IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received July 31, 2019, accepted August 21, 2019, date of publication September 4, 2019, date of current version October 4, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2939410

MOBIUS: Model-Oblivious Binarized Neural Networks

HIROMASA KITAI[1], JASON PAUL CRUZ[1], NAOTO YANAI[1], NAOHISA NISHIDA[2], TATSUMI OBA[2], YUJI UNAGAMI[2], TADANORI TERUYA[3], NUTTAPONG ATTRAPADUNG[3], TAKAHIRO MATSUDA[3], AND GOICHIRO HANAOKA[3]
[1]Graduate School of Information Science and Technology, Osaka University, Osaka 565-0871, Japan
[2]Panasonic Corporation, Osaka 571-8501, Japan
[3]Cyber Physical Security Research Center, National Institute of Advanced Industrial Science and Technology, Tokyo 135-0064, Japan

Corresponding author: Naoto Yanai (yanai@ist.osaka-u.ac.jp)

This work was supported in part by the Secom Science and Technology Foundation and in part by the JST CREST under Grant JPMJCR19F6.

ABSTRACT A privacy-preserving framework in which a computational resource provider receives encrypted data from a client and returns prediction results without decrypting the data, i.e., oblivious neural network or encrypted prediction, has been studied in machine learning. In this work, we introduce and explore a new problem called the *model-oblivious problem*, where a trainer can delegate a protected model to a resource provider without revealing the original model itself to the resource provider. The resource provider can then offer prediction on a client's input data, which is additionally kept private from the resource provider. To solve this problem, we present *MOBIUS* (Model-Oblivious BInary neUral networkS), a new system that combines *Binarized Neural Networks (BNNs)* and secure computation based on secret sharing as tools for scalable and fast privacy-preserving machine learning. BNNs improve computational performance by binarizing values in training to -1 and $+1$, while secure computation based on secret sharing provides fast and various computations under encrypted forms via modulo operations with a short bit length. However, combining these tools is not trivial because their operations have different algebraic structures. MOBIUS uses improved procedures of BNNs and secure computation that have compatible algebraic structures without downgrading prediction accuracy. We present an implementation of MOBIUS in C++ using the ABY library (NDSS 2015). Then, we conduct experiments using several datasets, including the MNIST, Cancer, and Diabetes datasets, and the results show that MOBIUS outperforms SecureML (IEEE S&P 2017), which is the only other work that can potentially tackle the model-oblivious problem, in terms of both accuracy and computational time. Compared with TAPAS (ICML 2018) as a state-of-the-art BNN-based system, MOBIUS is *three orders of magnitude faster* without downgrading the accuracy despite solving the model-oblivious problem.

INDEX TERMS Model obliviousness, neural network predictions, privacy-preserving machine learning, secure computation.

I. INTRODUCTION

A. BACKGROUND

Machine learning methods are widely used in various situations, such as healthcare, manufacturing, and financial services. Consequently, privacy has become a serious concern in the use of big data. Consider a prediction-as-a-service application of machine learning with three entities, namely, a *trainer*, a *resource provider*, and a *client*. A trainer trains a model with plaintexts and then uploads the model to a resource provider, e.g., a cloud service. A client can then utilize such model by accessing the cloud and providing input data to the resource provider for prediction. However, this kind of system is insecure and impractical when the resource provider is malicious. In general, the following two features are important for a practical and secure use of machine learning:

- Guarantee the privacy of a client's input data from a resource provider without downgrading prediction performance.
- Guarantee the privacy of a trained model from a resource provider and from clients.

The associate editor coordinating the review of this manuscript and approving it for publication was Mahmoud Barhamgi.

To satisfy the first feature, different privacy-preserving frameworks have been proposed to keep the input data of clients private from resource providers [1]–[8], but a privacy-preserving mechanism should be used in a prediction to guarantee the privacy of a client and the input data. However, a privacy-preserving mechanism may downgrade the throughput of a model and may thus not be used because of poor performance. To solve this dilemma, **a privacy-preserving mechanism that does not downgrade performance of a model is necessary**.

Regarding the second feature, in contrast, **no framework that makes the model itself private, i.e., the resource provider computing the prediction is oblivious about the model, has been proposed**.[1]

In general, training a model needs significant amounts of data and heavy computations, and thus the model itself is an important resource to a trainer. Several machine learning systems provide prediction as a service in the cloud, but the aforementioned problem regarding the second feature implies that the resource provider who manages a cloud server is highly trusted. For example, a resource provider can maliciously use a model stored in a cloud server it owns without permission from the trainer to provide any service. Therefore, a trainer who owns a dataset and trains a model has to completely trust a resource provider who provides a prediction service. Otherwise, a trainer who wants to maintain privacy and does not completely trust a resource provider will hesitate to outsource machine learning services. To solve this problem, a model should be encrypted to prevent unauthorized entities, including the resource provider, from accessing the model itself.

1) MOTIVATING EXAMPLE

The main goal of this work is to create a system that keeps a trained model private from the clients and the resource provider itself and allows the clients to keep their input data private from the resource provider. We call this the **model-oblivious** problem.

Figure 1 shows an example scenario describing the intuition behind the model-oblivious problem. In this figure, a hospital, a cloud server, and doctors represent a trainer, a resource provider, and clients, respectively. The hospital trains a model with datasets it collected, encrypts the model, and then publishes the encrypted model on a cloud server, such as AmazonEC2, to make it publicly available to doctors. The cloud server can then execute a prediction for an input provided by a doctor by using the encrypted model without decrypting it. Since the model is encrypted, the cloud server cannot extract information from it or use it for other purposes.

2) KEY QUESTION

The main technical challenge is solving the model-oblivious problem without drastically downgrading throughput caused

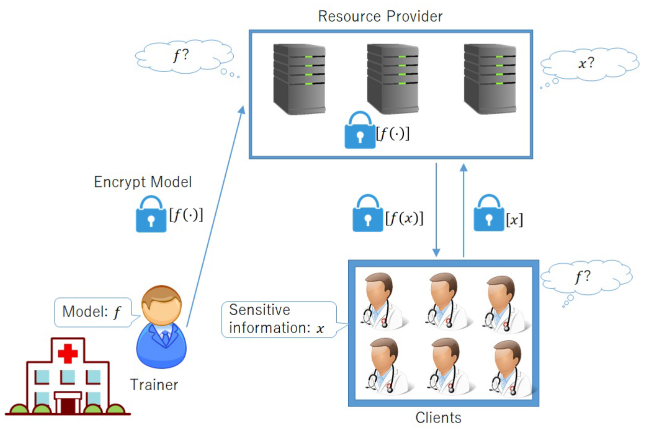


FIGURE 1. Example scenario of model-oblivious problem.

by the use of additional secure computation. Intuitively, making the computation of prediction oblivious about a model can be achieved by using secure computation in both the trained model and the input for the prediction. However, making the prediction oblivious about a model makes existing speed-up techniques for oblivious prediction [4], [8] unavailable. Consequently, solving the model-oblivious problem without downgrading performance (relative to the performance of related work) is difficult. We note that oblivious prediction [1]–[8] and encrypted training [10]–[14] do not imply the features of the model-oblivious problem.

B. CONTRIBUTION

In this work, we propose a new system named *Model-Oblivious BInarized neUral networkS (MOBIUS)*,[2] which enables scalable encrypted prediction and the use of an encrypted model, i.e., making prediction oblivious about the model. MOBIUS is scalable in the sense of being able to support an easy construction using a well-known approach for secure computation, and we will show a proof-of-concept of such implementation. MOBIUS uses *binarized neural networks (BNNs)* [15] and *secure computation based on secret sharing* as its main tools. BNNs are neural networks whose values for weight matrices and activation functions are binarized to $+1$ or -1. BNNs have an advantage over traditional neural networks in computational efficiency, since they are operated on binarized values, rather than real numbers. Secure computation based on secret sharing distributes input data from a client as shares such that an individual share leaks nothing about the original data, and it can evaluate the data without reconstructing the data via the homomorphism of the shares. A bit length of shares can be shortened in comparison with conventional cryptography, and thus the resulting secure computation can perform better than other cryptographic tools, such as fully homomorphic encryption (FHE) [16].

[1]The techniques used by SecureML [9] can potentially be used to secure a trained model, but this was not part of their motivation and was not discussed in their paper. See Section IV for details.

[2]We published a paper with the same title as this manuscript in arXiv. This manuscript is an improved version that includes additional experimental results.

MOBIUS guarantees that a prediction is oblivious about a trained model against an honest-but-curious adversary.

Below we will briefly describe that some difficulties arise when one attempts to apply secure computation techniques directly to the original BNN as per [15]. We resolve this by proposing *improved BNNs* that are efficiently compatible with secure computation.

We present an implementation of MOBIUS based on secret sharing and our improved BNNs using C++ and the ABY library [17]. We conducted experiments using several datasets, such as the MNIST dataset, and the results show that MOBIUS can perform a prediction within 0.76 seconds with 95.9% as accuracy. This performance is six times faster than SecureML [9], which is the only other work that can potentially address the model-oblivious problem, as well as improvements in accuracy. Moreover, MOBIUS outperforms TAPAS [14] and FHE-DiNN [12], which are works on privacy-preserving prediction based on BNNs, in terms of both accuracy and computational performance despite solving the model-oblivious problem. We demonstrate that MOBIUS is three orders of magnitude faster than TAPAS through experiments with several datasets, which have privacy implications. These results have been obtained even without optimizing our implementation (See Section VI-C for details).

