Scalable Binary Neural Network Applications in Oblivious Inference

XINQIAO ZHANG, UC San Diego and San Diego State University, USA
MOHAMMAD SAMRAGH and SIAM HUSSAIN, UC San Diego, USA
KE HUANG, San Diego State University, USA
FARINAZ KOUSHANFAR, UC San Diego, USA

Binary neural network (BNN) delivers increased compute intensity and reduces memory/data requirements for computation. Scalable BNN enables inference in a limited time due to different constraints. This paper explores the application of Scalable BNN in oblivious inference, a service provided by a server to mistrusting clients. Using this service, a client can obtain the inference result on his/her data by a trained model held by the server without disclosing the data or learning the model parameters. Two contributions of this paper are: (1) we devise lightweight cryptographic protocols explicitly designed to exploit the unique characteristics of BNNs. (2) we present an advanced dynamic exploration of the runtime-accuracy tradeoff of scalable 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 various computational budgets. Compared to CryptFlow2, the state-of-the-art technique in the oblivious inference of non-binary DNNs, our approach reaches 3× faster inference while keeping the same accuracy. Compared to XONN, the state-of-the-art technique in the oblivious inference of binary networks, we achieve 2× to 12× faster inference while obtaining higher accuracy.

CCS Concepts: • **Computing methodologies** → **Neural networks;**

Additional Key Words and Phrases: Scalable neural network, oblivious inference, secure computing

ACM Reference format:
Xinqiao Zhang, Mohammad Samragh, Siam Hussain, Ke Huang, and Farinaz Koushanfar. 2024. Scalable Binary Neural Network Applications in Oblivious Inference. *ACM Trans. Embedd. Comput. Syst.* 23, 3, Article 45 (May 2024), 18 pages.
https://doi.org/10.1145/3607192

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

This work was supported in part by ARO-MURI under award number W911NF-21-1-0322, NSF-CNS award number 2016737, and the Intel PrivateAI Collaborative Research Institute.
Authors' addresses: X. Zhang, UC San Diego and San Diego State University, 9500 Gilman Dr, La Jolla, CA, 92093, USA; email: x5zhang@ucsd.edu; M. Samragh, S. Hussain, and F. Koushanfar, UC San Diego, 9500 Gilman Dr, La Jolla, CA, 92093, USA; emails: {msamragh, s2hussai, farinaz}@ucsd.edu; K. Huang, San Diego State University, 5500 Campanile Dr, San Diego, CA, 92182, USA; email: khuang@sdsu.edu.

1539-9087/2024/05-ART45 $15.00
https://doi.org/10.1145/3607192

inference on their data. One major shortcoming of such a service is the leakage of clients' private data to the server, which can hinder commercialization in specific applications. For instance, in medical diagnosis [13], clients would need to expose their "plaintext" health information to the server, which violates patient privacy regulations such as HIPAA [43]. One attractive option for ensuring clients' content privacy is the use of modern cryptographic protocols, as they provide provable security guarantees. [4, 6, 7, 9, 12, 20, 23, 27, 31, 37, 41]. Let $f(\theta, x)$ be the inference result on the client's input x using the server's parameters θ. By executing cryptographically-secure operations, the 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 an exchange of $\sim$ 3.4 GB of data and takes $\sim$ 10 seconds [38]. Early research on oblivious inference mainly focused on developing protocols for inference of a given DNN model without making significant modifications to the model itself [4, 6, 7, 9, 12, 20, 23, 27, 31, 37, 41]. Recently, a body of work has explored modifying the DNN architecture such that the resulting model is more amenable to secure computation [17, 30, 33, 38]. Other potential directions for enhancing oblivious inference could include pruning [19], tensor decomposition [24], quantization [49], and **Binary Neural Networks (BNNs)** [11]. 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.

XONN first noted the benefits of employing BNNs for oblivious inference[38]. Despite achieving significant runtime improvement compared to non-binary DNN inference, there are opportunities provided by BNNs that XONN has not leveraged. 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 in which the underlying secure computation protocol is customized to each layer to minimize the total execution cost for oblivious inference on all layers. We design a composite custom secure execution protocol optimized for BNN operations using standard security primitives. Our protocol significantly improves the efficiency of XONN, as shown in our experiments.

One standing challenge in oblivious BNN inference is finding network architectures that are both accurate and amenable to secure computation. Since BNNs suffer from long training times and poor convergence, searching for such architectures could be inefficient. We address the search inefficiency challenge by training a *single BNN* that can operate under different computational budgets. Our adaptive BNN offers a trade-off between accuracy and inference time without requiring training separate models.

Slimmable neural networks [15] are a category of neural networks in that single networks can change the runtime width based on different conditions. Slimmable neural networks can adaptively choose the width of their model to optimize the accuracy-efficiency trade-offs. **Universally slimmable networks (US-Nets)** [47] enable a more flexible inference by using an arbitrary number of width instead of predefined width to get more favorable accuracy-efficiency trade-offs.

Figure 1 presents the trade-off 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. For example, we achieve $\sim 2\times$ faster oblivious inference at the same accuracy compared to Cryptflow2 [37], the state-of-the-art non-binary DNN inference framework, and $2\times$ to $11\times$ lower runtime compared to XONN, the previous work on oblivious BNN inference.

