ell + 3$
	OT [25]	$11(\ell-1)$	$\log\ell$
	SecBNN	$10\ell - 4\log\ell - 14$	$\log\ell$
Sign activation $\ell = 8$	GC [15]	2048 bits	2
	GC [16]	5120 bits	2
	GMW [31]	$\approx$ 6144 bits	6
	OT [25]	77 bits	3
	SecBNN	54 bits	3
Maxpooling on $\{0,1\}^{n\times n}$	GC [15], [16]	$2\lambda(n^2-1)$	2
	SecBNN	$4(n^2-1)$	$\log n^2$
Maxpooling $n = 2$	GC [15], [16]	768 bits	2
	SecBNN	12 bits	2

TABLE II

COMPARISON OF COMMUNICATION COST WITH PRIOR WORKS FOR LINEAR PROTOCOLS, WHERE THE INPUTS ARE $(m \times n)$-DIM $\mathbf{X}$ AND $(n \times k)$-DIM $\mathbf{Y}$. FOR CLARITY, WE DEFINE $\bar{\ell} = \lceil \log n \rceil + 8$ AND $\ell = \lceil \log(N+1) \rceil + 1$. FOR THE CONCRETE EXAMPLE, WE USE THE SECURITY PARAMETER $\lambda = 128$. * DENOTES OUR BINARY LINEAR PROTOCOL USING THE DIVIDE-AND-CONQUER OPTIMIZATION

Operation	Protocol	Communication	Round
Hidden linear layer	GC [15]	$\approx 5\lambda mnk$	2
	OT [16]	$mn(2k\ell + \lambda)$	2
	SecBNN	$nk(1 + m\ell)$	2
	SecBNN*	Equation 7	5
Hidden linear layer $196 \times 144 \times 32$	GC [15]	$\approx$ 68.90 MB	2
	OT [16]	2.37 MB	2
	SecBNN	0.97 MB	2
	SecBNN*	0.56 MB	5
First linear layer	OT [15], [16]	$mn(2k\bar{\ell} + \lambda)$	2
	SecBNN	$mn(1 + k\bar{\ell})$	2
First linear layer $64 \times 27 \times 1024$	OT [15], [16]	2.55 MB	2
	SecBNN	1.26 MB	2

binary matrices $\mathbf{X} \in \{-1,1\}^{m\times n}$ and $\mathbf{Y} \in \{-1,1\}^{n\times k}$ owned by the server and the client respectively, and outputs the secret shares of $\mathbf{Z} = \mathbf{X} \cdot \mathbf{Y}$. As illustrated in XONN [15], considering the boolean mapping of the inputs, this function can be computed via cost-free XNOR along with a sum operation over boolean shares. Thus, it is necessary to develop a secure protocol to efficiently compute sum.

This function occupies the major overhead in secure BNN inference. Existing works [15], [16] utilize GC or OT to evaluate it. However, these methods are costly in both communication and computation. In SecBNN, we attempt to exploit the advanced boolean-to-arithmetic (B2A) protocol [23], [25] to evaluate this function. The main insights are that the sum operation on arithmetic shares is free, and the B2A protocol can be efficiently built based on COT [28]. For ease of understanding, we talk about a simple vector case with size n, i.e., computing the arithmetic shares of $y = \sum_{i\in[n]} x_i$, where $x_i \in \{0,1\}$ is boolean-shared between the server and the client. Specifically, we first convert the boolean shares of each x_i to ℓ-bit arithmetic shares, where $\ell = \lceil \log(n+1) \rceil + 1$ to prevent overflow [16], and then sum the generated arithmetic shares for free. Furthermore, when extended to the matrix form, we observe that by carefully setting up the receiver's and sender's messages, multiple COT instances can be combined to reduce the communication cost. As a result, this solution obtains better communication and runtime performance than the prior protocols [15], [16].

Despite achieving overall advantages in different network environments (i.e., LAN and WAN) as shown in Section VII, the above sum protocol is sensitive to the network bandwidth. The primary reason is attributed to the $O(n\ell)$ asymptotic communication complexity of this protocol, which leads to an increase in runtime on the communication-limited network environment. Nevertheless, optimizing this overhead may be hard, since any reduction in n or ℓ will affect the correctness with a non-negligible probability. Rather, we propose a divide-and-conquer strategy, which reduces the communication cost by slightly increasing the communication round. The main insight is to sum on each n'-dimension ($n' < n$) sub-vector and hence to execute B2A on a smaller bitlength $\ell' = \lceil \log(n'+1) \rceil + 1$. After that, we obtain $\lceil \frac{n}{n'} \rceil$ summations, which are summed by first calling our optimized ℓ'-to-ℓ bitlength extension protocol. Totally, our communication complexity is $O(n\ell' + \frac{n}{n'}(\ell - \ell'))$. By picking n' appropriately, we save nearly half the communication compared to our basic solution. Note that SecBNN adaptively chooses the appropriate protocol with or without this optimization regarding the network environment, thereby carefully balancing the round and communication complexity for better performance. Table II shows the theoretical communication overhead of our protocol.

C. First Linear Layer

For the first linear layer of BNNs, where the inference samples are not necessarily binarized, a variant of the above matrix multiplication protocol for $\mathbf{Z} = \mathbf{X} \cdot \mathbf{Y}$ is required. In this function, $\mathbf{X} \in \{-1,1\}^{m\times n}$ is also a binary weight matrix owned by the server but $\mathbf{Y} \in \mathbb{Z}_{2^\ell}^{n\times k}$ as an inference sample is an *integer* matrix owned by the client.

The source of the inefficiency of prior solutions [15], [16] is that the usage of OT results in high communication costs.[4] To achieve a communication-efficient protocol, we aim to build a new COT-based evaluation by constructing a correlation function between the sender's messages. Specifically, we consider an instance that the server and client jointly compute a scalar multiplication $x \cdot y$, where $x \in \{-1,1\}$ and $y \in \mathbb{Z}_{2^\ell}$. To evaluate it, the client first constructs a correlation function $f(\alpha) = 2y + \alpha$. Then, the two parties call a 1-out-of-2 COT, where the client is the sender with the input $f(\cdot)$ and the server is the receiver with the choice

[4] XONN+ [16] proposed an offline/online protocol for the amortized setting. In the offline phase, the client and the server engage in a Random OT (ROT) protocol assuming the model weights are known in advance and are served as the choice bits. During the online phase, the sender only sends two masked messages. It is not hard to figure out that the overall communication overhead of this protocol equals the OT-based method in XONN [15].

bit b that is 0 if $x = -1$ and 1 otherwise. After that, they obtain the shares of $x \cdot y$ if the client sets the share as $[x \cdot y]_0 = -\alpha - y$ and the server sets the share as the output of COT, i.e., $[x \cdot y]_1 = b \cdot 2y + \alpha$. Note that the above design and function construction are essentially different from existing multiplexer [32] or boolean-integer multiplication [33] protocols, since our binary value is from $\{+1, -1\}$ rather than $\{0, 1\}$. When extending this idea to the matrix form, the *batching* technique [18], [33] can be used to further reduce the computation and communication complexities. Table II shows the detailed communication complexity analysis.

III. Preliminaries

A. Notations

Let λ be the security parameter. $[x]^\ell$ indicates that x is arithmetic shared in $\mathbb{Z}_{2^\ell}$ between the server and the client, while $[x]^B$ denotes the boolean shares in $\mathbb{Z}_2$. $[n]$ denotes the set $\{1, 2, \cdots, n\}$. $\wedge$ and $\oplus$ represent logical AND and XOR operations, respectively. $\lceil \cdot \rceil$ and $\lfloor \cdot \rfloor$ denote the ceiling and flooring functions, respectively. We use bold lower-case letters (e.g., $\mathbf{x}$) to represent vectors, and bold upper-case letters (e.g., $\mathbf{X}$) to represent matrices. $\mathbf{X}[i, \cdot]$ and $\mathbf{X}[\cdot, j]$ denote the i-th row and j-th column of $\mathbf{X}$, respectively. Given two matrices $\mathbf{X}$ and $\mathbf{Y}$, $\mathbf{X} \circ \mathbf{Y}$ refers to the Hadamard product (i.e., element-wise product). $\boxtimes$ indicates the vector-matrix multiplication without accumulation: given an m-length vector $\mathbf{x}$ and $\mathbf{Y}$ with size $m \times n$, $\mathbf{x} \boxtimes \mathbf{Y}$ outputs $\mathbf{Z}$ with size $m \times n$. $\boxplus$ denotes the matrix-vector XOR: given $\mathbf{Y}$ with size $m \times n$ and an n-length vector $\mathbf{x}$, $\mathbf{Y} \boxplus \mathbf{x}$ outputs $\mathbf{Z}$ with size $m \times n$. Besides, $\mathbf{X}[j, \cdot] = r$ means that each element in $\mathbf{X}[j, \cdot]$ equals r.

1) Fixed-Point Representation: Similar to existing secure inference works [15], [16], [25], we encode a real number, i.e., a float-point number, $\hat{x} \in \mathbb{R}$ as a field element $x \in \mathbb{F}_p$ using its fixed-point representation. Given the fractional bitlength s (also called *scale*), the mapping from reals to their field representation is $x = \lfloor \hat{x} \cdot 2^s \rfloor \bmod p$, and vice versa, the mapping from the field representation to reals is $(x - c \cdot p)/2^s$ where $c = 1\{x > (p-1)/2\}$.

B. Binary Neural Networks

Similar to traditional neural networks, a BNN comprises a sequence of linear and non-linear layers, except that the weights and activations of these layers are restricted to ± 1. Below, we briefly describe the functionalities of these layers.

1) Binary Linear Layers: A fully-connected layer (FC) takes as input an n-length activation vector $\mathbf{x} \in \{+1, -1\}^n$ and a weight matrix $\mathbf{W} \in \{+1, -1\}^{m \times n}$, and outputs $\mathbf{y} \in \mathbb{R}^m$ using a linear transformation $\mathbf{y} = \mathbf{W} \cdot \mathbf{x}$. The convolutional layer (CONV) is another form of linear transformation, which can be represented with matrix multiplication [22], [23].

2) Batch Normalization Layer: A batch normalization layer (BN) is typically applied to the output of linear layers to normalize the results. For the output $\mathbf{y}$ of FC, BN multiplies the i-th element of $\mathbf{y}$ by $\gamma[i]$ and adds $\beta[i]$ to the result, where $\gamma \in \mathbb{R}^m$ is the scaling vector and $\beta \in \mathbb{R}^m$ is the shift vector. For the output of CONV, BN multiplies all of the i-th channel's elements by a scalar $\gamma[i]$ and adds $\beta[i]$ to the resulting channel.

3) Sign Activation Layer: The output of BN is usually fed to a binary activation function. It takes as input $\mathbf{x} \in \mathbb{R}^n$ and maps it to $\mathbf{y} = \mathsf{sign}(\mathbf{x}) \in \{+1, -1\}^n$, where sign outputs either $+1$ or -1 based on the sign of $\mathbf{x}$.

4) Binary MAXPOOLING LAYER: Pooling layers operate on image channels outputted by CONV, which slides a window on the channels and aggregates the values into a single output. Maxpooling and averagepooling are two of the most common pooling operations. However, averagepooling is usually not used in BNNs since the average of multiple binary values is no longer binary.

