FSSiBNN: FSS-based Secure Binarized Neural Network Inference with Free Bitwidth Conversion

Peng Yang[1], Zoe Lin Jiang[1,2(✉)], Jiehang Zhuang[1], Junbin Fang[3], Siu-Ming Yiu[4], and Xuan Wang[1,2]

[1] Harbin Institute of Technology, Shenzhen, Shenzhen, China
{stuyangpeng,jiehangzhuang}@stu.hit.edu.cn
[2] Guangdong Key Laboratory of New Security and Intelligence Technology, Shenzhen, China; zoeljiang@hit.edu.cn, wangxuan@cs.hitsz.edu.cn
[3] Jinan University, Guangzhou, China; tjunbinfang@jnu.edu.cn
[4] The University of Hong Kong, HKSAR, China; smyiu@cs.hku.hk

Abstract. Neural network inference as a service enables a cloud server to provide inference services to clients. To ensure the privacy of both the cloud server's model and the client's data, secure neural network inference is essential. Binarized neural networks (BNNs), which use binary weights and activations, are often employed to accelerate inference. However, achieving secure BNN inference with secure multi-party computation (MPC) is challenging because MPC protocols cannot directly operate on values of different bitwidths and require bitwidth conversion. Existing bitwidth conversion schemes expand the bitwidths of weights and activations, leading to significant communication overhead.
To address these challenges, we propose FSSiBNN, a secure BNN inference framework featuring free bitwidth conversion based on function secret sharing (FSS). By leveraging FSS, which supports arbitrary input and output bitwidths, we introduce a bitwidth-reduced parameter encoding scheme. This scheme seamlessly integrates bitwidth conversion into FSS-based secure binary activation and max pooling protocols, thereby eliminating the additional communication overhead. Additionally, we enhance communication efficiency by combining and converting multiple BNN layers into fewer matrix multiplication and comparison operations. We precompute matrix multiplication tuples for matrix multiplication and FSS keys for comparison during the offline phase, enabling constant-round online inference.
In our experiments, we evaluated various datasets and models, comparing our results with state-of-the-art frameworks. Compared with the two-party framework XONN (USENIX Security '19), FSSiBNN achieves approximately 7× faster inference times and reduces communication overhead by about 577×. Compared with the three-party frameworks SecureBiNN (ESORICS '22) and FLEXBNN (TIFS '23), FSSiBNN is approximately 2.5× faster in inference time and reduces communication overhead by 1.3× to 16.4×.

Keywords: Secure neural network inference · Binarized neural network · Free bitwidth conversion · Function secret sharing.

1 Introduction

Neural network inference as a service is increasingly utilized in applications such as disease diagnosis, fraud detection, and risk management. In these scenarios, a cloud server typically hosts a well-trained neural network model and provides inference services to clients who supply the data [20,22]. For example, a patient may send private medical data to the cloud server, which then performs the neural network inference and returns the diagnosis results.

However, neural network inference services in cloud environments face significant privacy challenges. Both the client's data and the server's model are highly sensitive and cannot be shared openly due to privacy regulations and competitive advantage. To address these privacy concerns, various approaches have been explored, including secure multi-party computation (MPC) [29], homomorphic encryption (HE) [17], and trusted execution environments (TEE) [10]. Compared to HE-based approaches, MPC-based approaches have lower computational overhead. Additionally, unlike TEE-based approaches, MPC-based approaches do not rely on specialized hardware and offer provable security.

With the increasing size of neural network models, binarized neural networks (BNNs) with binary weights and activations (i.e., -1 or $+1$) have been proposed to address this issue, demonstrating considerable progress in recent years. Consequently, MPC-based BNN inference has gained significant attention. Existing MPC-based BNN inference frameworks require weights and activations to be encoded as either Boolean or fixed-point values [24,30]. Boolean encoding uses Boolean circuits to evaluate BNNs; however, BNNs involve numerous arithmetic operations (addition and multiplication), which are not well-suited for representation by Boolean circuits, resulting in high circuit complexity and increased communication costs. Fixed-point encoding requires a bitwidth expansion algorithm to extend the bitwidths of weights, undermining the efficiency advantages of BNNs. Additionally, BNN inference involves numerous non-linear layers, such as binary activation and max pooling layers. Current approaches use garbled circuits (GC) [29], secret sharing (SS) [13], or homomorphic encryption (HE) [17] to securely compute these non-linear operations [24,12,15], often incurring high communication or computation overhead.

To address the challenges in secure BNN inference, we propose FSSiBNN, a framework featuring free bitwidth conversion based on function secret sharing (FSS) [6,7]. By leveraging FSS, which supports arbitrary input and output bitwidths, we introduce a bitwidth-reduced parameter encoding scheme. This scheme seamlessly integrates bitwidth conversion into FSS-based secure binary activation and max pooling protocols, thereby eliminating the additional communication overhead typically associated with bitwidth conversion. Furthermore, we optimize the computation by combining and converting multiple BNN layer functions into fewer matrix multiplication and comparison operations. We also design secure BNN layer function computation protocols in the offline-online computation paradigm [13]. In the offline phase, we precompute matrix multiplication tuples for matrix multiplication operations and FSS keys for comparison operations. During the online phase, these precomputed elements are utilized, enabling constant-round online inference with low computational complexity.

We conducted experiments on various datasets and BNN models, comparing our results with state-of-the-art frameworks: XONN [24], SecureBiNN [30], and FLEXBNN [15]. The experimental results demonstrate that FSSiBNN outperforms these frameworks in both communication efficiency and inference time.

1.1 Related Work

Secure neural network inference based on MPC [23,27,16] has been a vibrant area of research in recent years. With the advancements in quantized neural networks, secure quantized neural network inference has also garnered significant attention. Our focus is specifically on secure BNN inference based on MPC.

FHE-DiNN [4] is the first to propose the secure quantized neural network inference, relying on computationally expensive homomorphic encryption (HE) techniques. XONN [24] is the first MPC-based secure quantized neural network inference framework targeting BNNs (1-bit quantization). Subsequently, FOBNN [9] further optimizes its performance. QUOTIENT [1] enables secure inference of ternarized neural networks (2-bit quantization), while SecureQ8 [12] focuses on 8-bit and 16-bit quantization. ABNN^2 [25] supports arbitrary-bitwidth quantized neural network inference. However, these frameworks often suffer from high computation overhead due to the use of HE or significant communication costs due to the use of GC, resulting in communication costs two orders of magnitude higher and run-time one order of magnitude higher than secret sharing-based approaches.

Leia [21] presents a secure BNN inference framework based on additive secret sharing (SS). Similarly, BANNERS [18] and SecureBiNN [30] tackle secure BNN inference by leveraging replicated secret sharing (RSS). These frameworks utilize circuit conversion between Boolean and arithmetic or sharing conversion between SS and RSS to evaluate BNNs, leading to additional communication rounds and high communication overhead. Leia [21] requires two non-colluding computational servers, while BANNERS [18] and SecureBiNN [30] operate in an honest-majority setting, which imposes a stronger security assumption than two-party computation (2PC) frameworks that assume a dishonest-majority.

Recently, a concurrent work, FLEXBNN [15], employs non-uniform bitwidth equipped with a seamless bitwidth conversion method and designs several specific optimizations for the basic operations. FLEXBNN [15] operates in a three-server setting (non-colluding and honest majority) and uses RSS-based MPC with online communication rounds linear in the multiplicative depth of the circuit. In contrast, our framework adopts a client-server setting (collusion-resistant and dishonest majority) and leverages FSS-based MPC, ensuring constant online communication rounds. Thus, our work and FLEXBNN [15] differ fundamentally in terms of problem settings and protocol assumptions.

Table 1 provides a comparison of secure quantized neural network inference frameworks based on MPC. Our framework can resist collusion attacks and supports non-uniform bitwidth arithmetic in a dishonest-majority setting.

Table 1. The secure quantized neural network inference frameworks based on MPC

Framework	Num.	Crypto.	Adversary	Dishonest Majority	Collusion Resistance*	Non-Uniform Bitwidth**
XONN [24]	2	GC	Semi-Honest	✓	✓	×
QUOTIENT [1]	2	GC	Semi-Honest	✓	✓	×
ABNN^2 [25]	2	GC	Semi-Honest	✓	✓	×
Leia [21]	2	SS	Semi-Honest	✓	×	×
FOBNN [9]	2	GC	Semi-Honest	✓	✓	×
BANNERS [18]	3	SS, RSS	Malicious	×	×	×
SecureBiNN [30]	3	SS, RSS	Semi-Honest	×	×	✓
FLEXBNN [15]	3	SS, RSS	Semi-Honest	×	×	✓
SecureQ8 [12]	N	SS, HE	Malicious	×	×	×
FSSiBNN (Ours)	2	SS, FSS	Semi-Honest	✓	✓	✓

* Whether it is secure against collusion attacks.
** Whether it supports secure non-uniform bitwidth arithmetic.

1.2 Our Contributions

In this work, we propose FSSiBNN, an FSS-based secure inference framework for BNNs, enabling the server to provide inference services to the client without compromising the privacy of either the server's model or the client's data.

Our contributions can be summarized as follows:

- **Secure BNN Inference with Free Bitwidth Conversion**. To address the problems of existing work that cannot effectively support secure non-uniform bitwidth computation and requires high overhead during the bitwidth conversion process, we leverage the property of FSS that supports arbitrary input and output bitwidths to propose a bitwidth-reduced parameter encoding scheme with free bitwidth conversion. We naturally embed the bitwidth conversion into the FSS-based secure binary activation and max pooling protocols, thereby avoiding the additional computational and communication overhead introduced by bitwidth conversion.
- **Constant-Round Online Inference based on FSS**. To solve the problems of high latency in BNN inference and high communication costs in secure BNN layer computation protocols, we combine and convert multiple BNN layer functions into fewer matrix multiplication and comparison operations. By precomputing matrix multiplication tuples for matrix multiplication and FSS keys for comparison in the offline phase, we achieve constant-round online inference with low computational complexity.