C. DIFFICULTIES AND OUR APPROACH

1) NEURAL NETWORKS

Traditional neural networks consist of *full connection* layers and *activation* layers, which correspond to matrix multiplications (between input vectors and model matrices) and non-linear operations, respectively. *Batch normalization* layers [18] are added so as to improve the prediction accuracy to traditional networks. *Binarized neural networks* (BNN) [15] consists of these three types of layers albeit in the *binarized* forms, rather than real numbers as in traditional networks; this was done so as to improve computational efficiency. In particular, due its binarized forms, BNN of [15] chose to perform batch normalization using the so-called shift-based batch normalization (SBN) method, which is a boolean operation involving bit shifting. Note that SBN somewhat generally degrades the prediction accuracy from traditional neural networks (although the experiments on *specific* networks in [15] achieved no accuracy loss.)

2) APPLYING SECURE COMPUTATION

Secret-sharing based secure computation can be executed in *boolean* or *arithmetic* forms [17]. On-the-fly conversions among these forms [17] are also available but they do not come for free. For full connection and batch normalization layers, naively, one may apply secret-sharing based secure computation to the original BNN of [15] in the following manners:

- *Boolean shares for both types of layers.* As matrix multiplications are arithmetic in nature, and it is known [17] that using boolean shares for arithmetic computation would result in computationally inefficient protocols, we can conclude that this method would be inefficient.
- *Arithmetic shares for full connection and boolean shares for batch normalization.* This method provides suitable formats for respective layers; however, a costly conversion for arithmetic-to-boolean shares (equivalent to bit decomposition protocols in shared forms [19]) must be applied; this again yields inefficient protocols.
- *Arithmetic shares for both types of layers.* This requires computing shift-based computation for SBN in the form of arithmetic shares. Intuitively, this involves secure multiplication with the values that are powers of 2. However, this would inherit the accuracy degradation from the original BNN of [15].

To conclude, the above naive methods would result in either inefficient or lower-accuracy protocols.

3) OUR APPROACH: IMPROVED BNNs

Our idea is to modify the original BNN of [15] so that it would inherit the advantage on fast computation of BNN (due to binarized forms) while also improve accuracy. We execute this idea by using arithmetic shares for both layers and observing that we are not confined to multiply to only the power of 2, as is the case in the shift-based method for SBN, anymore. In other words, we can turn back to use the ‘‘vanilla’’ batch normalization as in [18] instead of its approximation in the shift-based method, where the former would result in better accuracy in general (as we can use any parameters instead of one the power of 2 as in the SBN method).

The next problem arises as such a vanilla batch normalization is performed in real numbers (not only integers), and hence it is somewhat not fully compatible with secure computation protocols, which are usually defined on integers. We solve this by multiplying the batch normalization parameters (which are real numbers) by a constant and then rounding them down to the nearest integers. We note that this kind of approximation would generally cause correctness errors in normal secure computation protocols; however, we observe that, in our networks, each batch normalization layer will be followed exactly by an activation layer, of which computation will not by affected by approximating inputs, and hence the rounding technique can be used.

D. RELATED WORK

SecureML [9] is the closest work to ours. The main motivation of SecureML is to provide scalable encrypted training, and solving the model-oblivious problem is not one of their goals. Although encrypted training does *not* imply the model-oblivious problem, SecureML can potentially consider the model-oblivious problem. We also note that encrypted training is out of the scope of our work.

As related work on combining BNNs with cryptography, TAPAS [14] and FHE-DiNN [12] based on FHE have been concurrently proposed. FHE-DiNN utilized discretized neural networks where domains are defined from $-w$ to $+w$, but its experiments were conducted with -1 to $+1$ exactly the same as BNNs. These works mainly aim to deploy fast computation of FHE [16], [20] in BNNs, and they did not consider the use of BNNs under modulo operation and the model-oblivious problem.

For developments of secure computation based on secret sharing, the state-of-the-art library is ABY3 [21] which is secure against a *malicious adversary* who ignores a protocol specification. Although our current implementation is secure against an *honest-but-curious* adversary who follows a protocol specification, we can extend it to a secure implementation against a malicious adversary by re-constructing with ABY3 as well as throughput.

We note that secure computation protocols with fixed-point arithmetic are available [22], [23] and is potentially useful for dealing with real numbers. Our method is similar in scaling by a constant factor; the difference is that we use arbitrary constants and use rounding, while the protocol in [22], [23] use the constants that are power of 2 and use truncating. Note also that the frameworks in [22], [23] are intended for general protocols such as divisions, while ours is for the mere purpose of applying to batch normalization.

Another approach for preserving privacy is differential privacy [24], which can prevent a trained model from leaking an individual record by perturbing the records with randomized noise. Given this capability, many works on neural networks use differential privacy [11], [25]. There are also works on further applications of differential privacy, e.g., data collection on an untrusted server [26], [27] or general function release [28]. However, according to Dowlin *et al.* [10], *the notion of differential privacy is not useful in the prediction phase*. Moreover, preventing unauthorized entities from accessing a model is outside the scope of differential privacy.

Related systems consider model extraction attack [29] as a kind of malicious usage of a model. Model extraction attack allows an adversary to abuse query APIs of a model to steal the hosted model. However, this kind of attack is out of the scope of model obliviousness because model obliviousness does not protect information obtained via queries to a model. To protect a model from model extraction attack, restriction in the use of APIs [29] or query monitoring [30] may be necessary.

II. PRELIMINARIES

In this section, we provide background on neural networks and secure computation to help in understanding our work.

A. BINARIZED NEURAL NETWORK

Binarized neural networks (BNNs) [15] were proposed to reduce overhead by minimizing data sizes. To do this, values presented in neural networks are binarized to reduce the required computational resources.

The original work on BNNs [15] described methods to binarize three protocols, namely, *full connection*, *batch normalization*, and *activation*, which are required in standard neural networks. Full connection computes matrix multiplications between vectors and weight matrices. Batch normalization makes the distribution for nodes uniform in the training phase and contributes to speeding up both training and prediction. Activation applies non-linear processing to output vectors, and a sign function is utilized in BNNs. Among the protocols described above, batch normalization has adopted a bit-shift method to be computed in a binarized form, which is different from well-known batch normalization algorithms [18], because the operations in well-known batch normalization algorithms require real numbers, consequently creating a bottleneck in the computations.

B. CRYPTOGRAPHIC PRELIMINARIES

In this section, we describe the notations and terminologies used in secure computation based on secret sharing.

1) SECRET SHARING

A t-out-of-n secret sharing scheme over a finite domain D consists of the following two algorithms:

$(\llbracket x \rrbracket_1, \ldots, \llbracket x \rrbracket_n) \leftarrow \mathsf{Share}(x)$: Share takes $x \in D$ as input, and outputs $\llbracket x \rrbracket_1, \ldots, \llbracket x \rrbracket_n \in D$.

$x \leftarrow \mathsf{Reconst}(\llbracket x \rrbracket_1, \ldots, \llbracket x \rrbracket_t)$: $\mathsf{Reconst}$ takes $\llbracket x \rrbracket_1, \ldots, \llbracket x \rrbracket_t \in D$ as input, and outputs $x \in D$.

In these algorithms, for $i \in \{1, \ldots, n\}$, $\llbracket x \rrbracket_i$ is called the i-th share of x. We denote $\llbracket x \rrbracket = (\llbracket x \rrbracket_1, \ldots, \llbracket x \rrbracket_n)$ as their shorthand. Any less than t shares of x over the t-out-of-n secret sharing scheme jointly give no information on x, whereas any $\geq t$ shares jointly determine x by using $\mathsf{Reconst}$. Several secret sharing schemes that have been proposed typically have finite domains, e.g., the ring of integers $\mathbb{Z}_M$ modulo M, where M is a positive integer greater than 1, and an ℓ-length binary string [31]. An i-th share of an ℓ-dimensional vector $\boldsymbol{v} = (x_1, \ldots, x_\ell)$ over a domain D consists of i-th shares of its components and is denoted by $\llbracket \boldsymbol{v} \rrbracket_i := (\llbracket x_1 \rrbracket_i, \ldots, \llbracket x_\ell \rrbracket_i)$. Analogously, an i-th share of a matrix is defined in the same way. Therefore, a secret sharing scheme over vectors, matrices, and tensors, among others, can be defined.

2) SECURE COMPUTATION BASED ON SECRET SHARING

We define sub protocols of secure computation that we utilized in our work. The following computations are defined over the ring of integers $\mathbb{Z}_M = \{0, \ldots, M-1\}$ modulo M. Several efficient implementations of the protocols have been provided [17], [32].