C. Cryptographic Primitives

1) Secret Sharing: We adopt 2-out-of-2 secret sharing schemes over different power-of-2 rings [34], categorized into boolean sharing and arithmetic sharing. In the boolean sharing, the sharing algorithm $\mathsf{Shr}^{\mathsf{B}}(x)$ inputs x in $\mathbb{Z}_2$ and outputs two shares $[x]_0^B$ and $[x]_1^B$ satisfying $[x]_0^B \oplus [x]_1^B = x$ in $\mathbb{Z}_2$. The reconstruction algorithm $\mathsf{Rec}^{\mathsf{B}}([x]_0^B, [x]_1^B)$ takes as input the two shares and outputs x. Given two boolean-shared values, the XOR operation can be non-interactively evaluated, and AND requires invoking OT primitive. Besides, we denote the arithmetic shares of $x \in \mathbb{Z}_{2^\ell}$ by $[x]^\ell = ([x]_0^\ell, [x]_1^\ell)$ with $x = [x]_0^\ell + [x]_1^\ell \bmod 2^\ell$. The operations on arithmetic shares are similar to those of boolean shares, except that XOR and AND are replaced by addition and multiplication, respectively. We denote the arithmetic sharing and reconstruction algorithms as $\mathsf{Shr}^{\mathsf{A}}(\cdot)$ and $\mathsf{Rec}^{\mathsf{A}}(\cdot)$, respectively.

2) Oblivious Transfer: In the 1-out-of-2 oblivious transfer (OT) protocol [27], a *sender* inputs two ℓ-bit messages m_0, $m_1 \in \{0, 1\}^\ell$ and a *receiver* inputs a choice bit $b \in \{0, 1\}$. At the end of the protocol, the receiver obtains m_b and the sender receives nothing. SecBNN relies on two variants of OT [35], i.e., 1-out-of-2 correlated OT ($\binom{2}{1}$-COT_ℓ) and 1-out-of-2 random OT ($\binom{2}{1}$-ROT_ℓ). In the $\binom{2}{1}$-COT_ℓ, the sender inputs a correlation function $f(r) = x + r$, and the receiver inputs a choice bit b. After the protocol, the sender receives a random $r \in \{0, 1\}^\ell$ and the receiver obtains m_b, where $m_0 = r$ and $m_1 = x + r$. In the $\binom{2}{1}$-ROT_ℓ, the sender has no inputs and obtains random $m_0, m_1 \in \{0, 1\}^\ell$, while the receiver obtains m_b. The above OT primitives are usually realized with OT extension techniques (typically, IKNP [35] and silent OT extension [28], [36]). Like [25], we use the low-communication silent OT extension [28] as the underlying building block, where $\binom{2}{1}$-ROT_ℓ enjoys almost 0 amortized communication cost,[5] and $\binom{2}{1}$-COT_ℓ communicates $\ell + 1$ bits within 2 rounds.

D. 2PC Functionalities

SecBNN adopts the following 2-party functionalities, with detailed protocols outlined in Appendix A.

1) AND: The functionality $\mathcal{F}_{\mathsf{AND}}$ takes $[x]^B$ and $[y]^B$ as input and returns $[x \wedge y]^B$. Cheetah [25] gives the most communication-efficient protocol for $\mathcal{F}_{\mathsf{AND}}$ with two calls of silent $\binom{2}{1}$-ROT_1. This protocol requires 4 bits of communication within 2 rounds.

[5] Similar as the advanced Cheetah [25], the complexity analysis in SecBNN ignores the communication and round overhead introduced by $\binom{2}{1}$-ROT_ℓ.

2) Boolean to Arithmetic (B2A)*:* The ℓ-bit B2A functionality, $\mathcal{F}^{\ell}_{\mathsf{B2A}}$, takes $[x]^B$ as input and returns the arithmetic shares of the same value x, i.e., $[x]^{\ell}$. We use the silent $\binom{2}{1}$-COT$_{\ell}$-based protocol [25], which requires $\ell+1$ bits of communication within 2 rounds.

3) Bitlength Extension With Known MSB (SExt)*:* The functionality, $\mathcal{F}^{\ell',\ell}_{\mathsf{SExt}}$, takes ℓ'-bit shares $[x]^{\ell'}$ with *positive* x as input and returns ℓ-bit shares of the same value, i.e., $[x]^{\ell}$. We instantiate the protocol of [32] using the silent OT extension, which reduces the communication cost from $\ell-\ell'+2\lambda+2$ to $\ell-\ell'+3$ bits.

IV. Secure BNN Inference

This section details the threat model and the high-level overview of SecBNN.

A. Threat Model

The security of SecBNN is provably provided against a semi-honest probabilistic polynomial time adversary $\mathcal{A}$ that either corrupts the client or the server but not both. In this setting, $\mathcal{A}$ strictly follows the specification of designed protocols, but attempts to infer more private information from the received messages. Security is modeled in the simulation paradigm [37], [38], which defines a *real* interaction and an *ideal* interaction. In the real interaction, the parties execute protocols according to the specification in the presence of $\mathcal{A}$ and the environment $\mathcal{Z}$. In the ideal interaction, the parties send their inputs to an ideal functionality that faithfully executes the operation. Secure inference requires that no environment can computationally distinguish between real and ideal interactions. The protocols in SecBNN invoke multiple sub-protocols, and we use the *hybrid model* to describe them like prior works [23], [32]. This is analogous to the real interaction, except that sub-protocols are replaced by the corresponding ideal functionalities. By convention, a protocol invoking a functionality $\mathcal{F}$ is referred to as the "$\mathcal{F}$-hybrid model".

Remark: As with all semi-honest secure protocols [15], [16], [23], [25], our SecBNN is not designed to defend against active adversaries or side-channel attacks. In the following, we discuss possible solutions to deal with these malicious behaviors. (1) Active adversaries: For adversaries who might deviate from protocol specifications to gain unauthorized information, a possible solution is to utilize the zero-knowledge proof techniques [39], [40], combined with specific consistency check strategies, to ensure data integrity [41], [42]. However, even with the most advanced methods, the protocol overhead will increase by several orders of magnitude. (2) Side-channel attacks: To our knowledge, no existing 2PC-based secure inference protocols currently address the impact of side-channel attacks. These attacks exploit secret-dependent side information generated during the execution of cryptographic protocols to perform secret recovery attacks. A potential solution could involve introducing random dummy operations or masks into the side information, thereby weakening the direct correlation between the side information and the secret. This might be a valuable direction for future research.

B. Overview of SecBNN

SecBNN provides an end-to-end private inference service via securely evaluating each layer of BNNs using efficient protocols, as shown in Figure 1. Recall that a BNN simply contains a sequence of connected layers of appropriate dimensions. When securely realizing each layer, SecBNN maintains the following invariant[6]: the client and the server begin with boolean/arithmetic shares of the input to the layer, and end with boolean/arithmetic shares of the output of the layer after the protocol. This allows us to stitch protocols for proper layers sequentially to obtain a complete secure inference scheme. The semi-honest security will follow trivially from the sequential composability of individual sub-protocols [38], [43].

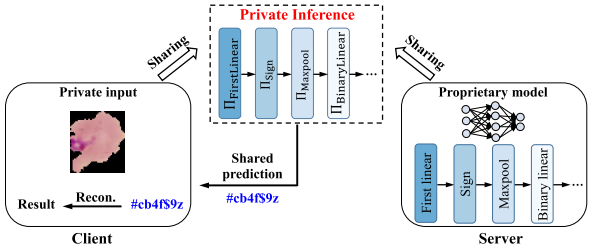


Fig. 1. Overview of secure BNN inference in SecBNN.

V. Protocols for Non-Linear Layers

This section presents efficient protocols for the non-linear layers of BNNs including (1) sign activation layers and (2) binary maxpooling layers. The corresponding functionalities are shown in Table III.

A. Sign Activation Protocol

The sign activation function computes $y=\mathsf{sign}(x)$ for each input x, and outputs $+1$ if $x\geq 0$ and -1 otherwise. With the boolean encoding of the output, this function can be simplified to extract the most significant bit (MSB) of x, i.e.,

$$y=\mathsf{MSB}(x)\oplus 1. \tag{1}$$

Formally, the MSB protocol takes as input ℓ-bit shares $[x]^{\ell}$ and outputs $[\mathsf{MSB}(x)]^B$. For efficient MSB evaluation, in the following, we represent it as a boolean adder, and propose a new construction logic and efficient protocol designs, assuming ℓ is a power of 2 (general cases will be discussed later).

Let $e_{\ell},\ldots,e_1$ and $f_{\ell},\ldots,f_1$ are the bitwise representation of $[x]^{\ell}_0$ and $[x]^{\ell}_1$ respectively, and hence $\mathsf{MSB}(x)=c_{\ell}\oplus e_{\ell}\oplus f_{\ell}$, where $c_{\ell}=c_{\ell-1}\wedge(e_{\ell-1}\oplus f_{\ell-1})\oplus(e_{\ell-1}\wedge f_{\ell-1})$ is the ℓ-th carry bit. Given $g_i=e_i\wedge f_i$ and $p_i=e_i\oplus f_i$ for $i\in[\ell]$, c_{ℓ} can be reformulated as

$$c_{\ell}=g_{\ell-1}\oplus(p_{\ell-1}\wedge g_{\ell-2})\oplus\cdots\oplus(p_{\ell-1}\wedge\cdots\wedge p_2\wedge g_1). \tag{2}$$

To obtain c_{ℓ}, we construct a $\log\ell$-layer tree and traverse the tree from the leaves until reaching the root. The leaves adhere to the following operation

$$g^1_k=g_k\quad p^1_j=p_{2j-1}\wedge p_{2j-2}, \tag{3}$$

where $k\in[\ell-1]$ and $j\in[2,\ell/2]$. Then in the r-th layer ($r=2$), the tree needs to update g and p as follows

$$\begin{aligned} g^r_k&=g^{r-1}_{2k-1}\oplus(g^{r-1}_{2k-2}\wedge p^{r-1}_{2k-1})\\ g^r_1&=g^r_2\oplus(g^{r-1}_2\wedge p^{r-1}_2)\\ p^r_j&=p^{r-1}_{2j}\wedge p^{r-1}_{2j-1}, \end{aligned} \tag{4}$$

[6]An exception is the first linear layer that takes the client's samples and the server's model weights as input.

TABLE III
IDEAL FUNCTIONALITIES OF NON-LINEAR FUNCTIONS

(a) $\mathcal{F}_{\mathsf{Sign}}$

Parameters: Bitlength ℓ and size n.
Inputs: The arithmetic shares $[\mathbf{x}]_0^\ell \in \mathbb{Z}_{2^\ell}^n$ and $[\mathbf{x}]_1^\ell \in \mathbb{Z}_{2^\ell}^n$ from the client and the server, respectively.
Outputs: $[\mathbf{y}]_0^B = \mathbf{r} \in \mathbb{Z}_2^n$ to the client and $[\mathbf{y}]_1^B = \mathsf{sign}(\mathbf{x}) \oplus \mathbf{r} \in \mathbb{Z}_2^n$ to the server, where $\mathbf{r}$ is sampled from $\mathbb{Z}_2^n$ uniformly at random.

(b) $\mathcal{F}_{\mathsf{Maxpool}}$

Parameters: Window size $n \times n$ and number m.
Inputs: The boolean shares $[\mathbf{x}_1]_0^B \in \mathbb{Z}_2^{n\times n}, \cdots, [\mathbf{x}_m]_0^B \in \mathbb{Z}_2^{n\times n}$ from the client and $[\mathbf{x}_1]_1^B \in \mathbb{Z}_2^{n\times n}, \cdots, [\mathbf{x}_m]_1^B \in \mathbb{Z}_2^{n\times n}$ from the server.
Outputs: For $i \in [m]$, $[y_i]_0^B = r_i \in \mathbb{Z}_2$ to the client and $[y_i]_1^B = \mathsf{max}(\mathbf{x}_i) \oplus r_i \in \mathbb{Z}_2$ to the server, where r_i is sampled from $\mathbb{Z}_2$ uniformly at random.

where $k \in [2, \ell/2^{r-1}]$ and $j \in [2, \ell/2^r]$. When $r \geq 3$, we also follow the Equation 4 except that $g_k^r = g_{2k}^{r-1} \oplus (g_{2k-1}^{r-1} \wedge p_{2k}^{r-1})$ if $r \geq 3$. Finally, c_ℓ equals $g_1^{\log \ell}$, and $\mathsf{MSB}(x)$ can be further obtained via cost-free XOR operations.

