SecureBiNN: 3-Party Secure Computation for Binarized Neural Network Inference

Wenxing Zhu[1,2], Mengqi Wei[1], Xiangxue Li[1,3](✉), and Qiang Li[4]

[1] School of Software Engineering, East China Normal University, Shanghai, China
xxli@cs.ecnu.edu.cn

[2] Shanghai Key Laboratory of Privacy-Preserving Computation, MatrixElements Technologies, Shanghai, China

[3] Shanghai Key Laboratory of Trustworthy Computing, Shanghai, China

[4] Institute of Cyber Science and Technology, Shanghai Jiao Tong University, Shanghai, China
qiangl@sjtu.edu.cn

Abstract. The paper proposes SecureBiNN, a novel three-party secure computation framework for evaluating privacy-preserving binarized neural network (BiNN) in semi-honest adversary setting. In SecureBiNN, three participants hold input data and model parameters in secret sharing form, and execute secure computations to obtain secret shares of prediction result without disclosing their input data, model parameters and the prediction result. SecureBiNN performs linear operations in a computation-efficient and communication-free way. For non-linear operations, we provide novel secure methods for evaluating activation function, maxpooling layers, and batch normalization layers in BiNN. Communication overhead is significantly minimized comparing to previous work like XONN and Falcon. We implement SecureBiNN with tensorflow and the experiments show that using the Fitnet structure, SecureBiNN achieves on CIFAR-10 dataset an accuracy of 81.5%, with communication cost of 16.609MB and runtime of 0.527s/3.447s in the LAN/WAN settings. More evaluations on real-world datasets are also performed and other concrete comparisons with state-of-the-art are presented as well.

Keywords: Privacy-preserving machine learning · Secure multi-party computation · Binarized neural network

1 Introduction

Machine Learning as a Service (MLaaS) has created huge economic benefits and been widely used in image classification, disease diagnosis, etc. In MLaaS, both model owner and data owner would suffer from privacy leakage if the model or input data could be accessed by others arbitrarily. Cryptographic techniques, e.g., secure multiparty computation (SMC) and homomorphic encryption (HE), are good solutions to the problem. The community really sees many cryptography-enabled protocols [7,15,23,24,27,34], and most of them focus on naive neural networks (NN) and stipulate the parameters in fixed-point numbers. Some work [8,28] implements binarized neural network (BiNN) (model parameters take the values of ± 1). BiNN has simpler calculation process than NN and one may expect more efficient implementation of BiNN (than NN) with SMC and HE.

This paper proposes SecureBiNN, a novel three-party framework for BiNN inference, secure under non-colluding semi-honest adversary setting (same as SecureNN [34] and XONN [28]). SecureBiNN has three semi-honest participants, each of which holds secret shares of the input data and of the model. We evaluate BiNNs with replicated secret share technique [5], reduce communication cost and enhance computation efficiency by ruling out garbled circuit (GC) and HE. After the framework evaluation, each party outputs its secret share of prediction result. SecureBiNN can be applied to inference on sensitive data without compromising privacy. For example, three banks could use their own data to predict customers for financial fraud, or three hospitals could use patients' private data to make better diagnoses without revealing their privacy.

Our contributions can be summarized in the following two aspects:

– Framework: We propose SecureBiNN, a secure computation framework for BiNN inference. To reduce communication cost, SecureBiNN chooses ring size according to BiNN architecture. Furthermore, we propose new protocols for three-party oblivious transfer, secure activation, and secure maxpooling. To further improve SecureBiNN performance under WAN settings, we use 3-input AND gate to reduce the number of communication rounds. In particular, our maxpooling operation requires only one private comparison operation. We put batch normalization and binarized activation together, and thereby use one addition and one private comparison to achieve batch normalization and activation operation.
– Practicality: SecureBiNN is computation-efficient and communication-cheap. We implement SecureBiNN with numpy [36], Tensorflow [4] and run experiments on three ecs.c7.2xlarge servers from Alibaba Cloud. We evaluate various networks on MNIST, CIFAR-10 and real-world medical datasets. The results of experiments in Sect. 4 show the practical feasibility of SecureBiNN under LAN/WAN settings. We provide our source code on Github[1].

We recap the related work in the Appendix A due to space limitation.

2 Preliminaries

We review security model, correlated randomness, and secret sharing used in this paper. We denote three participants as P_0, P_1, and P_2. $P_{(i+1) mod\,3}$ and $P_{(i-1) mod\,3}$ denote the next and previous parties for P_i respectively. For simplicity, we omit *mod 3* in the subscript of P_i and other variables (e.g., x_i, r_i).

[1] https://github.com/Wixee/SecureBiNN.

2.1 Security Model

We assume that each participant is *honest-but-curious* (a.k.a. *semi-honest*). In other words, each participant implements the protocols honestly and will not attack others actively (e.g., capture others' secret keys, monitor others' communications, or construct malicious input data). However, they might try to infer information of others as much as possible. Such model is commonly assumed in [22,23,28,34]. We also adopt the same assumption as in other three-party protocols [16,24,35] that there is no collusion between any two participants.

As previous work and other cryptographic protocols, SecureBiNN ensures input data and model parameters will not be leaked during protocol executions. We will not discuss how to protect model structure or to defend other attacks like model retrieval attack [33], membership inference attack [31] and data poisoning attacks [11], which go apparently out of scope of the framework.

2.2 Correlated Randomness

In this work, random values are generated by a PRF (Pseudo-Random Function) R with two inputs x, y, denoted as $R(x, y)$. Given an input pair, R will return a specific value. R's output can be viewed as a uniform distribution on its domain. We assume that $\Pr[(x, y) \leftarrow \mathcal{A}(R(x, y))]$ is negligible for any PPT adversary $\mathcal{A}$.

Suppose P_i and P_{i+1} negotiate a secret random seed r_{i+1}. P_i holds (r_i, r_{i+1}), $i = 0, 1, 2$. All participants maintain a counter cnt, incremented by one after each invocation. For a modulo m, we have two types of correlated randomness.
3-out-of-3 randomness: P_i calculates $a_i = R(r_{i+1}, cnt) - R(r_i, cnt) \mod m$. Note that $a_0 + a_1 + a_2 \equiv 0 \mod m$.
2-out-of-3 randomness: P_i calculates $(a_i, a_{i+1}) = (R(r_i, cnt), R(r_{i+1}, cnt))$. By 2-out-of-3 randomness, P_i, P_{i+1} can generate a common random value a_{i+1} without any communication cost, and P_{i-1} does not have the knowledge of a_{i+1}.

2.3 Two-party Secret Sharing

For a modulo $m = 2^l, l \in \mathbb{Z}^+$, a secret $x \in \mathbb{Z}_m$ is shared by sampling two random x_0 and x_1 s.t. $x_0 + x_1 = x$ over $\mathbb{Z}_m$. Denote x shared by two parties as $[\![x]\!]_2^m$. When $l = 1$, we can denote $[\![x]\!]_2^B = [\![x]\!]_2^2 = (x_0, x_1)$ where $x_0 \oplus x_1 = x$ and $x, x_0, x_1 \in \{0, 1\}$. More details can be found in [6,13,23]. The linear operation for two-party secret sharing is trivial, and the multiplication can be implemented by using Beaver's Triplet [6].