2 Preliminaries

Notations. Let $\mathbb{Z}_{2^n}$ be a ring with each element identified by its n-bit binary representation. Unless otherwise specified, we parse $x \in \{0,1\}^n$ as $x_{n-1}||\cdots||x_0$

where $||$ denotes string concatenation and x_{n-1} is the most significant bit (MSB). For $0 \leq i < j \leq n$, $x_{[i]} \in \mathbb{Z}_2$ denotes x_i and $x_{[i,j)} \in \mathbb{Z}_{2^{j-i}}$ denotes the ring element corresponding to the bit-string $x_{j-1}|| \cdots ||x_i$. Denote scalar, vector, and matrix by lowercase letter x, lowercase bold letter $\mathbf{x}$, and uppercase bold letter $\mathbf{X}$, respectively. Let $\mathbf{X}_{ij}$ denote the element at the i-th row and j-th column in matrix $\mathbf{X}$. Denote random sampling by $\in_R$, the security parameter by λ, and $\mathbf{1}\{b\}$ by the indicator function that outputs 1 when b is true and 0 otherwise.

2.1 Binarized Neural Networks

Binarized neural networks (BNNs) are a subtype of neural networks with binary weights and activations (i.e., $\{-1, +1\}$) [11]. A BNN is composed of multiple layers, such as fully connected, convolutional, binary activation, batch normalization, and pooling. The following is a brief description of the specific operations in each layer function of a BNN, with an emphasis on bitwidth representation.

Fully Connected and Convolutional Layers. Fully connected (FC) and convolutional (Conv) layers, called linear layers, perform linear combinations of the inputs and binary weights. Given the l-th layer's input $\mathbf{X}^{(l-1)}$ and binary weight $\mathbf{W}^{(l)}$ (for simplicity, we assume a bias $\mathbf{B}$ is already embedded in $\mathbf{W}$), FC can be computed as matrix multiplication $\mathbf{W}^{(l)} \times \mathbf{X}^{(l-1)}$. Conv can also be implemented as matrix multiplication using an unrolling technique. In the input layer ($l = 1$), the linear layer's input $\mathbf{X}^{(0)}$ is usually a floating-point number and needs to be converted to a fixed-point integer using fixed-point encoding (refer to Section 2.1 in [26]). In the hidden and output layers ($2 \leq l \leq L$), the linear layer's input $\mathbf{X}^{(l-1)}$ takes the value $\{-1, +1\}$.

Batch Normalization and Binary Activation Layers. Batch normalization (BN) layers usually follow linear layers to normalize the output. The operation is defined as $y = \gamma \frac{x-\mu}{\sqrt{\sigma^2+\epsilon}} + \beta$, where γ and β are learnable parameters, μ and σ are parameters determined during the training process, and ϵ is a small positive constant. During the inference process, the parameters of batch normalization are fixed values (usually fixed-point numbers). Thus, the batch normalization operation can be rewritten as $y = \gamma' x + \beta'$ where $\gamma' = \frac{\gamma}{\sqrt{\sigma^2+\epsilon}}$ and $\beta' = \beta - \frac{\mu\gamma}{\sqrt{\sigma^2+\epsilon}}$.

The binary activation (BA) layer follows the BN layer, and its operation is equivalent to the sign function, which is defined as:

$$\mathsf{Sign}(x) = \begin{cases} +1 & x \geq 0 \\ -1 & x < 0 \end{cases} \tag{1}$$

It can be seen that the output of the BA layer is constrained to $\{-1, +1\}$.

Max Pooling Layers. Pooling layers usually follow BA layers and are used to reduce the dimensions of outputs. Max pooling and average pooling are two of the more commonly used pooling methods. We adopt max pooling (Maxpool) which uses the maximum value of each cluster of neurons in the feature map.

2.2 Additive Secret Sharing

Additive secret sharing is a cryptographic method of dividing a secret into multiple parts, where the sum of all parts reconstructs the original secret, but individual parts reveal no information about it. In a 2-out-of-2 secret sharing [14], party P_0 and P_1, with secret shares $\langle x\rangle_0^n$ and $\langle x\rangle_1^n$ respectively, share the secret value $x \in \mathbb{Z}_{2^n}$, such that $x = (\langle x\rangle_0^n + \langle x\rangle_1^n) \bmod 2^n$. We say that P_0 and P_1 hold $\langle x\rangle^n$, meaning that P_0 holds $\langle x\rangle_0^n$ and P_1 holds $\langle x\rangle_1^n$.

Sharing and Reconstruction. To realize the functionality $\mathcal{F}_{\mathsf{Share}}$ which additively shares a secret value $x \in \mathbb{Z}_{2^n}$, protocol Π_{Share} works as follows: the secret owner samples a random value $r \in \mathbb{Z}_{2^n}$, and sends $\langle x\rangle_b^n = (x - r) \bmod 2^n$ to P_b and sends $\langle x\rangle_{1-b}^n = r$ to P_{1-b}. To realize the functionality $\mathcal{F}_{\mathsf{Recon}}$ which reconstructs an additively shared value $\langle x\rangle^n$, protocol Π_{Recon} works as follows: P_b sends $\langle x\rangle_b^n$ to P_{1-b}, who computes $(\langle x\rangle_0^n + \langle x\rangle_1^n) \bmod 2^n$ for $b \in \{0, 1\}$. In the following text, we omit the modular operation for simplicity.

Addition and Multiplication. Functionalities $\mathcal{F}_{\mathsf{Add}}$ and $\mathcal{F}_{\mathsf{Mul}}$ add and multiply two shared values $\langle x\rangle^n$ and $\langle y\rangle^n$ respectively. It is easy to non-interactively add the shared values by having P_b compute $\langle z\rangle_b^n = \langle x\rangle_b^n + \langle y\rangle_b^n$. To realize $\mathcal{F}_{\mathsf{Mul}}$, taking the advantage of Beaver's precomputed multiplication triples technique [3], the specific protocol Π_{Mul} works as follows: assume that P_0 and P_1 hold multiplication triples $\langle u\rangle^n, \langle v\rangle^n, \langle uv\rangle^n$ where $u, v \in_R \mathbb{Z}_{2^n}$, P_b locally computes $\langle e\rangle_b^n = \langle x\rangle_b^n - \langle u\rangle_b^n$ and $\langle f\rangle_b^n = \langle y\rangle_b^n - \langle v\rangle_b^n$ and then the two parties reconstruct $\langle e\rangle^n, \langle f\rangle^n$ to get e, f. Finally, P_b lets $\langle z\rangle_b^n = b \cdot e \cdot f + f \cdot \langle u\rangle_b^n + e \cdot \langle v\rangle_b^n + \langle uv\rangle_b^n$.

2.3 Function Secret Sharing

A two-party function secret sharing (FSS) scheme [8,5] splits a function $f \in \mathcal{F}$ into two shares f_0, f_1 such that (1)each f_b hides f; (2)for each input x, $f_0(x) + f_1(x) = f(x)$. A two-party FSS scheme consists of the key generation algorithm Gen and the function evaluation algorithm Eval. We directly follow the definition of the algorithms $(\mathsf{Gen}, \mathsf{Eval})$ in [5].

Distribute Comparison Function (DCF). A comparison function $f^{<}_{\alpha,\beta}(x) : \mathbb{Z}_{2^m} \rightarrow \mathbb{Z}_{2^n}$ outputs β if $x < \alpha$ and 0 otherwise, where $x, \alpha \in \mathbb{Z}_{2^m}$ and $\beta \in \mathbb{Z}_{2^n}$. We refer to an function secret sharing scheme for comparison functions as distributed comparison function (DCF). And the variant of DCF, called dual distributed comparison function (DDCF), is considered and denoted by $f^{<}_{\alpha,\beta_1,\beta_2}(x)$ that outputs β_1 for $0 \leq x < \alpha$ and β_2 for $x \geq \alpha$. Obviously, $f^{<}_{\alpha,\beta_1,\beta_2}(x) = \beta_2 + f^{<}_{\alpha,\beta_1-\beta_2}(x)$ and thus DDCF can be constructed by DCF.

FSS-based Secure Two-party Computation. Recent work by Boyle et al. [8,5] shows that FSS can be used to efficiently evaluate some function families within the offline-online computation paradigm [13]. Specifically, Gen and Eval correspond to the offline and online phases, respectively. In the offline phase, a trusted dealer randomly samples a mask r^{in} for each input wire w_{in} and $\mathsf{r}^{\mathsf{out}}$ for each output wire w_{out} in the computation circuit. For each gate g with w_{in} and w_{out}, the dealer constructs the *offset function* $g^{[\mathsf{r}^{\mathsf{in}},\mathsf{r}^{\mathsf{out}}]}(x) := g(x - \mathsf{r}^{\mathsf{in}}) + \mathsf{r}^{\mathsf{out}}$, and runs Gen to generate FSS keys (k_0, k_1) corresponding to $g^{[\mathsf{r}^{\mathsf{in}},\mathsf{r}^{\mathsf{out}}]}$. Then the dealer sends k_b to P_b and the corresponding mask r to P_b for circuit input and output wires w owned by P_b. In the online phase, P_b calculates the masked wire value $\hat{x} = x + \mathsf{r}^{\mathsf{in}}$ for each w_{in} with r^{in} owned by P_b, and sends it to P_{1-b}. Starting from the input gates, P_0 and P_1 compute gates in topological order to obtain masked output wire values. To process a gate g with w_{in} and w_{out}, P_b uses Eval with FSS key k_b and the masked input wire value $\hat{x} = x + \mathsf{r}^{\mathsf{in}}$ to obtain the masked output wire value $g(x) + \mathsf{r}^{\mathsf{out}}$. For output wires, they subtract the corresponding mask received from the dealer to obtain the plaintext output values.

