On the Application of Binary Neural Networks in Oblivious Inference

Mohammad Samragh*
UC San Diego
msamragh@ucsd.edu

Siam Hussain*
UC San Diego
s2hussai@ucsd.edu

Xinqiao Zhang
UC San Diego, San Diego State University
x5zhang@ucsd.edu

Ke Huang
San Diego State University
khuang@sdsu.edu

Farinaz Koushanfar
UC San Diego
farinaz@ucsd.edu

Abstract

This paper explores the application of Binary Neural Networks (BNN) in oblivious inference, a service provided by a server to mistrusting clients. Using this service, a client can obtain the inference result on her data by a trained model held by the server without disclosing the data or leaning the model parameters. We make two contributions to this field. First, we devise light-weight cryptographic protocols designed specifically to exploit the unique characteristics of BNNs. Second, we present dynamic exploration of the runtime-accuracy tradeoff of BNNs in a single-shot training process. While previous works trained multiple BNNs with different computational complexities (which is cumbersome due to the slow convergence of BNNs), we train a single BNN that can perform inference under different computational budgets. Compared to CryptFlow2, the state-of-the-art in oblivious inference of non-binary DNNs, our approach reaches $2\times$ *faster inference at the same accuracy. Compared to XONN, the state-of-the-art in oblivious inference of binary networks, we achieve* $2\times$ *to* $11\times$ *faster inference while obtaining higher accuracy.*

1. Introduction

There is an increasing surge in cloud-based inference services that employ deep learning models. In this setting, the server trains and holds the DNN model and clients query the model to perform inference on their data. One major shortcoming of such service is the leakage of clients' private data to the server, which can hinder commercialization in certain applications. For instance, in medical diagnosis [1], clients would need to expose their "plaintext" health information to the server, which violates patient privacy regulations such as HIPAA [2].

One attractive option for ensuring clients' content privacy is the use of modern cryptographic protocols as they provide provable security guarantees [3–13]. Let $f(\theta, x)$ be the inference result on client's input x using server's parameters θ. By executing cryptographically-secure operations, client and server can jointly compute $f(\theta, x)$ without revealing x to the server or θ to the client. We refer to this process as *oblivious inference* in the remainder of the paper. Unlike plaintext inference, oblivious inference protects the privacy of both parties. The challenge, however, is the excessive computation and/or communication overhead associated with privacy-preserving computation. For example, the contemporary state-of-the-art for performing oblivious inference on a single CIFAR-10 image requires exchange of ~ 3.4 GB of data and takes ~ 10 seconds [14].

Early research on oblivious inference mostly focused on developing protocols for inference of a given DNN model, without making major modifications to the model itself [3–13]. Recently, a body of work has explored modifying the DNN architecture such that the resulting model is more amenable to secure computation [14–17]. Other potential directions for enhancing oblivious inference could include pruning [18], tensor decomposition [19], quantization [20], and Binary Neural Networks (BNNs) [21]. In this work, we study BNN as a candidate for fast and scalable oblivious inference. We show that a BNN has several unique characteristics that allow translating its computations to simple and efficient cryptographic protocols.

The benefits of employing BNNs for oblivious inference were first noted by XONN [14]. Despite achieving significant runtime improvement compared to non-binary DNN inference, there are opportunities provided by BNNs that have not been leveraged by XONN. Part of the inefficiency of XONN is due to the usage of a single secure computation protocol as a blackbox for all neural network layers after the input layer. In this work, we introduce a new hybrid approach where the underlying secure computation protocol is customized to each layer, such that the total execution cost for oblivious inference on all layers is minimized. We design a composite custom secure execution protocol, specifically optimized for BNN operations, using standard

*indicates equal contribution

security primitives. Our protocol significantly improves the efficiency of XONN as we show in our experiments.

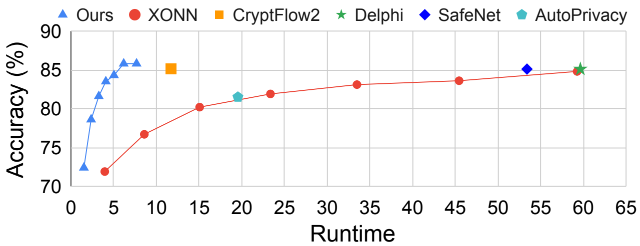


Figure 1: Accuracy and runtime of our oblivious BNN inference, compared with contemporary research that have the same server-client scenario setting as us (two-party, honest but curious). Among these, XONN [14] evaluates BNNs, whereas Cryptflow2 [12], Delphi [17], SafeNet [15], and AutoPrivacy [13] evaluate non-binary models.

One standing challenge in oblivious inference is finding architectures that are both accurate and amenable to secure computation. Since BNNs suffer from long training time and poor convergence, searching for such architectures could be quite inefficient. We address the search inefficiency challenge by training a *single BNN* that can operate under different computational budgets. Our adaptive BNN offers a tradeoff between accuracy and inference time, without requiring to train separate models. Figure 1 presents the tradeoff achieved by our flexible BNN on the 7-layer VGG network trained on CIFAR-10. With the combined power of our custom oblivious inference protocols and adaptive BNN training schemes, our method outperforms prior art both in terms of accuracy and runtime. Our solution is $\sim 2\times$ faster than Cryptflow2 [12], the state-of-the-art non-binary DNN inference framework, and $2\times$ to $11\times$ faster than XONN, the previous oblivious BNN inference framework.

2. Scenario and Threat Model

Figure 2 presents the scenario in oblivious inference. The neural network architecture f is known by both server and client. The server holds the set of trained parameters, i.e., $\theta = \{\theta^1, \ldots, \theta^L\}$, and the client holds the input query to the neural network, i.e., x. The two parties engage in a secure function evaluation protocol, where the client learns the inference result $y = f(\theta, x)$. Similar to prior work, we consider the honest-but-curious scenario [10–17]. In this threat model, the two parties follow the protocol that they agree upon to compute the output, yet they may try to learn about the other party's data as much as they can. As such, the protocol should guarantee the following requirements:

- x or $f(\theta, x)$ are not revealed to the server.
- θ is not revealed to the client.
- Client and server do not learn intermediate activations.