2.4 Three-party Secret Sharing

Three-party secret sharing is proposed in [5]. For a modulo $m = 2^l$, a secret $x \in \mathbb{Z}_m$ is shared by sampling three random $x_0, x_1, x_2 \in \mathbb{Z}_m$, $x = x_0 + x_1 + x_2$, and then P_i holds (x_i, x_{i+1}). Denote such kind of secret shared value x as $[\![x]\!]_3^m$, and the tuple (x_0, x_1, x_2) represents its shares. Three-party secret sharing supports linear operations and multiplications between the shared values. Given public a, b and c, we have the following for secretly shared $[\![x]\!]_3^m$ and $[\![y]\!]_3^m$.

- Linear operation: participants can compute $[\![ax+by+c]\!]_3^m := (ax_0+by_0+c,\ ax_1+by_1, ax_2+by_2)$ locally to achieve linear operation.
- Multiplication: P_i computes and outputs $z_i = x_iy_i + x_iy_{i+1} + x_{i+1}y_i$ where $\sum_{i=0}^{2} z_i = z = xy$. If the participants need to restore the output to the form of $[\![z]\!]_3^m$, they should invoke 3-out-of-3 randomness to get a_0, a_1, a_2, and P_i sends $z_i + a_i$ to P_{i-1}. At last, P_i outputs $(z_i + a_i,\ z_{i+1} + a_{i+1})$.

Both two-party and three-party secret sharing schemes can be trivially extended to matrix computations.

3 The SecureBiNN Framework

3.1 Highlights

For a BiNN with n layers, we denote the input and the weights of the i-th layer as X_{i-1} and W_i respectively. X_{i-1}s and W_is are encoded into an integer ring and secretly shared by participants who run multiplication protocol of secret sharing to calculate W_iX_{i-1}. This method can be easily generalized to convolutional neural network. The activation function used in BiNN is

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

When running the *sign* function, we convert three-party secret sharing into two-party secret sharing between P_0 and P_1 to evaluate part of parallel prefix adder circuit. We can get the result of the sign function according to the MSB (most significant bit) of its input. After that, three participants utilize a three-party oblivious transfer (OT) to convert the result of the circuit into three-party secret sharing again. The three-party OT we use originates from [24]. However, we alter its input and make some modifications so that it also works well in SecureBiNN. We will show details in the following part. Note that the *sign* function is used in the maxpooling layers as well.

Before implementing SecureBiNN, a model owner should encode its model in the form of three-party secret sharing. Input data should also be encoded in the same way. After the protocol, each party holds a share of the result, and all participants would then send their shares to those who should know the result. This is application-dependent and here we do not consider the source of the input and how participants deal with the results.

3.2 Parameters Encoding

We encode all model parameters into $\mathbb{Z}_{2^l}$. Among all n layers, l's value decided in the i-th layer might be independent of that in the j-th layer $(1 \leq i, j \leq n, i \neq j)$. In the input layer and output layer, we use standard fixed-point arithmetic encoding scheme. In other layers, however, we can use a smaller l to reduce parameter size. For example, we encode parameters of the input layer and output layer into $\mathbb{Z}_{2^{16}}$ or $\mathbb{Z}_{2^{32}}$, while the parameters of other layers are encoded into $\mathbb{Z}_{2^9}$ or $\mathbb{Z}_{2^{14}}$. We will describe more details below.

Encoding Input and Output Layer Parameters. In the input and output layers, we use standard fixed-point encoding, and a bit string of length l can represent a fixed-point number in complement form. In most cases, the value of l is 32. The MSB of some number represents the sign of the number: it is non-negative if its MSB is 0, and negative otherwise. We say a number has l_D bits of precision ($0 \leq l_D < (l-1)/2$) if the l_D bits at the far right represent the fractional part and the remaining $l-1-l_D$ bits represent the integer part.

In SecureBiNN, we ditch truncation operations (used generally in fixed-point arithmetic multiplications), for we only use fixed-point encoding in input/output layers and no overflow occurs. Our method requires that for addition operations in input/output layers, the operands should have same precision. Take $Y = WX + b$ for instance. If W and X have l_D^W and l_D^X bits of precision respectively, then WX has $l_D^W + l_D^X$ bits of precision. In order to remove truncation and make addition result correct, b should also have $l_D^W + l_D^X$ bits of precision.

Encoding Hidden Layer Parameters. In SecureBiNN, the parameters of each hidden layer are represented by signed integers of l bits. Herein, l could be chosen independently in each layer. In hidden fully connected layers, every entry in the input X_{i-1} and the weight W_i takes 1 or –1. If the shapes of W_i and X_{i-1} are (n, m) and (m, k) respectively, then each entry in $W_i X_{i-1}$ is an signed integer in $[-m, m]$. We can thus choose the smallest l s.t. $2^{l-1} - 1 \geq m$ and encode W_i and X_{i-1} into the ring Z_{2^l}. In case that the hidden layer performs a convolution operation and filter size is (n_{in}, h, w, n_{out}), we require $2^{l-1} - 1 \geq n_{in} \times h \times w$. Then we use l bits to represent any signed integer parameter in the layer. W.l.o.g., this can also be seen as 0-bit precision fixed-point encoding.

This method has relatively low fault tolerance. Indeed, the number of neurons in previous layer completely determines the ring size, and we do not consider addition operations in subsequent batch normalization (see the coming sections), which might cause overflow. This little gap could be handled heuristically. If the hidden layer is a fully connected layer or a convolutional layer followed by a normalization layer, then we add a constant δ to l to relax the restriction on l. Our experiments show that setting $\delta = 2$ already provides expected robustness.

3.3 Fully Connected Layer and Convolutional Layer

Consider an arbitrary fully connected layer. All its input X, weight W (and bias b for input/output layers) are secretly shared over $\mathbb{Z}_{2^l}$ among the participants who collaboratively invoke multiplication protocol (see Algorithm 1) to calculate the (shared) result. The protocol could be easily extended to convolution operations. In the protocol, b should have $l_W + l_X$ bits of precision if the fixed-point W and X have l_W and l_X bits of precision respectively.

3.4 Secure 3-Input and Gate

In order to further improve the performance in the WAN setting, we use 3-input AND gate technique [25] to reduce communication rounds in implementing

Algorithm 1. Fully Connected Layer Inference: Π_{fc}

Input: P_i inputs the weight share (W_i, W_{i+1}), data share(X_i, X_{i+1}) (and bias share b_i if the layer is input layer or output layer) s.t. $\sum_{j=0}^{2} W_j = W$, $\sum_{j=0}^{2} X_j = X$, $\sum_{j=0}^{2} b_j = b$, $i \in \{0, 1, 2\}$.
Output: P_i outputs Z_i, secret shares of fully connected layer output.

```
1: $P_i : Z_i = W_i X_i + W_{i+1} X_i + W_i X_{i+1}$
2: **if** it is not a hidden layer **then**
3:     $Z_i = Z_i + b_i$
4: **end if**
```