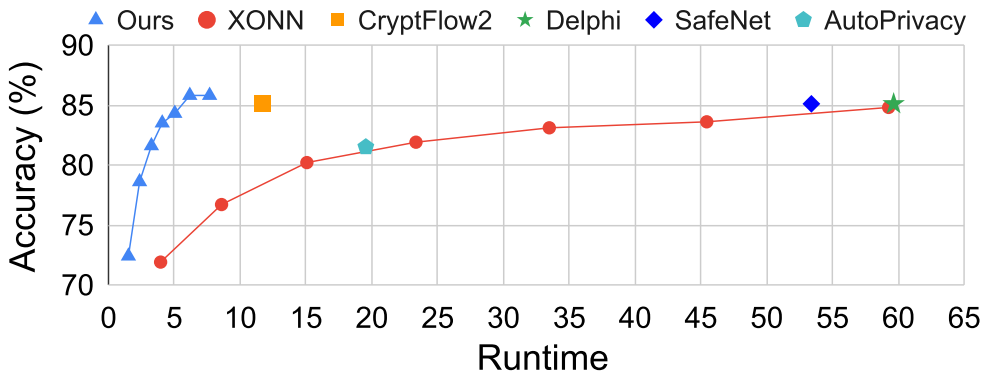


Fig. 1. Accuracy and runtime of our oblivious BNN inference, compared with contemporary research with the same server-client scenario setting as us (two-party, honest but curious). Among these, XONN [38] evaluates BNNs, whereas CryptFlow2 [37], Delphi [33], SafeNet [30], and AutoPrivacy [31] evaluate non-binary models.

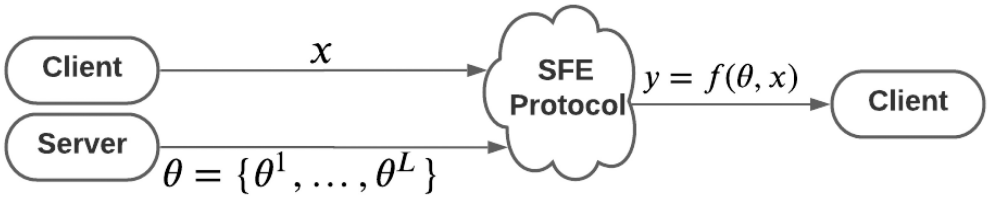


Fig. 2. The server and client use a secure function evaluation (SFE) protocol to perform oblivious inference. At the end of the protocol, the client learns $y = f(\theta, x)$ without learning the server's parameters θ or revealing x to the server.

In brief, the contributions of this paper are as follows:

- Developing adaptive BNNs which explore the accuracy-runtime trade-off in a one-shot training process and provide accuracy-runtime trade-off at test time.
- Devising a hybrid oblivious BNN inference framework that enables custom cryptographic protocols for BNN layers.
- Introducing a custom protocol for evaluating matrix multiplication in BNNs, which uses favorable characteristics of BNNs to gain efficiency.
- Evaluating our oblivious inference protocol and training scheme on three visual datasets, namely, CIFAR10 as a comparison baseline to prior work, FaceScrub [14, 36] as an example of oblivious face recognition, and Malaria Infection Dataset [1] as an example of private medical diagnosis.

2 SCENARIO AND THREAT MODEL

Figure 2 presents the scenario in oblivious inference. The neural network architecture f is known by both the server and the 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 in privacy-preserving inference, we consider the honest-but-curious scenario [17, 23, 27, 30, 31, 33, 37, 38]. In this threat model, the two parties follow the protocol they

agree upon to compute the output, yet they may try to learn about the other party's data as much as possible. As such, the protocol should guarantee the following security 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.

3 BACKGROUND

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

Scalable neural networks. In [48], an approach for training a single neural network that can infer at a different width was proposed to provide immediate and adaptive accuracy-efficiency trade-offs. The width can be defined as the portion of machine learning models that can be chosen from a predefined width set. For image recognition tasks such as classification, scalable neural networks, also known as slimmable neural networks, offer similar accuracy to traditional neural networks. In [47], an advanced version of scalable neural networks was proposed by using a systematic approach to train Universally Slimmable Networks (US-Nets). US-Nets provides flexible inference at any width, enabling better adaptability of accuracy-efficiency trade-offs.

Secure Function Evaluation Protocol. During oblivious inference, Alice and Bob engage in a **Secure Function Evaluation (SFE)** protocol, 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 computation result can be exposed to both parties, only one of them or neither.

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 $x = [\![x]\!]_A + [\![x]\!]_B$. Standard SFE protocols exist to perform addition and multiplication on secret-shared data such that the result is shared between the two parties. We employ these protocols in oblivious inference to ensure that neither a layer's input nor output is revealed to the involved parties. We refer curious readers to [3] for more details.

Oblivious Transfer (OT) is a protocol between two parties to exchange information in such a way that one party learns nothing about the other party's information. There are several variations of OT, but the most commonly studied are 1-out-of-2 and k-out-of-n OT. One common construction of 1-out-of-2 OT is based on the use of trapdoor functions, which are one-way functions that can be easily inverted by someone who knows a secret key. A sender who has two messages (μ_0, μ_1), and a receiver who has a selection bit $i \in \{0, 1\}$. The receiver obtains the intended message μ_i through OT without revealing the selection bit i to the sender. The receiver can then use the value they selected to decrypt the message they received. The security of this construction is based on the fact that the trapdoor function is computationally hard to invert without the secret key, and the receiver can't learn anything about the other message since they never received it. The receiver does not learn the other message μ_{1-i}.