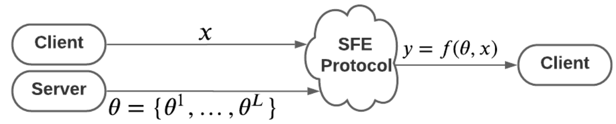


Figure 2: The server and client use a secure function evaluation (SFE) protocol to perform oblivious inference. At the end of the protocol, client learns $y = f(\theta, x)$ without learning server's parameters θ or revealing x to server.

3. Background

This section provides a high-level outline of the necessary terminologies. Following the convention in secure computation literature, we refer to server and client as Alice and Bob, respectively.

Secure Function Evaluation Protocol. During oblivious inference, Alice and Bob engage in a Secure Function Evaluation (SFE) protocol, which is essentially a set of rules specifying the messages communicated between them. By following these rules, they jointly compute the output of a function that takes the inputs from both of them without disclosing any information about Alice's data to Bob and vice versa. Depending on protocol agreements, the result of the computation can be exposed to both parties, only one of them, or neither of them.

Additive Secret Sharing (AS) is a method for distributing a secret x between Alice and Bob such that Alice holds $[\![x]\!]_A = x + r$ and Bob holds $[\![x]\!]_B = -r$, where r is a random value. Individually, both $[\![x]\!]_A$ and $[\![x]\!]_B$ are random values, hence, Alice and Bob cannot independently decipher the original message x. Only by combining $[\![x]\!]_A$ and $[\![x]\!]_B$ can one recover the actual secret as $x = [\![x]\!]_A + [\![x]\!]_B$. There exists standard SFE protocols to perform addition and multiplication on secret-shared data such that the result is also shared between the two parties. We employ these protocols in oblivious inference to ensure that neither the input nor the output of a layer is revealed to the involved parties. We refer curious readers to [22] for more details.

Oblivious Transfer (OT) is a protocol between two parties – a sender (Bob) who has two messages (μ_0, μ_1), and a receiver (Alice) who has a selection bit $i \in \{0, 1\}$ [23]. Through OT, Alice obtains the intended message μ_i, without revealing the selection bit i to Bob. Alice does not learn the other message μ_{1-i}. OT requires public key cryptography, which is costly in general. In the following, we introduce more efficient methods for OT computation.

OT extension enables extending a constant number of 'base OTs' to a large number of OTs through cheaper symmetric key cryptography [24]. The first step in OT-extension is called Random OT (ROT) [25]. In ROT, Alice provides the selection bit i and Bob does not provide any input. After ROT execution, Bob receives two random 128-

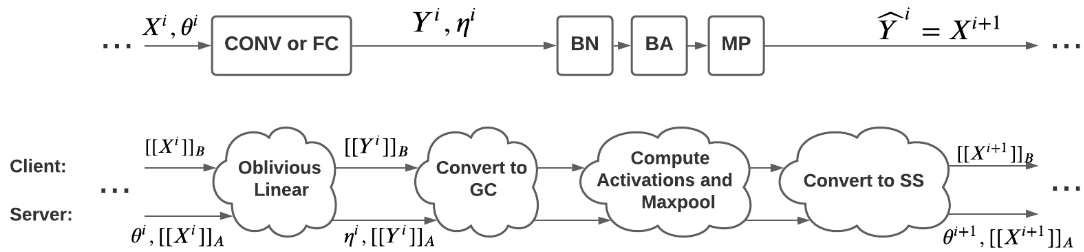


Figure 3: Illustration of plaintext inference (top) and our proposed equivalent oblivious inference (bottom). We denote linear layers by *CONV* and *FC*, Batch-Normalization by *BN*, Binary Activation by *BA*, and Max-Pooling by *MP*. Here, X^i, and Y^i, and θ^i are the input, output, and weight/bias parameters of the linear layer, respectively. η^i denotes BN parameters, and $\widehat{Y}$ is the output of binary activation.

bit keys (k_0, k_1) and Alice receives k_i. The final step of OT-extension is as follows: Bob computes $\{v_0, v_1\} = \{\mu_0 \oplus H(k_0), \mu_1 \oplus H(k_1)\}$, where the $\oplus$ operator denotes bit-wise XOR and $H(k)$ is a cryptographically-secure random number generator [26] with k as the seed. Bob transmits $\{v_0, v_1\}$ to Alice, who computes $\mu_i = v_i \oplus H(k_i)$.

In Section 4.1, we design a protocol for oblivious matrix multiplication, which enables oblivious evaluation of convolution and fully-connected layers. We build our protocol by only using OT and AS which we outlined above. However, AS and OT are not efficient for evaluating non-linear activations and Max-pooling.

Garbled Circuit (GC) is an SFE protocol that can be used for evaluation of an arbitrary function (linear or non-linear). The downside of GC is its heavy communication overhead. Therefore, we limit its usage to nonlinear operations. We refer curious readers to [27–29] for more details about GC.

4. Cryptographically Secure BNN Inference

BNNs were originally introduced to minimize memory footprint and computation overhead of plaintext inference. In this section, we provide insights on why BNNs are also useful for very efficient and fast oblivious inference.

The first favorable property of BNNs is enforcing the weights to +1 or -1. With this restriction, multiplying a feature x by a weight w is equivalent to computing either $+x$ or $-x$. This simple property becomes useful when computing vector dot products of the form $\sum_{i=1}^{N} w_i x_i$, which can be computed via N conditional additions/subtractions. We show in Section 4.1 that conditional summations can be computed using OT and AS, both of which are known to be very efficient and light-weight cryptographic tools.