- $\llbracket c \rrbracket \leftarrow \mathsf{ADD}(\llbracket a \rrbracket, \llbracket b \rrbracket)$: ADD takes shares $\llbracket a \rrbracket$ and $\llbracket b \rrbracket$ of $a \in \mathbb{Z}_M$ and $b \in \mathbb{Z}_M$, respectively, as inputs, then outputs a share $\llbracket c \rrbracket$ of $a + b = c \in \mathbb{Z}_M$.
- $\llbracket c \rrbracket \leftarrow \mathsf{ADDConst}(\llbracket a \rrbracket, b)$: $\mathsf{ADDConst}$ takes share $\llbracket a \rrbracket$ of $a \in \mathbb{Z}_M$ and $b \in \mathbb{Z}_M$ as inputs, then outputs a share $\llbracket c \rrbracket$ of $a + b = c \in \mathbb{Z}_M$.
- $\llbracket c \rrbracket \leftarrow \mathsf{MUL}(\llbracket a \rrbracket, \llbracket b \rrbracket)$: MUL takes shares $\llbracket a \rrbracket$ and $\llbracket b \rrbracket$ of $a \in \mathbb{Z}_M$ and $b \in \mathbb{Z}_M$, respectively, as inputs, then outputs a share $\llbracket c \rrbracket$ of $a \times b = c \in \mathbb{Z}_M$.
- $\llbracket c \rrbracket \leftarrow \mathsf{MULConst}(\llbracket a \rrbracket, b)$: $\mathsf{MULConst}$ takes share $\llbracket a \rrbracket$ of $a \in \mathbb{Z}_M$ and $b \in \mathbb{Z}_M$ as inputs, then outputs a share $\llbracket c \rrbracket$ of $a \times b = c \in \mathbb{Z}_M$.

- $[\![c]\!] \leftarrow \mathsf{CMP}([\![a]\!], [\![b]\!])$: CMP takes shares $[\![a]\!]$ and $[\![b]\!]$ of $a \in \mathbb{Z}_M$ and $b \in \mathbb{Z}_M$, respectively, as inputs, then outputs a share $[\![1]\!]$ if $a < b$ over the integers, $[\![0]\!]$ otherwise.
- $[\![c]\!] \leftarrow \mathsf{Half}([\![a]\!])$: Half takes a share $[\![a]\!]$ of $a \in \mathbb{Z}_M$ as input, then outputs a share $[\![1]\!]$ if $a \leq \lfloor M/2 \rfloor$ over the integers, $[\![0]\!]$ otherwise.

In our implementation, we utilize the ABY library [17], which is based on a two-party setting (See Section VI-A for details) and supports secure computation over the ring of integers modulo $M = 2^m$ ($m = 8, 16, 32$, or 64). Here, Half can be instantiated by the use of CMP although it is not originally included in the ABY library.

C. SECURITY AND NETWORK SETTINGS

In this paper, we focus on the semi-honest adversary. More precisely, we consider the adversary who follows protocols but curiously learn client's or trainer's data. As mentioned above, in our proposed protocol, there are three parties: the client, the service provider, and the trainer, and note that there are n servers in the cloud hosted by the service provider.

The trainer locally trains with plaintexts, i.e., non-encrypted training, and constructs a model of a BNN. Then the trainer computes shares of the model with respect to an underlying t-out-of-n secure computation scheme, and then uploads the resulting shares to the servers. Namely, the adversary cannot learn the model as long as the adversary corrupts less than t servers.

The client computes shares of its query of the prediction on trainer's model with respect to the underlying t-out-of-n secure computation scheme, and then sends the resulting shares to the cloud. More than $t - 1$ servers jointly compute a protocol of the prediction with input the shares of model and the shares of query, and then output its result. Namely, the adversary cannot learn client's query as long as the adversary corrupts less than t servers.

However, similar to previous proposals [4], [9], we do not aim to hide the size of client's query, the network architecture of trainer's model, and which secure computation protocols are used. The authors of MiniONN [4] suggested that such information can be protected by adding dummy layers, which can also be integrated with our proposed protocol.

Finally, we assume the use of secure channel, which can be instantiated by the transport layer security (TLS) [33]. This setting is the same as that in other literature [4], [9].

III. OUR MAIN IDEA

1) TECHNICAL PROBLEM

This work aims to create a system that achieves both the performance and the security of a trained model by using BNNs. The values in the operations of BNNs are binarized into $+1$ or -1 and may seem to be compatible with the algebraic structures of secure computation. However, the processes of the original batch normalization [18] that improve the performance of neural networks are linear operations in real numbers, making them somewhat incompatible with secure computation in integers.

2) TRANSFORMATION INTO INTEGERS

To solve the compatibility problem, we transform the parameters of batch normalization into linear operations in integers by multiplying the parameters by a constant and rounding them down to the nearest integers. Heuristically, we can deduce that such transformation has small influence on the accuracy because errors can be "reset" by using non-linear processing in an activation function after the batch normalization. In particular, the possibility that the truncation of digits changes the sign of the output of batch normalization (i.e., from positive to negative and vice versa) and influence the activation function is negligible. The output of the batch normalization in the output layer is identical to that of BNNs, and the maximized value in these output vectors can be finally obtained as a prediction result. The possibility that the index of a maximized value is changed is negligible, and thus the truncation of digits does not affect the prediction result. In actual applications, the sizes of the parameters can be chosen such that the decline in the accuracy in a trained BNN model is minimal.

The method described above solves the incompatibility problem between the algebraic structures of the operations of BNNs and secure computation. Moreover, this method achieves a higher accuracy than the bit-shift method in the original BNNs [15] because the standard batch normalization can clip distribution with a higher accuracy. Finally, we can construct MOBIUS by combining an efficient and scalable secure computation based on secret sharing and the improved BNNs.

IV. BINARIZED NEURAL NETWORKS COMPATIBLE WITH SECURE COMPUTATION

In this section, we describe our *improved BNNs* that will be suitable for applying secure computation protocols, and hence will be used in MOBIUS. The intuition on our improved BNNs follows from Section I-C. Below we first explain our approach on batch normalization with integers (rounding parameters), and then describe our improved BNNs. We finally describe an instantiation on an architecture for MNIST, which is a large database of handwritten digits, as a concrete example of the proposed improved BNNs.

A. NOTATIONS

For convenience, we summarize parameters used in subsequent sections as follows.

Notation	Description
M	the modulus
$\mathbb{Z}_M$	the ring of integers modulo M
$\boldsymbol{W}$	the weight matrix used in full connection
$\gamma, \mu, \sigma, \epsilon, \beta$	the vanilla batch normalization parameters
d	the size of hidden layer
q	the scale parameter of MOBIUS
s, t, s', t'	batch normalization parameters of MOBIUS

Algorithm 1 BinaryFullConnection

```
Input: $\boldsymbol{a} \in \mathbb{Z}^{d_{in} \times 1}$ : input vector
    $\boldsymbol{W} \in \{-1, 1\}^{d_{out} \times d_{in}}$ : weight matrix
Output: $\boldsymbol{c} \in \mathbb{Z}^{d_{out} \times 1}$
Procedure:
1: $\boldsymbol{c} \leftarrow \boldsymbol{Wa}$
```

B. BATCH NORMALIZATION WITH INTEGERS

In this section, we explain our approach to perform batch normalization with integers. We refer the 'vanilla'' (or ordinary) batch normalization to [18] and briefly recapitulate it here. The ordinary batch normalization of [18] defines $\gamma_{(i)}, \beta_{(i)}, \mu_{(i)}, \sigma_{(i)}$, and ϵ as learned parameters. All parameters are real numbers while ϵ is a small positive value.

In the i-th layer, with an integer input $x_{(i)}$, it performs the following computation and outputs

$$\hat{x}_{(i)} = \gamma_{(i)} \frac{x_{(i)} - \mu_{(i)}}{\sqrt{\sigma_{(i)}^2 + \epsilon}} + \beta_{(i)}. \tag{1}$$

By replacing coefficients, Equation (1) can be transformed into $\hat{x}_{(i)} = s_{(i)} x_{(i)} + t_{(i)}$ where

$$s_{(i)} = \frac{\gamma_{(i)}}{\sqrt{\sigma_{(i)}^2 + \epsilon}}, \quad t_{(i)} = \beta_{(i)} - \frac{\gamma_{(i)}\mu_{(i)}}{\sqrt{\sigma_{(i)}^2 + \epsilon}}. \tag{2}$$

By substituting $s'_{(i)}, t'_{(i)}$ for integers $s_{(i)}, t_{(i)}$ using an appropriate integer q, which we call a scale parameter, we obtain an alternative integer $\hat{x}'_{(i)}$ for $\hat{x}_{(i)}$ as follows:

$$\hat{x}'_{(i)} = s'_{(i)} x_{(i)} + t'_{(i)} \quad \left(s'_{(i)} = \lfloor q s_{(i)} \rfloor, t'_{(i)} = \lfloor q t_{(i)} \rfloor\right) \tag{3}$$

Although the value of q can be determined layerwise or even nodewise, the same q is used in every node for simplicity in this paper. Intuitively, as the value q increases, the deterioration of BNN prediction accuracy decreases. However, the bit length of modulus M increases as q increases, consequently increasing memory requirements and calculation costs. Therefore, q should be as small as possible to maintain high prediction accuracy.