To evaluate the above AND operations, we provide two customized protocols based on the silent OT extension [28], as we identify two different input forms in the tree. Specifically, for the intermediate nodes, as well as p_j^1 for $j \in [2, \ell/2]$ in the leaves, we use the generic Π_{AND} protocol [25] to perform AND operations, since the input of each node is boolean shared between the parties. Then we observe that g_k^1 for $k \in [\ell - 1]$ in the leaves can be computed using our simplified Π_{AND^*} protocol in Algorithm 6 of Appendix A, since the inputs e_i and f_i are held by the server and the client, respectively. The protocol Π_{AND^*} only requires one call of $\binom{2}{1}$-ROT_1 with 2-bit communication, saving half of the communication and computation overhead compared with Π_{AND}.

Algorithm 1 details our sign protocol. We would like to emphasize that our new designs significantly improve the current advanced adder, i.e., PPA [29], [44], in both computation and communication performance. Specifically, (1) our approach achieves better computation performance by reducing the required interaction rounds, thereby minimizing the effect of network latency. PPA consists of two sequential processes to compute MSB: an input computation step that needs one round of interaction, followed by a circuit evaluation step that involves $\log \ell$ rounds of interaction. Overall, $\log \ell + 1$ interaction rounds are required. By contrast, our evaluation logic saves a round of interaction. (2) For communication performance dominated by the AND evaluation, our approach requires fewer AND gates, and provides efficient and customized protocols for two types of AND with different input forms. In comparison, PPA needs more AND operations due to the redundant circuit, and it fails to capture the input feature of AND gates, resorting instead to a generalized protocol for all AND gates. The experimental results in Section VII also implicitly demonstrate the superiority of our protocol.[7]

Remark: We would like to clarify that the above adder-based method provides the same level of security as GC and OT, specifically resistance to probabilistic polynomial-time adversaries. Specifically, on the one hand, the adder is composed only of AND and XOR gates. The AND gates are evaluated using the protocol of Cheetah [25], while the evaluation of XOR gates is a local operation that does not require interaction between the parties. Thus, the security of the AND gate evaluation is ensured by the AND protocol in Cheetah, and the XOR gates, which incur no communication cost, remain secure due to the absence of information exchange between parties. On the other hand, like existing secure inference works [15], [16], [23], [25] and GC/OT protocols, all intermediate results in our adder-based protocol are represented as secret shares or are masked by randomness. In summary, the adder-based protocol we employed offers security on par with GC- and OT-based methods.

1) Correctness and Security: Given the correctness of c_ℓ, it is not hard to deduce that $y = \mathsf{MSB}(x) \oplus 1 = (e_\ell \oplus f_\ell \oplus c_\ell) \oplus 1$. Now we prove the correctness of the carry bit c_ℓ, i.e., $\mathbf{g}[1]$ in Algorithm 1. Specifically, in Algorithm 1, the server and the client participate in $\log \ell$ iterations and jointly evaluate Equations 3 or 4 in each iteration to update the vectors $\mathbf{g}$ and $\mathbf{p}$. For clarity, let $g_i = \mathbf{g}[i]$, $p_i = \mathbf{p}[i]$, and $\mathbf{g}^r$, $\mathbf{p}^r$ denote $\mathbf{g}$ and $\mathbf{p}$ that are waiting for update in Round r, $r \in [\log \ell]$, respectively. Then we have that $\mathbf{g}[1]$ in Round $\log \ell$ equals to $g_1^{\log \ell}$, where $[g_1^{\log \ell}]^B = [g_2^{\log \ell}]^B \oplus ([g_1^{\log \ell - 1}]^B \wedge [p_2^{\log \ell - 1}]^B) = ([g_{2*2}^{\log \ell - 1}]^B \oplus [g_{2*2-1}^{\log \ell - 1}]^B \wedge [p_{2*2}^{\log \ell - 1}]^B) \oplus [([g_{2*1}^{\log \ell - 2}]^B \oplus [g_{2*1-1}^{\log \ell - 2}]^B \wedge [p_{2*1}^{\log \ell - 2}]^B) \wedge ([p_{2*2}^{\log \ell - 2}]^B \wedge [p_{2*2-1}^{\log \ell - 2}]^B)] = \cdots = [g_{\ell-1}]^B \oplus ([g_{\ell-2}]^B \wedge [p_{\ell-1}]^B) \oplus \cdots \oplus ([g_1]^B \wedge [p_2]^B \wedge \cdots \wedge [p_{\ell-1}]^B)$. Given the correctness of $\mathcal{F}_{\mathsf{AND}}$ and $\mathcal{F}_{\mathsf{AND}^*}$, the above equation is equal to Equation 2, thus proving correctness. The security follows easily the $(\mathcal{F}_{\mathsf{AND}^*}, \mathcal{F}_{\mathsf{AND}})$-hybrid.

2) Communication Complexity: The client and the server communicate in this protocol only for $\mathcal{F}_{\mathsf{AND}^*}$ and $\mathcal{F}_{\mathsf{AND}}$. To compute $[g_k^1]^B$ for $k \in [\ell - 1]$, We need $\ell - 1$ AND* gates. Then the leaves of the adder also needs $(\frac{\ell}{2} - 1)$ AND gates to compute p_j^1 for $j \in [2, \ell/2]$. Further, in the r-th layer, $r \in \{2, 3, \cdots, \log \ell\}$, $\frac{\ell}{2^r} \times 3 - 1$ AND gates are needed. Overall, we require $\ell - 1$ instances of $\mathcal{F}_{\mathsf{AND}^*}$ and $2\ell - \log \ell - 3$ instances of $\mathcal{F}_{\mathsf{AND}}$, and this gives us total communication of $2(\ell - 1) + 4(2\ell - \log \ell - 3) = 10\ell - 4\log \ell - 14$ bits.

3) General Case: The bitlength of $[x]^\ell$, i.e., ℓ, maybe (1) powers of 2 (Algorithm 1 is applicable to this case), (2) even but not powers of 2, and (3) odd. In each round $r \in [\lceil \log \ell \rceil]$, the evaluation of $\{p^r\}$ and $\{g^r\}$ is identical in the three cases, with only slight differences in computing g_1^r. We can make minor modifications on protocol Π_{Sign} to deal with the latter two cases. Specifically, for case (2), in Round $r \in [2, \lceil \log \ell \rceil]$, if $\lceil \frac{\ell}{2^{r-1}} \rceil$ is even, $[g_1^r]^B = [g_2^r]^B \oplus [g_1^{r-1}]^B \wedge [p_2^{r-1}]^B$ will be evaluated via invoking line 9 of Algorithm 1, otherwise, $[g_1^r]^B$

[7] We do not report the experimental results of using PPA to evaluate MSB. Nonetheless, our scheme does improve this approach, due to the state-of-the-art counterpart in Cheetah [25], superior to the PPA-based method, still has performance disadvantages over our protocol as shown in Section VII.

will not need to be computed since $[g_1^r]^B = [g_1^{r-1}]^B$. Except for this step, other operations are the same as in case (1). For case (3), in Round 2, $[g_1^2]^B = [g_2^1]^B \oplus [g_1^1]^B \wedge [p_2^1]^B$ will be evaluated via invoking $\mathcal{F}_{\mathsf{AND}}$, and other operations are the same as in case (2).

Algorithm 1 Secure Sign Activation Protocol Π_{Sign}

Input: Arithmetic shares $[x]^\ell \in \mathbb{Z}_{2^\ell}$
Output: Boolean shares $[y]^B = \mathsf{sign}(x) \in \mathbb{Z}_2$

1: The client and the server parse $[x]_0^\ell$ and $[x]_1^\ell$ to $e_\ell||e_{\ell-1}||\cdots||e_1$ and $f_\ell||f_{\ell-1}||\cdots||f_1$, respectively.
2: The client and the server initiate $\{[\mathbf{g}]_0^B, [\mathbf{p}]_0^B = \{e_i|i \in [\ell]\}\}$ and $\{[\mathbf{g}]_1^B, [\mathbf{p}]_1^B = \{f_i|i \in [\ell]\}\}$, respectively.
3: The client and the server invoke $\mathcal{F}_{\mathsf{AND}^*}([\mathbf{p}[i]]_0^B, [\mathbf{p}[i]]_1^B)$ to obtain $[\mathbf{g}[i]]^B$ for $i \in [\ell-1]$, and $\mathcal{F}_{\mathsf{AND}}([\mathbf{p}[2i-1]]^B, [\mathbf{p}[2i-2]]^B)$ to obtain $[\mathbf{p}[i]]^B$ for $i \in \{2, \cdots, \frac{\ell}{2}\}$.
▷ Round 1
4: **for** $r \in \{2, \cdots, \log \ell\}$ **do** ▷ Round 2 ∼ Round $\log \ell$
▷ *Evaluate* $\{g^r\}$ *and* g_1^r *in each round* r
5: **if** $r = 2$ **then**
6: The client and the server invoke $\mathcal{F}_{\mathsf{AND}}([\mathbf{g}[2i-2]]^B, [\mathbf{p}[2i-1]]^B)$ and obtain $[a]^B$, and then learn $[\mathbf{g}[i]]^B = [\mathbf{g}[2i-1]]^B \oplus [a]^B$ locally, for $i \in [2, \frac{\ell}{2^{r-1}}]$.
7: **else**
8: The client and the server invoke $\mathcal{F}_{\mathsf{AND}}([\mathbf{g}[2i-1]]^B, [\mathbf{p}[2i]]^B)$ and obtain $[a]^B$, and then learn $[\mathbf{g}[i]]^B = [\mathbf{g}[2i]]^B \oplus [a]^B$ locally, for $i \in [2, \frac{\ell}{2^{r-1}}]$.
9: The client and the server invoke $\mathcal{F}_{\mathsf{AND}}([\mathbf{g}[1]]^B, [\mathbf{p}[2]]^B)$ and obtain $[a]^B$, and then learn $[\mathbf{g}[1]]^B = [\mathbf{g}[2]]^B \oplus [a]^B$ locally.
▷ *Evaluate* $\{p^r\}$ *in each round* r
10: The client and the server invoke $\mathcal{F}_{\mathsf{AND}}([\mathbf{p}[2i]]^B, [\mathbf{p}[2i-1]]^B)$ and obtain $[\mathbf{p}[i]]^B$, for $i \in [2, \frac{\ell}{2^r}]$.
11: **end for**
12: The client generates $[y]_0^B = e_\ell \oplus [\mathbf{g}[1]]_0^B$ and the server generates $[y]_1^B = f_\ell \oplus [\mathbf{g}[1]]_1^B$.
13: **Return** $[y]^B$

4) Integrate Batch Normalization for Free: In BNNs, BN is useful to normalize feature x before applying the activation function. We can apply the BN fusion technique [15] to evaluate this layer for free. Specifically, a BN followed by an activation layer is equivalent to $y = \mathsf{sign}(\gamma \cdot x + \beta) = \mathsf{sign}(x + \frac{\beta}{\gamma})$, where the latter equation works since γ is a positive value. Thus, the fusion of BN and activation layers is realized by a single invocation of protocol Π_{Sign}.