In oblivious inference, nonlinear operations are evaluated through heavy cryptographic primitives such as GC, resulting in large runtime and communication overheads. The large communication cost of GC is directly related to the bit-widths of GC inputs. The second advantage of BNNs is their 1-bit hidden layer feature representation, which significantly reduces the GC evaluation cost when compared to non-binary features. In Section 4.2, we expand on low-bit nonlinear operations and their efficient GC evaluation.

We present the overall flow for oblivious BNN inference in Figure 3. The inputs and outputs of all layers are in AS format, e.g., server and client have $[\![Y^i]\!]_A$ and $[\![Y^i]\!]_B$ rather than Y^i. To obliviously evaluate linear layers (CONV or FC), we propose a novel custom protocol for binary matrix multiplication that directly works on AS data. We merge batch normalization (BN), binary activation (BA), and max-pooling (MP) into a single nonlinear function $f(\cdot)$. To securely evaluate $f([\![Y^i]\!]_A, [\![Y^i]\!]_B)$, three consecutive steps should be taken:

1. Securely translating the input from AS to GC. This step prepares the data to be processed by GC.
2. Computing the nonlinear layer through GC protocol.
3. Securely translating the result of the GC protocol to AS. This step prepares the data to be processed in the following linear layer.

Using this hybrid approach, we achieve a significantly faster oblivious inference compared to the state-of-the-art [14].

4.1. Linear Layers

Fully-connected and convolutional layers require computing $Y = WX$, with weight matrix W and input X. In secure matrix multiplication, the input is secret shared between the server and the client, i.e., $X = [\![X]\!]_A + [\![X]\!]_B$. Bob (the client) has $[\![X]\!]_B$ whereas Alice (the server) has the weight W and $[\![X]\!]_A$[1]. The matrix multiplication is computed as follows:

$$W([\![X]\!]_A + [\![X]\!]_B) = W[\![X]\!]_A + W[\![X]\!]_B \qquad (1)$$

Alice can compute $W[\![X]\!]_A$ locally and only $W[\![X]\!]_B$ needs secure evaluation. After evaluating $Y = WX$,

- Alice gets $[\![Y]\!]_A$ but does not learn $[\![X]\!]_B$ or $[\![Y]\!]_B$.

[1] At the first layer, only client has the input share, hence $[\![X]\!]_A = 0$

Algorithm 1: One-time setup for matrix-mult.

Input: from Alice $W \in \{-1,+1\}^{M\times N}$
Output: to Alice $K_A \in \mathbb{Z}^{M\times N}$
Output: to Bob $\{K_B^0 \in \mathbb{Z}^{M\times N}, K_B^1 \in \mathbb{Z}^{M\times N}\}$
Remark:

$$K_A[m,n] = \begin{cases} K_B^0[m,n] & if \;\; W[m,n] = -1 \\ K_B^1[m,n] & if \;\; W[m,n] = 1 \end{cases}$$

for $m \in [M]$ **do**
  for $n \in [N]$ **do**
    Alice and Bob engage in ROT where:
    • Alice inputs $i = \frac{W[m,n]+1}{2}$
    • Alice receives $K_A[m,n]$
    • Bob receives $\{K_B^0[m,n], K_B^1[m,n]\}$

Algorithm 2: Secure binary matrix-mult.

Input: from Alice $W \in \{-1,+1\}^{M\times N}$
Input: from Alice $K_A \in \mathbb{Z}^{M\times N}$
Input: from Alice $[\![X]\!]_A \in \mathbb{Z}^{N\times L}$
Input: from Bob $K_B^0 \in \mathbb{Z}^{M\times N}$
Input: from Bob $K_B^1 \in \mathbb{Z}^{M\times N}$
Input: from Bob $[\![X]\!]_B \in \mathbb{Z}^{N\times L}$
Output: to Alice $[\![Y]\!]_A \in \mathbb{Z}^{M\times L}$
Output: to Bob $[\![Y]\!]_B \in \mathbb{Z}^{M\times L}$
Remark: j is the number of inferences so far
Remark: $[\![Y]\!]_A + [\![Y]\!]_B = W([\![X]\!]_A + [\![X]\!]_B)$

1 Alice locally sets $[\![Y]\!]_A = W[\![X]\!]_A \in \mathbb{Z}^{M\times L}$
2 Bob locally sets $[\![Y]\!]_A = \mathbf{0} \in \mathbb{Z}^{M\times L}$
3 **for** $m \in [M]$ **do**
4   Alice locally sets $[\![y]\!]_A = [\![Y(m,:)]\!]_A$
5   Bob locally sets $[\![y]\!]_B = [\![Y(m,:)]\!]_B$
6   **for** $n \in [N]$ **do**
7   Bob generates random vector $r \in \mathbb{Z}^L$
8   Bob computes:
$$\begin{cases} v_0 = H(j, K_B^0[m,n]) \oplus (r - [\![X[n,:]]\!]_B) \\ v_1 = H(j, K_B^1[m,n]) \oplus (r + [\![X[n,:]]\!]_B) \end{cases}$$
9   Bob sends v_0, v_1 to Alice
10   Knowing $i = \frac{W[m,n]+1}{2}$, Alice computes: $\mu_i = H(j, K_A[m,n]) \oplus v_i$
11   Alice locally updates $[\![y]\!]_A = [\![y]\!]_A + \mu_i$
12   Bob locally updates $[\![y]\!]_B = [\![y]\!]_B - r$
13   Alice locally updates $[\![Y(m,:)]\!]_A = [\![y]\!]_A$
14   Bob locally updates $[\![Y(m,:)]\!]_B = [\![y]\!]_B$

- Bob gets $[\![Y]\!]_B$ but does not learn W or $[\![Y]\!]_A$.