Algorithm 2. Secure 3-input AND Gate: $\Pi_{3-inputANDgate}$

Input: P_0, P_1 input $[\![a]\!]_2^B$, $[\![b]\!]_2^B$, $[\![c]\!]_2^B$, P_2 inputs $\perp$.
Output: P_0 and P_1 ouput $[\![z]\!]_2^B$ where $z = abc$, P_2 ouputs $\perp$.

```
1: $P_0$, $P_1$ generate $[\![a']\!]_2^B$, $[\![b']\!]_2^B$, $[\![c']\!]_2^B$, $[\![a'b']\!]_2^B$, $[\![a'c']\!]_2^B$, $[\![b'c']\!]_2^B$, $[\![a'b'c']\!]_2^B$ with the help
   of $P_2$ by utilizing *2-out-of-3 randomness*.
2: $P_0$, $P_1$ calculate $[\![a \oplus a']\!]_2^B$, $[\![b \oplus b']\!]_2^B$, $[\![c \oplus c']\!]_2^B$.
3: $P_0$, $P_1$ reconstruct and open $(a \oplus a')$, $(b \oplus b')$, $(c \oplus c')$.
4: $P_0$ and $P_1$ calculate and output $[\![z]\!]_2^B$ according to the Eq. (2).
```

secure activation function. It can be seen as an extension of Beaver's Triplet technique [6]. The formula for 3-input AND gate can be written as

$$\begin{aligned} z = abc = &(a \oplus a')(b \oplus b')(c \oplus c') \oplus (c \oplus c')a'b' \oplus (b \oplus b')a'c' \\ &\oplus (a \oplus a')b'c' \oplus (a \oplus a')(b \oplus b')c' \\ &\oplus (a \oplus a')(c \oplus c')b' \oplus (b \oplus b')(c \oplus c')a' \oplus a'b'c'. \end{aligned} \tag{2}$$

Here, a, b, c represent the inputs and z represents the output, a', b' and c' are random masks used to hide a, b, c respectively in SMC. Before implementing such a 3-input AND gate, P_0 and P_1 take $[\![a]\!]_2^B$, $[\![b]\!]_2^B$ and $[\![c]\!]_2^B$ as inputs, P_2 plays the role of a helper, generates $[\![a']\!]_2^B$, $[\![b']\!]_2^B$, $[\![c']\!]_2^B$, $[\![a'b']\!]_2^B$, $[\![a'c']\!]_2^B$, $[\![b'c']\!]_2^B$, $[\![a'b'c']\!]_2^B$ and send these secret shares to P_0 and P_1.

In SecureBiNN, with the help of *2-out-of-3 randomness*, communication cost of implementing a 3-input AND gate can be further optimized. By *2-out-of-3 randomness*, P_2 can reduce communication cost of secret sharing distribution. P_0 and P_1 generate common uniform randomness with P_2 respectively, and take them as $[\![a']\!]_2^B$, $[\![b']\!]_2^B$ and $[\![c']\!]_2^B$. Then P_2 reconstructs a', b', c' and calculates $a'b'$, $a'c'$, $b'c'$, $a'b'c'$. P_0 and P_2 then generate common randomness again, P_0 takes these randomness as shares of $a'b'$, $a'c'$, $b'c'$, $a'b'c'$, P_2 calculates and sends appropriate shares of above terms to P_1. In the naive method, P_2 needs to send a total of 14 bit, while in above optimized method, 4 bits suffice. Algorithm 2 shows the details of implementing a 3-input AND gate with the above optimization.

It is easy to prove the correctness, and we omit the details and just focus on its security proof below.

Algorithm 3. Ideal Function of Three-Party Oblivious Transfer: F_{3-OT}

```
Input: Sender inputs ($m_0$, $m_1$), Receiver inputs $b$, Helper inputs $\perp$.
Output: Receiver outputs $m_b$, Sender and Helper output $\perp$.
```

Algorithm 4. Three-Party Oblivious Transfer: Π_{3-OT}

```
Input: Sender inputs ($m_0$, $m_1$), Receiver inputs $b$, Helper inputs $\perp$.
Output: Receiver outputs $m_b$, Sender and Helper output $\perp$.
1: Receiver and Sender generate common ranom bit $r$, common random bit string $mask_0$ and $mask_1$.
2: Sender computes $s_i = m_{i \oplus r} \oplus mask_{i \oplus r}$, $i \in \{0, 1\}$.
3: Sender sends $(s_0, s_1)$ to Helper.
4: Receiver sends $b \oplus r$ to Helper.
5: Helper sends $s_{b \oplus r}$ to Receiver.
6: Receiver calculates $m_b = s_{b \oplus r} \oplus mask_{b \oplus r}$.
```

Theorem 1. *The above optimized secret sharing used for implementing 3-input AND gate is secure, i.e., these secret shares in performing 3-input AND gate will not leak secret information.*

Proof. Since the *2-out-of-3 randomness* generates uniform randomness, a', b', c', $a'b'$, $a'c'$, $b'c'$, $a'b'c'$ and the corresponding secret shares also conform to uniform distribution. When implementing 3-input AND gate, messages are well masked to follow uniform distribution, thus P_0 and P_1 cannot infer each other's secret share based on the transcripts (see interaction details in [25]). Therefore, it is trivial to construct a simulator of the real-ideal paradigm [9]. P_2 plays the role of a helper and does not participate in the execution of AND gate itself, so P_2 can not infer the secrets. $\square$

3.5 Three-party Oblivious Transfer

We use three-party oblivious transfer (OT) in follow-up secure activation function. In our three-party OT, we have a sender holding two messages m_0 and m_1, a receiver holding a choice bit b (that decides the message m_b it wants from the sender), and a helper holding null input. The corresponding ideal function is shown in Algorithm 3, and the detail of the three-party OT is shown in Algorithm 4.

After the protocol, the receiver gets the message m_b with the requirements that the sender does not know which message is selected by the receiver, that the receiver knows nothing about m_{1-b}, and that the helper does not learn any knowledge of the messages and the choice bit. Table 1 gives the comparison of our three-party OT and that in ABY^3 from the perspectives of communication overhead, communication round, and the requirement on the choice bit.

Note that any two parties (e.g., the sender and the receiver in three-party OT) can take a communication-free invocation of *2-out-of-3 randomness* to generate common randomness. So the first step in Algorithm 4 is communication free.

Table 1. Comparison of OT in SecureBiNN and 1-out-of-2 OT in ABY^3, l represents the length of a single message

	Comm. cost	Round	Helper knows the choice bit?
Ours	$3l + 1$	2	×
ABY^3	$3l$	2	✓

Thus, if the bit lengths of both m_0 and m_1 are l, overall communication takes $3l+1$ bits. The correctness and security of the protocol are obvious and intuitive. This protocol will be used in secure activation function.

In our protocol, not only the sender but the helper cannot tell the index of the exact message selected by the receiver, which is different from previous three-party oblivious transfer protocols in ABY^3 [24] and Falcon [35][2]. The counterparts in them call an explicit and simple conversion from two-party secret sharing to three-party secret sharing so that the helper and the receiver reach a common bit and further take it as the choice bit. Thus, those schemes require one more round of communication before the oblivious transfer begins.

Theorem 2. *Π_{3-OT} is secure against non-collusion semi-honest adversaries.*

Proof. Due to the common uniformly random masks generated by Sender and Receiver, the execution of the Π_{3-OT} protocol is quite simple and one may easily construct such a simulator. Sender and Receiver can generate common uniformly random bit r, common uniformly random string $mask_0$ and $mask_1$ with the common random seed. This can be achieved by a trivial simulation (omitted in the following discussion). Next, we discuss from the perspectives of three parties.