B. Binary Maxpooling Protocol

The binary maxpooling layer is used to obtain the maximum value among the binary activations within an $n \times n$ sliding window. This operation can be expressed as $y = \mathsf{max}(\mathbf{x})$, where $\mathbf{x} = (x_1, \ldots, x_{n\times n}) \in \{+1, -1\}^{n\times n}$. Given the boolean encoding ($+1 \rightarrow 1$ and $-1 \rightarrow 0$), this function can be represented only with AND and NOT gates, as follows,

$$\mathsf{max}(\mathbf{x}) = \neg\left(\neg x_1 \wedge \neg x_2 \wedge \ldots \wedge \neg x_{n\times n}\right). \quad (5)$$

Prior protocols [15], [16] utilized GC to evaluate this equation. However, the inherent limitations of GC are the high communication cost, e.g., $2\lambda(n^2-1)$ bits as reported in Table I, due to transmitting garbled tables [26], and the heavy computation overhead due to invoking encryption operations.

We attempt to construct a max protocol built on the silent OT primitive, and use a tree-reduction mode for better round complexity. We observe that in SecBNN, the inputs of the max function are boolean shares $[\mathbf{x}]_0^B$ and $[\mathbf{x}]_1^B$, on which the NOT gate is cost-free and the AND gate can be efficiently implemented by invoking $\mathcal{F}_{\mathsf{AND}}$. Based on this observation, our solution only requires $4(n^2-1)$ bits of communication and achieves $\frac{\lambda}{2}\times$ communication improvement, i.e., at least $64\times$, compared with GC-based methods [15], [16]. Further, instead of sequentially evaluating AND gates on $n\times n$ elements, we use a tree-reduction mode to reduce communication rounds. The insight is to recursively partition the input into two halves and then evaluate the elements of each half. Specifically, assuming n is a power of 2 (general cases will be discussed below), the parties arrange the $n \times n$ values into a 2-ary tree with the depth of $\log n^2$, and evaluate the tree in a top-down fashion. Algorithm 2 details our maxpooling protocol.

1) Correctness and Security: From Algorithm 2, $\mathbf{t} = \mathsf{Rec}^{\mathsf{B}}([\mathbf{t}]_0^B, [\mathbf{t}]_1^B) = [\mathbf{x}]_0^B \oplus [\mathbf{x}]_1^B \oplus 1 = \neg\mathbf{x}$. Then given the correctness of $\mathcal{F}_{\mathsf{AND}}$, after the tree evaluation we have $y = \mathsf{Rec}^{\mathsf{B}}([y]_0^B, [y]_1^B) = [\mathbf{t}[0]]_0^B \oplus [\mathbf{t}[0]]_1^B \oplus 1 = (\mathbf{t}[1] \wedge \mathbf{t}[2] \wedge \ldots \wedge \mathbf{t}[n\times n]) \oplus 1 = \neg(\neg\mathbf{x}[1] \wedge \neg\mathbf{x}[2] \wedge \ldots \wedge \neg\mathbf{x}[n\times n]) = \mathsf{max}(\mathbf{x})$. Security follows trivially that in the $\mathcal{F}_{\mathsf{AND}}$-hybrid.

2) Communication Complexity: The client and the server communicate in this protocol only for $\mathcal{F}_{\mathsf{AND}}$ (line 3). Overall, our protocol requires n^2-1 AND gates for each window with the size of $n \times n$. Therefore, given m sliding windows, the overall communication cost is $4m(n^2-1)$ bits, and the number of communication round is $\log n^2$.

3) General Case: Protocol Π_{Maxpool} can be easily extended to the setting where the window size is not a power of 2. In this case, we do not have a perfect binary tree of recursion, and need to slightly change our recursion/tree traversal. Inspired by [23], our method is to construct multiple maximal possible perfect binary sub-trees where the leaves are disjoint subsets of the values within the window. The sub-trees can be evaluated using Algorithm 2 to obtain the corresponding values. Then, in the same manner, these values as leaves continue the tree evaluation until the final result is obtained.

Algorithm 2 Secure Binary Maxpooling Protocol Π_{Maxpool}

Input: Boolean shares $[\mathbf{x}]^B$ of size $n \times n$.
Output: Boolean shares $[y]^B = \mathsf{max}(\mathbf{x})$.

1: The client and the server set $[\mathbf{t}]_0^B = [\mathbf{x}]_0^B$ and $[\mathbf{t}]_1^B = [\mathbf{x}]_1^B \oplus 1$, respectively.
2: **for** $j = 1$ to $\log n^2$ **do**
3: The client and the server invoke $\mathcal{F}_{\mathsf{AND}}([\mathbf{t}[2(k-1)+1]]^B, [\mathbf{t}[2(k-1)+2]]^B)$ for $k \in [\frac{n^2}{2^j}]$, and obtain $[\mathbf{t}[k]]^B$.
4: **end for**
5: The client and the server compute $[y]_0^B = [\mathbf{t}[0]]_0^B$ and $[y]_1^B = [\mathbf{t}[0]]_1^B \oplus 1$, respectively.
6: **Return** $[y]^B$.

TABLE IV
IDEAL FUNCTIONALITIES OF LINEAR FUNCTIONS

(a) $\mathcal{F}_{\text{BinaryLinear}}$

Parameters: Dimension m, n, k, and bitlength $\ell = \lceil \log(n+1) \rceil + 1$. Original weight $\mathbf{W}'$ and input $\mathbf{X}'$.
Inputs: The encoded boolean shared $[\mathbf{X}]_0^B \in \{0,1\}^{n\times k}$ from the client and $[\mathbf{X}]_1^B \in \{0,1\}^{n\times k}$ from the server. The encoded weight $\mathbf{W} \in \{0,1\}^{m\times n}$ from the server.
Outputs: $[\mathbf{Y}]_0^B = \mathbf{R} \in \mathbb{Z}_{2^\ell}^{m\times k}$ to the client and $[\mathbf{Y}]_0^B = \mathbf{W}' \cdot \mathbf{X}' - \mathbf{R} \in \mathbb{Z}_{2^\ell}^{m\times k}$ to the server, where $\mathbf{R}$ is sampled from $\mathbb{Z}_{2^\ell}^{m\times k}$ uniformly at random.

(b) $\mathcal{F}_{\text{FirstLinear}}$

Parameters: Dimension m, n, $k > 0$, input's bitlength $\ell > 0$ and output's bitlength $\ell' = \ell + \lceil \log n \rceil$.
Inputs: $\mathbf{X} \in \mathbb{Z}_{2^\ell}^{n\times k}$ from the client and $\mathbf{W} \in \{+1,-1\}^{m\times n}$ from the server.
Outputs: $[\mathbf{Y}]_0^{\ell'} = \mathbf{R} \in \mathbb{Z}_{2^{\ell'}}^{m\times k}$ to the client and $[\mathbf{Y}]_1^{\ell'} = \mathbf{W} \cdot \mathbf{X} - \mathbf{R} \in \mathbb{Z}_{2^{\ell'}}^{m\times k}$ to the server, where $\mathbf{R}$ is sampled from $\mathbb{Z}_{2^{\ell'}}^{m\times k}$ uniformly at random.

VI. PROTOCOLS FOR LINEAR LAYERS

This section presents efficient protocols for the linear layers of BNNs including (1) binary linear layers and (2) the first linear layer. The corresponding functionalities are shown in Table IV.

A. Binary Linear Protocol

Binary linear layers can be formalized as $\mathbf{Y} = \mathbf{W} \cdot \mathbf{X}$, where $\mathbf{W} \in \{-1,+1\}^{m\times n}$ is the model weight owned by the server, and $\mathbf{X} \in \{-1,+1\}^{n\times k}$ is secret-shared between the server and the client. As illustrated in [15], this matrix multiplication operation can be evaluated with the XNOR-PopCount paradigm, if $\mathbf{W}$ and $\mathbf{X}$ are encoded to boolean values. For ease of understanding, we illustrate a vector case in Figure 2, i.e., $\mathbf{W}[j,\cdot] \cdot \mathbf{X}[\cdot,i]$. Briefly, element-wise XNOR operations between the encoded $\mathbf{W}[j,\cdot]$ and $\mathbf{X}[\cdot,i]$ are first evaluated. Next, PopCount is executed: first computing p as the sum of the XNOR outputs, and then outputting $2p - n$. In this paradigm, XNOR is cost-free, and hence the challenge is to efficiently compute the sum p.

We explore the boolean-to-arithmetic (B2A) protocol since the sum operation on arithmetic shares is free and this protocol can be built upon the efficient COT primitive [28]. Specifically, we first convert the boolean shares to ℓ-bit arithmetic shares with $\ell = \lceil \log(n+1) \rceil + 1$ to prevent overflow [16], and then sum the generated arithmetic shares for free. When extending to the matrix form, by carefully setting up the receiver's and sender's messages in COT, the batching technique can be adopted to reduce the computation and communication complexities. The main insight is that, when evaluating XNOR, i.e., $\mathbf{Z}[j,\cdot] = \mathbf{W}[j,\cdot] \oplus \mathbf{X}[\cdot,i] \oplus 1$ for each $j \in [m]$, $[\mathbf{Z}[j,\cdot]]_1^B = \mathbf{W}[j,\cdot] \oplus [\mathbf{X}[\cdot,i]]_1^B \oplus 1$, but $[\mathbf{Z}[j,\cdot]]_0^B$ always equals $[\mathbf{X}[\cdot,i]]_0^B$. It means that the same choice bit of $[\mathbf{X}[\cdot,i]]_0^B$ is used for the corresponding column of $[\mathbf{Z}]_1^B$. Thus, we can evaluate $\mathbf{Z}$ with nk invocations of $\binom{2}{1}$-COT$_{m\ell}$, rather than mnk invocations of $\binom{2}{1}$-COT$_\ell$. More importantly, in the COT implementation, the mask bitlength, i.e., the AES block size, is typically 128. Therefore, the masks in our protocol can be generated using $\frac{m\ell}{128}$ AES evaluations, achieving a $\frac{128}{\ell}\times$ computation improvement.

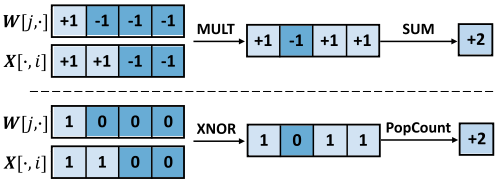


Fig. 2. Equivalence between the binary linear function and the XNOR-PopCount operation.

Despite the overall advantages, our protocol consumes $O(n\ell)$ of the asymptomatic communication complexity in the above vector example. As discussed in Section II-B, when working in a communication-limited network environment, this complexity results in a rise in runtime due to the low bandwidth. To alleviate the communication overhead without sacrificing correctness, we propose a divide-and-conquer strategy. The main insight is to first perform B2A on a small bitlength $\ell' = \lceil \log(n'+1) \rceil + 1$ ($\ell' < \ell$), and then sum on each n'-dimension ($n' < n$) sub-vector.[8] Finally, we sum the $\lceil \frac{n}{n'} \rceil$ summations by first invoking the ℓ'-to-ℓ bitlength extension protocol. In our setting, the inputs to the extension protocol are always positive, thus we can adopt the MSB-known bitlength extension protocol in SIRNN [32] and further optimize it using silent OT as shown in Algorithm 8 of Appendix A. As a result, this protocol requires $\ell - \ell' + 3$ bits of communication within 3 rounds. Note that in our evaluation, we adaptively choose the appropriate protocol with or without the divide-and-conquer strategy regarding the network environment, to carefully balance the round and communication complexity, thereby optimizing overall performance. The detailed secure binary linear protocol is given in Algorithm 3.