The above computation is performed in in two phases: (1) the setup phase, shown in Algorithm 2, where Alice and Bob perform ROTs. Note that the setup phase only depends on the weight matrix which remains unchanged over a large number of inferences. Therefore this phase is performed only once and the cost is amortized among all future oblivious inferences. (2) the inference phase, shown in Algorithm 2, which is performed separately for each inference. Initially, Alice sets her output share to $W[\![X]\!]_A$ (line 1) and Bob sets his share to zero (line 2). Next, they obliviously evaluate $W[\![X]\!]_B$ one row at a time in the outer loop of Algorithm 2 (lines 3-14). Specifically, the m-th iteration of the outer loop evaluates the m-th row of the output as:

$$y_n = [\![y_n]\!]_A + [\![y_n]\!]_B = \sum_{n=1}^{N} W[m,n]X[n,:]$$

The inner loop of Algorithm 2 (lines 6-12) computes the above summation by running OT for $n \in [N]$. After each OT invocation, Alice receives either $\mu_0 = r - [\![X[n,:]]\!]_B$ or $\mu_1 = r + [\![X[n,:]]\!]_B$ depending on the selection bit. It is easy to see that μ_i (known by Alice) and $-r$ (known by Bob) are the arithmetic shares of $W[m,n][\![X[n,:]]\!]_B$.

4.2. Nonlinear Layers

In this section, we outline and leverage characteristics of BNNs for oblivious inference of nonlinear layers. The cascade of batch normalization (BN) and binary activation (BA) takes input feature y and returns $\widehat{y} = sign(\alpha y + \beta) = sign(y + \frac{\beta}{\alpha})$, where α and β are the BN parameters. Since both α and β belong to the server, the parameter $\eta = \frac{\beta}{\alpha}$ can be computed offline. The GC evaluation of BN and BA only entails adding η to y and computing the sign of the result, which can be evaluated by relatively low GC cost [30]. Moreover, binary Max-Pooling can be efficiently evaluated at the bit-level. Taking the maximum in a window of binarized scalars is equivalent to performing logical `OR` among the values, which is also efficient in GC [30].

Algorithm 3 presents our efficient protocol for oblivious evaluation of nonlinear layers in BNNs, which levereges the insights discussed above. Our protocol receives secret-shared data $[\![Y]\!]_A$ and batch-normalization parameter values $\eta = \frac{\beta}{\alpha}$ from the server, as well as $[\![Y]\!]_B$ from the client. It then computes $\widehat{Y}$ by applying batch normalization, binary activation, and max-pooling on Y. Upon completion of the protocol, server and client receive $[\![\widehat{Y}]\!]_A$ and $[\![\widehat{Y}]\!]_B$, respectively, which they use to evaluate the proceeding layer.

4.3. Communication Cost

Recall that each layer execution is done via SFE protocol, where the two involved parties cooperatively compute output shares of their own. During the protocol, each party may perform certain computation, storage, or random data generation internally on their own device. In privacy-preserving computation, these type of local processes are deemed as *free* operations. In practice, the runtime of the

Algorithm 3: Protocol for secure non-linear operations.

Input: from Alice $[\![Y]\!]_A$
Input: from Alice η
Input: from Bob $[\![Y]\!]_B$
Output: to Alice $[\![\widehat{Y}]\!]_A$
Output: to Bob $[\![\widehat{Y}]\!]_B$
Remark: $[\![\widehat{Y}]\!]_A + [\![\widehat{Y}]\!]_B = f([\![Y]\!]_A + [\![Y]\!]_B + \eta)$
Remark: $f(\cdot)$ denotes BN, BA, and optional MP.

Alice locally computes $[\![Y]\!]_A + \eta$
Bob locally generates random tensor R
Alice and Bob engage in GC where:
  • Alice inputs $[\![Y]\!]_A + \eta$
  • Bob inputs $[\![Y]\!]_B$ and R
  • GC computes $F = R + f([\![Y]\!]_A + \eta + [\![Y]\!]_B)$
  • GC returns F only to Alice
Alice sets $[\![\widehat{Y}]\!]_A = F$
Bob sets $[\![\widehat{Y}]\!]_B = -R$

Table 1: Communication Cost for different stages of our oblivious inference protocols. Here, b is the bitwidth for arithmetic sharing[2]. κ is a security parameter, and its standard value is 128 in recent literature. For max-pooling, w is the window size. In cases where max-pooling is applied, the dimensionality is reduced from L to $L' \approx \frac{L}{w^2}$.

Stage	Underlying Operation	Communication (bits)
Mat-Mult	$Y \leftarrow W([\![X]\!]_A + [\![X]\!]_B)$	$NbML$
BN+BA	$\widehat{Y} \leftarrow sign([\![Y]\!]_A + \eta + [\![Y]\!]_B)$	$5\kappa bML$
MP	$\widehat{Y} \leftarrow maxpool_{w\times w}(\widehat{Y})$	$2(w^2-1)\kappa ML'$
SS	$[\![\widehat{Y}]\!]_A \leftarrow \widehat{Y} + R$	$3\kappa bML'$

process is dominated by the exchange of messages between the two parties, not the internal computations. In our protocols (Algorithms 2& 3), message exchanges occur during OT or GC invocations. We provide the communication cost of our protocols in Table 1. By plugging in the parameters of this table, one can compute the total execution cost for oblivious inference of a given BNN architecture. As we show in our experiments, the communication cost is closely tied with the runtime of our protocols.

5. Training Adaptive BNN

One of the primary challenges of BNNs is to ensure inference accuracy comparable to the non-binarized model. Since the introduction of BNNs, there have been tremendous efforts to improve inference accuracy by increasing the number of channels per convolution layer [31], increasing the number of computation bits [32], or introducing new connections and nonlinear layers [33, 34], to name a few. In this paper, we improve the accuracy of the base BNN by multiplying its width, e.g., by training an architecture with twice as many neurons at each layer. In practice, specifying the appropriate width for a BNN architecture requires exploring models with various widths, which can be quite time-consuming and cumbersome. Each model with a certain width should be trained and stored separately. What aggravates the problem is that BNNs suffer from convergence issues unless the data augmentation and training hyperparameters are carefully selected [35].