1. If adversary $\mathcal{A}$ corrupts Sender: $\mathcal{A}$ sends two masked messages s_0 and s_1 to the simulator $\mathcal{S}$. $\mathcal{S}$ can extract m_0, m_1 from s_0 and s_1 since r, $mask_0$ and $mask_1$ are known to both $\mathcal{A}$ and $\mathcal{S}$. $\mathcal{S}$ provides the input to the ideal functionality $\mathcal{F}_{3-OT}$, then the ideal functionality will send the output to the party which represents the receiver in the real world, thus honest parties in the ideal world receive correct ouput. Since the sender receives no messages, the simulator $\mathcal{S}$ sends the adversary $\mathcal{A}$ nothing and thereby $\mathcal{A}$ cannot distinguish the ideal world and the real execution.
2. If adversary $\mathcal{A}$ corrupts Helper: $\mathcal{S}$ just sends a uniformly random bit and two uniformly random strings which represent $b \oplus r$, s_0 and s_1 (we assume that the lengths of s_0 and s_1 are public). In the real world, $b \oplus r$, s_0 and s_1 all follow uniformly random distribution. So $\mathcal{A}$ cannot tell whether it is interacting with the real protocol or simulator $\mathcal{S}$.
3. If adversary $\mathcal{A}$ corrupts Receiver: $\mathcal{A}$ sends $b \oplus r$ to the simulator $\mathcal{S}$. $\mathcal{S}$ extracts b with the knowledge of r and inputs b to the ideal functionality $\mathcal{F}_{3-OT}$, and can then receive the ideal output m_b. After that, $\mathcal{S}$ generates $m_b \oplus mask_{b\oplus r}$ according to r and $mask_{b\oplus r}$ (since r and $mask_b \oplus r$ are known to Sender

[2] OT protocols in ABY^3 and Falcon are secure against semi-honest adversaries.

and Receiver). Finally, $\mathcal{S}$ sends $m_b \oplus mask_{b\oplus r}$ to the adversary $\mathcal{A}$ who opens the message and gets the correct output. r and $mask_{b\oplus r}$ simulated by $\mathcal{S}$ are uniformly random, which is the same as in the real execution. So the adversary $\mathcal{A}$ cannot tell whether it is interacting with the real protocol or simulator $\mathcal{S}$.

This completes the security proof of Π_{3-OT}. □

3.6 Secure Activation Function

Suppose that the participants finish fully connected layer inference or convolutional layer inference and now activation layer follows. Each party holds a share of the evaluation result (in fact, a series of shares of the result's entries) after evaluating a fully connected layer or a convolutional layer. Suppose that P_i holds the share z_i. Now the participants want $[\![Sign(z)]\!]_3^{m'}$ given $[\![z]\!]_3^m$. Here $m = 2^l$ represents the modulo of the last layer, m' represents the modulo of the next layer. To the purpose, we have three phases: first convert three-party sharing $[\![z]\!]_3^m$ (among P_0, P_1, P_2) to two-party secret sharing $[\![z]\!]_2^m$ (between P_0 and P_1); then get $[\![MSB(z)]\!]_2^B$ by evaluating a parallel adder circuit on $\mathbb{Z}_2$ with 3-input AND gate technique; and finally get $[\![Sign(z)]\!]_3^{m'}$ by invoking three-party oblivious transfer Π_{3-OT}. The following presents the details.

From Three-Party Secret Sharing to Two-Party Secret Sharing. All participants jointly call the *3-out-of-3 randomness* protocol and each gets a random value a_i (such that $a_1 + a_2 + a_3 = 0$), then P_2 sends $z_2 + a_2$ to one of the remaining parties (say P_0). This will not leak any information as $z_2 + a_2$ sent by P_2 is distributed uniformly at random. Now P_0 holds $s_0 = (z_0 + a_0) + (z_2 + a_2)$ and P_1 holds $s_1 = z_1 + a_1$. We have $s = s_0 + s_1 = \sum(z_i) + \sum(a_i) = \sum(z_i) = z$ and thereby manage two-party secret sharing, i.e., s_0 and s_1 are secret shares of z (and s).

Achieving MSB Extraction with Specific Part of Parallel Prefix Adder Circuit. By converting three-party secret sharing to two-party secret sharing, P_0 and P_1 reach $[\![s]\!]_2^m = (s_0, s_1)$ where $m = 2^l$. One may view s_i ($i = 0, 1$) as a bit string of length l. As $s_0 = s_0 \oplus 0$, we then have $[\![s_0]\!]_2^B$, a toy "two-party secret sharing" of s_0 such that P_0 holds s_0 and P_1 holds 0. Similarly, we have a toy "two-party secret sharing" $[\![s_1]\!]_2^B$. One might gain the advantage of communication round complexity in the MSB extraction protocol by the joint exploit of toy "two-party secret sharing" and standard secret sharing.

Now, the participants P_0, P_1 (and P_2) are ready to extract the MSB of s (see Algorithm 5). We use $s_0[i]$ and $s_1[i]$ to denote the i-th bits of s_0 and of s_1 respectively. It is obvious that $MSB(s) = s_0[l-1] \oplus s_1[l-1] \oplus c$, where c is a carry bit generated from lower $l-1$ bit pairs. Three parties then use 3-input AND gate to evaluate the parallel prefix adder circuit to compute the carry bit c. For a 3-input AND gate with one free input in the circuit, we can replace it with a 2-input AND gate by using well known Beaver's Triplet technique [6] (a

Algorithm 5. MSB Extraction: Π_{msb}

```
Input: $P_0$ and $P_1$ input $[\![s_0]\!]_2^B$ and $[\![s_1]\!]_2^B$, $s_i \in \mathbb{Z}_{2^l}$, $P_2$ inputs $\perp$.
Output: $P_0$ and $P_1$ achieve $[\![b]\!]_2^B$, $b$ is the MSB of $s$ where $s = s_0 + s_1$.
1: $P_0$, $P_1$ input $[\![s_0]\!]_2^B$ and $[\![s_1]\!]_2^B$ to the parallel prefix adder circuit which is composed
   by 2-input and 3-input secure AND gates, $P_2$ plays as a helper. They collaborate
   to compute $[\![carry\_bit]\!]_2^B$ with SMC.
2: $[\![b]\!]_2^B = [\![s_0[l-1]]\!]_2^B \oplus [\![s_1[l-1]]\!]_2^B \oplus [\![carry\_bit]\!]_2^B$.
```