Additive Secret Sharing works by breaking up a secret value into shares, where each party holds a share of the secret. The shares are then combined using addition to compute the function on the secret value, without any party knowing the value of the secret itself. This technique has the advantage of being simple and efficient, with low overhead and minimal communication cost.

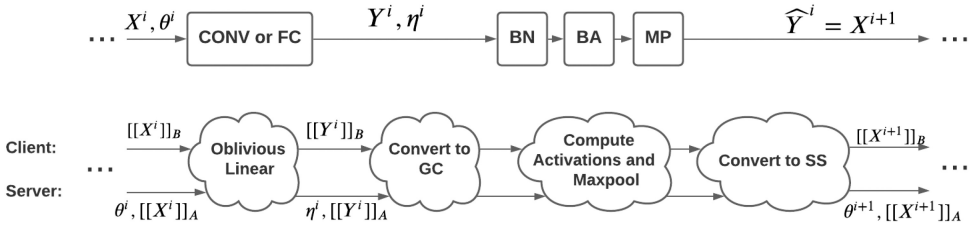


Fig. 3. Illustration of plaintext inference (top) and our proposed equivalent oblivious inference (bottom). We denote linear layers by *CONV* and *FC*, Batch-Normalization by *BN*, Binary Activation by *BA*, and Max-Pooling by *MP*. Here, X^i, Y^i, and θ^i are the linear layer's input, output, and weight/bias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

However, it has the disadvantage that it only works for functions that can be computed using addition. Oblivious Transfer, on the other hand, works by enabling one party to choose one of two inputs to send to another party, without revealing which input was chosen to anyone else. This technique is more flexible than Additive Secret Sharing, as it can be used to compute any function, not just functions that can be computed using addition. However, Oblivious Transfer is more complex and has a higher overhead than Additive Secret Sharing, as it requires multiple rounds of communication between the parties. The type of information that will be leaked for the gain of low overhead depends on the specific implementation of the technique and the security model being used. In general, techniques with lower overhead and communication cost may leak more information about the inputs or the computation being performed, as they may rely on weaker security assumptions or make fewer rounds of communication between the parties.

Also, the use of secure functions such as AS and OT can help maintain the privacy and security of data, but may also introduce some degree of inaccuracy and latency in the computation. The extent of this impact on accuracy depends on the specific implementation and use case of these techniques.

We refer curious readers to [2, 22, 35] for details about OT, its variants, and their implementations.

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 the **oblivious transfer (OT)** and **additive secret sharing (AS)** outlined above. However, AS and OT are not efficient for evaluating nonlinear activations and Max-pooling.

Garbled Circuit (GC), also known as Yao's protocol, is a cryptographic technique used to perform secure computation on private data. GC is the most efficient method of SFE protocol that can be used for the evaluation of an arbitrary function (linear or nonlinear) [25, 26]. The main concept of GC is to describe a function f as a Boolean circuit which processes two private inputs $[\![x]\!]_A$ by Alice and $[\![x]\!]_B$ by Bob, where each gate in the circuit is represented by two garbled tables. The tables correspond to the possible values of the inputs to the gate, and are constructed in such a way that they only reveal the output of the gate when the correct input values are provided. The tables are then sent to the other party, who uses them to evaluate the circuit on their own input. In other words, Alice and Bob jointly compute $y = f(x)$ without learning x or y. Nonetheless, the disadvantage of GC lies in its substantial communication overhead, which pertains to the extensive communication required between the parties involved. GCs are typically used for small computations on small datasets. Therefore, we limit its usage to only nonlinear operations. We refer curious readers to [25, 26, 45, 46] for more details about GC.

4 CRYPTOGRAPHICALLY SECURE BNN INFERENCE

BNNs were initially introduced to minimize plaintext inference's memory footprint and computation overhead. This section provides insights into why BNNs are also helpful for 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, which are very efficient and lightweight cryptographic tools.

In oblivious inference, nonlinear operations are evaluated through heavy cryptographic primitives such as GC, resulting in significant runtime and communication overheads. The high 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 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.

We achieve a significantly faster oblivious inference using this hybrid approach compared to the state-of-the-art [38].

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:

$$W([\![X]\!]_A + [\![X]\!]_B) = W[\![X]\!]_A + W[\![X]\!]_B \tag{1}$$

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

Algorithm 1 presents the secure matrix multiplication protocol for the class of binary weights. The secure matrix multiplication protocol includes the following steps: (1) Alice and Bob construct the Garbled Circuit for binary matrix multiplication using the Yao's Garbled Circuit construction.

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

ALGORITHM 1: Protocol for secure binary matrix multiplication

Input: from Alice $W \in \{-1, +1\}^{M \times N}$
Input: from Alice $[\![X]\!]_A \in \mathbb{Z}^{N \times L}$
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: $[\![Y]\!]_A + [\![Y]\!]_B = W([\![X]\!]_A + [\![X]\!]_B)$
1 Alice locally sets $[\![Y]\!]_A = W[\![X]\!]_A \in \mathbb{Z}^{M \times L}$
2 Bob locally sets $[\![Y]\!]_B = \mathbf{0} \in \mathbb{Z}^{M \times L}$
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 Alice and Bob engage in OT where:
9 • Bob inputs $\{\mu_0, \mu_1\} = \{r \pm [\![X(n,:)]\!]_B\}$
10 • Alice inputs $i = \frac{W(m,n)+1}{2}$
11 • Alice receives μ_i
12 Alice locally updates $[\![y]\!]_A = [\![y]\!]_A + \mu_i$
13 Bob locally updates $[\![y]\!]_B = [\![y]\!]_B - r$
14 Alice locally updates $[\![Y(m,:)]\!]_A = [\![y]\!]_A$
15 Bob locally updates $[\![Y(m,:)]\!]_B = [\![y]\!]_B$