A related field of research is training dynamic DNNs [36], with the goal of providing flexibility at inference time. In this realm, we find Slimmable Networks [37] quite compatible to our problem setting and adapt them to BNNs. Our goal is to train a single network with certain maximum width, say $4\times$ the base network, in a way that the model can still deliver acceptable accuracy at lower widths, e.g., $1\times$ or $2\times$ the base network. Once this model is trained, it can operate under any of the selected widths, thus, providing a tradeoff between accuracy and runtime.

Slimmable BNNs Definition. Let us denote the base BNN as M_1 and represent BNNs with $s\times$ higher width at each layer with M_s. Our goal is to train $M_{s_1} \subset M_{s_2} \subset M_{s_n}$ for a number of widths $\{s_i\}_{i=1}^n$. The weights of M_{s_i} are a subset of the weights of $M_{s_{i+1}}$. Therefore, having M_{s_n} we can configure it to operate as any M_{s_i} for $i \le n$.

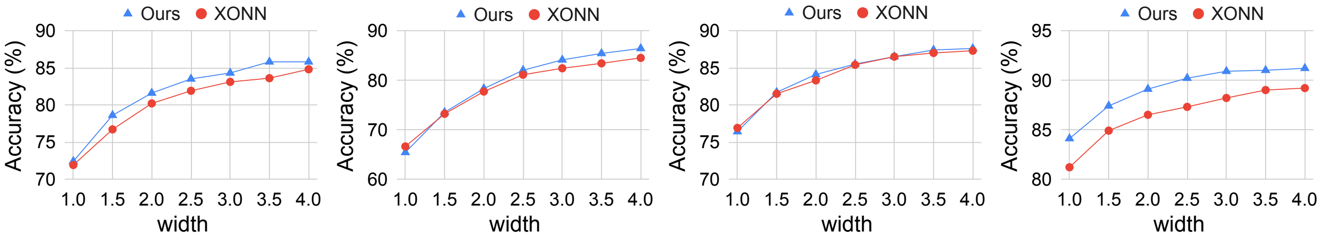
Training Slimmable BNNs. For a given minibatch X, each subset model computes the output as $\widetilde{Y}_{s_i} = M_{s_i}(X)$, resulting in $\{\widetilde{Y}_{s_1}, \dots, \widetilde{Y}_{s_n}\}$ computed by $M_{s_1} \dots M_{s_n}$. The ground-truth label Y is then used to compute the cumulative loss function as $\sum_{i=1}^{n} \mathcal{L}(Y, \widetilde{Y}_{s_i})$, where $\mathcal{L}(\cdot,\cdot)$ represents cross-entropy. The BNN weights are then updated using the standard gradient approximation rule suggested in [21].

6. Evaluations

Standard Benchmarks. We perform our evaluation on several networks trained on the CIFAR-10 dataset, shown in Table 2. The BC1 network has been evaluated by the majority oblivious inference papers [10–15, 17, 38, 39]. Other models are evaluated by XONN [14], the state-of-the-art for oblivious inference of *binary* networks. For brevity, we omit details about layer-wise configurations and refer curious readers to [14] for further information.

Training. For all benchmarks, we use standard backpropagation algorithm proposed by [21] to train our binary networks. We split the CIFAR10 dataset to 45k training examples, 5k validation examples, and 10k testing examples, and train each architecture for 300 epochs. We use Adam optimizer with initial learning rate of 0.001, and the learning rate is multiplied by 0.1 after 101, 142, 184 and 220 epochs. The batch size is set to 128 across all CIFAR10 training ex-

[2]To ensure correctness, b should be set to $\lceil Log(N)+1\rceil$. In practice, software libraries only support multipliers of 8. Hence, we set b to the smallest multiplier of 8 bigger than or equal to $\lceil Log(N)+1\rceil$.



(a) BC1 (b) BC2 (c) BC3 (d) BC4

Figure 4: CIFAR-10 test accuracy of each architecture at different widths. Our Adaptive BNN trains a single network that can operate at all widths, whereas previous work (XONN) trains a separate BNN per width.

Table 2: Summary of the trained binary network architectures evaluated on the CIFAR-10 dataset.

Arch.	Previous Papers	Description
BC1	[10], [38], [39], [11], [14], [17], [12], [15], [13]	7 CONV, 2 MP, 1 FC
BC2	[14]	9 CONV, 3 MP, 1 FC
BC3	[14]	9 CONV, 3 MP, 1 FC
BC4	[14]	11 CONV, 3 MP, 1 FC

periments. The training data is augmented by zero padding the images to 40×40, and randomly cropping a 32×32 window from each zero-padded image.

Evaluation Setup. The training codes are implemented in Python using the Pytorch Library. We use a single Nvidia Titan Xp GPU to train all benchmarks. We design a library for oblivious inference in C++. For implementation of OT and GC, we use the standard emp-toolkit [40] library. To run oblivious inference, we translate the model description and trained parameters from Pytorch to the equivalent description in our C++ library. For measurements, we run our oblivious inference code on a computer with 2.2 GHz Intel Xeon CPU and 16 GB RAM. For runtime measurements, we consider two real-world network settings, namely LAN with a throughput of 1.25 GBps, round trip time of 0.25ms, and WAN with a throughput of 20 MBps, round trip time of 50ms. Reported runtimes do not include the setup time.