3 Secure BNN Inference Framework

We present our FSSiBNN framework for secure BNN inference in Section 3.1, which includes two submodules, described in Section 3.2 and Section 3.3.

3.1 The FSSiBNN Overview

In inference as a service, a client $\mathcal{C}$ provides data to a cloud server $\mathcal{S}$, which performs the inference using the pre-trained models and returns the result to $\mathcal{C}$. To ensure the privacy of both the client's data and the server's model, we introduce our FSSiBNN framework for secure BNN inference. As shown in Fig. 1, FSSiBNN works as follows: $\mathcal{S}$ and $\mathcal{C}$ first input the model and data respectively, then perform secure BNN inference, and finally, $\mathcal{C}$ receives the inference results.

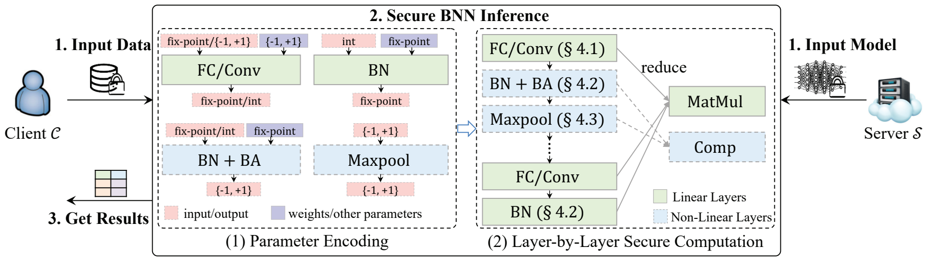

Fig. 1. The overview of FSSiBNN framework

In secure BNN inference, after receiving the data and model, FSSiBNN determines the data control flow and encodes parameters such as inputs, outputs, and model weights. Once the parameters' bitwidths are determined, each BNN layer's operations are computed sequentially to generate the final inference result. Therefore, our secure BNN inference module is divided into (1) the parameter encoding submodule and (2) the layer-by-layer secure computation submodule.

The *parameter encoding submodule* encodes inputs, model parameters, and outputs of each BNN layer to determine their bitwidths. These parameters have different ranges and precisions: the client's inference input is usually a fixed-point integer (comprising integer and fractional parts), the server's model weights are binary (i.e., $\{-1, +1\}$), batch normalization parameters are fixed-point integers, and binary activation outputs are binary. Additionally, some layers' inputs and outputs are integers (without fractional parts). Therefore, it is necessary to design protocols that support secure computation with non-uniform bitwidths and bitwidth conversion to accommodate these different ranges and precisions.

The *layer-by-layer secure computation submodule* computes each BNN layer by designing secure computation protocols for each BNN operator, presented in Section 4. Specifically, we reduce linear layers (fully connected, convolutional, and batch normalization layers) to matrix multiplication (MatMul) and non-linear layers (binary activation and max pooling layers) to comparison (Comp), as illustrated in Fig. 1 (2). Additionally, we combine some BNN layers, such as batch normalization and binary activation layers, for computation. By leveraging the offline-online computation paradigm, matrix multiplication can be implemented with one online communication round. However, it is challenging to design secure comparison protocols that enable online-efficient secure inference.

3.2 Bitwidth-reduced Parameter Encoding Scheme with Free Bitwidth Conversion

The weights and activations of BNNs are constrained to $\{-1, +1\}$, allowing BNN operators to be computed with a small bitwidth. To take advantage of these small bitwidths, prior work [24,18,21] uses Boolean circuits or Boolean-arithmetic circuits to evaluate BNNs. However, BNN inference involves a large number of arithmetic operations that are not suitable for computation with Boolean circuits, resulting in significant communication overhead.

We propose a bitwidth-reduced parameter encoding scheme, which not only represents these parameters with an appropriate and small bitwidth, but also uses secure non-uniform bitwidth arithmetic to efficiently evaluate BNNs. Fig. 2 illustrates our scheme. For simplicity, we slightly abuse the terminology and refer to the entire first layer as the input layer.

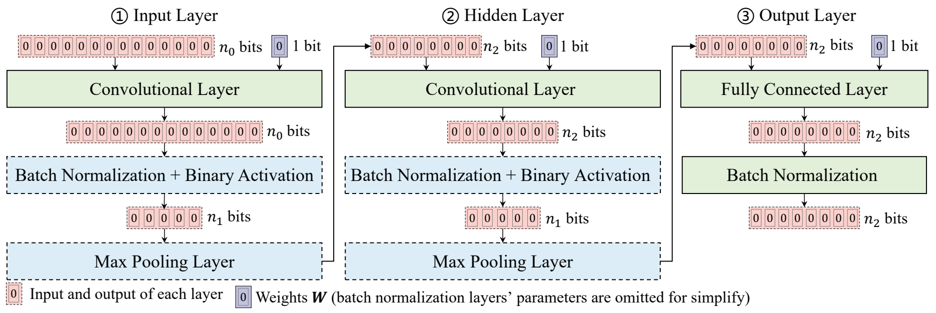
① Input Layer
② Hidden Layer
③ Output Layer
n_0 bits
1 bit
n_2 bits
n_1 bits
Convolutional Layer
Fully Connected Layer
Batch Normalization + Binary Activation
Batch Normalization
Max Pooling Layer
Input and output of each layer
Weights $\boldsymbol{W}$ (batch normalization layers' parameters are omitted for simplify)

Fig. 2. Bitwidth-reduced parameter encoding scheme applied to an example BNN

We use 1 bit to represent the binary weight $\mathbf{W}$ and an appropriate and small bitwidth to represent other values, usually depending on the range of values and the bitwidth required in layer function computations. Specifically, n_0, n_1, and n_2 represent the output bitwidths of different layers in BNN. Firstly, the input layer receives fixed-point inference input alongside binary weights, typically necessitating a long bitwidth n_0 (e.g., $n_0 = 32$ bits) for encoding fixed-point values. After computing the convolutional layer, the output remains fixed point, thus requiring the representation bitwidth n_0. Secondly, after processing through the batch normalization and binary activation layers, the output becomes binary (i.e., $+1, -1$). Considering that the output will serve as the input of the max pooling layer and requires further computation, we use a small bitwidth n_1 (e.g., $n_1 = 8$ bits) instead of one bit to represent it. Finally, the output of the max pooling layer, which is also binary, serves as the input for the next convolutional layer, and its kernel size dictates the output value range, typically necessitating a medium bitwidth n_2 (e.g., $n_2 = 16$ bits), and so forth for the remaining layers.

Furthermore, as illustrated in Fig. 2, the bitwidths are required to be frequently converted. For example, bitwidth n_0 is converted to bitwidth n_1 in the input layer, and bitwidth n_2 is converted to bitwidth n_1 in the hidden layer. Observer that bitwidth conversion occurs during the computation of binary activation and max pooling. In FSSiBNN, secure binary activation and max pooling protocols are reduced to secure comparison protocols and implemented using function secret sharing (FSS). The construction of FSS supports arbitrary input and output bitwidths [5], allowing bitwidth conversion to be embedded into the FSS-based comparison protocol. Based on these observations, we propose a free bitwidth conversion scheme that avoids introducing additional overhead by naturally embedding bitwidth conversion into the FSS-based secure activation and max pooling (see Section 4.2 and Section 4.3).

Comparison with SecureBiNN [30]. SecureBiNN analyzes the parameter range in BNNs and uses small bitwidths to represent these parameters. In SecureBiNN, to facilitate computing fully connected and convolutional layers, the model weights are not represented using 1 bit like in our scheme but are encoded with a specific bitwidth. For example, the weights in the hidden layer are encoded as 14, 15, or 17 bits (see Section 4 of [30]), incurring additional communication overhead.

Comparison with FLEXBNN [15]. A concurrent work, FLEXBNN, also proposes a similar flexible bitwidth scheme and implements bitwidth conversion through Boolean-arithmetic share conversion when computing binary activation. FLEXBNN actually transfers the overhead of bitwidth conversion to the process of share conversion, which requires linear communication rounds and incurs high communication costs. Moreover, its techniques are applicable to a three-server setting and cannot be used in the client-server setting.

3.3 Online-efficient Secure Non-linear BNN Layers via FSS

BNNs involve many non-linear layers, such as binary activation and max pooling layers. In prior implementations [24,30,15], the main source of inefficiency is that secure non-linear layer computation protocols require linear communication rounds, which incurs high communication costs. To address this, we design constant-round secure computation protocols by leveraging FSS.

We assume that a well-trained BNN model is known to the server $\mathcal{S}$ in advance, allowing us to precompute the correlated randomness for matrix multiplication and comparison operations. Specifically, the process is as follows:

- In the offline phase, $\mathcal{C}$ and $\mathcal{S}$ precompute the correlated randomness. For matrix multiplication, $\mathcal{C}$ first chooses $\mathbf{R}$ randomly, and runs a two-party protocol with $\mathcal{S}$ to gets $\mathbf{U}$ and $\mathcal{S}$ gets $\mathbf{V}$ where $\mathbf{U}+\mathbf{V}=\mathbf{W}\times\mathbf{R}$. $(\mathbf{U},\mathbf{V})$ is called *matrix multiplication tuples*. For comparison, $\mathcal{C}$ and $\mathcal{S}$ engage in a FSS key generation protocol $\mathsf{Gen}_m^<(\cdot)$ to generate *FSS keys* (k_0,k_1).
- In the online phase, $\mathcal{C}$ with inference input $\mathbf{X}$ compute layer functions with $\mathcal{S}$. For multiplication, $\mathcal{C}$ lets the share of $\mathbf{W}\times\mathbf{X}$ be $\langle\mathbf{Z}\rangle_0^n=\mathbf{U}$ and sends $\mathbf{X}-\mathbf{R}$ to $\mathcal{S}$ who computes $\langle\mathbf{Z}\rangle_1^n=\mathbf{W}\times(\mathbf{X}-\mathbf{R})+\mathbf{V}$. For comparison, $\mathcal{S}$ and $\mathcal{C}$ open $\hat{x}=x+r$ where x is the input and r is the mask, and then respectively compute $\mathsf{Eval}_m^<(\cdot)$ locally to get the share of $\mathsf{Sign}(x)$.