This involves representing the matrix multiplication algorithm as a circuit of binary gates, such as AND and XOR gates. (2) Alice and Bob use the Garbled Circuit to compute the product of their matrices. This is done by evaluating the Garbled Circuit, where each party independently evaluates their own shares of the circuit using their input matrix as input to the circuit. (3) During the evaluation process, Alice and Bob perform **Garbled Circuit OT (GCOT)** to securely obtain the necessary values for the circuit. Specifically, Alice and Bob use GCOT to retrieve the values for the intermediate matrices and XOR the obtained values to obtain the final result. (4) At the end of the evaluation, Alice and Bob each has a share of the result matrix Y. They then perform secure addition of their shares to obtain the final result of Y. (5) Finally, Alice and Bob verify the correctness of the result by comparing their shares of the result with each other. Particularly, Alice first 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 1 (lines 3–15). Explicitly, in line 10, Alice computes i according to the current binary weight value of -1 or +1, then the m-th iteration of the outer loop evaluates the m-th row of the output as:

$$y = [\![y]\!]_A + [\![y]\!]_B = \sum_{n=1}^{N} W(m,n)X(n,:) \tag{2}$$

The inner loop of Algorithm 1 (lines 6–13) computes the above summation by N invocations of OT. 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 $W(m,n)[\![X(n,:)]\!]_B$.

Security. The security of our binary matrix multiplication is guaranteed as follows: first, OT guarantees that Alice's selection bit is not revealed to Bob. Hence, Alice's binary weight remains secret

ALGORITHM 2: Protocol for secure nonlinear 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.

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

to her. Second, before participating in OT, Bob adds a random vector to $[\![X(n,:)]\!]_B$. After OT invocation, Alice receives one and only one of the two messages $\{r + [\![X(n,:)]\!]_B, r - [\![X(n,:)]\!]_B\}$, where r is random. Therefore, Alice cannot learn $[\![X(n,:)]\!]_B$ from the received message. Third, Alice does not communicate $[\![X]\!]_A$ to Bob. Thus, her input share is also kept private. Last, Alice and Bob do not communicate $[\![Y]\!]_A$ and $[\![Y]\!]_B$. Hence, their output shares are kept private.

4.2 Nonlinear Layers

In this section, we outline and leverage the 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 [34]. 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 [34].

Algorithm 2 presents our efficient protocol for oblivious evaluation of nonlinear layers in BNNs, which leverages 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.

Security. GC inherently guarantees the security of our protocol for inference of nonlinear layers. During GC, no information about one party's input is revealed to the other party and vice versa. Hence, $[\![Y]\!]_A$, $[\![Y]\!]_B$, and η are kept private to their owners. After GC, Alice receives $R + \widehat{Y}$. Alice does not know R, so she cannot recover $\widehat{Y}$. Bob knows the random share R but does not know $R + \widehat{Y}$ because GC returns it only to Alice. Therefore, Bob does not know the output either. Therefore, Bob's share $[\![\widehat{Y}]\!]_B = -R$ along with Alice's share $[\![\widehat{Y}]\!]_A = R + \widehat{Y}$ are secure additive shares.

Table 1. Communication Cost for Different Stages of our Oblivious Inference Protocols

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

For matrix multiplication, N, M, L are the dimensions from $W_{M\times N}$ and $X_{N\times L}$, and b is the bitwidth for arithmetic sharing.[2] For batch normalization and binary activation, κ is a security parameter, and its standard value is 128 in the literature. For max-pooling, w is the window size, and $L' \approx \frac{L}{w^2}$ is represented with a different notation than L to account for the lower output resolution. Depending on whether or not max-pooling was applied, the secret sharing cost can be $3\kappa bML'$ or $3\kappa bML'$.

4.3 Communication Cost

Recall that each layer execution is done via SFE protocol, where the two involved parties co-operatively compute output shares of their own. During the protocol, each party may perform computation, storage, or random data generation internally on their device. In privacy-preserving computation, these local processes are deemed *free* operations. In practice, the process's runtime is dominated by the exchange of messages between the two parties, not the internal computations. In our protocols (Algorithms 1 & 2), message exchanges occur during OT or GC invocations. We provide the communication cost of our protocols in Table 1. The communication cost of the protocol can be calculated as the number of bits exchanged between Alice and Bob during the protocol. Specifically, the communication cost can be computed as the sum of the bits sent from Alice to Bob and the bits sent from Bob to Alice.

Plugging in this table's parameters allows one to 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 to the runtime of our protocols.

5 BNN MODEL TRAINING

5.1 Training Slimmable 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 [32], increasing the number of computation bits [16], or introducing new connections and nonlinear layers [5, 29], 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 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 [42].

A related field of research is training dynamic DNNs [28] to provide flexibility at inference time. In this realm, we find Slimmable Networks [48] quite compatible with our problem setting and adapt them to BNNs. Our goal is to train a single network with certain maximum width, say 4×, the base network, so that the model can still deliver acceptable accuracy at lower widths, e.g.,

[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 $\lceil Log(N)+1 \rceil$.

1× or 2× 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 $i \leq n$.