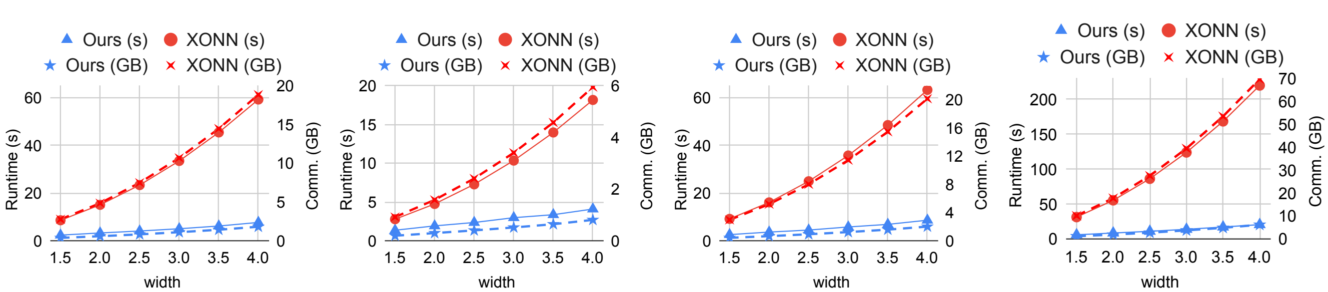
6.1. Evaluating Flexible BNNs

Let us start by evaluating our adaptive BNN training. We train slimmable networks with maximum $4\times$ width of the base models presented in Table 2. During training, we reiterate through subsets of widths $\{1\times, 1.5\times, \ldots, 4\times\}$ and perform gradient updates as explained in Section 5.

Figure 4 presents the test accuracy of each network at different widths. We also report the accuracy of independetly trained networks reported by XONN. The test accuracy of a particular base BNN architecture can be improved by increasing its width. Our adaptive networks obtain better accuracy than independently trained BNNs at each width. Once the adaptive network is trained, the server can provide oblivious inference service to clients, which we discuss in the following section.

6.2. Oblivious Inference

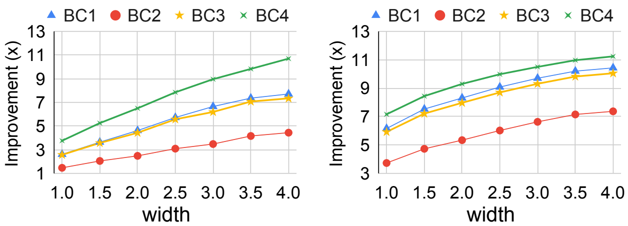
Recall that the runtime of oblivious inference is dominated by data exchange between client and server. We compare the communication cost and runtime of our custom protocol with XONN's GC implementation in Figure 5. The horizontal axis in each figure presents the network width. The left and right vertical axes respectively show the runtime (in seconds) and communication (in Giga-Bytes). The figure shows that for all the benchmarks, the runtime and communication of our method are significantly smaller than XONN. As seen, increasing the network width results in higher communication and runtime, which is the cost we



(a) BC1 (b) BC2 (c) BC3 (d) BC4

Figure 5: Runtime and communication cost of each architecture at different widths.

pay for higher inference accuracy.



(a) Runtime (b) Communication

Figure 6: Improvements in LAN runtime and communication compared to XONN. Our protocols achieve $2\times$ to $11\times$ in runtime and $4\times$ to $11\times$ communication reduction.

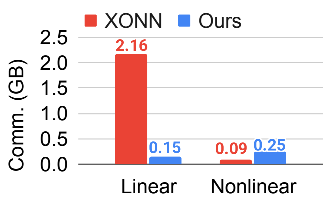


Figure 7: Breakdown of communication cost at linear and nonlinear layers for BC2 network. Our protocol significantly reduces XONN's GC-based linear layer cost, with a slight increase in nonlinear layer cost.

Figure 6 summarizes the performance boost achieved by our protocols, i.e., $2\times$ to $11\times$ lower runtime and $4\times$ to $11\times$ lower communication compared to XONN. The enhancement is more significant at higher widths, which shows the scalability for our method. To illustrate the reason behind our protocol's better performance, we focus our attention to the BC2 network at width 2.5, and show the breakdown of its communication cost in Figure 7. For the XONN protocol, most of the cost is from linear operations, which we reduce from 2.16GB to 0.15GB. In nonlinear layers, our cost is slightly more that XONN's, i.e., 0.25GB versus 0.09GB, which is due to the extra cost of conversion between AS and GC. Overall, the total communication is reduced from 2.25GB to 0.4GB compared to XONN.

Comparison to Non-binary Models. Among the architectures presented in Table 2, BC1 has been commonly evaluated in contemporary oblivious inference research. In Figure 1 we compare the performance of our method to the best-performing earlier work on this benchmark. The vertical and horizontal axes in the figure represent test accuracy and runtime, hence, points to the top-left corner are more desirable. Our method achieves a better accuracy/runtime tradeoff than all contemporary work while providing flexibility. Compared to Cryptflow2 (the most recent oblivious inference framework at the time of this paper), our method achieves $\sim 2\times$ faster inference at the same accuracy.

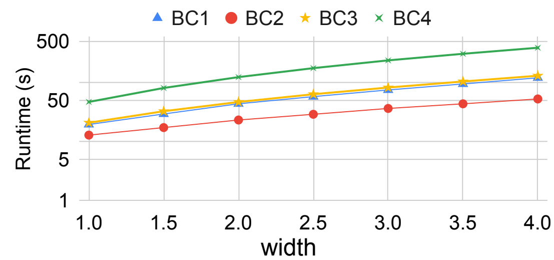


Figure 8: Inference runtime in WAN setting with $\sim$ 20 MBps bandwidth and $\sim$ 50 ms network delay.

Evaluation in Wide Area Network (WAN). So far we reported our runtimes for the setting where client and server are connected via LAN, which is the most common assumption among prior work. We now extend our evaluation to the WAN setting, where the bandwidth is $\sim$ 20MBps and the delay is $\sim$ 50ms. The aforesaid bandwidth and delay correspond to the connection speed between two AWS instances located in "US-West-LA-1a" and "US-East-2a". Runtimes are reported in Figure 8, showing varying inference time from 13 to 367 seconds depending on architecture and width. The results show the great potential of BNNs for commercial use. Indeed, the delay introduced by oblivious inference might not be tolerable in many applications that require real-time response, e.g., Amazon Alexa. However, there exist many applications where guaranteeing privacy is much more crucial than runtime, and several seconds or even minutes of delay can be tolerated. We evaluate two such applications in the following section.