C. IMPROVED BINARIZED NEURAL NETWORKS

In this section, we describe the binary full connection, batch normalization, and activation algorithms used in the proposed BNNs. The binary full connection algorithm is shown in Algorithm 1. This algorithm takes an integer vector $\boldsymbol{a}$ and a learned weight matrix $\boldsymbol{W}$ as inputs, then outputs the result of matrix multiplication $\boldsymbol{Wa}$.

The batch normalization algorithm is shown in Algorithm 2. This algorithm takes an integer vector $\boldsymbol{c}$, which is usually an output of binary full connection, and batch normalization parameters s', t' as inputs, then outputs the result of batch normalization. Batch normalization parameters s', t' are obtained as described in Section IV-B.

The activation algorithm is shown in Algorithm 3. This algorithm takes an integer vector $\boldsymbol{b}$ as input, then outputs a

Algorithm 2 BatchNormalization

```
Input: $\boldsymbol{c} \in \mathbb{Z}^{d}$ : input vector
    $\boldsymbol{s'}, \boldsymbol{t'} \in \mathbb{Z}^{d}$: batch normalization parameters
Output: $\boldsymbol{b} \in \mathbb{Z}^{d}$
Procedure:
1: for $i = 1$ to $d$, $b_{(i)} \leftarrow s'_{(i)} * c_{(i)} + t'_{(i)}$
```

Algorithm 3 Activation

```
Input: $\boldsymbol{b} \in \mathbb{Z}^{d}$: input vector
Output: $\boldsymbol{a} \in \{-1, 1\}^{d}$
Procedure:
1: for $i = 1$ to $d$, $a_{(i)} \leftarrow \begin{cases} -1 & (b_{(i)} < 0) \\ 1 & (b_{(i)} \geq 0) \end{cases}$
```

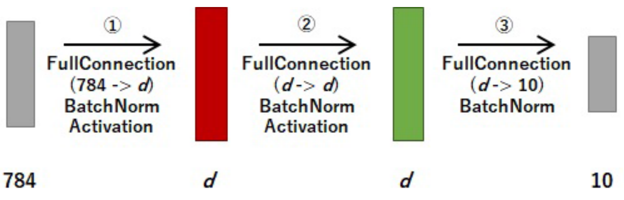


FIGURE 2. Architecture of the BNNs for MNIST dataset.

binary vector that represents the signs of each element of the input vector $\boldsymbol{b}$.

D. BINARIZED NEURAL NETWORKS FOR MNIST DATASET

In Section IV-C, we described the algorithms used in the proposed BNNs. To use BNNs for learning or predicting data, we need to instantiate a concrete architecture and determine the entire procedure. We instantiate an architecture for MNIST dataset image classification (See Section VI-B for details on the MNIST dataset). Consider a typical architecture with an input layer of size 784, two hidden layers of size d, and an output layer of size 10, as shown in Figure 2. In the hidden layers, the full connection, batch normalization, and activation algorithms are executed in order. In the output layer, only the full connection and batch normalization algorithms are executed. In this architecture, even though the maximum value index of the output vector is the result of the prediction, we omit this process because the proposed method is designed to return output vectors as the result of secure computation.

The weight matrices $\boldsymbol{W_{(1)}}, \boldsymbol{W_{(2)}}$, and $\boldsymbol{W_{(3)}}$ used in algorithm 4 are learned parameters, and the batch normalization parameters $\boldsymbol{s'_{(j)}}, \boldsymbol{t'_{(j)}}$ can be calculated as described in Section IV-B. In the case of $d = 128, 1000$ (d is the size of hidden layers), we confirm experimentally that the deterioration of prediction accuracy towards test data is negligible when a scale parameter $q = 10,000$. Therefore, we use $q = 10,000$ in all experiments in this work.

Algorithm 4 Binarized Neural Network for MNIST

```
Input: $input \in \mathbb{Z}^{784\times 1}$ : input vector
    $W_{(1)} \in \{-1,1\}^{d\times 784}, s'_{(1)} \in \mathbb{Z}^{d\times 1}, t'_{(1)} \in \mathbb{Z}^{d\times 1}$,
    $W_{(2)} \in \{-1,1\}^{d\times d}, s'_{(2)} \in \mathbb{Z}^{d\times 1}, t'_{(2)} \in \mathbb{Z}^{d\times 1}$,
    $W_{(3)} \in \{-1,1\}^{10\times d}, s'_{(3)} \in \mathbb{Z}^{10\times 1}, t'_{(3)} \in \mathbb{Z}^{10\times 1}$
Output: $output \in \mathbb{Z}^{10\times 1}$ : output vector
Procedure:
1: $c_{(1)} \leftarrow \mathsf{FullConnection}\left(input, W_{(1)}\right)$
2: $b_{(1)} \leftarrow \mathsf{BatchNormalization}\left(c_{(1)}, s'_{(1)}, t'_{(1)}\right)$
3: $a_{(1)} \leftarrow \mathsf{Activation}\left(b_{(1)}\right)$
4: $c_{(2)} \leftarrow \mathsf{FullConnection}\left(a_{(1)}, W_{(2)}\right)$
5: $b_{(2)} \leftarrow \mathsf{BatchNormalization}\left(c_{(2)}, s'_{(2)}, t'_{(2)}\right)$
6: $a_{(2)} \leftarrow \mathsf{Activation}\left(b_{(2)}\right)$
7: $c_{(3)} \leftarrow \mathsf{FullConnection}\left(a_{(2)}, W_{(3)}\right)$
8: $output \leftarrow \mathsf{BatchNormalization}\left(c_{(3)}, s'_{(3)}, t'_{(3)}\right)$
```

V. MOBIUS DESIGN

In this section, we describe the design of MOBIUS. We first describe share generation of a trained model in the pre-processing phase, and then show its main algorithms.

MOBIUS is composed of protocols we call *secure full connection*, *secure batch normalization*, and *secure activation*. The main sequences of these protocols are almost the same as those described in the previous section, but we utilize secure computation in the internal processes.

A. SECRET SHARING A MODEL

We first construct shares of parameters, which are learned in plaintexts, except for that of batch normalization by utilizing secret sharing described in Section II-B. In this construction, let M be a modulus of the secret sharing. Moreover, for any a, $[\![a]\!]$ is a secret share if $a > 0$ and $[\![M \; \mathcal{C} \; a]\!]$ is a secret share if $a < 0$. Hereinafter, we denote $0 \leq a \leq \lfloor \frac{M}{2} \rfloor$ as a non-negative integer and $\lfloor \frac{M}{2} \rfloor < a < M$ as a negative integer.

Learned weight matrices $W_{(i)}$ $(i = 1, \cdots ; L-1)$ are shared using secret sharing and are stored in each server in a distributed manner. Parameters of the batch normalization are computed using the computation in Section IV-B, and its resulting parameters $s_{(i)}$, $t_{(i)}$ $(i = 1, \cdots, L-1)$ are stored in each sever as shares by utilizing the secret sharing. Finally, the size information $(L, n_{(0)}, \cdots, n_{(L)})$ of the shares themselves are not shared, i.e., they are stored as plaintexts.

B. MODEL-OBLIVIOUS PREDICTION

The construction of a prediction protocol for MNIST in MOBIUS is shown in Algorithm 5. The secure full connection, secure batch normalization, and secure activation are denoted by SecureFC, SecureBN, and SecureAct, respectively. Moreover, for any matrix X, $X_{(i,j)}$ indicates an element of the i-th row and j-th column and $X_{(i)}$ indicates an element of the i-th column.

Algorithm 5 SecureBinaryNN for MNIST

```
Input: $[\![input]\!] \in \mathbb{Z}_M^{784}$: Shares of Input Vectors
Output: $[\![output]\!] \in \mathbb{Z}_M^{10}$: Prediction Results
Procedure:
1: $[\![c_{(1)}]\!] \leftarrow \mathsf{SecureFC}\left([\![input]\!], [\![W_{(1)}]\!]\right)$
2: $[\![b_{(1)}]\!] \leftarrow \mathsf{SecureBN}\left([\![c_{(1)}]\!], [\![s_{(1)}]\!], [\![t_{(1)}]\!]\right)$
3: $[\![a_{(1)}]\!] \leftarrow \mathsf{SecureAct}\left([\![b_{(1)}]\!]\right)$
4: $[\![c_{(2)}]\!] \leftarrow \mathsf{SecureFC}\left([\![a_{(1)}]\!], [\![W_{(2)}]\!]\right)$
5: $[\![b_{(2)}]\!] \leftarrow \mathsf{SecureBN}\left([\![c_{(2)}]\!], [\![s_{(2)}]\!], [\![t_{(2)}]\!]\right)$
6: $[\![a_{(2)}]\!] \leftarrow \mathsf{SecureAct}\left([\![b_{(2)}]\!]\right)$
7: $[\![c_{(3)}]\!] \leftarrow \mathsf{SecureFC}\left([\![a_{(2)}]\!], [\![W_{(3)}]\!]\right)$
8: $[\![output]\!] \leftarrow \mathsf{SecureBN}\left([\![c_{(3)}]\!], [\![s_{(3)}]\!], [\![t_{(3)}]\!]\right)$
```

Algorithm 6 SecureFullConnection

```
Input: $[\![input]\!] \in \mathbb{Z}_M^{d_{in}}$: Shares of Input Vectors
    $[\![W]\!] \in \mathbb{Z}_M^{d_{out}\times d_{in}}$: Shares of Weight Matrices
Output: $[\![output]\!] \in \mathbb{Z}_M^{d_{out}}$
Procedure:
1: for $i = 0$ to $d_{out}$ do
2:     for $j = 0$ to $d_{in}$ do
3:         $[\![X_{(i)}]\!] \leftarrow \mathsf{MUL}([\![W_{(i,j)}]\!], [\![input_{(j)}]\!])$
4:         $[\![output_{(i)}]\!] \leftarrow \mathsf{ADD}([\![output_{(i)}]\!], [\![X_{(i)}]\!])$
5:     end for
6: end for
```

The secure full connection protocol is described in Algorithm 6. The matrix multiplication between shares is computed similarly as in Algorithm 1.