Algorithm 6. Convert $[\![MSB(z)]\!]_2^B$ to $[\![Sign(z)]\!]_3^{m'}$

```
Input: $P_0$ and $P_1$ input the shares $b_0, b_1$ in $[\![MSB(z)]\!]_2^B$ respectively ($MSB(z) = 0$ if
the said neuron is activated, and 1 otherwise), $P_2$ inputs $\perp$.
Output: $[\![t]\!]_3^{m'}$ among $P_0, P_1, P_2$ s.t. $\sum t_i = Sign(z)$. $m'$ is the modulo.
Parameter: $deact\_val = 0$ if maxpooling layer follows and $-1$ otherwise.
1: $P_1$ selects a secret value $r \in_R \mathbb{Z}_{m'}$ and calculates $m_0 = (1 \oplus b_1)act\_val + (0 \oplus
   b_1)deact\_val - r \ (mod\ m')$, $m_1 = (0 \oplus b_1)act\_val + (1 \oplus b_1)deact\_val - r \ (mod\ m')$.
2: Call $\Pi_{3-OT}$ ($P_0$ acts as receiver, $P_1$ sender, and $P_2$ helper), and $P_0$ receives $m_{b_0}$.
3: $P_0$ sets $t'_0 = m_{b_0}$, $P_1$ sets $t'_1 = r$, $P_2$ sets $t'_2 = 0$.
4: $P_i$ generates $a_i \in \mathbb{Z}_m$ with 3-out-of-3 randomness.
5: $P_i$ sets $t_i = t'_i + a_i$, and sends $t_i$ to $P_{i-1}$, then $P_i$ holds ($t_i$, $t_{i+1}$).
```

simplified version of 3-input AND gate) to further reduce the communication cost. The same circuit is used in [26] to handle the case where l is set as 32 or 64. In contrast, SecureBiNN always takes a smaller value (e.g., 16 or 18) of l determined by the structure of the previous layer. In other words, SecureBiNN has a smaller circuit size and this will further contribute to lower communication cost and less interaction rounds.

Converting $[\![MSB(z)]\!]_2^B$ to $[\![Sign(z)]\!]_3^{m'}$. Given $[\![MSB(z)]\!]_2^B$ between P_0 and P_1, we can now get $[\![Sign(z)]\!]_3^{m'}$ among P_0, P_1, P_2. Recall that if the MSB is 0, then the neuron hereof should be activated, and deactivated otherwise. As we use the activation function *Sign*, we have the activation value $act\_val = 1$ and the deactivation value $deact\_val = -1$. If a maxpooling layer follows however, we set $deact\_val$ as 0 (rather than -1) in order to adapt to maxpooling operation.

The details are shown in Algorithm 6. To achieve $[\![Sign(z)]\!]_3^{m'}$, we let P_1 generate symmetric messages by using its secret knowledge (secret share in $[\![MSB(z)]\!]_2^B$ and one-time randomness) along with activation/deactivation values. We then call our three-party oblivious transfer protocol Π_{3-OT} by viewing P_1 as the sender, P_0 the receiver, and P_2 the helper. The particular constructing of the messages enables P_0 to get the exact message decided by its secret share (in $[\![MSB(z)]\!]_2^B$) so that a simple conversion from two-party secret sharing to three-party sharing leads to the expected $[\![Sign(z)]\!]_3^{m'}$.

3.7 Batch Normalization

In SecureBiNN, batch normalization is always followed by an activation layer so we can put batch normalization operation and activation operation together. Consider the following formula for batch normalization [17]:

$$Y = \gamma \frac{X - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta. \tag{3}$$

Herein, γ and β are trainable parameters in the batch normalization, ϵ is a small constant to avoid the "Divide by Zero" error, μ and σ are parameters decided in the training process. Thus, all these parameters would be set as fixed values during inference. And we can rewrite Eq. (3) as

$$Y = \frac{\gamma}{\sqrt{\sigma^2 + \epsilon}} X + \beta - \frac{\gamma\mu}{\sqrt{\sigma^2 + \epsilon}}. \tag{4}$$

Let $\gamma' = \frac{\gamma}{\sqrt{\sigma^2+\epsilon}}$, $\beta' = \beta - \frac{\gamma\mu}{\sqrt{\sigma^2+\epsilon}}$, then we have

$$Y = \gamma' X + \beta'. \tag{5}$$

In most cases, γ' is positive and batch normalization layer is followed by an activation layer. It holds that

$$Sign(\gamma' X + \beta') = Sign(X + \frac{\beta'}{\gamma'}). \tag{6}$$

Thus, the model owner first encodes $\frac{\beta'}{\gamma'}$ into a fixed-point number over the ring $\mathbb{Z}_m$ and then uniformly samples θ_i s.t. $\sum_{i=0}^{2} \theta_i = \frac{\beta'}{\gamma'}$ and sends θ_i to P_i. If the participants need to perform batch normalization between a fully connected layer and an activation function, P_i only needs to add θ_i to the output share of the fully connected layer.[3]

In the input layer, X and $\frac{\beta'}{\gamma'}$ are both fixed-point numbers, but in the hidden layers, X is an integer and $\frac{\beta'}{\gamma'}$ is not. In this case, if $Sign(X + \frac{\beta'}{\gamma'}) = 1$ (which means $X + \frac{\beta'}{\gamma'} \geq 0$), there is

$$X \geq \lceil -\frac{\beta'}{\gamma'} \rceil \geq -\frac{\beta'}{\gamma'}. \tag{7}$$

So Eq. (8) holds and we can implement it instead.

$$Sign(\gamma' X + \beta') = Sign(X - \lceil -\frac{\beta'}{\gamma'} \rceil). \tag{8}$$

[3] Same for convolutional layers.

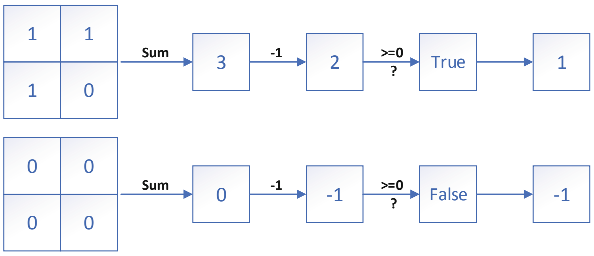

Fig. 1. Two examples of maxpooling

3.8 Maxpooling

We know that the neurons hereof should be activated (i.e., $act\_val = 1$) if $MSB(z) = 0$, and deactivated (i.e., $deact\_val = -1$) otherwise. Suppose that maxpooling layer follows now. If there exists 1 in some maxpooling step, then the max value is 1 and a standard maxpooling step would output 1, and -1 otherwise. This can be done generally by several *comparator* operations [35]. Whereas, our framework takes a different trick. Now we set the deactivation value as 0 rather than -1[4] and then check whether there exists 1 in the pool. If yes, then the sum of these entries minus 1 should be greater than or equal to 0. After a convolutional layer, participants convert the maxpooling layer to a 'sumpooling layer', i.e., they replace the 'Max' operation in the maxpooling layer with an 'Add' operation (see Fig. 1) and then apply $Sign$ function to the output of the 'Add' operation. Therefore, a single step of maxpooling layer can be done by an 'Add' operation and a MSB extraction operation. Prior frameworks, like MiniONN [21], generally lean on the secure comparison protocols (i.e., *comparator*) and secure multiplications to find the largest value in the pool, and the numbers of comparator and multiplication calls have a linear relationship with filter size.

There is another optimization. The participants need not convert the shares to three-party secret shares after an activation operation which is always followed by a maxpooling layer. P_0 and P_1 can *locally* implement a 'sumpooling layer' and then participants implement the activation operation again.

4 Experiment Results and Analysis

We execute SecureBiNN with Python (about 3k lines of code), and computations are implemented through numpy 1.19.0 [36] and tensorflow 2.5.1 [4]. We run our experiments on three ecs.c7.2xlarge servers from Alibaba Cloud, each with 8vCPU and 16GB RAM. We evaluate SecureBiNN in the following settings:

[4] The maxpooling operation comes on the heels of the activation layer, and can be achieved by changing the final output of the activation layer as in Algorithm 6.