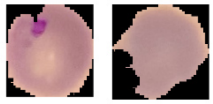
6.3. Evaluation on Private Tasks

In this section, we study the application of oblivious inference in face authentication and medical data analysis. Both applications involve sensitive features that the client wishes to keep secret: revealing medical data is against the HIPPA [2] regulation, and facial features can be used by malicious hackers to authenticate into the client's personal accounts. Since we do not have access to real private data, our best choice is to simulate these tasks using similar datasets that are publicly available to the research community. We evaluate our method on FaceScrub [41,42] and Malaria Cell Infection [43] as representatives for face authentication and medical diagnosis, respectively.

Figure 9 shows example samples from each dataset. We were able to download $\sim$ 57,000 images from the links provided by FaceScrub authors, of which we use 45000 for training, 6000 for validation, and 6000 for testing. The Malaria dataset is split to $\sim$ 24800 samples for training, $\sim$ 1300 for evaluation, and $\sim$ 1300 for testing. We train the *BC2* architecture at width 3 and 1 on FaceScrub and Malaria. The accuracy and performance results in the WAN setting are summarized in Table 3. Our model reaches 70.2.1% inference accuracy on FaceScrub and 94.7% ac-

Andrea Bowen | Jodi Long | Bernard Hill

(a) Facescrub



infected | benign

(a) Malaria Cells

Figure 9: examples of input samples and labels from each dataset. For training, we resize Facescrub and Malaria cell images to 50×50 and 32×32, respectively.

curacy on Malaria infection detection. The networks incur runtimes of 1-3 and 10-30 seconds in LAN and WAN settings, showing great potential for practical deployment. Note that in a commercial application the network architecture can be selected more carefully and more training data can be collected to achieve a better accuracy and runtime.

Table 3: Example BNNs trained for face recognition and medical application. We use the *BC2* architecture at width 3 and 1 for FaceScrub and Malaria, respectively. Runtimes are measured in the WAN setting.

Task	**Classes**	**Accuracy**	**Comm.**	**Runetime (s)**	
				LAN	**WAN**
FaceScrub	530	70.8%	404 MBs	2.2	32.2
Malaria	2	94.7%	80.5 MBs	0.7	11.5

7. Related Work

Oblivious inference was shown to be conceptually practical for small sized neural networks in CryptoNets [44]. Using CryptoNets, an inference on MNIST data would take $\sim$ 300 seconds, which motivated researcher to invest in the field. Since then, a plethora of more efficient protocols for oblivious inference have been proposed [3–13]. These works mainly focus on optimization of security primitives for oblivious inference, without making major modifications to the model.

A second line of research has been focused on identifying DNN models that are inherently amenable to secure execution protocols. Several DNN modification examples include replacing ReLU operations with square function [15, 17, 44], using dimensionality reduction at the input layer [45], and neural architecture search [16]. Concurrently, researchers in ML community have devised DNN optimization techniques such as pruning [18], quantization [20], tensor factorization [19], and binary neural networks [21]. Among the above, BNNs are especially compelling candidates for oblivious inference, since they translate linear arithmetic to bitwise operations. XONN [14] was the first work to notice the especial use case of binary networks for cryptographically secure inference using GC [28], noting that `XNOR` operations that frequently appear in BNNs can be evaluated for free in GC.

Despite improving oblivious inference time, XONN does not completely utilize the full set of opportunities provided by BNNs. Instead of using GC as a black box, we propose a hybrid protocol where GC is only used for nonlinear operations. We propose a novel protocol for matrix multiplication based on secret sharing and oblivious transfer. By exploiting the characteristics of BNN linear operations, our protocol achieves up to $11\times$ reduction in runtime compared to XONN. A remaining challenge with BNNs is their low inference accuracy, which XONN addresses partially by brute-force training of many BNN models, and choosing the one with proper accuracy/runtime for deployment. Alternatively, we show that BNNs can be trained via the Slimmable Network training technique [37]. We provide accurate and efficient BNN benchmarks for oblivious inference, that offer a tradeoff between execution cost and inference accuracy.

Last but not least, variants of BNNs are being developed to enhance inference accuracy, opening exciting avenues for future research. Developing custom protocols to securely evaluate residual connections [33], residual activation binarization [32], and PReLU nonlinearity [34] are interesting future directions for oblivious BNN inference. Our current oblivious inference implementation does not support these operations. However, the aforementioned techniques can be integrated and tested in future work, which may or may not result in improved accuracy-runtime tradeoff.

8. Conclusion

This paper studies the application of binary neural networks in oblivious inference, where a server provides a privacy-preserving inference service to clients. Using this service, clients can run the neural network owned by the server, without revealing their data to the server or learning the parameters of the model. We explore favorable characteristics of BNNs that make them amenable to oblivious inference, and design custom cryptographic protocols to leverage these characteristics. In contrast to XONN [14], which uses GC to evaluate both linear and non-linear layers, we use GC only for nonlinear layers. We present a custom protocol for linear layers using OT and AS, which leads to $2\times$ to $11\times$ performance improvement compared to XONN. We also address the problem of low inference accuracy by training adaptive BNNs, where a single model is trained to be evaluated under different computational budgets. Finally, we extend our evaluations to computer vision tasks that perform inference on private data, i.e., face authentication and medical data analysis.

References

[1] Andre Esteva, Alexandre Robicquet, Bharath Ramsundar, Volodymyr Kuleshov, Mark DePristo, Katherine Chou, Claire Cui, Greg Corrado, Sebastian Thrun, and Jeff Dean. A guide to deep learning in healthcare. *Nature medicine*, 25(1):24, 2019.

[2] The HIPAA Privacy Rule. https://www.hhs.gov/hipaa/for-professionals/privacy/index.html.

[3] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. Cryptodl: Deep neural networks over encrypted data. *arXiv preprint arXiv:1711.05189*, 2017.

[4] Alon Brutzkus, Ran Gilad-Bachrach, and Oren Elisha. Low latency privacy preserving inference. In *International Conference on Machine Learning*, pages 812–821. PMLR, 2019.