The secure batch normalization protocol is described in Algorithm 7. Although the original batch normalization [18] requires computations of root or division, the secure batch normalization protocol can be performed with only addition and multiplication by performing the computation in Equation (2) in advance.

The secure activation protocol is described in Algorithm 8. This algorithm outputs $+1$ if the input is greater than or equal to zero or -1 otherwise. As described above, a non-negative integer is represented by $\{0, \ldots, \lfloor \frac{M}{2} \rfloor\}$ and a negative integer is represented by $\{\lfloor \frac{M}{2} \rfloor + 1, \ldots, M-1\}$. Therefore, the algorithm is performed by a comparison operation with $\lfloor \frac{M}{2} \rfloor + 1$ in secure computation.

C. DETERMINATION OF MODULUS SIZE

To contain non-negative integers within $\left\{0, \ldots, \lfloor \frac{M}{2} \rfloor\right\}$ and negative integers within $\left\{\lfloor \frac{M}{2} \rfloor + 1, \ldots, M-1\right\}$, we need to

Algorithm 7 SecureBatchNormalization

```
Input: $[\![c]\!] \in \mathbb{Z}_M^d$: Shares of Input Vectors
       $[\![s]\!]$, $[\![t]\!] \in \mathbb{Z}_M^d$: Batch Normalization Parameters
Output: $[\![output]\!] \in \mathbb{Z}_M^d$
Procedure:
1: for $j = 0$ to $d$ do
2:     $[\![X_{(j)}]\!] \leftarrow \mathsf{MUL}([\![c_{(j)}]\!], [\![s_{(j)}]\!])$
3:     $[\![output_{(i)}]\!] \leftarrow \mathsf{ADD}([\![X_{(j)}]\!], [\![t_{(j)}]\!])$
4: end for
```

Algorithm 8 SecureActivation

```
Input: $[\![b]\!] \in \mathbb{Z}_M^d$: Shares of Input Vectors
Output: $[\![a]\!] \in \mathbb{Z}_M^d$
1: for $j = 0$ to $d$ do
2:     $[\![X_{(j)}]\!] \leftarrow \mathsf{Half}([\![b_{(j)}]\!])$
3:     $[\![Y_{(j)}]\!] \leftarrow \mathsf{MULConst}([\![X_{(j)}]\!], 2)$
4:     $[\![a_{(j)}]\!] \leftarrow \mathsf{ADDConst}([\![Y_{(j)}]\!], -1)$
5: end for
```

determine the size of modulus M adequately. Such an adequate modulus M can be computed through a maximized (or minimized) value during execution of BNNs without secure computation.

For convenience, we describe a computation with a network model described in Fig. 2. Let s and t be parameters for batch normalization. The maximized value for MOBIUS is identical to that of b in the second line of Algorithm 5 and becomes $(784 \times x \times s_{max} + t_{max}) * 2 + 1$, where x is the size of input data, i.e., 8 bits, and s_{max} and t_{max} are the maximized values for s and t, respectively. When we conducted an experiment with a scale parameter $q = 10000$ and 128 neurons, the values for s_{max} and t_{max} are 6 and 20233, respectively. Since the input data is a gray-scale image with 0–255, we can set a modulus as $M = 2^{22}$, which is greater than $(784 \times 255 \times 6 + 20233) * 2 + 1 \simeq 2^{21.2}$.

D. SECURITY ANALYSIS

To show the security of our proposed protocol against semi-honest adversary, we follow the same security discussion described in SecureML [9]. Their security achievements are different from our protocol, but the underlying adversarial model and theoretical security notions are identical to ours. Following the composition theorems [34], if building blocks are secure, then there is a secure composition of the protocols. We can then construct a secure construction of MOBIUS against a semi-honest adversary by utilizing secure computation, whose security can be proven against the semi-honest adversary, in the operations described in Section II-B. A construction against a malicious adversary can also be created by utilizing a secure computation protocol against a malicious adversary, such as ABY3 [21].

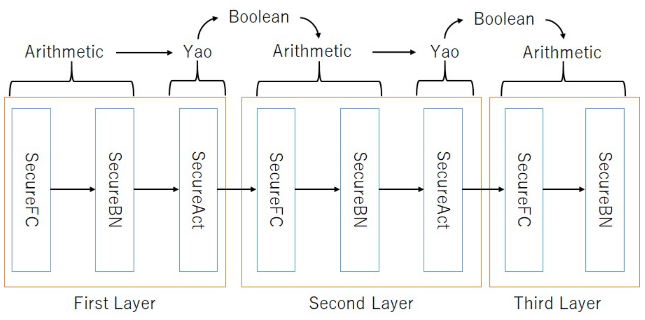


FIGURE 3. Flow of implementation of MOBIUS with arithmetic and Yao shares.

VI. EXPERIMENT

In this section, we describe the implementation of MOBIUS and the results of experiments using the MNIST dataset.

A. IMPLEMENTATION

1) LANGUAGE AND LIBRARY

MOBIUS is implemented in C++ with the ABY library [17] for secure computation. The ABY library is a secure computation framework with two-party setting and contains three types of shares, namely, *arithmetic*, *boolean*, and *Yao*. These shares have different operations, and the ABY library provides efficient conversions between them. We refer the readers to the paper [17] for details on the ABY framework.

We briefly describe several parts related to our implementation below. The arithmetic shares can be used in arithmetic operations, such as addition and multiplication. Therefore, the secure full connection and secure batch normalization are implemented with arithmetic shares. In terms of share size, the ABY library includes four parameters as a modulus M, i.e., 8, 16, 32, and 64 bits. Although we omit the details due to space limitation, the MNIST dataset is available with 32-bit parameter as described in the previous section. The secure activation requires a comparison operation of secure computation, which can be computed with boolean or Yao shares. In the ABY library, arithmetic shares cannot be directly converted into boolean shares, i.e., the arithmetic shares are first converted to Yao shares and then from Yao shares to boolean shares. Therefore, since the conversion of arithmetic shares to boolean shares requires two conversions, we used Yao shares in the secure activation. Besides, according to the benchmark of the ABY library [17], a comparison operation using Yao shares can be computed faster than using boolean shares. We therefore implemented the secure activation with Yao shares.

2) OVERVIEW OF IMPLEMENTATION

We show the implementation flows of shares in Figure 3 and Figure 4. As described in the previous section, we utilized Yao shares in the secure activation function even though boolean shares can also be used. In the algorithms of the original BNNs [15], secure activation should be performed with boolean shares because of the bit-shift operations.

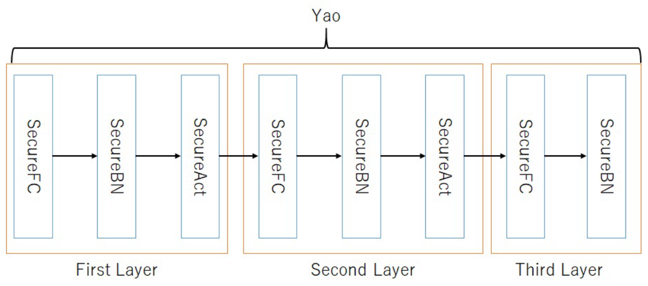


FIGURE 4. Flow of implementation of MOBIUS with only Yao shares.

The extra conversion to boolean shares and the overhead operations of boolean shares may downgrade the computational performance. Therefore, we considered two cases, namely, implementation with both arithmetic shares and Yao shares and that with only Yao shares.

The implementation of MOBIUS was created by simply using the available ABY library and is therefore not optimized unlike SecureML [9]. Therefore, the performance of the following experiments can be improved by optimizing our implementation. We plan to publish our source codes for subsequent works. The training phase is out of the scope of this work, and therefore a model is trained in advance. Shares of the model and input from a client are generated by the PutSIMDINGate function of the ABY library. Since the full connection protocol requires matrix multiplication between shares, it is implemented by PutMULGate for multiplication between shares and PutADDGate for addition between shares. Similarly, the batch normalization protocol is based on addition and multiplication between shares and hence is implemented with PutMULGate and PutADDGate. Finally, the activation protocol requires a sign function, and hence is implemented with PutGTGate for comparing shares. To compute the activation protocol, a comparison operation is necessary and Yao shares are utilized. Therefore, the conversion of arithmetic shares to Yao shares is unavoidable. The comparison operation is faster when Yao shares, instead of boolean shares, are used. Moreover, the conversion of arithmetic shares to Yao shares is faster than that to boolean shares.