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 $\{\widetilde{Y}_{s_1}, \ldots, \widetilde{Y}_{s_n}\}$ computed by $M_{s_1} \ldots M_{s_n}$. 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 [11].

5.2 Training Universally Slimmable BNN

Inspired by [47], we use the *sandwich rule* and *inplace distillation* [47] to train our universally slimmable BNN.

Sandwich rule. Sandwich rule represents that in every training epoch, we train the model at the smallest width, the largest width, and some random widths between the smallest one and the largest one because [47] indicates that performance at all widths has an upper bound and lower bound, which are the model with the smallest width and the model with the largest width, respectively. Sandwich rule-based training also offers better convergence behavior and better performance compared with slimmable neural networks.

Inplace distillation. The idea of *inplace distillation* is similar to *transfer knowledge* [47]. During the training process, the predicted label of the model at the largest width is used as the training label for the other widths training, while the model with the largest width is trained with the ground truth label. *Inplace distillation* is single-slot compared to other works such as two-step knowledge distillation [21]. One benefit of *inplace distillation* is that it does not need additional computation or memory cost. This method can be applied to many other applications, e.g., deep reinforcement learning and image classification.

Using the *sandwich rule* and *inplace distillation*, we first define the *width range*, for example [×1, ×5], and the number of sampled widths per training iteration n. Compared with slimmable BNN during training, we randomly sample $(n - 2)$ widths as width samples. For each random sample, we execute a sub-network at that *width* and then accumulate gradients by loss calculation. Our universally slimmable BNN training enables scalability for different applications and tasks. GPU memory cost in universally slimmable BNN is maintained at a similar level compared to single BNN training since the same batch size is used. In addition, the universally slimmable BNN training also uses consistent hyper-parameters for all architectures used for training. For detailed algorithms on universally slimmable network training, please refer to [47].

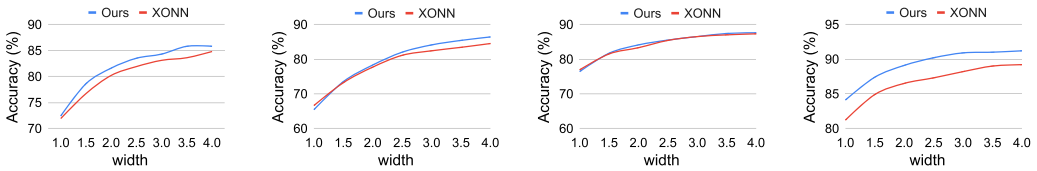
The main challenge of training a scalable DNN would be tuning the hyper-parameters such as leaning rate, number of epochs, batch size, dropout rate, and so on. In general, a single neural network is trained with an optimal set of hyperparameters, but finding the set of hyperparameters that can be applied to networks of different widths by training them only once can be challenging.

6 EVALUATIONS

Standard Benchmarks. We perform our evaluation on several networks trained on the CIFAR-10 dataset, shown in Table 2. These benchmarks provide a rich set of comparison baselines as they are commonly used in prior work. Specifically, the BC1 network has been evaluated by most oblivious inference papers [8, 23, 27, 30, 31, 33, 37–39]. Other models are evaluated by XONN [38],

Table 2. Summary of the Trained Binary Network Architectures Evaluated on the CIFAR-10 Dataset

Arch.	Previous Papers	Description
BC1	[27], [39], [8], [23], [38], [33], [37], [30], [31]	7 CONV, 2 MP, 1 FC
BC2	[38]	9 CONV, 3 MP, 1 FC
BC3	[38]	9 CONV, 3 MP, 1 FC
BC4	[38]	11 CONV, 3 MP, 1 FC



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

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

the state-of-the-art for oblivious inference of *binary* networks. We omit details about layer-wise configurations for brevity and refer curious readers to [38] for further information.

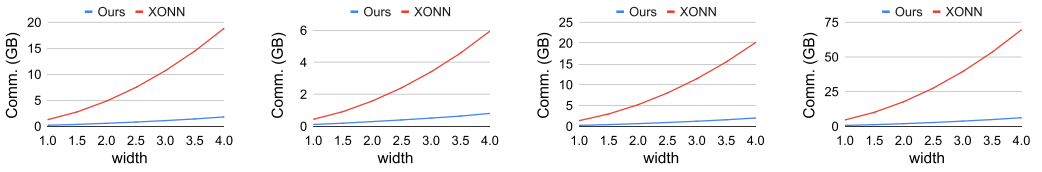
Training. For all benchmarks, we use the standard backpropagation algorithm proposed by [11] to train our binary networks. We split the CIFAR10 dataset into 45k training examples, 5k validation examples, and 10k testing examples and train each architecture for 300 epochs. We use the Adam optimizer with an 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 experiments. 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++. We use the standard emp-toolkit [44] library to implement OT and GC. To run oblivious inference, we translate the model description and trained parameters from Pytorch to the equivalent description in our C++ library. We run our oblivious inference code for measurements on a computer with a 2.2 GHz Intel Xeon CPU and 16 GB RAM.