4 Secure BNN Inference Protocol

As discussed in Section 2.1, a BNN comprises multiple layers. In this section, we sequentially present the secure computation protocols for each layer of BNNs.

4.1 Secure Fully Connected and Convolutional Layers

The fully connected and convolutional layers can be computed using matrix multiplication. Secure fully connected layers (FC) and secure convolutional layers (Conv) can be reduced to secure matrix multiplication (MatMul). Given the binary weight $\mathbf{W}^B\in\mathbb{Z}_2^{d_1\times d_2}$ and the input $\mathbf{X}\in\mathbb{Z}_{2^n}^{d_2\times d_3}$, where $\mathbf{W}^B$ is the 0-1 encoding of $\mathbf{W}$, the functionality $\mathcal{F}_{\mathsf{MatMul}}$ computes $\mathbf{W}\times\mathbf{X}\in\mathbb{Z}_{2^n}^{d_1\times d_3}$. To realize $\mathcal{F}_{\mathsf{MatMul}}$ and further realize $\mathcal{F}_{\mathsf{FC}}$ and $\mathcal{F}_{\mathsf{Conv}}$, we present protocol Π_{MatMul}, which is divided into an offline phase and an online phase.

Offline Phase. In the offline phase, the functionality $\mathcal{F}_{\mathsf{Gen}^{\mathsf{mmt}}}$ generates matrix multiplication tuples $(\mathbf{U},\mathbf{V})$ such that $\mathbf{U}+\mathbf{V}=\mathbf{W}\times\mathbf{R}$ and sends $\mathbf{U}$ and $\mathbf{V}$ to $\mathcal{S}$ and $\mathcal{C}$, respectively. To implement $\mathcal{F}_{\mathsf{Gen}^{\mathsf{mmt}}}$, we propose protocol $\Pi_{\mathsf{Gen}^{\mathsf{mmt}}}$ where $\mathcal{C}$ first samples a matrix $\mathbf{R}$ and computes $\mathbf{W}\times\mathbf{R}$ with $\mathcal{S}$, who holds $\mathbf{W}$.

We first compute the shares of the product $w_{ij}\cdot r_{ik}$ in Protocol 1 where w_{ij} is the (i,j)-th element in $\mathbf{W}$ and r_{jk} is the (j,k)-th element in $\mathbf{R}$, and it can be easily extended to compute $\mathbf{W}\times\mathbf{R}$. Since $w_{ij}\in\{-1,+1\}$ is encoded to $w_{ij}^B\in\{0,1\}$ by the bijective mapping $\{-1\leftrightarrow 0,+1\leftrightarrow 1\}$, we need to compute $(-1)^{\neg w_{ij}^B}\cdot r_{jk}$, which can be computed using the 1-out-of-2 correlated oblivious transfer (COT) [2] functionality $\mathcal{F}_{\mathsf{COT}_n}$. Functionality $\mathcal{F}_{\mathsf{COT}_n}$ is defined as follows: the sender inputs an n-bit message $m_0\in\mathbb{Z}_{2^n}$ and a correlation function f, the receiver inputs a choice bit $b\in\{0,1\}$, and the functionality outputs m_b to the receiver, where $m_1=f(m_0)$. $\mathcal{F}_{\mathsf{COT}_n}$ can be implemented by leveraging the VOLE-style OT generation scheme proposed in Ferret [28].

```
Protocol 1 Matrix multiplication tuple generation via COT: $\Pi_{\mathsf{Gen}^{\mathsf{mmt}}}(w_{ij}^B, r_{jk})$
Input: $w_{ij}^B$ be the $(i,j)$-th element in $\mathbf{W}^B$ and $r_{jk}$ be the $(j,k)$-th element in $\mathbf{R}$.
Output: $\mathcal{S}$ and $\mathcal{C}$ get $u$ and $v$ respectively, where $u + v = (-1)^{\neg w_{ij}^B} \cdot r_{jk} = w_{ij} \cdot r_{jk}$.
1: $\mathcal{C}$ chooses $s_j \in \mathbb{Z}_{2^n}$ randomly and sets the correlation function of $\mathcal{F}_{\mathsf{COT}_n}$ to $f_j(x) = -(2s_j + x) \bmod 2^n$, and sets $m_0 = -(r_{jk} + s_j)$; $\mathcal{S}$ sets the choose bit $b = w_{ij}^B$.
2: $\mathcal{C}$ and $\mathcal{S}$ run $(\perp; m_b) \leftarrow \mathcal{F}_{\mathsf{COT}_n}(m_0, f_j(x); b)$ where $m_b = (-1)^{\neg w_{ij}^B} \cdot r_{jk} - s_j$.
3: $\mathcal{S}$ let $u = (-1)^{\neg w_{ij}^B} \cdot r_{jk} - s_j$ and $\mathcal{C}$ let $v = s_j$.
```

Online Phase. In the online phase, $\mathcal{S}$ and $\mathcal{C}$ perform matrix multiplication by using the matrix multiplication tuples generated in the offline phase. Note that the input $\mathbf{X}^{(0)} = \mathbf{X}$ is held by $\mathcal{C}$ in the input layer ($l = 1$), and the input $\mathbf{X}^{(l-1)}$ is shared between $\mathcal{C}$ and $\mathcal{S}$ in the hidden and output layers ($l = 2, \cdots, L$). Therefore, there are two different procedures for different layers:

– In the input layer, $\mathcal{C}$ holds $\mathbf{X}^{(0)}$ and $\mathcal{S}$ holds $\mathbf{W}^{(1)}$. Given a matrix multiplication tuple $(\mathbf{U}, \mathbf{V})$ such that $\mathbf{U} + \mathbf{V} = \mathbf{W}^{(1)} \times \mathbf{R}^{(0)}$, $\mathcal{C}$ sends $\mathbf{X}^{(0)} - \mathbf{R}^{(0)}$ to $\mathcal{S}$ and lets $\langle \mathbf{Z}^{(1)} \rangle_0^n = \mathbf{V}$ and $\mathcal{S}$ lets $\langle \mathbf{Z}^{(1)} \rangle_1^n = \mathbf{W}^{(1)} \times (\mathbf{X}^{(0)} - \mathbf{R}^{(0)}) + \mathbf{U}$.
– In the hidden and output layers, $\mathcal{C}$ and $\mathcal{S}$ hold the share $\langle \mathbf{X}^{(l-1)} \rangle^n$. Given matrix multiplication tuple $(\mathbf{U}, \mathbf{V})$ such that $\mathbf{U} + \mathbf{V} = \mathbf{W}^{(l)} \times \mathbf{R}^{(l-1)}$, $\mathcal{C}$ sends $\langle \mathbf{X}^{(l-1)} \rangle_0^n - \mathbf{R}^{(l-1)}$ to $\mathcal{S}$ and let $\langle \mathbf{Z}^{(l)} \rangle_0^n = \mathbf{V}$, and $\mathcal{S}$ lets $\langle \mathbf{Z}^{(l)} \rangle_1^n = \mathbf{W}^{(l)} \times ((\langle \mathbf{X}^{(l-1)} \rangle_0^n - \mathbf{R}^{(l-1)}) + \langle \mathbf{X}^{(l-1)} \rangle_1^n) + \mathbf{U} = \mathbf{W}^{(l)} \times (\mathbf{X}^{(l-1)} - \mathbf{R}^{(l-1)}) + \mathbf{U}$.

For a matrix multiplication, it requires 2 rounds with $d_1 d_2 (\lambda + n d_3)$ bits of communication in the offline phase, and 1 round with $d_2 d_3 n$ bits of communication in the online phase.

4.2 Secure Batch Normalization and Binary Activation Layers

In the input and hidden layers, the batch normalization layer is followed by the binary activation layer, whereas in the output layer, batch normalization appears alone. Therefore, we propose the secure batch normalization protocol (Π_{BN}) for the output layer. For the input and hidden layers, we combine the binary activation and batch normalization layers to propose the secure binary activation and batch normalization protocol (Π_{BNBA}).

– Secure batch normalization Π_{BN}: Given input share $\langle x \rangle^n$ and the parameters γ' and β', $\mathcal{F}_{\mathsf{BN}}$ computes $\gamma' x + \beta'$. It is easy to realize $\mathcal{F}_{\mathsf{BN}}$ by performing $\Pi_{\mathsf{BN}}(\langle x \rangle^n) = \Pi_{\mathsf{Mul}}(\langle x \rangle^n, \gamma') + \beta'$ where Π_{Mul} is secure multiplication protocol.
– Secure binary activation and batch normalization Π_{BNBA}: Given input share $\langle x \rangle^n$ and the parameters γ' and β', $\mathcal{F}_{\mathsf{BNBA}}$ computes $\mathsf{BNBA}(x) = \mathsf{BA}(\mathsf{BN}(x)) = \mathsf{BA}(\gamma' x + \beta')$ where $\mathsf{BA}(x) = \mathsf{Sign}(x)$. It holds that $\mathsf{BA}(\gamma' x + \beta') = \mathsf{BA}(x + \frac{\beta'}{\gamma'})$ since γ' is positive. To realize $\mathcal{F}_{\mathsf{BNBA}}$, Π_{BNBA} is proposed as follows:
  • In the input layer, x and $\frac{\beta'}{\gamma'}$ are both fixed-point numbers, and $\Pi_{\mathsf{BNBA}}(\langle x \rangle^n) = \Pi_{\mathsf{BA}}(\langle x + \frac{\beta'}{\gamma'} \rangle^n)$.