1) Correctness and Security: The multiplication between $\mathbf{W}$ and $\mathbf{X}$ can be split into k matrix-vector multiplications. Below, we focus on the correctness of the i-th matrix-vector multiplication, denoted by $\mathbf{W} \cdot \mathbf{X}[\cdot,i]$. First, we prove the correctness of our B2A gadget, i.e., $\mathbf{M} = [\mathbf{W} \boxplus \mathbf{X}[\cdot,i] \oplus 1]_0^B + [\mathbf{W} \boxplus \mathbf{X}[\cdot,i] \oplus 1]_1^B - 2[\mathbf{W} \boxplus \mathbf{X}[\cdot,i] \oplus 1]_0^B \circ [\mathbf{W} \boxplus \mathbf{X}[\cdot,i] \oplus 1]_1^B$. Specifically, let $\mathbf{Z}_0^B$ denote the choice bit matrix, where $\mathbf{Z}_0^B = \begin{bmatrix} [\mathbf{X}[1,i]]_0^B & \cdots & [\mathbf{X}[n,i]]_0^B \\ \vdots & \ddots & \vdots \\ [\mathbf{X}[1,i]]_0^B & \cdots & [\mathbf{X}[n,i]]_0^B \end{bmatrix}$. From line 7 of Algorithm 3 and the correctness of *batched* $\binom{2}{1}$-COT, we have

$$\begin{aligned} \mathbf{M} &= \mathsf{Rec}^{\ell'}([\mathbf{M}]_0^{\ell'}, [\mathbf{M}]_1^{\ell'}) \\ &= (-\mathbf{T}_s + [\mathbf{Z}]_0^B) + ([\mathbf{Z}]_1^B + \mathbf{T}_c) \\ &= (-\mathbf{T}_s + [\mathbf{Z}]_0^B) + ([\mathbf{Z}]_1^B + \mathbf{T}_s - 2\mathbf{B} \circ [\mathbf{Z}]_1^B) \\ &= [\mathbf{Z}]_1^B + [\mathbf{Z}]_0^B - 2[\mathbf{Z}]_0^B \circ [\mathbf{Z}]_1^B. \end{aligned} \tag{6}$$

Observe that $[\mathbf{Z}]_0^B$ and $[\mathbf{Z}]_1^B$ are boolean shares of $\mathbf{W} \boxplus \mathbf{X}[\cdot,i] \oplus 1$ from line 3 in Algorithm 3, which proves our B2A gadget. Next the parties accumulate each row of $[\mathbf{M}]^\ell$ and obtain the final result $[\mathbf{Y}[\cdot,i]]^\ell$. Given the correctness

[8] We set the optimal n' with the goal of minimizing communication.

Algorithm 3 Secure Binary Linear Protocol $\Pi_{\mathsf{BinaryLinear}}$

Input: Encoded weight $\mathbf{W} \in \{0,1\}^{m\times n}$ from the server. Encoded input's shares $[\mathbf{X}]_0^B, [\mathbf{X}]_1^B \in \{0,1\}^{n\times k}$ from the client and the server, respectively.

Output: Arithmetic shares $[\mathbf{Y}]^\ell = \mathbf{W}'\cdot\mathbf{X}' \in \mathbb{Z}_{2^\ell}^{m\times k}$, where $\mathbf{W}'$ and $\mathbf{X}'$ are original weight and input, respectively.

1: **for** $i \in [k]$ **do**
2: The client and the server initiate empty $m\times n$ matrices $\mathbf{T}_c$ and $\mathbf{T}_s$, respectively.
3: The client sets $[\mathbf{Z}[j,\cdot]]_0^B = [\mathbf{X}[\cdot,i]]_0^B$, for $j \in [m]$, and the server sets $[\mathbf{Z}]_1^B = (\mathbf{W} \boxplus [\mathbf{X}[\cdot,i]]_1^B) \oplus 1$.
4: **for** $j \in [n]$ **do**
5: The server constructs a correlation function $f_j(\alpha) = \alpha - 2[\mathbf{Z}[\cdot,j]]_1^B$, for $\alpha \in \mathbb{Z}_{2^{\ell'}}^m$; the client sets $b_j = [\mathbf{Z}[0,j]]_0^B$.
6: The client and the server run $\binom{2}{1}$-$\mathsf{COT}_{m\ell'}$, where the server is the sender with input f_j and the client is the receiver with choice bit b_j. The server sets its output as $\mathbf{T}_s[\cdot,j]$, while the client sets its output as $\mathbf{T}_c[\cdot,j]$.
7: The server sets $[\mathbf{M}[\cdot,j]]_0^{\ell'} = [\mathbf{Z}[\cdot,j]]_0^B - \mathbf{T}_s[\cdot,j] \bmod 2^{\ell'}$; the client sets $[\mathbf{M}[\cdot,j]]_1^{\ell'} = [\mathbf{Z}[\cdot,j]]_1^B + \mathbf{T}_c[\cdot,j] \bmod 2^{\ell'}$.
8: **end for**
9: For $j \in [\lceil \frac{n}{n'} \rceil]$, the client and the server locally compute $[\mathbf{Q}[q,j]]^{\ell'} = \sum_{t\in[n']}[\mathbf{M}[q, j\cdot n'+t]]^{\ell'} \bmod 2^{\ell'}$, for $q \in [m]$.
10: The client and the server invoke $\mathcal{F}_{\mathsf{SExt}}^{\ell',\ell}([\mathbf{Q}]^{\ell'})$, and obtain $[\mathbf{Q}]^\ell$.
11: For $j \in [m]$, the client and the server locally compute $[p]^\ell = \sum_{t=1}^{\lceil n/n' \rceil}[\mathbf{Q}[j,t]]^\ell \bmod 2^\ell$, and $[\mathbf{Y}[j,i]]^\ell = 2[p]^\ell - n \bmod 2^\ell$.
12: **end for**
13: **Return** $[\mathbf{Y}]^\ell$

of $\mathcal{F}_{\mathsf{SExt}}^{\ell',\ell}$, $\mathsf{Rec}^\ell([\mathbf{Y}[\cdot,i]]_0^\ell, [\mathbf{Y}[\cdot,i]]_1^\ell) = \mathbf{W}\cdot\mathbf{X}[\cdot,i]$. Security follows trivially that in the $(\binom{2}{1}$-$\mathsf{COT}_{m\ell'}, \mathcal{F}_{\mathsf{SExt}}^{\ell',\ell})$-hybrid.

2) Communication Complexity: We first give the communication cost without our divide-and-conquer strategy, where the client and the server communicate only for nk calls to $\binom{2}{1}$-$\mathsf{COT}_{m\ell}$. Thus, the total communication is $nk(m\ell+1)$ bits. When adopting the divide-and-conquer optimization, our protocol consists of two communication processes, i.e., B2A and the bitlength extension. B2A requires nk calls to $\binom{2}{1}$-$\mathsf{COT}_{m\ell'}$, which introduces $nk(m\ell'+1)$ communication bits within 2 rounds and $\ell' = \lceil \log(n'+1)\rceil + 1$. The bitlength extension process requires $mk\lceil \frac{n}{n'}\rceil$ calls to $\mathcal{F}_{\mathsf{SExt}}^{\ell',\ell}$, which introduces $mk\lceil \frac{n}{n'}\rceil(3+\ell-\ell')$ bits within 3 rounds and $\ell = \lceil \log(n+1)\rceil + 1$. Overall, the communication cost (bit) is given in Equation 7 within 5 rounds.

$$nk(m\ell'+1) + mk\lceil \frac{n}{n'}\rceil(3+\ell-\ell') \tag{7}$$

B. First Linear Protocol

The first linear layer, comprising either an FC or a CONV layer, can be formulated as $\mathbf{Y} = \mathbf{W}\cdot\mathbf{X}$, where $\mathbf{W} \in \{-1,+1\}^{m\times n}$ is the weight owned by the server and $\mathbf{X} \in \mathbb{Z}_{2^\ell}^{n\times k}$ is the input sample owned by the client. The inefficiency of previous protocols [15], [16] stems from the high communication overhead introduced by OT.

To achieve a communication-efficient protocol, we propose a new COT-based solution along with batching optimizations. For clarity, we take a scalar multiplication instance, i.e., $w\cdot x$, where $w \in \{-1,+1\}$ and $x \in \mathbb{Z}_{2^\ell}$. The main idea is to construct a correlation function between the sender's messages and then incorporate local post-processing. To this end, we first construct a correlation function $f(\alpha) = 2x + \alpha$. Then, the client and the server call a $\binom{2}{1}$-COT_ℓ, where the client is the sender with the input f and the server is the receiver with the choice bit b ($b=1$ if $w=+1$ and 0 otherwise). After that, the two parties can obtain the shares of $w\cdot x$ if the client sets $[w\cdot x]_0^\ell = -\alpha - x$ and the server sets $[w\cdot x]_1^\ell$ as the output of COT, i.e., $[w\cdot x]_1^\ell = \alpha + b\cdot 2x$. Further, since the matrix multiplication operation involves the dot product of each row of $\mathbf{W}$ with k columns of $\mathbf{X}$, we can evaluate $\mathbf{Y}$ with mn invocations of $\binom{2}{1}$-$\mathsf{COT}_{k\ell}$, rather than mnb invocations of $\binom{2}{1}$-COT_ℓ. Similar to our binary linear protocol, this batching optimization achieves a $\frac{128}{\ell}\times$ computation improvement.

1) Correctness and Security: The multiplication of $\mathbf{W} \in \{-1,+1\}^{m\times n}$ and $\mathbf{X} \in \mathbb{Z}_{2^\ell}^{n\times k}$ can be split into m vector-matrix multiplications, i.e., multiplying each row of $\mathbf{W}$ by $\mathbf{X}$. Now we focus on the correctness of the i-th vector-matrix multiplication denoted as $\mathbf{W}[i,\cdot]\cdot\mathbf{X}$. Let $\mathbf{B}$ denote the choice bit matrix. By the correctness of batched $\binom{2}{1}$-$\mathsf{COT}_{k\ell}$, we have $\mathbf{M} = \mathsf{Rec}^\ell([\mathbf{M}]_0^\ell, [\mathbf{M}]_1^\ell) = -\mathbf{X} + 2\mathbf{X}\circ\mathbf{B}$. As shown in line 5 of Algorithm 4, all elements of the j-th row in $\mathbf{X}$ are multiplied by the same bit b_j. Thus, $\mathbf{M}$ can be expressed as

$$\mathbf{M} = -\mathbf{X} + 2\mathbf{X}\circ\begin{bmatrix} b_1 & \cdots & b_1 \\ \vdots & \ddots & \vdots \\ b_n & \cdots & b_n \end{bmatrix}. \tag{8}$$

Observe that if $b_j = 0$, $\mathbf{M}[j,\cdot] = -\mathbf{X}[j,\cdot] + 2\mathbf{X}[j,\cdot]\circ[0,\cdots,0] = -\mathbf{X}[j,\cdot]$, otherwise $\mathbf{M}[j,\cdot] = -\mathbf{X}[j,\cdot] + 2\mathbf{X}[j,\cdot]\circ[1,\cdots,1] = \mathbf{X}[j,\cdot]$. Therefore $\mathbf{M} = \mathsf{Rec}^\ell([\mathbf{M}]_0^\ell, [\mathbf{M}]_1^\ell) = \mathbf{W}[i] \boxtimes \mathbf{X}$. We next accumulate each column of $[\mathbf{M}]^\ell$ and obtain the final result $[\mathbf{Y}[i,\cdot]]^\ell$. Security follows trivially that in the $\binom{2}{1}$-$\mathsf{COT}_{k\ell}$-hybrid.