6.1 Evaluating Flexible BNNs

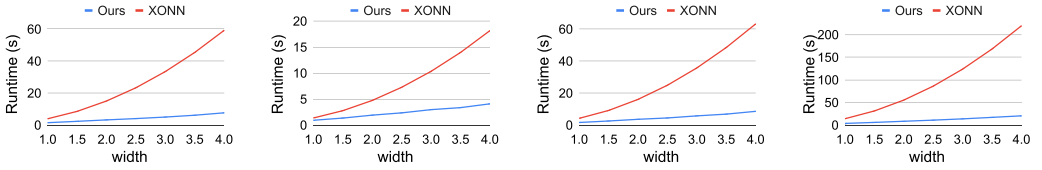
Let us start by evaluating our adaptive BNN training. We train slimmable networks with maximum 4× width of the base models presented in Table 2. During training, we re-iterate through subsets of widths $\{1\times, 1.5\times, \ldots, 4\times\}$ and perform gradient updates as explained in Section 5.

Figure 4 presents the test accuracy of each network at different widths. We also report the accuracy of independently 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. In particular, the BC4 architecture is more complex than other architectures (BC1-BC3) considered in our experiments. Since our framework uses slimmable neural networks which can adaptively choose the width of the model to optimize the accuracy-efficiency trade-offs, the effect of run-time width optimization is more pronounced



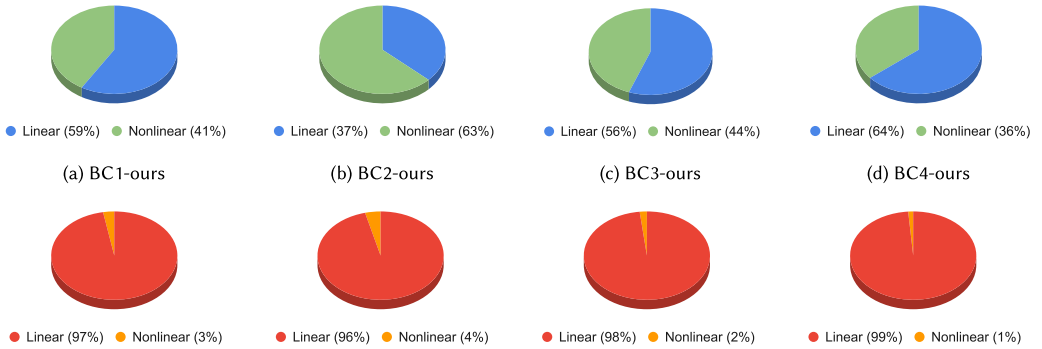
(a) BC1 (b) BC2 (c) BC3 (d) BC4

Fig. 5. Communication cost of each architecture at different widths.



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

Fig. 6. The runtime of each architecture at different widths.



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

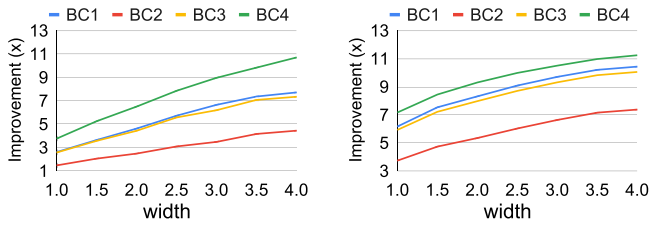
(e) BC1-XONN (f) BC2-XONN (g) BC3-XONN (h) BC4-XONN

Fig. 7. The proportions of linear and nonlinear operation of each architecture.

with a more complex architecture (e.g., neural network with large width value). Consequently, our proposed framework shows improved accuracy with more complex architectures such as BC4. Once the adaptive network is trained, the server can provide oblivious inference service to clients, which we discuss in the following section.

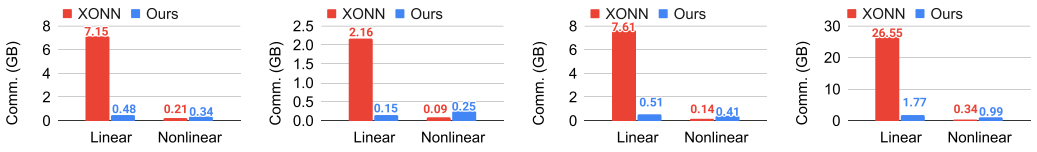
6.2 Oblivious Inference

Recall that the runtime of oblivious inference is dominated by data exchange between the client and server. We compare our custom protocol's communication cost and runtime with XONN's GC implementation in Figures 5 and 6. In Figure 5, the horizontal axes present the network width. The left vertical axes show the communication (in Giga-Bytes). While in Figure 6, the left vertical axes indicate the runtime (in seconds), and the horizontal axis in each figure represents the network width. Both Figures 5 and 6 show that for all the benchmarks, the runtime and communication of our method are significantly smaller than XONN. The communication cost of GC is proportional to the size of the circuit and the number of gates in the circuit. For each gate in the circuit, two garbled tables need to be exchanged, resulting in a high communication cost. On the other



(a) Runtime (b) Communication

Fig. 8. Improvements in LAN runtime and communication compared to XONN. Our protocols achieve 2× to 11× in runtime and 4× to 11× communication reduction.



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

Fig. 9. Breakdown of communication cost at linear and nonlinear layers. Our protocol significantly reduces XONN's GC-based linear layer cost, with a slight increase in nonlinear layer cost.

hand, SFE involves computing a function on two private inputs by exchanging messages between the parties, where each message reveals partial information about the output of the function. The communication cost of SFE is proportional to the number of messages exchanged between the parties, which is typically much lower than the number of gates in a circuit. Therefore, GC has a higher communication cost than SFE as it involves exchanging a large number of garbled tables, which are typically much larger than the messages exchanged in SFE. The increased network width results in higher communication and runtime, which is the cost for higher inference accuracy.