Table 2. Evaluation results of SecureBiNN under LAN setting on MNIST and comparisons with prior work. Runtime is reported in seconds and Comm in MB. $*$: Falcon does not report the consumption (runtime and communication cost) required in its offline phase. The results hereof are only for its online phase. $\triangle$: the protocols can be secure in both semi-honest adversary model and malicious adversary model, and we only consider their costs under the semi-honest model

2PC/3PC	Framework	Network-A			Network-B			Network-C		
		Time(s)	Comm. (MB)	Acc.	Time	Comm.	Acc.	Time	Comm.	Acc.
2PC	EzPC ($\triangle$)	0.7	76	0.976	0.6	70	0.99	5.1	501	0.99
	Gazelle	0.09	0.5	0.976	0.29	8	0.99	1.16	70	0.99
	MiniONN	1.04	15.8	0.976	1.28	47.6	0.990	9.32	657.5	0.99
	XONN	0.13	4.29	0.976	0.16	38.3	0.986	0.15	32.1	0.99
3PC	ABY^3 ($\triangle$)	0.008	0.5	–	0.01	5.2	–	–	–	–
	SecureNN	0.043	2.1	0.934	0.076	4.05	0.988	0.13	8.86	0.99
	Falcon($\triangle$, $*$)	0.011	0.012	0.974	0.009	0.049	0.978	0.042	0.51	0.986
	Secure BiNN(ours)	0.003	0.005	0.973	0.007	0.032	0.972	0.020	0.357	0.984

Table 3. Evaluation results of SecureBiNN under WAN setting on MNIST and comparisons. Runtime is reported in seconds and Comm in MB. $*$, $\triangle$: same as in Table 2

2PC/3PC	Framework	Network-A			Network-B			Network-C		
		Time(s)	Comm. (MB)	Acc.	Time	Comm.	Acc.	Time	Comm.	Acc.
2PC	EzPC ($\triangle$)	1.7	76	0.976	1.6	70	0.99	11.6	501	0.99
3PC	SecureNN	243	2.1	0.934	3.06	4.05	0.988	3.93	8.86	0.99
	Falcon($\triangle$, $*$)	0.99	0.012	0.974	0.76	0.049	0.978	3.0	0.51	0.986
	Secure BiNN(ours)	0.248	0.005	0.973	0.440	0.032	0.972	1.15	0.36	0.984

- LAN: We set up three servers in Ulanqab, and the network latency and bandwidth are 0.2 ms and 625MBps respectively.
- WAN: We set up three servers in Ulanqab, Hangzhou and Shanghai, the latency and the bandwidth between the servers are 36 ms and 5MBps respectively. Note that these parameters are close to those of the Internet in daily use, which further proves our solution is available in practical settings.

At present, there are few secure inference frameworks applied to BiNNs, so in addition to XONN [28] (a 2-party scheme for BiNNs), we select other works for comparison as well. We choose 3-party frameworks including ABY^3 [24], SecureNN [34], Falcon [35], Chameleon [29], and some well known 2-party protocols, i.e., EzPC [10], Gazelle [18], MiniONN [21], for sufficient comparisons.

We measure running time, communication volume and accuracy. We repeat each experiment 10 times, and then take average running time. Although we need secret shares of helping terms in implementing secure AND gates, we emphasize those are done online, and *no offline phase is required.*

Table 4. Performance comparisons among different frameworks on Lenet-5. $*$, $\triangle$: same as in Table 2

Framework	SecureNN	CrypTFlow	Falcon($\triangle$, $*$)	SecureBiNN
Time(LAN, s)	0.23	0.058	0.047	0.025
Time(WAN, s)	4.08	–	3.06	0.602
Comm.(MB)	18.94	–	0.74	0.522
Acc	0.991	–	0.991	0.989

4.1 Experimental Evaluation on MNIST

We evaluate three different NN on MNIST dataset [20] (60,000 training samples and 10,000 test samples). Each sample is a 28×28 handwritten digital image. We use SecureBiNN to make inference on these networks and measure its running time and communication cost. Bellow are the architectures of three models:

Network-A: FC(128) − FC(128) − FC(10);
Network-B: Conv(28×28, 5 channels) − FC(100) − FC(10);
Network-C: Conv(28×28, 16 channels) − MP(2×2) − Conv(12×12, 16 channels) − MP(2×2) − FC(100) − FC(10).

Network-A is a 3 layer fully-connected network, Network-B is a 3 layer network with a single convolution layer followed by 2 fully connected layers, and Network-C is a network with 2 convolutional layers, 2 maxpooling layers and 2 fully-connected layers. These models are used in many prior work, and we also compare our experimental results with some of them. Tables 2 and 3 show the comparisons under LAN and WAN settings. Further, we compare the performance of SecureBiNN with SecureNN [34], CryptTFlow [27], Falcon [35] on Lenet-5 [20], and the results are shown in Table 4. Accuracy rate of each framework is only for reference as it may vary with the changes of model parameters.

Comparing to prior work, SecureBiNN shows its competitive performance partially due to the tricky encoding, i.e., the parameters are encoded over a ring with a smaller size. Other frameworks like Falcon, ABY^3 use $\mathbb{Z}_{2^{32}}$, meaning that parameters are represented with 32 bits. The advantages of parameter encoding lead to significant reduction in the amount of communication consumption between participants. Another advantage goes to maxpooling. In prior protocols, maxpooling cost is generally exorbitant as each pooling step requires overabundant MSB extraction operations (pool size minus 1 times). However, one single MSB extraction is needed in each pooling step of SecureBiNN.

4.2 Experimental Evaluation on CIFAR-10

Table 5 evaluates more networks on CIFAR-10 [19], including binarized versions of Fitnet 1–4 [30]. We train BiNN according to each structure. In the experiments, we encode the parameters of input/output layers over $\mathbb{Z}_{2^{32}}$ and set the precision of these parameters as 13 bits. By the architecture of these networks

Table 5. Evaluations of SecureBiNN under LAN/WAN on CIFAR-10. Runtime is in seconds and Comm in MB. s means that the number of neurons in fully connected layers (or the number of filters in convolutional layers) is increased by a factor of s

Arch.	s	Time(s, LAN)	Time(s, WAN)	Comm.(MB)	Acc.
Fitnet 1	1	0.112	2.776	3.074	0.688
	2	0.204	2.958	6.364	0.778
	3	0.527	3.447	16.609	0.815
Fitnet 2	1	0.173	2.901	5.178	0.765
	2	0.351	3.419	10.773	0.797
	3	0.527	3.769	16.609	0.812
Fitnet 3	1	0.367	3.739	11.806	0.811
	2	0.810	5.244	24.654	0.834
	3	1.368	7.313	39.878	0.836
Fitnet 4	1	0.430	6.393	13.660	0.789
	2	0.909	6.062	29.111	0.808
	3	1.560	8.500	48.195	0.810

in Table 5, when the scale factors are 1 and 2, most of the weighted parameters in the hidden layers are encoded over $\mathbb{Z}_{2^{14}}$, $\mathbb{Z}_{2^{15}}$; for the factor of 3, most parameters are over $\mathbb{Z}_{2^{17}}$. Many previous tests have been done on Fitnet 1, and we compare SecureBiNN with these work. The results are shown in Table 6.