B. EXPERIMENTAL SETTING

1) MACHINE ENVIRONMENTS

We conducted experiments with the MNIST dataset using the algorithms described in Section IV-D on two AmazonEC2 c4.8xlarge machines, both of which are running Linux and have 60 GB of RAM. The two machines are hosted in the same region as a LAN setting. The bandwidth is 1 GB/s, and the neural network has two hidden layers with 128 neurons in each layer. This setting is identical to that of SecureML [9]. We also utilize the sign function as the activation function. The neural network is fully connected. We them compare the performance of our protocol with SecureML and other state-of-the-art protocols with cryptography [4], [12], [14].

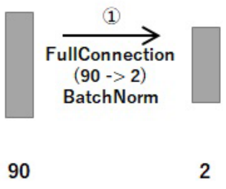


FIGURE 5. Architecture of the BNNs for Cancer dataset.

2) DATASETS

We evaluated three datasets, namely, Cancer, Diabetes, and MNIST datasets, which are the same datasets utilized in TAPAS [14]. The Cancer and Diabetes datasets contain health care information and therefore have privacy implications. The MNIST dataset is the standard benchmark dataset. We shifted the number of neurons for experiments with the MNIST dataset, but not for experiments with the Cancer and Diabetes datasets. These settings are identical to those of TAPAS. Details of each dataset are described below.

a: CANCER

The Cancer dataset[3] contains 569 data points with each point containing 30 real-valued features. The prediction task is to predict whether a tumor is cancerous or benign. Similar to TAPAS [14], we divided the dataset into a training set and a test in a 70:30 ratio. For all real-valued features, we divided the range of each feature into three equal-spaced binaries and one-hot encoded each feature by its membership. We then created a 90-dimensional binary vector for each sample. We utilized a single fully connected layer from 90 to two neurons followed by a batch normalization layer. An architecture of networks with the Cancer dataset is shown in Figure 5.

b: DIABETES

The Diabetes dataset[4] contains data of 100,000 patients with diabetes. The prediction task is to predict one of three possible labels regarding hospital readmission after release. Similar to TAPAS [14], we divided patients into an 80/20 train/test split. We bin categorical features similarly to the Cancer dataset, and then obtained a 1704-dimensional binary data point for each sample. We utilized a network consisting of a fully connected layer from 1704 to ten neurons, a batch normalization layer, an activation function, followed by another fully connected layer from ten to three neurons, and a batch normalization layer. An architecture of networks with the Diabetes dataset is shown in Figure 6.

c: MNIST

The MNIST dataset contains 70,000 images of handwritten digits from 0 to 9. In particular, the MNIST dataset has 60,000 training samples and 10,000 test samples, each with 784 features representing 28×28 pixels in the image.

[3]https://tinyurl.com/gl3yhzb
[4]https://tinyurl.com/m6upj7y

TABLE 1. Performance of MOBIUS with MNIST dataset in comparison with related work.

	Model Obliviousness	Secure Computation	Accuracy [%]	Neurons	On-line Time [sec]	Off-line Time [sec]	Total Time [sec]
SecureML [9]	✓	SS	93.1	128	0.18	4.7	4.88
MiniONN [4]		SS	97.6	128	0.14	0.9	1.04
GAZELLE [8]		HE, GC	-	128	0.03	0	0.03
TAPAS [14]		FHE	97.3	-	-	-	147
FHE-DiNN [12]		FHE	96.3	100	-	-	1.64
MOBIUS (only Yao)	✓	SS	95.9	128	21.1	39.0	60.1
			96.7	200	35	66	101
			97.7	1000	NA	NA	NA
MOBIUS (arithmetic + Yao)	✓	SS	95.9	128	0.06	0.71	0.77
			96.7	200	0.11	1.29	1.4
			97.7	1000	0.85	10.04	10.89

The second column refers to the capability to encrypt a model. The third column refers to types of secure computation, where SS, FHE, and GC indicates secret sharing, fully homomorphic encryption, and garbled circuit, respectively. The fourth column refers to the evaluation of the model as output at the prediction phase. The fifth column refers to the number of neurons that are used in each experiment. The sixth and seventh columns refer to on-line time and off-line time for secure computation. The final column refers to elapsed time of execution in the real time. Although several source codes of the other systems were obtained, we could not execute them and the values in the Table were obtained from their papers. Hence, we simply refer to the values described in each work except for MOBIUS. Here, the symbol "-" means that the value has not been described in the corresponding work and "NA" in the line of MOBIUS (only Yao) means that the results could not be obtained.

TABLE 2. Performance of MOBIUS with Cancer dataset in comparison with related work.

	Model Obliviousness	Secure Computation	Accuracy [%]	On-line Time [sec]	Off-line Time [sec]	Total Time [sec]
TAPAS [14]		FHE	97.1	-	-	3.5
MOBIUS (only Yao)	✓	SS	97.7	0.18	0.07	0.25
MOBIUS (arithmetic + Yao)	✓	SS	97.7	0.001	0.004	0.005

The setting is almost the same as that in Table 1. However, the Cancer dataset was used only in TAPAS [14] and the number of neuron is fixed at only two neurons in the output layer. We also note that we could not execute TAPAS even though we obtained the source codes.

TABLE 3. Performance of MOBIUS with Diabetes dataset in comparison with related work.

	Model Obliviousness	Secure Computation	Accuracy [%]	On-line Time [sec]	Off-line Time [sec]	Total Time [sec]
TAPAS [14]		FHE	54.9	-	-	283
MOBIUS (only Yao)	✓	SS	54.9	11	4.4	15.4
MOBIUS (arithmetic + Yao)	✓	SS	54.9	0.13	0.01	0.14

The setting is almost the same as that in Table 1. Similar to Table 2, the Diabetes dataset was used only in TAPAS [14] and the number of neuron is fixed at ten neurons in the middle layers and three neurons in the output layer. We also note that we could not execute TAPAS even though we obtained the source codes.

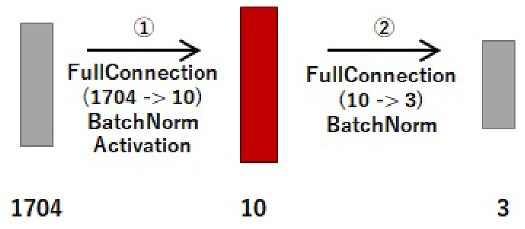


FIGURE 6. Architecture of the BNNs for Diabetes dataset.

Each feature is a grayscale between 0–255. An architecture of networks with the MNIST dataset is shown in Figure 2.

C. RESULTS

The experimental results are shown in Table 1, Table 2, Table 3, Figure 7, and Figure 8. Table 1 shows a comparison of different protocols based on capability to encrypt a model, accuracy, and computational time. We also note that we were not able to execute their prototypes even though several source codes were obtained, and therefore we referred to the values described in their papers. In Table 2 and Table 3, we compared the performance of our system with only TAPAS, which, to the best of our knowledge, is the only other related work that used the Cancer and Diabetes datasets. Figure 7 shows a comparison of MOBIUS and the original BNNs based on prediction accuracy with respect to the number of neurons with the MNIST dataset. Since the accuracy of MOBIUS with arithmetic shares and Yao shares is identical to that of MOBIUS with only Yao shares as shown in Table 1, Table 2 and Table 3, we contain only a single and common curve for MOBIUS in Figure 7. Likewise, Figure 8 shows a comparison of MOBIUS and the original BNNs based on computational time for prediction with respect to the number of neurons with the MNIST dataset. Since the computational time of MOBIUS with arithmetic shares and

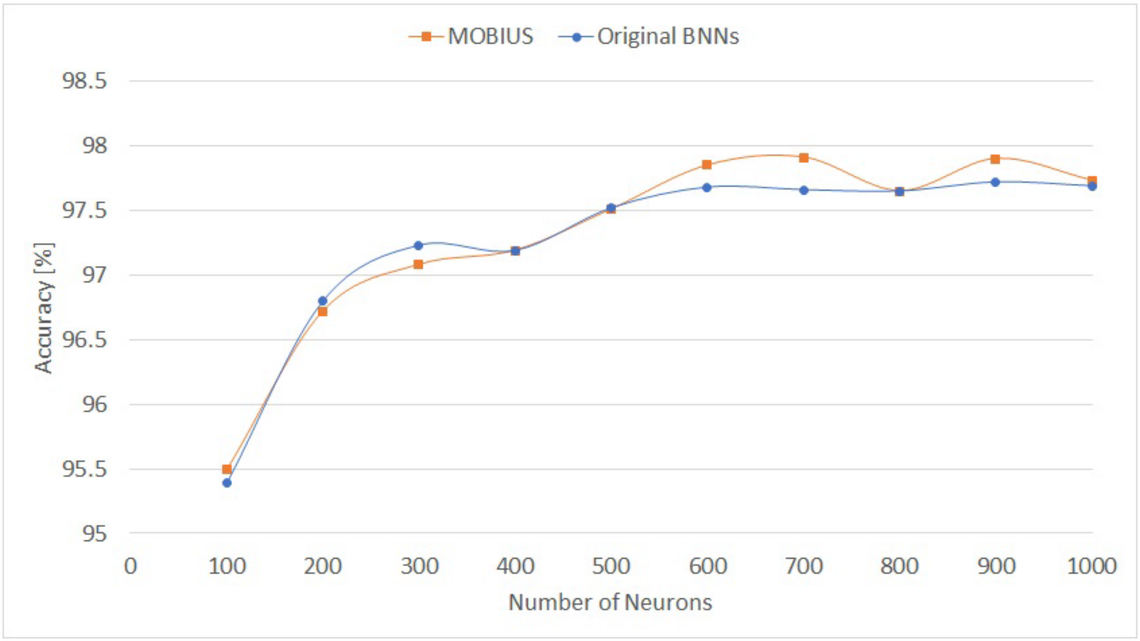


FIGURE 7. Prediction accuracy of MOBIUS with MNIST.