- In the hidden layers, x is an integer and $\frac{\beta'}{\gamma'}$ is a fixed-point number. In this case, $\mathsf{BA}(x+\frac{\beta'}{\gamma'})$ is equivalent to $\mathsf{BA}(x-\lceil -\frac{\beta'}{\gamma'}\rceil)$. Thus, $\Pi_{\mathsf{BNBA}}(\langle x\rangle^n) = \Pi_{\mathsf{BA}}(\langle x-\lceil -\frac{\beta'}{\gamma'}\rceil\rangle^n)$.

Therefore, the protocol Π_{BNBA} can be reduced to the protocol Π_{BA}. To implement Π_{BA}, we propose a distributed comparison function (DCF, defined in Section 2.3) scheme for the sign function (i.e., binary activation function, see Eq. (1)) and then design an online-efficient secure binary activation protocol.

Sign Function Gate. To construct the DCF for the sign function, we present the sign function gate $\mathcal{G}_{\mathsf{sign}}$. $\mathcal{G}_{\mathsf{sign}}$ is the family of functions $g_{\mathsf{sign},m,n} : \mathbb{S}_{2^m} \to \mathbb{S}_{2^n}$, given by $g_{\mathsf{sign},m,n}(x) := 1 - 2\cdot \mathbf{1}\{x<0\}$, where $\mathbb{S}_{2^m}$ and $\mathbb{S}_{2^n}$ are the signed m-bit and n-bit integer sets and $x\in\mathbb{S}_{2^m}$. We denote the corresponding offset gate class by $\hat{\mathcal{G}}_{\mathsf{sign}}$, and its component offset functions by $\hat{g}^{[\mathsf{r}^{\mathsf{in}},\mathsf{r}^{\mathsf{out}}]}_{\mathsf{sign},m,n} : \mathbb{U}_{2^m} \to \mathbb{U}_{2^n}$, where $\mathbb{U}_{2^m}$ and $\mathbb{U}_{2^n}$ are the unsigned m-bit and n-bit integer sets and $\mathsf{r}^{\mathsf{in}}\in\mathbb{U}_{2^m}$, $\mathsf{r}^{\mathsf{out}}\in\mathbb{U}_{2^n}$. Given an unsigned integer $x\in\mathbb{U}_{2^m}$ and a signed integer $x'\in\mathbb{S}_{2^m}$ such that $(x-\mathsf{r}^{\mathsf{in}}) \bmod 2^m = x' \bmod 2^m$, it holds that $(\hat{g}^{[\mathsf{r}^{\mathsf{in}},\mathsf{r}^{\mathsf{out}}]}_{\mathsf{sign},m,n}(x) - \mathsf{r}^{\mathsf{out}}) \bmod 2^n = g_{\mathsf{sign},m,n}(x') \bmod 2^n = 1-2\cdot(x_{[m-1]}\oplus r_{[m-1]}\oplus c)$, where $r=(2^m-\mathsf{r}^{\mathsf{in}}) \bmod 2^m$ and $c=\mathbf{1}\{2^{m-1}-x_{[0,m-1)}-1<r_{[0,m-1)}\}$. The proof is in Appendix A.

To compute $(\hat{g}^{[\mathsf{r}^{\mathsf{in}},\mathsf{r}^{\mathsf{out}}]}_{\mathsf{sign},m,n}(x)-\mathsf{r}^{\mathsf{out}}) \bmod 2^n = 1-2\cdot(x_{[m-1]}\oplus r_{[m-1]}\oplus c)$, we first compute $c=\mathbf{1}\{2^{m-1}-x_{[0,m-1)}-1<r_{[0,m-1)}\}$ by using the DDCF scheme in Protocol 2 (from BCG+21 [5]), where a distributed comparison function (DCF) scheme $(\mathsf{Gen}^{<}_m(1^\lambda,\alpha,\beta_0-\beta_1,\mathbb{U}_{2^n}),\mathsf{Eval}^{<}_m(b,k_b^{(m)},x))$ is used to computed $(\beta_0-\beta_1)\cdot\mathbf{1}\{x<\alpha\}$. We directly use the DCF scheme proposed in BCG+21 [5] but let $\mathbb{G}^{\mathsf{in}}=\mathbb{U}_{2^m}$ and $\mathbb{G}^{\mathsf{out}}=\mathbb{U}_{2^n}$ to support non-uniform bitwidth computation.

Protocol 2 DDCF from [5]: $(\mathsf{Gen}^{\mathsf{DDCF}}_m, \mathsf{Eval}^{\mathsf{DDCF}}_m)$

```
• $\mathsf{Gen}^{\mathsf{DDCF}}_m(1^\lambda,\alpha,\beta_0,\beta_1,\mathbb{U}_{2^n})$
 1: Compute $(k_0^{(m)},k_1^{(m)}) \leftarrow \mathsf{Gen}^{<}_m(1^\lambda,\alpha,\beta_0-\beta_1,\mathbb{U}_{2^n})$.
 2: Sample $r_0,r_1 \in_R \mathbb{U}_{2^n}$ such that $r_0+r_1=\beta_1$.
 3: Let $k_b = k_b^{(m)}||r_b$ for $b\in\{0,1\}$.
 4: return $(k_0,k_1)$.
• $\mathsf{Eval}^{\mathsf{DDCF}}_m(b,k_b,x)$
 1: Parse $k_b = k_b^{(m)}||r_b$.
 2: Compute $y_b^{(m-1)} \leftarrow \mathsf{Eval}^{<}_m(b,k_b^{(m)},x)$.
 3: return $y_b^{(m-1)}+r_b$.
```

Based on the DDCF scheme in Protocol 2, we propose the sign function gate in Protocol 3. Our sign function gate scheme is similar to the signed integer comparison gate scheme in BCG+21 [5] (see Fig. 8 in BCG+21 [5]), but the scheme in BCG+21 [5] only supports uniform bitwidth signed integer comparison. Our sign function gate Sign supports non-uniform bitwidth computation. The sign function gate Sign requires 1 call to DDCF, and the total key sizes are $(m-1)(\lambda+n+2)+\lambda+n$ bits per party.

Protocol 3 Sign function gate Sign:$(\mathsf{Gen}^{\mathsf{Sign}}_{m,n}, \mathsf{Eval}^{\mathsf{Sign}}_{m,n})$

```
• $\mathsf{Gen}^{\mathsf{Sign}}_{m,n}(1^\lambda, \mathsf{r}^{\mathsf{in}}, \mathsf{r}^{\mathsf{out}})$
1: Let $r = (2^m - \mathsf{r}^{\mathsf{in}}) \bmod 2^m$, and $\alpha^{(m-1)} = r_{[0,m-1)}$.
2: $(k_0^{(m-1)}, k_1^{(m-1)}) \leftarrow \mathsf{Gen}^{\mathsf{DDCF}}_{m-1}(1^\lambda, \alpha^{(m-1)}, \beta_0, \beta_1, \mathbb{U}_{2^n})$, where $\beta_0 = 1 \oplus r_{[m-1]} \in \mathbb{U}_{2^n}, \beta_1 = r_{[m-1]} \in \mathbb{U}_{2^n}$.
3: Sample randoms $\mathsf{r}_0, \mathsf{r}_1 \in \mathbb{U}_{2^n}$ such that $\mathsf{r}_0 + \mathsf{r}_1 = \mathsf{r}^{\mathsf{out}}$.
4: Let $k_b = k_b^{(m-1)} || \mathsf{r}_b$ for $b \in \{0,1\}$.
5: return $(k_0, k_1)$.
• $\mathsf{Eval}^{\mathsf{Sign}}_{m,n}(b, k_b, x)$
1: Parse $k_b = k_b^{(m-1)} || \mathsf{r}_b$.
2: $z_b^{(n-1)} \leftarrow \mathsf{Eval}^{\mathsf{DDCF}}_{m-1}(b, k_b^{(m-1)}, x^{(m-1)})$, where $x^{(m-1)} = 2^{m-1} - x_{[0,m-1)} - 1$.
3: Let $v_b = b - 2 \cdot (b \cdot x_{[m-1]} + z_b^{(n-1)} - 2 \cdot x_{[m-1]} \cdot z_b^{(n-1)}) + \mathsf{r}_b \in \mathbb{U}_{2^n}$.
4: return $v_b$.
```

Secure Binary Activation. Based on the sign function gate Sign in Protocol 3, we propose protocol Π_{BA} to implement the secure binary activation functionality $\mathcal{F}_{\mathsf{BA}}$, which computes Eq. (1). The protocol Π_{BA} is detailed in Protocol 4, which calls 1 Sign instance (the key sizes is $(m-1)(\lambda+n+2)+\lambda+n$ bits) in the offline phase, and requires 1 round with m bits of communication in the online phase.