Figure 7 represents the proportions of linear and nonlinear operations of each architecture. In Figure 7, (a)-(d) represent the proportions of linear and nonlinear operations of our proposed architecture, and (e)-(h) indicate the proportions of the same operations of XONN architecture. Nonlinear operations take most of the operations in most of our architectures, while XONN architectures constantly have over 96% linear operations, which explains the runtime improvements offered by the proposed approach.

Figure 8 summarizes the performance boost achieved by our protocols, i.e., 2× to 11× lower runtime and 4× to 11× lower communication compared to XONN. The enhancement is more significant at higher widths, which shows the scalability of our method. To illustrate the reason behind our protocol's better performance, we show the breakdown of all architecture's communication costs in Figure 9. For the XONN protocol, most of the cost is from linear operations, which we reduce from 7.15GB to 0.21GB, 2.16GB to 0.15GB, 7.61GB to 0.51GB, 26.55GB to 1.77GB for BC1, BC2, BC3, and BC4, respectively. In nonlinear layers, our cost is slightly more than XONN's, i.e., 0.25GB versus 0.09GB for BC2, due to the extra conversion cost between AS and GC.

Overall, the total communication is reduced from 7.36GB to 0.81GB, from 2.25GB to 0.4GB, from 7.75GB to 0.92GB, from 26.89GB to 2.76GB for BC1, BC2, BC3 and BC4, respectively 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

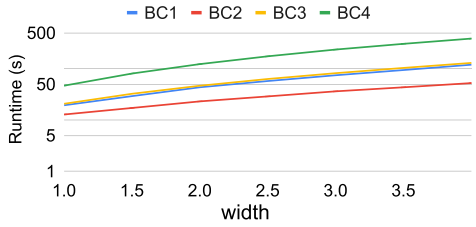


Fig. 10. Runtime in WAN setting with ∼20 MBps bandwidth and ∼50 ms network delay.

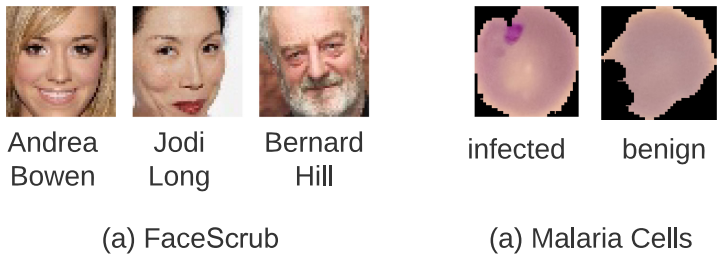
the performance of our method to the best-performing earlier work on this benchmark. The vertical and horizontal axes in the figure represent test accuracy and runtime. Hence, points to the top-left corner are more desirable. Our method achieves a better accuracy/runtime tradeoff than all contemporary work while providing flexibility. Compared to CryptFlow2 (the most recent oblivious inference framework at the time of this paper), our method achieves ∼2× faster inference at the same accuracy.

Evaluation in Wide Area Network (WAN). So far, we reported our runtimes for the setting where the 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 ∼20MBps, and the delay is ∼50ms. The bandwidth mentioned above and delay correspond to the connection speed between two AWS instances in "US-West-LA-1a" and "US-East-2a". Runtimes are reported in Figure 10, showing varying inference times 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, many applications exist 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 of 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 [43] regulation, and malicious hackers can use facial features to authenticate into the client's 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 [14, 36] and Malaria Cell Infection [1] as representatives for face authentication and medical diagnosis, respectively.

Figure 11 shows example samples from each dataset. The tasks are to identify 530 different actors in FaceScrub and classify infected cells from benign ones in Malaria. We download ∼57,000 images from the links provided by FaceScrub authors, of which we use 45,000 for training, 6,000 for validation, and 6,000 for testing. The Malaria dataset is split into ∼24,800 samples for training, ∼1,300 for evaluation, and ∼1,300 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 70.8% inference accuracy on FaceScrub and 94.7% accuracy 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 the network architecture can be selected more carefully in a commercial application, and more training data can be collected to achieve better accuracy and runtime.



(a) FaceScrub (a) Malaria Cells

Fig. 11. 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.

Table 3. Example BNNs Trained for Face Recognition and Medical Application

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

We use the *BC2* architecture at width 3 and 1 for FaceScrub and Malaria, respectively. Runtimes are measured in the WAN setting.

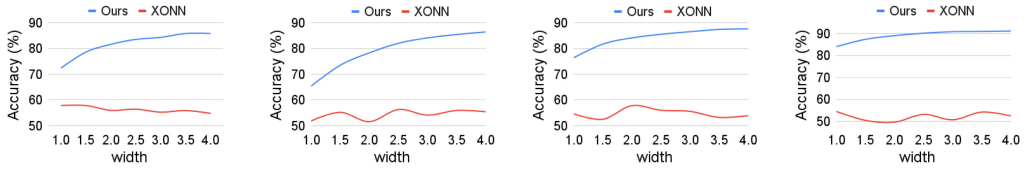
6.4 Evaluation on Large Dataset

In this section, we study the application of oblivious inference on a relatively large dataset: the STL-10 dataset [10]. The STL-10 dataset is an image recognition dataset for developing unsupervised feature learning, deep learning, and self-taught learning algorithms. The dataset contains 10 classes, including airplane, bird, car, cat, deer, dog, horse, monkey, ship, truck, and has 100k unlabeled images from the ImageNet dataset. Figure 12 shows the accuracy of architectures BC1 to BC4 computed with different width values for STL-10 dataset using the proposed approach and XONN. The experimental results show that our approach consistently outperforms the state-of-the-art technique in a challenging large data set with images from multiple classes and complex backgrounds.