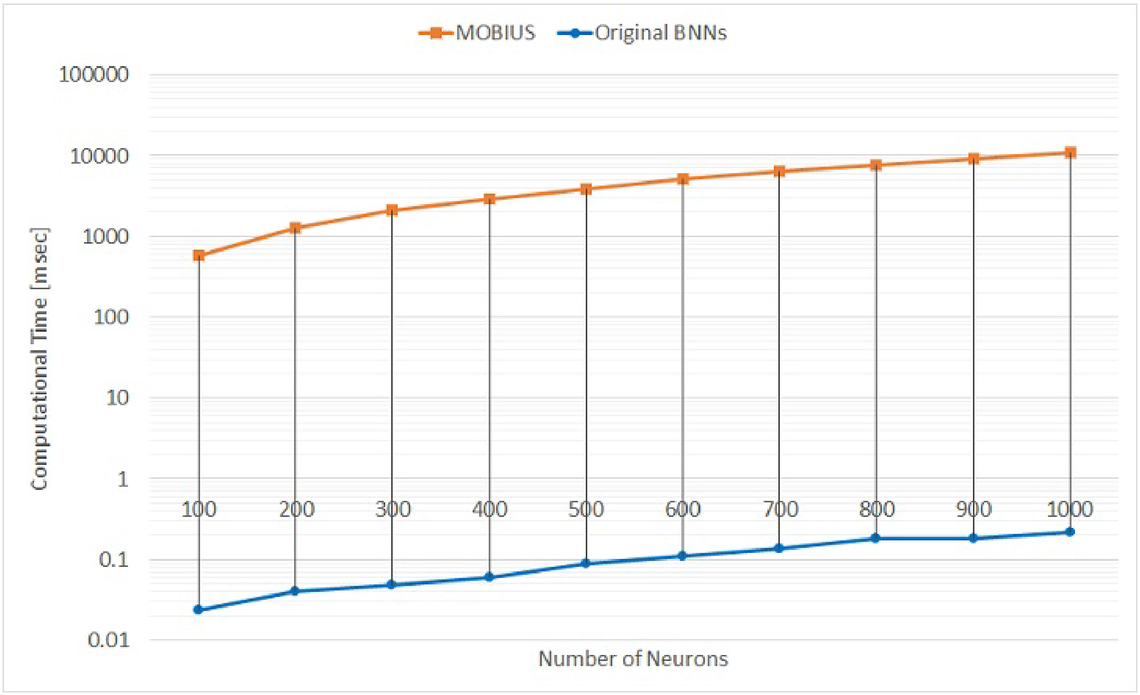


FIGURE 8. Computational time of MOBIUS with MNIST.

Yao shares is faster than that of MOBIUS with only Yao shares, we contain a single curve for MOBIUS in Figure 8.

As shown in Table 1, MOBIUS is the fastest system that combines BNNs and secure computation based on secret sharing despite having the capability to encrypt a model. In comparison with other works based on BNNs, i.e., TAPAS [14] and FHE-DiNN [12], MOBIUS with arithmetic shares and Yao shares obtained better accuracy and faster prediction by adjusting the number of neurons. For example, compared with FHE-DiNN, MOBIUS is at least twice as fast for 100 neurons and rigorously performs better in terms of accuracy and computational time for 200 neurons. By increasing neurons to 1000, MOBIUS also outperforms TAPAS in terms of accuracy and computational time.

In comparison with GAZELLE [8], we can estimate a cost to introduce model obviousness. In particular, according to the authors of GAZELLE, GAZELLE utilizes the same environment as that of SecureML, and hence MOBIUS costs about twice as much in terms of computational time to introduce model obviousness (although GAZELLE did not clearly show its accuracy in their paper). Finally, we note that the MOBIUS with only Yao shares could not return results because of an error from the ENCRYPTO_utils library,[5] which is a submodule of the ABY library, when the number of neuron is greater than 600. We consider that the accuracy of MOBIUS may be improved by optimizing our implementation.

In the comparison of MOBIUS and TAPAS for the Cancer dataset and the Diabetes dataset shown in Table 2 and Table 3, MOBIUS based on arithmetic shares and Yao shares showed better accuracy, and its computational throughput was approximately 700 times faster for the Cancer dataset and 2000 times faster for the Diabetes dataset. We note that precise prediction for the Diabetes dataset is difficult and even the accuracy of the original prediction, i.e., without secure computation, is only 55.6% according to the authors of TAPAS [14].

As shown in Figure 7, MOBIUS has better prediction accuracy than the original BNNs. Finally, as shown in Figure 8, the computational time of MOBIUS seems to be linear with respect to the number of neurons, although the computational time becomes 100 times longer than the original BNNs. We can thus approximately measure performance for any number of neurons.

VII. CONCLUSION

In this work, we presented MOBIUS (Model-Oblivious BInarized neUral networkS), a system that enables scalable encrypted prediction and encryption of a trained model. As our main technical contribution, we presented new algorithms of BNNs that are compatible with secure computation by representing all parameters in integers and removing the bit-shift method used in the original BNNs [15]. We then designed the main construction of MOBIUS with secure computation based on arithmetic shares and Yao shares. We also conducted experiments using several datasets, including the MNIST, Cancer, and Diabetes datasets, and the results show that MOBIUS achieves higher computational performance and higher accuracy than SecureML, [9] which is the only other system that considers the model-oblivious problem. Even in comparison with TAPAS [14] and FHE-DiNN [12], which are state-of-the-art protocols based on BNNs, MOBIUS can achieve higher computational performance and higher accuracy by adjusting the number of neurons. As future work, we plan to conduct experiments on more complicated datasets, such as CIFAR10. We also plan to implement with ABY3 [21] as a state-of-the-art library for secure computation to achieve better computational performance as well as security against a malicious adversary.

[5] https://github.com/encryptogroup/ENCRYPTO_utils/blob/9bb8af36cde8c23465c385145561bcba5adf217d/src/ENCRYPTO_utils/cbitvector.h

REFERENCES

[1] M. Barni, C. Orlandi, and A. Piva, "A privacy-preserving protocol for neural-network-based computation," in *Proc. MM&Sec*, Sep. 2006, pp. 146–151.

[2] C. Orlandi, A. Piva, and M. Barni, "Oblivious neural network computing via homomorphic encryption," *EURASIP J. Inf. Secur.*, vol. 1, Dec. 2007, Art. no. 037343.

[3] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser, "Machine learning classification over encrypted data," in *Proc. NDSS*, Feb. 2015, p. 4325.

[4] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via minionn transformations," in *Proc. CCS*, Oct. 2017, pp. 619–631.

[5] B. D. Rouhani, M. S. Riazi, and F. Koushanfar, "Deepsecure: Scalable provably-secure deep learning," May 2017, *arXiv:1705.08963*. [Online]. Available: https://arxiv.org/abs/1705.08963

[6] N. Chandran, D. Gupta, A. Rastogi, R. Sharma, and S. Tripathi, "EzPC: Programmable, efficient, and scalablesecure two-party computation for machine learning," iACR ePrint Arch., MSR India, Hyderabad, India, Tech. Rep. 2017/1109, 2017. [Online]. Available: https://eprint.iacr.org/2017/1109

[7] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar, "Chameleon: A hybrid secure computation framework for machine learning applications," in *Proc. ASIACCS*, Jun. 2018, pp. 707–721.

[8] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "Gazelle: A low latency framework for secure neural network inference," Jan. 2018, *arXiv:1801.05507*. [Online]. Available: https://arxiv.org/abs/1801.05507

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

[10] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *Proc. ICML*, Jun. 2016, pp. 201–210.