2) Communication Complexity: The client and the server communicate only for mn calls to $\binom{2}{1}$-$\mathsf{COT}_{k\ell}$ in line 5 of Algorithm 4. Therefore, the overall communication overhead is $mn(k\ell+1)$ bits within 2 rounds.

VII. Evaluation

A. Experiment Setup

1) Implementation: SecBNN is built on top of the EMP toolkit[9] in C++. To privately evaluate Python-based models, we use the EzPC framework[10] to translate the model description and trained parameters from Pytorch to the equivalent description in C++, which is then executed by our designed cryptographic backends. Like existing secure inference works [15], [16], [25], we simulate both LAN and WAN

[9] https://github.com/emp-toolkit
[10] https://github.com/mpc-msri/EzPC

Algorithm 4 Secure First Linear Protocol $\Pi_{\mathsf{FirstLinear}}$

Input: The binary weight $\mathbf{W} \in \{-1, +1\}^{m \times n}$ from the server, and the input $\mathbf{X} \in \mathbb{Z}_{2^\ell}^{n \times k}$ from the client.

Output: Arithmetic shares $[\mathbf{Y}]^\ell = \mathbf{W} \cdot \mathbf{X} \in \mathbb{Z}_{2^\ell}^{m \times k}$.

1: **for** $i \in [m]$ **do**
2: The client and the server initiate empty $n \times k$ matrices $\mathbf{T}_c$ and $\mathbf{T}_s$, respectively.
3: **for** $j \in [n]$ **do**
4: The client constructs a correlation function $f_j(\alpha) = 2\mathbf{X}[j, \cdot] + \alpha$, for $\alpha \in \mathbb{Z}_{2^\ell}^k$; the server sets $b_j = 1$ if $\mathbf{W}[i, j] = +1$ and 0 otherwise.
5: The client and the server run $\binom{2}{1}$-COT$_{k\ell}$, where the client is the sender with input f_j and the server is the receiver with choice bit b_j. The client sets its output as $\mathbf{T}_c[j, \cdot]$, while the server sets its output as $\mathbf{T}_s[j, \cdot]$.
6: **end for**
7: The client sets $[\mathbf{M}]_0^\ell = -\mathbf{T}_c - \mathbf{X}$ and the server sets $[\mathbf{M}]_1^\ell = \mathbf{T}_s$ where $\mathbf{M} = \mathbf{W}[i, \cdot] \boxtimes \mathbf{X}$.
8: For $j \in [k]$, the client and the server locally compute $[\mathbf{Y}[t, j]]^\ell = \sum_{t=1}^n [\mathbf{M}[t, j]]^\ell$.
9: **end for**
10: **Return** $[\mathbf{Y}]^\ell$

settings. Under LAN, the bandwidth is 384MBps and the latency is 0.3ms. Under WAN, the bandwidth is 44MBps and the latency is 40ms. All experiments are performed on AWS c5.9xlarge instances with Intel Xeon 8000 series CPUs at 3.6GHz. The reported results of all experiments represent the mean overhead, averaged over 10 runs.

2) Datasets and models: We evaluate SecBNN on MNIST and CIFAR10 datasets, which are two popular benchmarks considered in prior secure BNN inference works [15], [16]. We adopt the representative BNN models from XONN [15], most of which are also used in XONN+ [16]. The architectures are denoted as BM2 and BM3 for MNIST, BC2, BC3, BC4 and BC5 for CIFAR10. In Appendix B, we detail the model architectures we used. We emphasize that these models achieve comparable accuracy [15] to that of fixed-point full-precision models in prior secure inference works [23], [24].

B. Microbenchmark Evaluation

We compare the performance of our protocols with the advanced counterparts in XONN [15], XONN+ [16] and Cheetah [25]. All results are evaluated using a single thread.

1) Sign Activation Protocol: We compare the proposed sign activation protocol with the GC-based counterparts of XONN and XONN+ under different bitlengths in Table V. Note that although XONN and XONN+ follow the same circuit logic, XONN+ needs to reconstruct the inputs in GC, which consumes additional overhead. We can observe that SecBNN achieves 31.64 $\sim$ 41.37$\times$ communication performance improvement over XONN. Besides, SecBNN gains runtime speedups on both LAN and WAN. Compared to XONN+, SecBNN exhibits better performance advantages.

Beyond application to BNNs, our sign protocol can serve as a general building block for many operations such as ReLU [25]. To illustrate its efficiency, we also compare it with the state-of-the-art silent OT-based solution of Cheetah in Table V. We observe that the communication cost of our protocol always outperforms that of Cheetah, which is in line with the theoretical analysis in Table I. Similar conclusions apply to the runtime overhead. The main reason is that the communication cost of Cheetah is at a disadvantage due to the usage of costly 1-out-of-2^m OT primitives. Note that the bitlength in SecBNN is small, more precisely, less than 14 in our evaluation.[11]

TABLE V

COMPARISON OF OUR NON-LINEAR PROTOCOLS WITH XONN [15], XONN+ [16] AND CHEETAH [25] UNDER DIFFERENT BITLENGTHS OR WINDOW SIZES. THE NUMBER OF INSTANCES IS 10^5

(a) Sign activation protocol

Method	Comm. (MB)		LAN (ms)		WAN (ms)	
	Comm.	Speedup	Time	Speedup	Time	Speedup
Bitlength: 8						
XONN+	50.11	84.93×	364.12	3.09×	3126.01	13.67×
XONN	24.41	41.37×	232.06	1.97×	1894.23	8.28×
Cheetah	0.90	1.53×	170.15	1.44×	249.98	1.09×
SecBNN	0.59	1×	117.97	1×	228.74	1×
Bitlength: 10						
XONN+	62.70	68.15×	441.64	2.53×	3549.16	9.75×
XONN	30.51	33.16×	282.89	1.62×	2190.06	6.01×
Cheetah	1.38	1.50×	260.95	1.50×	379.97	1.04×
SecBNN	0.92	1×	174.29	1×	364.18	1×
Bitlength: 14						
XONN+	108.53	80.39×	698.63	2.58×	5843.05	12.75×
XONN	42.72	31.64×	432.47	1.60×	3498.66	7.63×
Cheetah	1.90	1.41×	357.78	1.32×	477.28	1.04×
SecBNN	1.35	1×	270.99	1×	458.28	1×

(b) Binary maxpooling protocol

Method	Comm. (MB)		LAN (ms)		WAN (ms)	
	Comm.	Speedup	Time	Speedup	Time	Speedup
Window size: 2×2						
XONN(+)	9.15	65.36×	29.27	1.18×	503.89	4.79×
SecBNN	0.14	1×	24.75	1×	105.13	1×
Window size: 3×3						
XONN(+)	24.41	64.24×	71.86	1.14×	1146.34	5.11×
SecBNN	0.38	1×	63.01	1×	224.44	1×

2) Binary Maxpooling Protocol: In Table V(b), we compare the proposed binary maxpooling protocol with GC-based solutions (XONN, XONN+) under different window sizes. XONN and XONN+ utilize the same GC logic for maxpooling, and neither requires the reconstruction of secret-shared inputs. We observe that SecBNN reduces the communication cost by about 65×, in line with the theoretical analysis in Table I. This communication advantage also significantly boosts the runtime of our protocol especially under WAN (about 5×).

[11] We set the minimal bitlength of arithmetic shares according to the size of linear layers so that the protocol's outputs are bitwise equivalent to the cleartext execution without balancing efficiency and accuracy. Specifically, the bitlength is reset for each linear layer, where the main operation related to the bitlength is the bitcounting of a boolean vector with size n, and n depends on the size of the linear layer. To represent the output with a signed value without overflow, we set the bitlength as $\ell = \lceil \log(n+1) \rceil + 1$, which is minimal for perfect correctness. ℓ is further used in the following non-linear operation. In our BNN models, $n < 2^{12}$ and hence $\ell < 14$.

TABLE VI

COMPARISON OF OUR LINEAR PROTOCOLS WITH XONN [15] AND XONN+ [16] UNDER DIFFERENT DIMENSIONS

(a) Binary linear protocol

Method	Comm. (KB)		LAN (ms)		WAN (ms)	
	Comm.	Speedup	Time	Speedup	Time	Speedup
(H, W, C) = (8, 8, 32), Window: 3 × 3, **Stride:** 1, **Kernels:** 48						
XONN+	2376.00	2.20×	34.84	1.24×	379.64	3.33×
SecBNN	1081.70	1×	28.05	1×	114.13	1×
(H, W, C) = (16, 16, 16), Window: 3 × 3, **Stride:** 1, **Kernels:** 32						
XONN+	2664.00	2.05×	42.48	1.24×	386.23	3.14×
SecBNN	1296.57	1×	34.26	1×	123.05	1×
(H, W, C) = (32, 32, 16), Window: 3 × 3, **Stride:** 1, **Kernels:** 16						
XONN+	5220.00	2.01×	82.55	1.18×	538.16	2.86×
SecBNN	2592.29	1×	70.18	1×	187.84	1×

(b) First linear protocol

Method	Comm. (KB)		LAN (ms)		WAN (ms)	
	Comm.	Speedup	Time	Speedup	Time	Speedup
(H, W, C) = (28, 28, 1), Window size: 5 × 5, **Stride:** 2, **Kernels:** 5						
XONN(+)	123.28	4.31×	2.07	3.23×	82.42	2.02×
SecBNN	28.60	1×	0.64	1×	40.78	1×
(H, W, C) = (32, 32, 3), Window size: 3 × 3, **Stride:** 1, **Kernels:** 64						
XONN(+)	6128.00	2.18×	69.31	1.35×	554.32	3.62×
SecBNN	2808.22	1×	51.52	1×	153.21	1×
(H, W, C) = (32, 32, 3), Window size: 5 × 5, **Stride:** 1, **Kernels:** 36						
XONN(+)	8833.38	2.28×	86.18	1.38×	794.97	4.01×
SecBNN	3876.32	1×	62.58	1×	198.05	1×

3) Binary Linear Protocol: Table VI(a) compares our binary linear protocol with the OT-based solution of XONN+. Our protocol is superior to XONN+ by at least 2× for the communication performance, and achieves 2.86 ∼ 3.33× runtime speedups under WAN. We omit the comparison with the GC-based method in XONN since optimized full adder circuits in GC are not given. Nonetheless, our protocol does improve the counterpart of XONN, since XONN+, inferior to our protocol, still has overall performance advantages over XONN.

We further justify the advantage of our divide-and-conquer strategy under a communication-limited setting, i.e., WAN, in Table VII. The optimized protocol boasts a communication cost about 2× lower than the non-optimized method while achieving better runtime performance. The exception is under LAN, where the usage of this strategy additionally increases 25% runtime on average. This is because, in a low-latency environment, communication is no longer the primary factor affecting the runtime. As a result, SecBNN adaptively configures this strategy based on the network environment.

4) First Linear Protocol: Table VI(b) reports the performance of our secure first linear protocol and the OT-based methods in XONN and XONN+. XONN+ implements this operation in the offline/online paradigm. For a fair evaluation, we focus on the overall overhead, which is the same as that of XONN. Our protocol provides at least 2.18× communication performance improvement over these two methods. Such communication advantage also significantly improves the runtime, since the communication dominates the main overhead of OT-based solutions. Moreover, while slow-down is observed under LAN, our solution still outperforms prior works by 1.35 ∼ 3.23×.