Table 6. Comparisons among different frameworks on Fitnet 1 under the LAN setting. s means that the number of neurons in fully connected layers (or the number of filters in convolutional layers) is increased by a factor of 3. △: same as in Table 2

Framework	MiniONN	Chameleon	EzPC (△)	Gazelle	XONN	SecureBiNN		
					$s=3$	$s=1$	$s=2$	$s=3$
Time(s)	544	52.67	265.6	15.48	5.79	0.112	0.204	0.527
Comm.(MB)	9272	2650	40683	1236	2599	3.074	6.364	16.60
Acc	0.816	0.816	0.816	0.816	0.819	0.688	0.778	0.815

Now, the advantages of our solution become more explicit, because Fitnet has more convolution, activation and maxpooling operations than Networks A, B, C (Tables 2, 3). XONN uses GC to evaluate the BiNN, the garbled table and keys transmission are known to be of lavish spending. Comparing to XONN, SecureBiNN needs far less communication overhead. Furthermore, when we increase the number of neurons in each layer by a factor of 3, most of the weighted parameters are encoded over $\mathbb{Z}_{2^{17}}$. This means that in an activation operation, we only need to input a 17-bit number instead of a 32-bit number into the parallel prefix adder circuit (the bit length is almost half of the latter). This substantially reduces the cost of the activation function, which is always overbearing consumption and bottleneck of such kinds of protocols. This special feature offers a significant advantage to Fitnet 1 architecture compared to prior work.

Table 7. Experiment results of SecureBiNN on real-world medical datasets. We drop the rows with null values from the sets and perform min-max scale on validation sets

Dataset	# of Samples		Evaluation results for a single input			
	Tr	Val	Time (s, LAN)	Time (s, WAN)	Comm. (MB)	Acc.
Breast cancer [1]	455	114	0.005	0.014	0.002	0.991
Diabetes [32]	614	154	0.005	0.014	0.002	0.812
Liver [2]	463	116	0.005	0.017	0.003	0.741
Malaria [3]	22048	5512	0.072	0.377	1.861	0.930

4.3 Experimental Evaluation on Real-World Medical Datasets

One of the most common scenarios where such schemes can be used is to aid in diagnosis. To demonstrate the effectiveness of SecureBiNN in real-world scenarios, we select below four real-world medical datasets to simulate real-world usage scenarios: breast cancer dataset [1], Pima Indians diabetes dataset [32], Indian liver patient records [2] and malaria cell images dataset [3]. The first three sets consist of several numerical features related to patients' information. The last one consists of the images of different sizes, so we reshape all the images to 32×32. All tasks are to predict whether a patient is infected with the corresponding disease. We evaluate the following models for each dataset:

Breast Cancer: FC(16) − FC(16) − FC(2);
Diabetes: FC(20) − FC(20) − FC(2); Liver: FC(32) − FC(32) − FC(2);
Malaria: CONV(5×5, 36 channels) − MP(2×2) − CONV(5×5, 36 channels) − MP(2×2) − FC(72) − FC(2).

We split each dataset into training and validation portions before evaluation, and the partition details and the evaluation results are shown in Table 7.

5 Conclusion and Future Work

The paper proposes the first privacy-preserving three-party discrete neural network inference framework supporting fast execution and low communication cost. As most existing SMC proposals, our framework SecureBiNN is secure against semi-honest adversary. Due to the parameter distribution of binarized neural network, fewer bits suffice in SecureBiNN to represent a parameter, reducing further the lengths of the messages generated in the participant interactions. Experiments confirm its lower communication cost at similar accuracy to state-of-the-art. In this paper, P_0 and P_1 are able to implement secure AND gate and

3-party OT with the help of P_2. However, in the case where P_2 is a malicious adversary, the method proposed in this paper cannot be directly used. More attempts might be made to construct actively secure algorithms by introducing consistency checking mechanisms against malicious adversary.

Acknowledgement. The work is supported by the National Natural Science Foundation of China (Grant No. 61971192), Shanghai Municipal Education Commission (2021-01-07-00-08-E00101), and Shanghai Trusted Industry Internet Software Collaborative Innovation Center.

A Related Work

The privacy-preserving neural network inference technology is mainly divided into two routes, one is based on HE, and another on SMC.

In the former route, one commonly used HE algorithm is CKKS [12], a computation-expensive leveled-FHE scheme with multiplication depth being kept within certain range. In 2016, Nathan et al. propose Cryptonets [12] using the CKKS algorithm. Since CKKS can only support addition and multiply operations, it is difficult to implement the Sigmoid or the ReLU activation functions, and only the square function can be used which makes low model accuracy.

A representative example in SMC-based route goes to SecureML [23] which uses Beaver's Triplet [6] to realize multiplication. As it requires numerous multiplication triples, SecureML supports limited practicability. Subsequent schemes (e.g., ABY [13]) significantly reduce the running time and communication cost. Other frameworks including BiNN inference framework XONN [28] mainly rely on GC. Some 3PC frameworks (e.g., ABY3 [24] and Falcon [35]) use replicated secret sharing [14]. Therein, three parties can directly perform privacy-preserving multiplications locally according to the input to obtain the output and no interaction is required. Thus, these 3PC frameworks are generally more efficient and faster than those 2PC frameworks, an advantage meeting actual needs.

References

1. Breast cancer wisconsin (diagnostic) data set (1995). Accessed 25 Apr 2022. https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29
2. Indian liver patient records (2013). Accessed 25 Apr 2022. https://archive.ics.uci.edu/ml/datasets/liver+disorders
3. Malaria cell images dataset (2019). Accessed 25 Apr 2022. https://www.kaggle.com/datasets/iarunava/cell-images-for-detecting-malaria
4. Abadi, M., et al.: Tensorflow: a system for large-scale machine learning. In: Keeton, K., Roscoe, T. (eds.) 12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, 2–4 November 2016, pp. 265–283. USENIX Association (2016). https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi

5. Araki, T., Furukawa, J., Lindell, Y., Nof, A., Ohara, K.: High-throughput semi-honest secure three-party computation with an honest majority. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, 24–28 October 2016, pp. 805–817. ACM (2016). https://doi.org/10.1145/2976749.2978331
6. Beaver, D.: One-time tables for two-party computation. In: Hsu, W.-L., Kao, M.-Y. (eds.) COCOON 1998. LNCS, vol. 1449, pp. 361–370. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-68535-9_40
7. Boemer, F., Costache, A., Cammarota, R., Wierzynski, C.: ngraph-he2: a high-throughput framework for neural network inference on encrypted data. In: Brenner, M., Lepoint, T., Rohloff, K. (eds.) Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography, WAHC@CCS 2019, London, UK, 11–15 November 2019, pp. 45–56. ACM (2019). https://doi.org/10.1145/3338469.3358944
8. Bourse, F., Minelli, M., Minihold, M., Paillier, P.: Fast homomorphic evaluation of deep discretized neural networks. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10993, pp. 483–512. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96878-0_17
9. Canetti, R.: Universally composable security. J. ACM **67**(5) (2020). https://doi.org/10.1145/3402457
10. Chandran, N., Gupta, D., Rastogi, A., Sharma, R., Tripathi, S.: Ezpc: programmable and efficient secure two-party computation for machine learning. In: IEEE European Symposium on Security and Privacy, EuroS&P 2019, Stockholm, Sweden, 17–19 June 2019, pp. 496–511. IEEE (2019). https://doi.org/10.1109/EuroSP.2019.00043
11. Chen, X., Liu, C., Li, B., Lu, K., Song, D.: Targeted backdoor attacks on deep learning systems using data poisoning. CoRR abs/1712.05526 (2017). https://arxiv.org/abs/1712.05526
12. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10624, pp. 409–437. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70694-8_15
13. Demmler, D., Schneider, T., Zohner, M.: ABY - a framework for efficient mixed-protocol secure two-party computation. In: 22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, 8–11 February 2015. The Internet Society (2015). https://www.ndss-symposium.org/ndss2015/aby--framework-efficient-mixed-protocol-secure-two-party-computation
14. Furukawa, J., Lindell, Y., Nof, A., Weinstein, O.: High-throughput secure three-party computation for malicious adversaries and an honest majority. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10211, pp. 225–255. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56614-6_8
15. Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K.E., Naehrig, M., Wernsing, J.: Cryptonets: applying neural networks to encrypted data with high throughput and accuracy. In: Balcan, M., Weinberger, K.Q. (eds.) Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, 19–24 June 2016. JMLR Workshop and Conference Proceedings, vol. 48, pp. 201–210. JMLR.org (2016). https://proceedings.mlr.press/v48/gilad-bachrach16.html