[11] M. Chase, R. Gilad-Bachrach, K. Laine, K. Lauter, and P. Rindal, "Private collaborative neural network learning," iACR ePrint Arch., Microsoft Res., Redmond, WA, USA, Tech. Rep. 2017/762, 2017. [Online]. Available: https://eprint.iacr.org/2017/762

[12] F. Bourse, M. Minelli, M. Minihold, and P. Paillier, "Fast homomorphic evaluation of deep discretized neural networks," iACR ePrint Arch., Orange Labs, Appl. Crypto Group, Cesson-Sévigné, France, Tech. Rep. 2017/1114, 2017. [Online]. Available: https://eprint.iacr.org/2017/1114

[13] S. Wagh, D. Gupta, and N. Chandran, "Securenn: Efficient and private neural network training," iACR ePrint Arch., Microsoft Res., Bengaluru, India, Tech. Rep. 2018/442, 2018. [Online]. Available: https://eprint.iacr.org/2018/442

[14] A. Sanyal, M. Kusner, A. Gascon, and V. Kanade, "Tapas: Tricks to accelerate (encrypted) prediction as a service," in *Proc. ICML 2018*, 2018. [Online]. Available: https://arxiv.org/abs/1806.03461

[15] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1," Feb. 2016, *arXiv:1602.02830*. [Online]. Available: https://arxiv.org/abs/1602.02830

[16] C. Gentry, "Fully homomorphic encryption using ideal lattice," in *Proc. STOC*, May 2009, pp. 169–178.

[17] D. Demmler, T. Schneider, and M. Zohner, "ABY-A framework for efficient mixed-protocol secure two-party computation," in *Proc. NDSS*, Feb. 2015.

[18] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. IML*, Jul. 2015, pp. 448–456.

[19] I. Damgård, M. Fitzi, E. Kiltz, J. B. Nielsen, and T. Toft, "Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation," in *Proc. Theory Cryptogr. Conf.*, 2006, pp. 285–304.

[20] S. Halevi and V. Shoup, "Bootstrapping for HElib," in *Advances in Cryptology* (Lecture Notes in Computer Science), vol. 9056, E. Oswald and M. Fischlin, Eds. Berlin, Germany: Springer-Verlag, 2015, pp. 641–670.

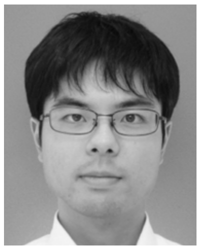
[21] P. Mohassel and P. Rindal, "Aby3: A mixed protocol framework for machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2018, pp. 35–52.

[22] O. Catrina and A. Saxena, "Secure computation with fixed-point numbers," in *Financial Cryptography and Data Security* (Lecture Notes in Computer Science), vol. 6052. Berlin, Germany: Springer, 2010, pp. 35–50.
[23] O. Catrina and S. de Hoogh, "Improved primitives for secure multiparty integer computation," in *Security and Cryptography for Networks* (Lecture Notes in Computer Science), vol. 6280. Berlin, Germany: Springer, 2010, pp. 182–199.
[24] C. Dwork, "Differential privacy," in *Proc. ICALP*, in Lecture Notes in Computer Science, vol. 4052. Springer, 2006, pp. 1–12.
[25] M. Abadi, A. Chu, I. Goodfellow, H. B. Mcmahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proc. CCS*, Oct. 2016, pp. 308–318.
[26] J. C. Duchi, M. I. Jordan, and M. J. Wainwright, "Local privacy and statistical minimax rates," in *Proc. FOCS*, Oct. 2013, pp. 429–438.
[27] B. Ding, H. Nori, P. Li, and J. Allen, "Comparing population means under local differential privacy: With significance and power," in *Proc. AAAI*, Apr. 2018, pp. 26–33.
[28] F. Aldà and B. I. Rubinstein, "The bernstein mechanism: Function release under differential privacy," in *Proc. AAAI*, Feb. 2017, pp. 1705–1711.
[29] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction apis," in *Proc. Usenix Secur.*, 2016, pp. 601–618.
[30] M. Kesarwani, B. Mukhoty, V. Arya, and S. Mehta, "Model extraction warning in mlaas paradigm," in *Proc. ACSAC*, Dec. 2018, pp. 371–380.
[31] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979.
[32] D. Bogdanov, S. Laur, and J. Willemson, "Sharemind: A framework for fast privacy-preserving computations," in *Proc. Computer Security—ESORICS* (Lecture Notes in Computer Science), vol. 5283. Berlin, Germany: Springer, 2008, pp. 192–206.
[33] E. Rescorla and T. Dierks, *The Transport Layer Security (TLS) Protocol Version 1.2 RFC*, document 5246, 2008. [Online]. Available: https://rfc-editor.org/rfc/rfc5246.txt
[34] E. Kushilevitz, Y. Lindell, and T. Rabin, "Information-theoretically secure protocols and security under composition," *SIAM J. Comput.*, vol. 39, no. 5, pp. 2090–2112, Sep. 2010.

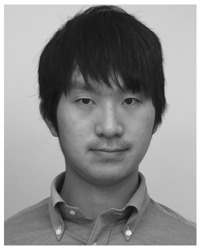
HIROMASA KITAI received the B.Eng. degree in engineering science from Osaka University, Japan, in 2018, where he is currently pursuing the master's degree with the Graduate School of Information Science and Technology. His research interests include machine learning and information security.

JASON PAUL CRUZ received the B.S. degree in electronics and communications engineering and the M.S. degree in electronics engineering from the Ateneo de Manila University, Quezon City, Philippines, in 2009 and 2011, respectively, and the Ph.D. degree in engineering from the Graduate School of Information Science, Nara Institute of Science and Technology, Nara, Japan, in 2017. He is currently a Specially Appointed Assistant Professor with Osaka University, Osaka, Japan. His current research interests include role-based access control, blockchain technology, hash functions and algorithms, and Android programming.

NAOTO YANAI received the B.Eng. degree from the National Institution for Academic Degrees and University Evaluation, Japan, in 2009, and the M.S.Eng. and Dr.E. degrees from the Graduate School of Systems and Information and Engineering, University of Tsukuba, Japan, in 2011 and 2014, respectively. He is currently an Assistant Professor with Osaka University, Osaka, Japan. His research interests include the areas of cryptography and information security.



NAOHISA NISHIDA received the M.E. degree from the University of Electro-Communications, Tokyo, Japan, in 2016. He then joined Panasonic Corporation, where he has been a Researcher, since 2016. His research interests include the applications of secure computation and blockchain.



TATSUMI OBA received the master's degree in engineering from the Tokyo Institute of Technology, Japan, in 2012. He has five years of experience in developing and applying machine learning algorithms. He has been a Security and Machine Learning Researcher with Panasonic Corporation, since 2016. His current research interests include anomaly detection, sequential data modeling, and privacy-preserving machine learning.

YUJI UNAGAMI received the B.E. and M.E. degrees in industrial and management systems from Waseda University, in 2004 and 2006, respectively. In 2006, he joined the Corporate Research and Development Division, Matsushita Electric Industrial (now Panasonic) Company Ltd., Osaka, Japan. His research interests include cryptography and information security.

TADANORI TERUYA received the M.E. degree and the Ph.D. degree in engineering from the University of Tsukuba, Japan, in 2009 and 2012, respectively. He was a Postdoctoral Researcher with the Faculty of Engineering, Information and Systems, University of Tsukuba, from 2012 to 2013, and the National Institute of Advanced Industrial Science and Technology (AIST), Japan, from 2013 to 2016. He has been a Researcher with AIST, since 2016. His research interests include cryptography and information security, especially practical aspects based on elliptic curves and lattices, and the applications of secure computation.

NUTTAPONG ATTRAPADUNG received the bachelor's degree (Hons.) in electrical engineering from Chulalongkorn University, Thailand, in 2001, and the master's and Ph.D. degrees in information and communication engineering from The University of Tokyo, in 2004 and 2007, respectively. From 2007 to 2008, he was granted a JSPS Postdoctoral Fellowship. He has been with the National Institute of Advanced Industrial Science and Technology (AIST), Japan, since 2008, where he is currently the Leader of the Cryptography Platform Research Team, Cyber Physical Security Research Center. His research interests include theoretical and applied cryptography, especially cryptographic schemes with advanced functionalities such as attribute-based encryption. He received the Ericsson Young Scientist Award, in 2010, and the Young Scientists' Prize from The Commendation for Science and Technology by the Minister of Education, Culture, Sports, Science, and Technology (MEXT, Japanese Government), in 2017.

TAKAHIRO MATSUDA received the bachelor's, master's, and Ph.D. degrees in information and communication engineering from The University of Tokyo, in 2006, 2008, and 2011, respectively. He has been with AIST, since 2011, where he is currently a Senior Research Scientist. His research interests include the foundations of cryptography, in particular encryption and digital signature schemes with enhanced security and functionality properties. He received the 2016 Docomo Mobile Science Award (Advanced Technology Award of Excellence).

GOICHIRO HANAOKA graduated from the Department of Engineering, The University of Tokyo, in 1997. He received the Ph.D. degree from the University of Tokyo, in 2002. In 2005, he joined AIST, where he is currently the Leader of the Advanced Cryptosystems Research Group, Information Technology Research Institute. He engages in the Research and Development for encryption and information security technologies, including the efficient design and security evaluation of public key cryptosystems. He received numerous awards, including the DoCoMo Mobile Science Award from the Mobile Communication Fund, in 2016, the Wilkes Award from the British Computer Society, in 2007, the Best Paper Award from the Institute of Electronics, Information and Communication Engineers (IEICE), in 2008, and the Innovative Paper Awards at the Symposium on Cryptography & Information Security (SCIS), IEICE, in 2012 and 2014.