TABLE VII

EFFECT OF OUR DIVIDE-AND-CONQUER STRATEGY ON BINARY LINEAR LAYERS. SecBNN * IS THE OPTIMIZED PROTOCOL USING THIS STRATEGY, AND SecBNN IS THE NON-OPTIMIZED VARIANT

Method	Comm. (MB)	LAN (ms)	WAN (ms)
(H, W, C) = (2, 2, 512), Window size: 3 × 3, **Stride:** 1, **Kernels:** 512			
SecBNN	16.31	463.28	837.17
SecBNN *	8.60	542.33	775.81
(H, W, C) = (4, 4, 256), Window size: 3 × 3, **Stride:** 1, **Kernels:** 512			
SecBNN	29.91	672.28	1418.06
SecBNN *	15.51	927.60	1310.81
(H, W, C) = (4, 4, 512), Window size: 3 × 3, **Stride:** 1, **Kernels:** 512			
SecBNN	63.28	1349.86	2917.70
SecBNN *	28.97	1653.26	2218.63

TABLE VIII

COMPARISON WITH XONN [15] AND XONN+ [16] ON MNIST AND CIFAR10. THE REPORTED RESULTS OF XONN+ DO NOT INCLUDE THE OFFLINE OVERHEAD, AND THEREFORE SecBNN SHOULD HAVE BETTER ADVANTAGES

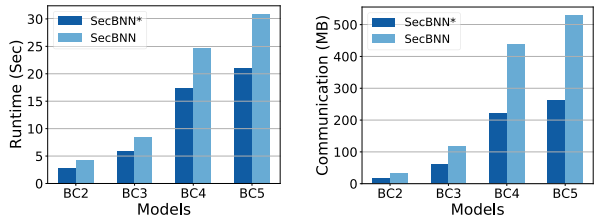
Method	Comm. (MB)		LAN (s)		WAN (s)	
	Comm.	Speedup	Time	Speedup	Time	Speedup
MNIST, BM2						
XONN	2.90	22.31×	0.12	12.00×	-	-
SecBNN	0.13	1×	0.01	1×	0.61	1×
MNIST, BM3						
XONN	17.59	17.59×	0.17	8.50×	-	-
SecBNN	1.00	1 ×	0.02	1×	1.13	1×
CIFAR10, BC2						
XONN	936.83	29.62×	2.75	10.19×	-	-
XONN+	204.78	6.47×	1.34	4.96×	17.14	4.09×
SecBNN	31.63	1×	0.27	1×	4.19	1×
CIFAR10, BC3						
XONN	2972.43	25.05×	9.19	11.49×	-	-
XONN+	395.67	3.33×	2.56	3.20×	32.51	3.84×
SecBNN	118.68	1×	0.80	1×	8.57	1×

C. End-to-End Inference Evaluation

1) Comparison With Secure BNN Inference Works: In Table VIII, we compare the performance of SecBNN with two advanced secure BNN inference works, XONN and XONN+, under the same settings as reported in XONN+ (LAN: 1.25 GBps, 0.25ms; WAN: 20 MBps, 50ms). Besides, we used a width of $s = 1.5$ for XONN+, consistent with the width used in our SecBNN to ensure a fair comparison. The communication and runtime costs of these works are extracted from Figures 5 and 8 of XONN+. Compared with XONN, SecBNN requires about 20× and 27× less communication on MNIST and CIFAR10, respectively. For the runtime, SecBNN is about 10× faster than XONN. Compared with XONN+ on CIFAR10,[12] SecBNN achieves 5× communication improvement on average, and also gains a boost (3 ∼ 4×) on the runtime.

2) Effect of Our Optimized Binary Linear Protocol: Recall that we optimize the proposed binary linear protocol by designing a divide-and-conquer strategy under the

[12]We omit the comparison with XONN+ on MNIST, since XONN+ did not conduct experiments on this dataset.



(a) Runtime (b) Communication

Fig. 3. Effect of our optimized binary linear protocol on different models with CIFAR10 under WAN.

communication-limited network setting. Figure 3 demonstrates the applicability of our optimized protocol under WAN. Observe that the optimized protocol always outperforms the non-optimized method under WAN. Moreover, the runtime advantage is greater when the model size increases. The main reason is that a larger model brings more communication overhead and causes higher delays in the communication-limited network environment. Overall, the optimized protocol achieves about $2\times$ communication and $1.5\times$ runtime improvement on those models.

VIII. Related Works

A quantity of secure two-party inference frameworks [15], [20], [24], [25], [45], [46] have been proposed by utilizing advanced cryptographic techniques as well as various model architecture optimizations. We briefly discuss them as follows.

A. Secure Inference With Advanced Cryptographic Techniques

CryptoNets [17] is perhaps the first work for secure neural network inference, which utilizes leveled homomorphic encryption (LHE) schemes and approximates non-linear functions with polynomials. While there are several subsequent works following this paradigm [47], [48], [49], the inherent limitations of LHE are high computation costs. To mitigate this problem, several works, like MiniONN [19], Chameleon [45], Delphi [24] and Gazelle [20], proposed hybrid protocols. Specifically, they used additively homomorphic encryption (AHE) to evaluate linear layers and GC for non-linear layers. Subsequently, CrypTFlow2 [23] designed optimized OT-based non-linear protocols, thereby significantly improving the performance of GC-based counterparts. Based on CrypTFlow2, Huang et al. recently presented the state-of-the-art 2PC inference framework, Cheetah [25], which improves the above OT-based protocols utilizing the advanced silent OT extension [28] and designs efficient AHE-based linear protocols.

B. Secure Inference With Model Architecture Optimizations

Many works [15], [16], [24], [33], [46], [50], [51] focus on the co-design of architecture-optimized neural networks and customized cryptographic protocols for better performance. COINN [46] proposed mixed low-precision quantization with clustered weights, and customized matrix multiplication protocol for evaluating these crypto-friendly linear functions. Quotient [33] presented secure inference and training strategies with ternary weights, i.e. $\{-1, 0, +1\}$. It introduced useful primitives such as repeated quantization to stabilize the optimization process and COT-based protocols for matrix multiplication. As a special flavor of quantization, BNNs quantize both the weight and activation with a 1-bit representation. The main advantage of such binary values is that the heavy matrix multiplication is replaced as cost-free XNOR operations. Utilizing the unique binary characteristic, XONN [15] proposed the first secure BNN inference framework, which purely exploits GC for linear and non-linear functions. Following this paradigm, [16] proposed XONN+, a hybrid optimized protocol where GC is used for non-linear operations and linear functions are evaluated by invoking OT. Most recently, [52] conducts secure BNN inference based on the techniques from XONN+, where the linear and non-linear layers are evaluated using OT and GC, respectively. The only difference between these two solutions is that XONN+ operates in the offline-online setting, while this work considers an end-to-end inference mechanism. Consequently, the overall performance of both solutions is nearly identical. Along this line, our work aims to further improve the performance of secure BNN inference, by designing fundamental cryptographic protocols.

IX. Conclusion

This work presents SecBNN, a practical secure two-party inference framework on BNNs. The main contributions of SecBNN are new insights and constructions for the secure and efficient evaluation of linear and non-linear layers of BNNs. In particular, we propose a secure sign protocol with a novel adder logic and customized evaluation algorithms for non-linear layers, and a new binary matrix multiplication protocol for linear layers, where a divide-and-conquer strategy is employed to recursively break down the evaluation into multiple sub-problems. Building upon these efficient components, we evaluate SecBNN over several representative benchmarks. Experimental results show that SecBNN significantly improves existing BNN inference works.

Below, we discuss the limitations of SecBNN and will leave addressing these limitations as future work. Specifically, (1) the inherent properties of BNNs make them more suitable for the relatively simple prediction tasks demonstrated in Section VII. When exploring more complex prediction tasks using BNNs, their inference accuracy will be a primary consideration. Recent advancements in various aspects, including model architectures, loss functions, and quantization errors, have greatly improved the inference accuracy [4] and could potentially alleviate this concern. In addition, the development of tailored secure protocols for these strategies will be necessary to strike a balance between accuracy and efficiency. (2) As described in Section IV-A, our protocols are custom-designed to defend against semi-honest adversaries and cannot defeat more powerful malicious adversaries. However, when considering the malicious adversaries, even with the most advanced techniques, the protocol overhead will increase by several orders of magnitude. These remain areas for future exploration.

Appendix

A. Supporting Protocols

1) Protocols for $\mathcal{F}_{\mathsf{AND}}$ and $\mathcal{F}_{\mathsf{AND}^*}$: Algorithm 5 shows the input-shared AND protocol of Cheetah [25]. We also propose a non-shared AND protocol in Algorithm 6.

2) Protocol for $\mathcal{F}_{\mathsf{B2A}}$: Algorithm 7 shows the $\binom{2}{1}$-COT_ℓ based $\mathsf{B2A}$ protocol from Cheetah [25].

3) Optimized Protocol for $\mathcal{F}^{\ell',\ell}_{\mathsf{SExt}}$: We present an optimized bitlength extension protocol, which utilizes silent COT and builds over the protocol in SIRNN [32].

Algorithm 5 Input-Shared AND Protocol Π_{AND}

Input: P_0 and P_1 hold $[x]^B$ and $[y]^B$, where $x, y \in \{0,1\}$.
Output: P_0 and P_1 learn $\langle z\rangle_0^B$ and $\langle z\rangle_1^B$, respectively, s.t. $z = x \wedge y$.

1: P_0 and P_1 invoke 2 calls to $\binom{2}{1}$-ROT_1 based Algorithm 1 of [35], to learn $([c]_0^B, [a]_0^B, [b]_0^B)$ and $([c]_1^B, [a]_1^B, [b]_1^B)$, respectively, satisfying $[c]_0^B \oplus [c]_1^B = \left([a]_0^B \oplus [a]_1^B\right) \wedge \left([b]_0^B \oplus [b]_1^B\right)$.
2: P_i, $i \in \{0,1\}$, locally computes $[e]_i^B = [x]_i^B \oplus [a]_i^B$ and $[f]_i^B = [y]_i^B \oplus [b]_i^B$, and sends $[e]_i^B$ and $[f]_i^B$ to P_{1-i}.
3: P_i, $i \in \{0,1\}$, locally compute $e = \mathsf{Rec}^{\mathsf{B}}([e]_0^B, [e]_1^B)$ and $f = \mathsf{Rec}^{\mathsf{B}}([f]_0^B, [f]_1^B)$.
4: P_i, $i \in \{0,1\}$, locally computes $[z]_i^B = (i \wedge e \wedge f) \oplus ([a]_i^B \wedge f) \oplus ([b]_i^B \wedge e) \oplus [c]_i^B$.

Algorithm 6 Non-Shared AND Protocol Π_{AND^*}

Input: P_0 and P_1 hold x and y respectively, where $x, y \in \{0,1\}$.
Output: P_0 and P_1 learn $\langle z\rangle_0^B$ and $\langle z\rangle_1^B$, respectively, s.t. $z = x \wedge y$.

1: P_0 and P_1 invoke 1 call to $\binom{2}{1}$-ROT_1 in Algorithm 1 of [35], to learn $([c]_0^B, a)$ and $([c]_1^B, b)$, respectively, satisfying $[c]_0^B \oplus [c]_1^B = a \wedge b$.
2: P_0 computes $e = x \oplus a$ and sends e to P_1.
3: P_1 computes $f = y \oplus b$ and sends f to P_0.
4: P_0 locally computes $[z]_0^B = (a \wedge f) \oplus [c]_0^B$. P_1 locally computes $[z]_1^B = (y \wedge e) \oplus [c]_1^B$.