Protocol 4 Secure Binary Activation: $\Pi_{\mathsf{BA}}(\langle x\rangle^m)$

```
Input: $\mathcal{S}$ and $\mathcal{C}$ hold $\langle x\rangle^m \in \mathbb{U}_{2^m}$.
Output: $\mathcal{S}$ and $\mathcal{C}$ hold $\langle z\rangle_b^n \in \mathbb{U}_{2^n}$ such that $\langle z\rangle_0^n + \langle z\rangle_1^n = \mathsf{Sign}(x)$.
• Offline Phase
1: Compute $(k_0, k_1) \leftarrow \mathsf{Gen}^{\mathsf{Sign}}_{m,n}(1^\lambda, \mathsf{r}^{\mathsf{in}}, \mathsf{r}^{\mathsf{out}})$.
2: Send $k_0$, $\langle \mathsf{r}^{\mathsf{in}}\rangle_0^m$, $\langle \mathsf{r}^{\mathsf{out}}\rangle_0^n$ to $\mathcal{C}$, and send $k_1$, $\langle \mathsf{r}^{\mathsf{in}}\rangle_1^m$, $\langle \mathsf{r}^{\mathsf{out}}\rangle_1^n$ to $\mathcal{S}$.
• Online Phase
1: $\mathcal{S}$ and $\mathcal{C}$ run $\Pi_{\mathsf{Recon}}(\langle x\rangle^m, \langle \mathsf{r}_1^{\mathsf{in}}\rangle^m)$ to get $x + \mathsf{r}^{\mathsf{in}}$.
2: $\mathcal{S}$ and $\mathcal{C}$ compute locally $\langle z\rangle_b^n \leftarrow \mathsf{Eval}^{\mathsf{Sign}}_{m,n}(b, k_b, x + \mathsf{r}^{\mathsf{in}}) - \langle \mathsf{r}^{\mathsf{out}}\rangle_b^n$ respectively.
```

4.3 Secure Max Pooling Layers

The max pooling (Maxpool) layer always follows the batch normalization and binary activation (BNBA) layer, so the input of the MaxPool layer is the output of the BNBA layer. To simplify the computation of the secure max pooling ($\mathsf{Maxpool}$), we modify the output of the BNBA layer to let each element be 0 or 1 instead of -1 or $+1$ (this step does not require communication). Then $\mathsf{Maxpool}$ can be calculated via secure addition and secure comparison [30].

Consider the case of a single channel: $\mathcal{F}_{\mathsf{Maxpool}}$ computes output numbers by sliding a window of size $k\times k$ over the input $\mathbf{X}^{d_1\times d_2}$ with stride s (typically $s = k$), and the output is $\mathbf{Z}^{d_1'\times d_2'}$ where $d_1' = \lfloor (d_1 - k)/s + 1\rfloor$ and $d_2' = \lfloor (d_2 - k)/s + 1\rfloor$. To implement $\mathcal{F}_{\mathsf{Maxpool}}$, Π_{Maxpool} invokes the secure addition protocol Π_{Add} in parallel and then invokes the secure comparison protocol Π_{BA} $d_1'd_2'$ times in parallel. Protocol Π_{Add} does not require any communication, and protocol Π_{BA} calls one sign function gate in the offline phase and requires one round with one ring element of communication in the online phase, Π_{Maxpool} invokes $d'_1 d'_2$ sign function gate Sign instances in the offline phase and requires one round with $d'_1 d'_2$ ring elements of communication in the online phase.

5 Theoretical Analysis and Experiment

5.1 Theoretical Analysis

We compare the online and offline communication complexities of FSSiBNN (ours) with the state-of-the-art frameworks XONN [24], SecureBiNN [30], and FLEXBNN [15]. XONN and FSSiBNN are two-party computation (2PC) frameworks, while SecureBiNN and FLEXBNN are three-party computation (3PC) frameworks. A detailed analysis of the online and offline computation complexities is provided in Appendix B.

Online Communication Complexity. In Table 2, we present the online communication complexity of the BNN operators, including MatMul, BN, BNBA, and $\mathsf{Maxpool}$. For round complexity, all BNN operators are calculated with constant online communication rounds in FSSiBNN, while linear functions (i.e., MatMul and BN) evaluation requires constant rounds and non-linear functions (i.e., BNBA and $\mathsf{Maxpool}$) evaluation requires $O(\log n)$ rounds in XONN, SecureBiNN, and FLEXBNN. For online communication cost, FSSiBNN achieves lower online communication cost in almost all BNN operators due to our bitwidth-reduced parameter encoding scheme and online-efficient secure computation protocol. SecureBiNN or FLEXBNN can take advantage of the 3PC framework to obtain "free" MatMul or BN, thus removing the communication overhead of computing fully connected and convolutional layers or batch normalization layers, but increasing the communication cost of computing other layer functions.

Table 2. Online communication complexity. $\mathsf{MatMul}_{d_1\times d_2, d_2\times d_3, n_0}$ is for the input layer, and $\mathsf{MatMul}_{d_1\times d_2, d_2\times d_3, n_2}$ is for the hidden and output layers, where d_1, d_2, and d_3 are the matrices' dimensions. n_i and n_o are the input and output bitwidths, and $d'_2 = \lceil \log_2(d_2(2^{n_0}-1))\rceil$. k is the kernel size, and s is the stride of the maxpool layer.

Operator	XONN [24]		SecureBiNN [30]		FLEXBNN [15]		FSSiBNN (Ours)	
	Rounds	Comm.	Rounds	Comm.	Rounds	Comm.	Rounds	Comm.
$\mathsf{MatMul}_{d_1\times d_2, d_2\times d_3, n_0}$	2	$2d_1 d'_2 d_3 \lambda$	0	0	1	$d_1 d_3 n_0$	1	$d_1 d_2 n_0$
$\mathsf{MatMul}_{d_1\times d_2, d_2\times d_3, n_2}$	1	$2d_1 d_2 d_3$	0	0	1	$d_1 d_3 \lceil \log_2(2d_2+1)\rceil$	1	$d_1 d_2 n_2$
$\mathsf{BN}_{d_1\times d_2, n_i}$	–	–	–	–	≈ 0	≈ 0	1	$d_1 d_2 n_i$
$\mathsf{BNBA}_{d_1\times d_2, n_i, n_o}$	2	$(\lambda+1) d_1 d_2 n_i$	$3+\log_2 n_i$	$d_1 d_2 (4n_i$ $+3n_o+1)$	$4+\log_2 n_i$	$d_1 d_2 (3n_i + n_o)$	1	$d_1 d_2 n_i$
$\mathsf{Maxpool}_{k,s,n_i}$	$\log_2(k^2)$	$2(k^2-1)$	$\log_2 n_i$	$\approx 3n_i$	$\log_2(ks)$	$ks-1$	1	n_i

Offline Communication Complexity. In the offline phase, the two-party frameworks, XONN [24] and FSSiBNN (Ours), need to generate correlated randomness (e.g., multiplication triples or DCF keys) to evaluate BNN. In contrast, SecureBiNN [30] and FLEXBNN [15] require smaller correlated randomness (e.g., 3-out-of-3 or 2-out-of-3 randomness). This discrepancy arises because the two-party frameworks are designed for a dishonest-majority setting, while the three-party frameworks operate under an honest-majority setting, which imposes a stronger security assumption. As a result, the two-party computing frameworks must rely on expensive public key cryptography to generate correlated randomness [19], whereas the three-party computing frameworks do not.

5.2 Experimental Results and Analysis

In this section, we present the implementation of FSSiBNN and provide detailed experimental results and analysis. The experiments were conducted on Aliyun ESC using ecs.hfr7.xlarge machines with 16 cores and 128 GB of CPU RAM in LAN settings. Our setup closely follows that of SecureBiNN [30]. We assess the secure inference of XONN [24], SecureBiNN [30], and FLEXBNN [15]. XONN is the state-of-the-art two-party framework, outperforming ABNN^2 [25] and Leia [21], while SecureBiNN and FLEXBNN are the state-of-the-art three-party frameworks, outperforming BANNERS [18].

We present the results of secure inference on the datasets MNIST, CIFAR-10, and Tiny ImageNet. We evaluate six networks: a 3-layer fully connected neural network (Network-A), a 3-layer convolutional neural network (Network-B), a 4-layer convolutional neural network (Network-C), LeNet, AlexNet, and VGG16. The architectures of Network-A, Network-B, and Network-C are the same as BM1, BM2, and BM3 in XONN [24]. We briefly discuss inference accuracy, and detailed experimental results are shown in Appendix C.

Evaluation on Small Neural Networks. In Table 3, we assess secure inference on MNIST using small neural network models (Network-A, Network-B, Network-C, and LeNet). Compared with the two-party framework XONN, FSSiBNN reduces the communication cost by 577× and is 7× faster. This is because XONN utilizes the Garbled Circuits (GC) protocol [29] to evaluate BNN, which involves lots of arithmetic operations. The computational and communication costs of using GC for these arithmetic operations are significantly high.

Table 3. Experimental results of various inference frameworks for small models on the MNIST dataset, with communication in MB and run time in seconds.

Framework	Num.	Network-A		Network-B		Network-C		LeNet	
		Comm.	Time	Comm.	Time	Comm.	Time	Comm.	Time
XONN [24]	2PC	4.290	0.130	38.280	0.160	32.130	0.150	–	–
SecureBiNN [30]	3PC	**0.005**	0.011	**0.032**	0.021	0.357	0.061	0.522	0.072
FLEXBNN [15]	3PC	0.008	**0.010**	0.043	**0.010**	0.430	**0.031**	0.610	0.074
FSSiBNN(Ours)	2PC	0.011	**0.010**	0.037	0.038	**0.133**	0.046	**0.206**	**0.062**

Compared with the three-party frameworks SecureBiNN and FLEXBNN, FSSiBNN performs slightly worse in terms of communication and run-time for Network-A and Network-B. However, it reduces communication costs by roughly

3× and has similar run-time for Network-C and LeNet. This discrepancy arises because Network-A and Network-B mainly consist of fully connected, convolutional, and batch normalization layers (i.e., MatMul and BN operators), and SecureBiNN and FLEXBNN have lower communication complexity for these operators (refer to Table 2). In contrast, Network-C and LeNet contain more binary activation layers and max pooling layers (i.e., BNBA and Maxpool operators). The efficient FSS-based nonlinear function calculation protocol in FSSiBNN allows for reduced communication costs under these conditions.

Evaluation on Large Neural Networks. In Table 4, we assess secure inference on the CIFAR-10 and Tiny ImageNet datasets using large neural network models (AlexNet and VGG16). Compared with FLEXBNN (XONN and SecureBiNN do not support these networks), FSSiBNN reduces the communication costs by $1.3 \sim 16.4\times$ and improves the run-time by up to $2.5\times$. The main reason is that these large networks involve more frequent bitwidth conversions and contain more activation layers and pooling layers. Due to our bitwidth-reduced parameter encoding scheme and the FSS-based online-efficient secure computation protocols proposed in this work, FSSiBNN can achieve BNN inference with low communication overhead.