16. Ibarrondo, A., Chabanne, H., Önen, M.: Banners: binarized neural networks with replicated secret sharing. In: Borghys, D., Bas, P., Verdoliva, L., Pevný, T., Li, B., Newman, J. (eds.) IH&MMSec 2021: ACM Workshop on Information Hiding and Multimedia Security, Virtual Event, Belgium, 22–25 June 2021, pp. 63–74. ACM (2021). https://doi.org/10.1145/3437880.3460394
17. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. In: Bach, F.R., Blei, D.M. (eds.) Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6–11 July 2015, JMLR Workshop and Conference Proceedings, vol. 37, pp. 448–456. JMLR.org (2015). https://proceedings.mlr.press/v37/ioffe15.html
18. Juvekar, C., Vaikuntanathan, V., Chandrakasan, A.: GAZELLE: a low latency framework for secure neural network inference. In: Enck, W., Felt, A.P. (eds.) 27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, 15–17 August 2018, pp. 1651–1669. USENIX Association (2018). https://www.usenix.org/conference/usenixsecurity18/presentation/juvekar
19. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images. Handb. Systemic Autoimmune Dis. **1**(4) (2009)
20. Lecun, Y., Bottou, L.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
21. Liu, J., Juuti, M., Lu, Y., Asokan, N.: Oblivious neural network predictions via minionn transformations. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, 30 October–03 November 2017, pp. 619–631. ACM (2017). https://doi.org/10.1145/3133956.3134056
22. Mishra, P., Lehmkuhl, R., Srinivasan, A., Zheng, W., Popa, R.A.: Delphi: a cryptographic inference system for neural networks. In: Zhang, B., Popa, R.A., Zaharia, M., Gu, G., Ji, S. (eds.) PPMLP 2020: Proceedings of the 2020 Workshop on Privacy-Preserving Machine Learning in Practice, Virtual Event, USA, November 2020, pp. 27–30. ACM (2020). https://doi.org/10.1145/3411501.3419418
23. Mohassel, P., Zhang, Y.: Secureml: a system for scalable privacy-preserving machine learning. In: 2017 IEEE Symposium on Security and Privacy (SP), pp. 19–38 (2017). https://doi.org/10.1109/SP.2017.12
24. Mohassel, P., Rindal, P.: Aby^3: a mixed protocol framework for machine learning. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, 15–19 October 2018, pp. 35–52. ACM (2018). https://doi.org/10.1145/3243734.3243760
25. Ohata, S., Nuida, K.: Communication-efficient (client-aided) secure two-party protocols and its application. In: Bonneau, J., Heninger, N. (eds.) FC 2020. LNCS, vol. 12059, pp. 369–385. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51280-4_20
26. Patra, A., Schneider, T., Suresh, A., Yalame, H.: ABY2.0: improved mixed-protocol secure two-party computation. In: Bailey, M., Greenstadt, R. (eds.) 30th USENIX Security Symposium, USENIX Security 2021, 11–13 August 2021, pp. 2165–2182. USENIX Association (2021). https://www.usenix.org/conference/usenixsecurity21/presentation/patra
27. Rathee, D., et al.: Cryptflow2: practical 2-party secure inference. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) CCS 2020: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, 9–13 November 2020, pp. 325–342. ACM (2020). https://doi.org/10.1145/3372297.3417274

28. Riazi, M.S., Samragh, M., Chen, H., Laine, K., Lauter, K.E., Koushanfar, F.: XONN: xnor-based oblivious deep neural network inference. In: Heninger, N., Traynor, P. (eds.) 28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, 14–16 August 2019, pp. 1501–1518. USENIX Association (2019). https://www.usenix.org/conference/usenixsecurity19/presentation/riazi
29. Riazi, M.S., Weinert, C., Tkachenko, O., Songhori, E.M., Schneider, T., Koushanfar, F.: Chameleon: a hybrid secure computation framework for machine learning applications. In: Proceedings of the 2018 on Asia Conference on Computer and Communications Security, ASIACCS 2018, pp. 707–721. Association for Computing Machinery, New York (2018). https://doi.org/10.1145/3196494.3196522
30. Romero, A., Ballas, N., Kahou, S.E., Chassang, A., Gatta, C., Bengio, Y.: Fitnets: hints for thin deep nets. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, 7–9 May 2015, Conference Track Proceedings (2015). https://arxiv.org/abs/1412.6550
31. Shokri, R., Stronati, M., Song, C., Shmatikov, V.: Membership inference attacks against machine learning models. In: 2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, 22–26 May 2017, pp. 3–18. IEEE Computer Society (2017). https://doi.org/10.1109/SP.2017.41
32. Smith, J., Everhart, J., Dickson, W., Knowler, W., Johannes, R.: Using the adap learning algorithm to forcast the onset of diabetes mellitus. In: Proceedings - Annual Symposium on Computer Applications in Medical Care, vol. 10 (1988)
33. Tramèr, F., Zhang, F., Juels, A., Reiter, M.K., Ristenpart, T.: Stealing machine learning models via prediction apis. In: Holz, T., Savage, S. (eds.) 25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, 10–12 August 2016, pp. 601–618. USENIX Association (2016). https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/tramer
34. Wagh, S., Gupta, D., Chandran, N.: Securenn: 3-party secure computation for neural network training. Proc. Priv. Enhanc. Technol. **2019**(3), 26–49 (2019). https://doi.org/10.2478/popets-2019-0035
35. Wagh, S., Tople, S., Benhamouda, F., Kushilevitz, E., Mittal, P., Rabin, T.: Falcon: Honest-majority maliciously secure framework for private deep learning. Proc. Priv. Enhanc. Technol. **2021**(1), 188–208 (2021). https://doi.org/10.2478/popets-2021-0011
36. van der Walt, S., Colbert, S.C., Varoquaux, G.: The numpy array: a structure for efficient numerical computation. Comput. Sci. Eng. **13**(2), 22–30 (2011). https://doi.org/10.1109/MCSE.2011.37