Algorithm 7 Boolean to Arithmetic Protocol Π_{B2A}

Input: P_0 and P_1 hold $[x]_0^B$ and $[x]_1^B$ respectively, where $x \in \{0,1\}$.
Output: P_0 and P_1 learn $\langle y\rangle_0^\ell$ and $\langle y\rangle_1^\ell$, respectively, s.t. $y = x$.

1: P_0 and P_1 invoke 1 call to $\binom{2}{1}$-COT_ℓ where P_0 is the sender with a correlation function $f(\alpha) = \alpha + [x]_0^B$ and P_1 is the receiver with input $[x]_1^B$. After $\binom{2}{1}$-COT_ℓ, P_0 learns r and sets $z_0 = 2^\ell - x$, and P_1 learns z_1.
2: P_i, $i \in \{0,1\}$, computes $[y]_i^\ell = [x]_i^B - 2 \cdot z_i$.

Algorithm 8 MSB-Known Extension $\Pi^{\ell',\ell}_{\mathsf{SExt}}$

Input: P_0 and P_1 hold $[x]_0^{\ell'}$ and $[x]_1^{\ell'}$ respectively, where $\mathsf{MSB}(x) = 0$.
Output: P_0 and P_1 learn $[y]_0^\ell$ and $[y]_1^\ell$, respectively, s.t. $y = x$.

1: P_0 and P_1 invoke $\mathcal{F}_{\mathsf{AND}^*}(\neg\mathsf{MSB}([x]_0^{\ell'}), \neg\mathsf{MSB}([x]_1^{\ell'}))$, and learn $[z]^B$.
2: P_0 and P_1 set $[w]_0^B = [z]_0^B \oplus 1$ and $[w]_1^B = [z]_1^B$, respectively, where $w = \mathsf{MSB}([x]_0^{\ell'}) \vee \mathsf{MSB}([x]_1^{\ell'})$.
3: P_0 and P_1 invoke $\mathcal{F}^{\ell-\ell'}_{\mathsf{B2A}}\left([w]^B\right)$ and learn $[w]^{\ell-\ell'}$.
4: P_i, $i \in \{0,1\}$, outputs $[y]_i^\ell = [x]_i^{\ell'} - 2^{\ell'} \cdot [w]_i^{\ell-\ell'} \bmod 2^\ell$.

B. Model Architecture

Table IX summarizes the model architectures employed for each dataset. The models are adopted from XONN [15], and detailed descriptions can be found in the appendix of XONN. These models are parameterized with s, which denotes the model width. Unless otherwise specified, we experimentally use $s = 1$ for MNIST and $s = 1.5$ for CIFAR10.

TABLE IX
MODEL ARCHITECTURES AND ACCURACY EMPLOYED FOR EACH DATASET

Datasets	Models	Accuracy (SecBNN)	Accuracy (XONN& XONN+)	Description
MNIST	BM2	0.97	0.97	1 CONV, 2 FC
	BM3	0.98	0.98	2 CONV, 2MP, 2FC
CIAFR10	BC2	0.73	0.73	9 CONV, 3 MP, 1 FC
	BC3	0.81	0.81	9 CONV, 3 MP, 1 FC
	BC4	0.85	0.85	11 CONV, 3 MP, 1 FC
	BC5	0.85	0.85	17 CONV, 3 MP, 1 FC

Table IX also provide concrete accuracy for each model we used. It is worth noticing that compared with existing secure BNN inference works [15], [16], our solution does not compromise model accuracy. The rationale is as follows. First, our solution and existing works all adopt 2PC techniques for protocol design. Second, our solution does not introduce any efficiency-driven modifications to the original BNN model structure. As a result, the improvements in communication and computational performance achieved by our solution, compared to previous works, are accomplished without any compromise in model accuracy.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017.
[3] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1," 2016, *arXiv:1602.02830*.
[4] H. Qin, R. Gong, X. Liu, X. Bai, J. Song, and N. Sebe, "Binary neural networks: A survey," *Pattern Recognit.*, vol. 105, Sep. 2020, Art. no. 107281.
[5] C. Yuan and S. S. Agaian, "A comprehensive review of binary neural network," 2021, *arXiv:2110.06804*.
[6] H. Le, R. K. Høier, C.-T. Lin, and C. Zach, "AdaSTE: An adaptive straight-through estimator to train binary neural networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2022, pp. 460–469.
[7] D. Kim and J. Choi, "Unsupervised representation learning for binary networks by joint classifier learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2022, pp. 9737–9746.
[8] J. Diffenderfer and B. Kailkhura, "Multi-prize lottery ticket hypothesis: Finding accurate binary neural networks by pruning a randomly weighted network," in *Proc. ICLR*, 2020, pp. 1–22.

[9] A. Bulat, B. Martinez, and G. Tzimiropoulos, "High-capacity expert binary networks," in *Proc. ICLR*, 2020, pp. 1–14.

[10] Z. Xu et al., "ReCU: Reviving the dead weights in binary neural networks," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 5178–5188.

[11] S. Xu, C. Liu, B. Zhang, J. Lü, G. Guo, and D. Doermann, "BiRe-ID: Binary neural network for efficient person re-ID," *ACM Trans. Multimedia Comput., Commun., Appl.*, vol. 18, no. 1s, pp. 1–22, Feb. 2022.

[12] B. Ferrarini, M. J. Milford, K. D. McDonald-Maier, and S. Ehsan, "Binary neural networks for memory-efficient and effective visual place recognition in changing environments," *IEEE Trans. Robot.*, vol. 38, no. 4, pp. 2617–2631, Aug. 2022.

[13] A. Wang, W. Xu, H. Sun, N. Pu, Z. Liu, and H. Liu, "Arrhythmia classifier using binarized convolutional neural network for resource-constrained devices," 2022, *arXiv:2205.03661*.

[14] Y. Ling, T. He, Y. Zhang, H. Meng, K. Huang, and G. Chen, "Lite-stereo: A resource-efficient hardware accelerator for real-time high-quality stereo estimation using binary neural network," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 12, pp. 5357–5366, Dec. 2022.

[15] P. Rizomiliotis, C. Diou, A. Triakosia, I. Kyrannas, and K. Tserpes, "Partially oblivious neural network inference," in *Proc. 19th Int. Conf. Secur. Cryptography*, 2022, pp. 158–169.

[16] M. Samragh, S. Hussain, X. Zhang, K. Huang, and F. Koushanfar, "On the application of binary neural networks in oblivious inference," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2021, pp. 4625–4634.

[17] 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. ICML*, 2016, pp. 1–20.

[18] 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.

[19] 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. 1–26.

[20] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "Gazelle: A low latency framework for secure neural network inference," in *Proc. USENIX Secur.*, 2018, pp. 1–13.

[21] 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*, Jun. 2019, pp. 496–511.

[22] 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.

[23] D. Rathee et al., "CrypTFlow2: Practical 2-party secure inference," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2020, pp. 1–26.

[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. 1–27.

[25] Z. Huang, W. Lu, C. Hong, and J. Ding, "Cheetah: Lean and fast secure two-party deep neural network inference," in *Proc. USENIX Secur.*, 2022, pp. 1–10.

[26] A. C. Yao, "Theory and application of trapdoor functions," in *Proc. 23rd Annu. Symp. Found. Comput. Sci.*, Nov. 1982, pp. 1–20.

[27] M. O. Rabin, "How to exchange secrets with oblivious transfer," *Cryptol. ePrint Arch.*, vol. 1, pp. 1–24, May 2005.

[28] K. Yang, C. Weng, X. Lan, J. Zhang, and X. Wang, "Ferret: Fast extension for correlated OT with small communication," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2020, pp. 1–20.

[29] P. Mohassel and P. Rindal, "Aby3: A mixed protocol framework for machine learning," in *Proc. ACM CCS*, 2018, pp. 1–18.

[30] A. Patra, T. Schneider, A. Suresh, and H. Yalame, "Aby2.0: Improved mixed-protocol secure two-party computation," in *Proc. USENIX Secur.*, 2021, pp. 1–6.

[31] J. Garay, B. Schoenmakers, and J. Villegas, "Practical and secure solutions for integer comparison," in *Proc. PKC*, 2007, pp. 1–18.

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

[33] N. Agrawal, A. Shahin Shamsabadi, M. J. Kusner, and A. Gascón, "Quotient: Two-party secure neural network training and prediction," in *Proc. ACM CCS*, 2019, pp. 1–22.

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

[35] 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.*, 2013, pp. 535–548.

[36] E. Boyle et al., "Efficient two-round OT extension and silent non-interactive secure computation," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2019, pp. 1–15.

[37] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Proc. 42nd IEEE Symp. Found. Comput. Sci.*, Apr. 2001, pp. 1–26.

[38] Y. Lindell, "How to simulate it—A tutorial on the simulation proof technique," in *Information Security and Cryptography*. Berlin, Germany: Springer, 2017, pp. 277–346.

[39] C. Weng, K. Yang, X. Xie, J. Katz, and X. Wang, "Mystique: Efficient conversions for zero-knowledge proofs with applications to machine learning," in *Proc. USENIX Secur.*, 2021, pp. 1–27.

[40] K. Yang, P. Sarkar, C. Weng, and X. Wang, "QuickSilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2021, pp. 1–24.

[41] M. Keller, E. Orsini, and P. Scholl, "MASCOT: Faster malicious arithmetic secure computation with oblivious transfer," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2016, pp. 1–32.

[42] M. Keller, V. Pastro, and D. Rotaru, "Overdrive: Making spdz great again," in *Proc. EUROCRYPT*, 2018, pp. 1–14.

[43] R. Canetti, "Security and composition of multiparty cryptographic protocols," *J. Cryptol.*, vol. 13, no. 1, pp. 143–202, Jan. 2000.

[44] D. Harris, "A taxonomy of parallel prefix networks," in *Proc. 37th Asilomar Conf. Signals, Syst. Comput.*, vol. 2, 2003, pp. 2213–2217.

[45] 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. 1–17.

[46] S. U. Hussain, M. Javaheripi, M. Samragh, and F. Koushanfar, "COINN: Crypto/ML codesign for oblivious inference via neural networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2021, pp. 1–34.

[47] R. Dathathri, B. Kostova, O. Saarikivi, W. Dai, K. Laine, and M. Musuvathi, "EVA: An encrypted vector arithmetic language and compiler for efficient homomorphic computation," in *Proc. 41st ACM SIGPLAN Conf. Program. Lang. Design Implement.*, Jun. 2020, pp. 1–33.

[48] F. Boemer, Y. Lao, R. Cammarota, and C. Wierzynski, "NGraph-HE: A graph compiler for deep learning on homomorphically encrypted data," in *Proc. 16th ACM Int. Conf. Comput. Frontiers*, Apr. 2019, pp. 1–37.

[49] R. Dathathri et al., "Chet: An optimizing compiler for fully-homomorphic neural-network inferencing," in *Proc. PLDI*, 2019, pp. 1–27.

[50] M. Keller and K. Sun, "Secure quantized training for deep learning," in *Proc. ICML*, 2022, pp. 1–24.

[51] N. K. Jha, Z. Ghodsi, S. Garg, and B. Reagen, "Deepreduce: Relu reduction for fast private inference," in *Proc. ICML*, 2021, pp. 1–26.

[52] X. Zhang, M. Samragh, S. Hussain, K. Huang, and F. Koushanfar, "Scalable binary neural network applications in oblivious inference," *ACM Trans. Embedded Comput. Syst.*, vol. 1, pp. 1–36, Jul. 2024.