Table 4. Experimental results of FLEXBNN and FSSiBNN for large models on CIFAR-10 and Tiny ImageNet dataset, with communication in MB and run time in seconds.

Framework	Num.	CIFAR-10 dataset				Tiny ImageNet dataset			
		AlexNet		VGG16		AlexNet		VGG16	
		Comm.	Time	Comm.	Time	Comm.	Time	Comm.	Time
FLEXBNN [15]	3PC	–	–	7.920	**1.520**	13.660	1.240	–	–
FSSiBNN(Ours)	2PC	0.455	0.144	**6.003**	2.158	**0.832**	**0.503**	25.323	8.782

Discussion. Based on the above analysis, our framework achieves significant performance advantages in inference on some large neural network models (e.g., LeNet, AlexNet, and VGG16), particularly when there are many binary activation and max pooling layers. Thus, our framework is well-suited for practical scenarios involving large models for inference. Additionally, for resource-constrained mobile devices, our approach significantly decrease communication costs and inference times, making it ideal for these environments.

6 Practical Demonstration of FSSiBNN

We give a practical demonstration of FSSiBNN in the online medical diagnosis scenario, and provide an evaluation of its limitations.

6.1 Online Medical Diagnosis using FSSiBNN

We consider a medical diagnosis scenario where a cloud server with a well-trained medical model provides an online diagnosis service to a patient with medical records. These records are highly sensitive and cannot be shared with the server, while the medical model, trained with significant computing power and large datasets, is also private. To protect both the patient's records and the server's model from being leaked, they can utilize the online diagnosis system shown in Fig. 3. to perform privacy-preserving inference.

In this setting, the patient inputs medical records and the server inputs the medical model, engaging in FSSiBNN for privacy-preserving BNN inference. Finally, the patient obtains the diagnosis results, and the server obtains nothing.

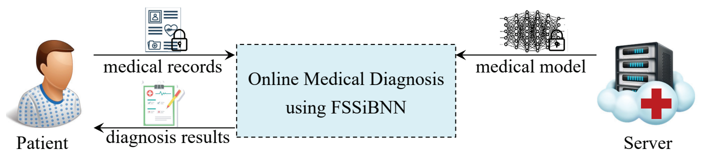

Fig. 3. The online medical diagnosis sysytem using FSSiBNN

We assume that a well-trained BNN model is known by the server in advance. The online medical diagnosis using FSSiBNN can be divided into two phases:

- **Setup Phase**: The patient and the server engage in a setup protocol to generate correlated randomness, such as matrix multiplication tuples and DCF keys. This phase corresponds to the offline phase of the FSSiBNN protocol.
- **Inference Phase**: The patient and the server engage in an inference protocol to perform online medical diagnosis. They use the correlated randomness generated in the setup phase to compute the BNN layer functions layer by layer, and finally, the patient obtains the diagnosis results. This phase corresponds to the online phase of the FSSiBNN protocol.

With this online medical diagnosis system, patients can use the server's medical diagnosis service to obtain results remotely, eliminating the need to visit a hospital. More importantly, the privacy of patients' medical records is preserved.

6.2 Evaluation of Limitations

Although the online medical diagnosis system using FSSiBNN shown in Fig. 3 can perform fast online diagnosis without leaking the privacy of the patient's medical records and the server's model, its main limitation is that the patient needs to perform certain calculations. Specifically, in the setup phase, the patient must interact with the server to generate and store correlated randomness, including matrix multiplication tuples and DCF keys, which are essential for the secure computation process. The patient must have a device capable of executing these tasks efficiently. While modern devices generally meet these requirements, the increasing complexity and size of models like Transformers pose challenges. Given current terminal device specifications, such as memory (4GB or larger), processors (capable of handling complex computing tasks), and storage (64GB or larger), this system can support the models tested in this paper. However, for larger models, such as Transformer models, additional work is needed to optimize the system's performance and resource requirements.

7 Conclusion

In this work, we propose a secure BNN framework, FSSiBNN, with free bitwidth conversion based on function secret sharing. FSSiBNN enables secure BNN inference service with low online latency and communication overhead. Experimental results show FSSiBNN outperforms the state-of-the-art solutions in both communication and time. Further attempts might be made to enable secure inference by accelerating the computation of FSS-based 2PC protocols with GPUs.

Acknowledgments. We sincerely thank the anonymous reviewers of ESORICS 2024 for their valuable comments. The work is supported by Shenzhen Science and Technology Major Project (KJZD20230923114908017), National Natural Science Foundation of China (62272131), and Guangdong Provincial Key Laboratory of Novel Security Intelligence Technologies (2022B1212010005).

A Proof of Sign Function Gate in Section 4.2

Proof. Given an unsigned integer $x \in \mathbb{U}_{2^m}$ and a signed integer $x' \in \mathbb{S}_{2^m}$ such that $(x - \mathsf{r}^{\mathsf{in}}) \bmod 2^m = x' \bmod 2^m$, the following relation holds:

$$\begin{aligned}
&(\hat{g}^{[\mathsf{r}^{\mathsf{in}},\mathsf{r}^{\mathsf{out}}]}_{\mathsf{sign},m,n}(x) - \mathsf{r}^{\mathsf{out}}) \bmod 2^n = g_{\mathsf{sign},m,n}(x') \bmod 2^n \\
&= 1 - 2 \cdot \mathbf{1}\{x' < 0\} = 1 - 2 \cdot \mathsf{MSB}\{(x - \mathsf{r}^{\mathsf{in}}) \bmod 2^m\} \\
&= 1 - 2 \cdot \mathsf{MSB}\{(x + 2^m - \mathsf{r}^{\mathsf{in}}) \bmod 2^m\} \\
&= 1 - 2 \cdot \mathsf{MSB}\{(x + r) \bmod 2^m\} \text{ where } r = (2^m - \mathsf{r}^{\mathsf{in}}) \bmod 2^m
\end{aligned}$$

Let $x = x_{[m-1]} \cdot 2^{m-1} + x_{[0,m-1)}$ where $x_{[0,m-1)} = x_{[m-2]}||\cdots||x_{[0]}$, $r = r_{[m-1]} \cdot 2^{m-1} + r_{[0,m-1)}$ and $r_{[0,m-1)} = r_{[m-2]}||\cdots||r_{[0]}$, it holds that:

$$\begin{aligned}
&(x + r) \bmod 2^m \\
=&((x_{[m-1]} \cdot 2^{m-1} + x_{[0,m-1)}) + (r_{[m-1]} \cdot 2^{m-1} + r_{[0,m-1)})) \bmod 2^m \\
=&((x_{[m-1]} + r_{[m-1]}) \cdot 2^{m-1} + (x_{[0,m-1)} + r_{[0,m-1)})) \bmod 2^m \\
=&((x_{[m-1]} + r_{[m-1]} + c) \cdot 2^{m-1} + (x_{[0,m-1)} + r_{[0,m-1)} - c \cdot 2^{m-1})) \bmod 2^m
\end{aligned}$$

where $c = \mathbf{1}\{2^{m-1} - 1 < x_{[0,m-1)} + r_{[0,m-1)}\} = \mathbf{1}\{2^{m-1} - x_{[0,m-1)} - 1 < r_{[0,m-1)}\}$.

Thus, it holds that $\mathsf{MSB}\{(x + r) \bmod 2^m\} = ((x_{[m-1]} + r_{[m-1]} + c) \cdot 2^{m-1}) \bmod 2^m = x_{[m-1]} \oplus r_{[m-1]} \oplus c$.

B Analysis of Computation Complexity

Online Computation Complexity. In the online phase, XONN [24], SecureBiNN [30], FLEXBNN [15], and FSSiBNN all have the same order of magnitude of online computation complexity since they all adopt the offline-online computation paradigm. However, XONN, SecureBiNN, and FLEXBNN only rely on lightweight computation (e.g., addition and multiplication), whereas FSSiBNN requires one calculation of $\mathsf{Eval}_{m,n}^{\mathsf{Sign}}$ per party in the online phase for comparison (e.g., binary activation and max pooling), and $\mathsf{Eval}_{m,n}^{\mathsf{Sign}}$ includes $m-1$ pseudo-random number generator (PRG) calls where m is the input size of $\mathsf{Eval}_{m,n}^{\mathsf{Sign}}$. Thus, FSSiBNN needs to additionally evaluate $m-1$ PRG. In practice, a typical three-layer BNN inference on the MNIST dataset requires roughly 12,900 $\mathsf{Eval}_{m,n}^{\mathsf{Sign}}$ calls. For the input $m = 8$ or 16, FSSiBNN requires local computation of 90,300 or 193,500 PRG calls per party. For the PRG implemented using AES, using an estimate of 360 million AES calls per second on a single-core 3.6 GHz machine [5], local computation would take roughly 0.25ms and 0.54ms, respectively, so there is no noticeable impact on inference time.

Offline Computation Complexity. In the offline phase, XONN [24] and FSSiBNN (Ours) need to generate correlated randomness (e.g., multiplication triples and DCF keys) to evaluate BNN, while SecureBiNN [30] and FLEXBNN [15] require smaller correlated randomness, which can be generated more efficiently using 2PC-based or 3PC-based offline phases. Therefore, XONN and FSSiBNN require more computation in the offline phase. Specifically, in FSSiBNN, the offline protocol needs to generate the multiplication tuples for the multiplication operations and generate the DCF keys for the comparison operations. The multiplication tuples are generated by directly using the VOLE-style OT generation scheme proposed in Ferret [28]. Considering the multiplication $\mathbf{W} \times \mathbf{R}$ where $\mathbf{W} \in \mathbb{Z}_2^{d_1 \times d_2}, \mathbf{R} \in \mathbb{Z}_{2^n}^{d_2 \times d_3}$, we need $d_1 d_2$ instances of OT, which require $d_1 d_2(\lambda + n d_3)$ bits of communication where λ is the security parameter. For the DCF keys, we leverage the distributed generation scheme proposed in [5] to generate these DCF keys. The communication and computation requirements of the corresponding protocol will be dominated by the $(n+1)$ secure evaluations of the pseudo-random generators (PRG). Setting $\lambda = 127$ and instantiating the PRG via two AES evaluations, as suggested previously, results in a necessary $2(n+1)$ secure evaluations of AES. Depending on the hardware, network, and on whether one targets semi-honest or malicious security, the throughput for state-of-the-art 2PC of AES is roughly $100 \sim 1000$ instances per second, with communication of roughly 200KB per instance [5].