[5] Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. Fast homomorphic evaluation of deep discretized neural networks. In *Annual International Cryptology Conference*, pages 483–512. Springer, 2018.

[6] Edward Chou, Josh Beal, Daniel Levy, Serena Yeung, Albert Haque, and Li Fei-Fei. Faster cryptonets: Leveraging sparsity for real-world encrypted inference. *arXiv preprint arXiv:1811.09953*, 2018.

[7] Amartya Sanyal, Matt Kusner, Adria Gascon, and Varun Kanade. Tapas: Tricks to accelerate (encrypted) prediction as a service. In *International Conference on Machine Learning*, pages 4490–4499. PMLR, 2018.

[8] Roshan Dathathri, Olli Saarikivi, Hao Chen, Kim Laine, Kristin Lauter, Saeed Maleki, Madanlal Musuvathi, and Todd Mytkowicz. Chet: an optimizing compiler for fully-homomorphic neural-network inferencing. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 142–156, 2019.

[9] Marshall Ball, Brent Carmer, Tal Malkin, Mike Rosulek, and Nichole Schimanski. Garbled neural networks are practical. *IACR Cryptol. ePrint Arch.*, 2019:338, 2019.

[10] Jian Liu, Mika Juuti, Yao Lu, and Nadarajah Asokan. Oblivious neural network predictions via minionn transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 619–631, 2017.

[11] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. {GAZELLE}: A low latency framework for secure neural network inference. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 1651–1669, 2018.

[12] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow2: Practical 2-party secure inference. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 325–342, 2020.

[13] Qian Lou, Bian Song, and Lei Jiang. Autoprivacy: Automated layer-wise parameter selection for secure neural network inference. In *Advances in Neural Information Processing Systems*, 2020.

[14] M Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin E Lauter, and Farinaz Koushanfar. Xonn: Xnor-based oblivious deep neural network inference. In *USENIX Security*, 2019.

[15] Qian Lou, Yilin Shen, Hongxia Jin, and Lei Jiang. {SAFEN}et: A secure, accurate and fast neural network inference. In *International Conference on Learning Representations*, 2021.

[16] Zahra Ghodsi, Akshaj Veldanda, Brandon Reagen, and Siddharth Garg. Cryptonas: Private inference on a relu budget. In *Advances in Neural Information Processing Systems*, 2020.

[17] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A cryptographic inference service for neural networks. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*, 2020.

[18] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1389–1397, 2017.

[19] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*, 2015.

[20] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

[21] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[22] Mikhail Atallah, Marina Bykova, Jiangtao Li, Keith Frikken, and Mercan Topkara. Private collaborative forecasting and benchmarking. In *Proceedings of the 2004 ACM workshop on Privacy in the electronic society*, pages 103–114, 2004.

[23] Moni Naor and Benny Pinkas. Computationally secure oblivious transfer. *Journal of Cryptology*, 18(1), 2005.

[24] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *Crypto*, volume 2729. Springer, 2003.

[25] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 535–548, 2013.

[26] Christof Paar and Jan Pelzl. *Understanding cryptography: a textbook for students and practitioners*. Springer Science & Business Media, 2009.

[27] Sophia Yakoubov. A gentle introduction to yao's garbled circuits, 2017.

[28] Andrew Yao. How to generate and exchange secrets. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*, 1986.

[29] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In *International Colloquium on Automata, Languages, and Programming*. Springer, 2008.

[30] Benjamin Mood, Debayan Gupta, Henry Carter, Kevin Butler, and Patrick Traynor. Frigate: A validated, extensible, and efficient compiler and interpreter for secure computation. In *EuroS&P)*, pages 112–127. IEEE, 2016.

[31] Asit Mishra, Eriko Nurvitadhi, Jeffrey J Cook, and Debbie Marr. Wrpn: Wide reduced-precision networks. *arXiv preprint arXiv:1709.01134*, 2017.

[32] Mohammad Ghasemzadeh, Mohammad Samragh, and Farinaz Koushanfar. Rebnet: Residual binarized neural network. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 57–64. IEEE, 2018.

[33] Joseph Bethge, Christian Bartz, Haojin Yang, Ying Chen, and Christoph Meinel. Meliusnet: Can binary neural networks achieve mobilenet-level accuracy? *arXiv preprint arXiv:2001.05936*, 2020.

[34] Zechun Liu, Zhiqiang Shen, Marios Savvides, and Kwang-Ting Cheng. Reactnet: Towards precise binary neural network with generalized activation functions. In *European Conference on Computer Vision*, pages 143–159. Springer, 2020.

[35] Wei Tang, Gang Hua, and Liang Wang. How to train a compact binary neural network with high accuracy? In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[36] Lanlan Liu and Jia Deng. Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[37] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. *arXiv preprint arXiv:1812.08928*, 2018.

[38] M Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M Songhori, Thomas Schneider, and Farinaz Koushanfar. Chameleon: A hybrid secure computation framework for machine learning applications. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pages 707–721, 2018.

[39] Nishanth Chandran, Divya Gupta, Aseem Rastogi, Rahul Sharma, and Shardul Tripathi. Ezpc: programmable, efficient, and scalable secure two-party computation for machine learning. *ePrint Report*, 1109, 2017.

[40] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. EMP-toolkit: Efficient MultiParty computation toolkit. https://github.com/emp-toolkit, 2016.

[41] FaceScrub. *The FaceScrub dataset*, 2020. http://engineering.purdue.edu/~mark/puthesis, (accessed July 3, 2020).

[42] Hong-Wei Ng and Stefan Winkler. A data-driven approach to cleaning large face datasets. In *IEEE international conference on image processing*, 2014.

[43] Malaria Cell Images, accessed on 01/20/2019. https://www.kaggle.com/iarunava/cell-images-for-detecting-malaria.

[44] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*, 2016.

[45] Bita Darvish Rouhani, M Sadegh Riazi, and Farinaz Koushanfar. Deepsecure: Scalable provably-secure deep learning. *arXiv preprint arXiv:1705.08963*, 2017.