C Evaluation and Analysis of Inference Accuracy

In this section, the inference accuracy of Network-A, Network-B, Network-C, and LeNet on dataset MNIST is presented in Table 5, with plaintext inference accuracy reported for comparison. The difference in accuracy between plaintext inference and private inference is 0.02% $\sim$ 0.07%, which is not significant, indicating that our method hardly affects inference accuracy. This is because we design the bitwidth encoding scheme in FSSiBNN to address multiplication overflow and truncation issues, and implement an accurate secure comparison protocol, ensuring that secure inference closely matches the computational accuracy of plaintext inference.

Table 5. The comparison of private accuracy and plaintext accuracy

Dataset	Model	Plaintext Accuracy	Private Accuracy (Ours)
MNIST	Network-A	96.17%	96.15% ($\downarrow$ 0.02%)
MNIST	Network-B	97.02%	96.95% ($\downarrow$ 0.07%)
MNIST	Network-C	98.30%	98.25% ($\downarrow$ 0.05%)
MNIST	LeNet	98.27%	98.22% ($\downarrow$ 0.05%)

References

1. Agrawal, N., Shahin Shamsabadi, A., Kusner, M.J., Gascón, A.: QUOTIENT: Two-Party Secure Neural Network Training and Prediction. In: 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 1231–1247. ACM, London UK (2019). https://doi.org/10.1145/3319535.3339819
2. Asharov, G., Lindell, Y., Schneider, T., Zohner, M.: More Efficient Oblivious Transfer and Extensions for Faster Secure Computation. In: 2013 ACM SIGSAC conference on Computer and Communications Security. pp. 535–548. ACM, Berlin Germany (2013). https://doi.org/10.1145/2508859.2516738
3. Beaver, D.: Efficient Multiparty Protocols using Circuit Randomization. In: Advances in Cryptology — CRYPTO 1991. pp. 420–432. Springer, Berlin, Heidelberg (1992). https://doi.org/10.1007/3-540-46766-1_34
4. Bourse, F., Minelli, M., Minihold, M., Paillier, P.: Fast Homomorphic Evaluation of Deep Discretized Neural Networks. In: Advances in Cryptology — CRYPTO 2018. pp. 483–512. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96878-0_17
5. Boyle, E., Chandran, N., Gilboa, N., Gupta, D., Ishai, Y., Kumar, N., Rathee, M.: Function Secret Sharing for Mixed-Mode and Fixed-Point Secure Computation. In: Advances in Cryptology – EUROCRYPT 2021. pp. 871–900. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77886-6_30
6. Boyle, E., Gilboa, N., Ishai, Y.: Function Secret Sharing. In: Advances in Cryptology – EUROCRYPT 2015. pp. 337–367. Springer, Berlin, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_12
7. Boyle, E., Gilboa, N., Ishai, Y.: Function Secret Sharing: Improvements and Extensions. In: 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 1292–1303. ACM, Vienna Austria (2016). https://doi.org/10.1145/2976749.2978429
8. Boyle, E., Gilboa, N., Ishai, Y.: Secure Computation with Preprocessing via Function Secret Sharing. In: Theory of Cryptography Conference. pp. 341–371. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-36030-6_14
9. Chen, X., Chen, Z., Dong, B., Wei, S., Chen, L., He, D.: FOBNN: Fast Oblivious Binarized Neural Network Inference (2024), https://arxiv.org/abs/2405.03136

10. Costan, V., Devadas, S.: Intel SGX Explained. Cryptology ePrint Archive, Paper 2016/086 (2016), https://eprint.iacr.org/2016/086
11. Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., Bengio, Y.: Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1 (2016), https://arxiv.org/abs/1602.02830
12. Dalskov, A., Escudero, D., Keller, M.: Secure Evaluation of Quantized Neural Networks. Proceedings on Privacy Enhancing Technologies pp. 355–375 (2020). https://doi.org/10.2478/popets-2020-0077
13. Damgård, I., Pastro, V., Smart, N., Zakarias, S.: Multiparty Computation from Somewhat Homomorphic Encryption. In: Advances in Cryptology – CRYPTO 2012. pp. 643–662. Springer, Cham (2012). https://doi.org/10.1007/978-3-642-32009-5_38
14. Demmler, D., Schneider, T., Zohner, M.: ABY – A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. In: Network and Distributed System Security Symposium (2015), https://encrypto.de/papers/DSZ15.pdf
15. Dong, Y., Chen, X., Song, X., Li, K.: FLEXBNN: Fast Private Binary Neural Network Inference with Flexible Bit-Width. IEEE Transactions on Information Forensics and Security pp. 2382 – 2397 (2023). https://doi.org/10.1109/TIFS.2023.3265342
16. Dong, Y., Xiaojun, C., Jing, W., Kaiyun, L., Wang, W.: Meteor: Improved Secure 3-Party Neural Network Inference with Reducing Online Communication Costs. In: Proceedings of the ACM Web Conference 2023. pp. 2087–2098. ACM, Austin TX USA (2023). https://doi.org/10.1145/3543507.3583272
17. Gentry, C.: Fully Homomorphic Encryption Using Ideal Lattices. In: 41st annual ACM symposium on Theory of computing. pp. 169–178. ACM, Bethesda MD USA (2009). https://doi.org/10.1145/1536414.1536440
18. Ibarrondo, A., Chabanne, H., Önen, M.: Banners: Binarized Neural Networks with Replicated Secret Sharing. In: 2021 ACM Workshop on Information Hiding and Multimedia Security. pp. 63–74. ACM, Virtual Event Belgium (2021). https://doi.org/10.1145/3437880.3460394
19. Lindell, Y.: Secure Multiparty Computation. Communications of the ACM pp. 86–96 (2020). https://doi.org/10.1145/3387108
20. Liu, J., Juuti, M., Lu, Y., Asokan, N.: Oblivious Neural Network Predictions via MiniONN Transformations. In: 2017 ACM SIGSAC conference on computer and communications security. pp. 619–631. ACM, Dallas Texas USA (2017). https://doi.org/10.1145/3133956.3134056
21. Liu, X., Wu, B., Yuan, X., Yi, X.: Leia: A Lightweight Cryptographic Neural Network Inference System at the Edge. IEEE Transactions on Information Forensics and Security pp. 237–252 (2022). https://doi.org/10.1109/TIFS.2021.3138611
22. Mishra, P., Lehmkuhl, R., Srinivasan, A., Zheng, W., Popa, R.A.: Delphi: A cryptographic inference service for neural networks. In: 29th USENIX Security Symposium. pp. 2505–2522. USENIX Association (2020), https://www.usenix.org/conference/usenixsecurity20/presentation/mishra
23. Mohassel, P., Zhang, Y.: SecureML: A System for Scalable Privacy-Preserving Machine Learning. In: 2017 IEEE Symposium on Security and Privacy. pp. 19–38. IEEE, San Jose USA (2017). https://doi.org/10.1109/SP.2017.12
24. Riazi, M.S., Samragh, M., Chen, H., Laine, K., Lauter, K., Koushanfar, F.: XONN: XNOR-based Oblivious Deep Neural Network Inference. In: 28th USENIX Security Symposium. pp. 1501–1518. USENIX Association, Santa Clara, CA (2019), https://www.usenix.org/conference/usenixsecurity19/presentation/riazi

25. Shen, L., Dong, Y., Fang, B., Shi, J., Wang, X., Pan, S., Shi, R.: ABNN^2: Secure Two-party Arbitrary-Bitwidth Quantized Neural Network Predictions. In: 59th ACM/IEEE Design Automation Conference. pp. 361–366. ACM, San Francisco California USA (2022). https://doi.org/10.1145/3489517.3530680
26. Storrier, K., Vadapalli, A., Lyons, A., Henry, R.: Grotto: Screaming fast $(2+1)$-PC for $\mathbb{Z}_{2^n}$ via (2, 2)-DPFs. In: 2023 ACM SIGSAC Conference on Computer and Communications Security. pp. 2143–2157. ACM, Copenhagen Denmark (2023). https://doi.org/10.1145/3576915.3623147
27. Wagh, S., Gupta, D., Chandran, N.: SecureNN: 3-Party Secure Computation for Neural Network Training. Proceedings on Privacy Enhancing Technologies pp. 26–49 (2019). https://doi.org/10.2478/popets-2019-0035
28. Yang, K., Weng, C., Lan, X., Zhang, J., Wang, X.: Ferret: Fast Extension for Correlated OT with Small Communication. In: 2020 ACM SIGSAC Conference on Computer and Communications Security. pp. 1607–1626. ACM, Virtual Event USA (2020). https://doi.org/10.1145/3372297.3417276
29. Yao, A.C.C.: How to Generate and Exchange Secrets. In: 27th Annual Symposium on Foundations of Computer Science (sfcs 1986). pp. 162–167. IEEE (1986). https://doi.org/10.1109/SFCS.1986.25
30. Zhu, W., Wei, M., Li, X., Li, Q.: SecureBiNN: 3-Party Secure Computation for Binarized Neural Network Inference. In: Computer Security–ESORICS 2022. pp. 275–294. Springer, Cham (2022). https://doi.org/https://doi.org/10.1007/978-3-031-17143-7_14