7 RELATED WORK

The oblivious inference was conceptually practical for small-sized neural networks in CryptoNets [18]. Using CryptoNets, an inference on MNIST data would take ~ 300 seconds, which motivated researchers to invest in the field. Since then, many more efficient protocols for oblivious inference have been proposed [4, 6, 7, 9, 12, 20, 23, 27, 31, 37, 41]. These works optimize security primitives for oblivious inference without significantly modifying the model.

The 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 [18, 30, 33], using dimensionality reduction at the input layer [40], and neural architecture search [17]. Concurrently, researchers in the ML community have devised DNN optimization techniques such as pruning [19], quantization [49], tensor factorization [24], and binary neural networks [11]. Among the above, BNNs are especially compelling candidates for oblivious inference since they translate linear arithmetic to bitwise operations. XONN [38] was the first work to notice the particular use case of binary networks for cryptographically secure inference using GC [46], noting that `XNOR` operations that frequently appear in BNNs can be evaluated for free in GC.



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

Fig. 12. The accuracy of each architecture at different widths for STL-10 dataset. Our Adaptive BNN trains a single network that can operate at all widths, whereas previous work (XONN) trains a separate BNN per width.

Despite improving oblivious inference time, XONN does not completely utilize the complete 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× 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 [48]. We provide accurate and efficient BNN benchmarks for the 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 [5], residual activation binarization [16], and PReLU nonlinearity [29] are interesting future directions for oblivious BNN inference. Our current oblivious inference implementation does not support these operations. However, since we use GC for nonlinear operations and GC can implement arbitrary functionalities, the techniques mentioned earlier can be integrated and tested in future work, which may or may not result in the 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 model's parameters. 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 [38], which uses GC to evaluate linear and nonlinear layers, we use GC only for nonlinear layers. We present a custom protocol for linear layers using OT and AS, which leads to 2× to 11× 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] (n.d.). Malaria Cell Images, accessed on 01/20/2019. https://www.kaggle.com/iarunava/cell-images-for-detecting-malaria.

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

[3] Mikhail Atallah, Marina Bykova, Jiangtao Li, Keith Frikken, and Mercan Topkara. 2004. Private collaborative forecasting and benchmarking. In *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society*. 103–114.

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

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

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

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

[8] Nishanth Chandran, Divya Gupta, Aseem Rastogi, Rahul Sharma, and Shardul Tripathi. 2017. EzPC: Programmable, efficient, and scalable secure two-party computation for machine learning. *ePrint Report* 1109 (2017).

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

[10] Adam Coates, Andrew Ng, and Honglak Lee. 2011. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. JMLR Workshop and Conference Proceedings, 215–223.

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

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

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

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

[15] Jonathan Frankle and Michael Carbin. 2018. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635* (2018).

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

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

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

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

[20] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. 2017. CryptoDL: Deep neural networks over encrypted data. *arXiv preprint arXiv:1711.05189* (2017).

[21] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* 2, 7 (2015).

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

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

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

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

[26] Y. Lindell and B. Pinkas. 2004. A proof of Yao's protocol for secure two-party computation. ECCC Report TR04-063. In *Electronic Colloquium on Computational Complexity (ECCC)*.

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

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

[29] Zechun Liu, Zhiqiang Shen, Marios Savvides, and Kwang-Ting Cheng. 2020. ReActNet: Towards precise binary neural network with generalized activation functions. In *European Conference on Computer Vision*. Springer, 143–159.
[30] Qian Lou, Yilin Shen, Hongxia Jin, and Lei Jiang. 2021. {SAFEN}et: A secure, accurate and fast neural network inference. In *International Conference on Learning Representations*.
[31] Qian Lou, Bian Song, and Lei Jiang. 2020. AutoPrivacy: Automated layer-wise parameter selection for secure neural network inference. In *Advances in Neural Information Processing Systems*.
[32] Asit Mishra, Eriko Nurvitadhi, Jeffrey J. Cook, and Debbie Marr. 2017. WRPN: Wide reduced-precision networks. *arXiv preprint arXiv:1709.01134* (2017).
[33] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. 2020. DELPHI: A cryptographic inference service for neural networks. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*.
[34] Benjamin Mood, Debayan Gupta, Henry Carter, Kevin Butler, and Patrick Traynor. 2016. Frigate: A validated, extensible, and efficient compiler and interpreter for secure computation. In *EuroS&P*. IEEE, 112–127.
[35] Moni Naor and Benny Pinkas. 2005. Computationally secure oblivious transfer. *Journal of Cryptology* 18, 1 (2005).
[36] Hong-Wei Ng and Stefan Winkler. 2014. A data-driven approach to cleaning large face datasets. In *IEEE International Conference on Image Processing*.
[37] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. 2020. CrypTFlow2: Practical 2-party secure inference. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 325–342.
[38] M. Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin E. Lauter, and Farinaz Koushanfar. 2019. XONN: XNOR-based oblivious deep neural network inference. In *USENIX Security*.
[39] M. Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M. Songhori, Thomas Schneider, and Farinaz Koushanfar. 2018. Chameleon: A hybrid secure computation framework for machine learning applications. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*. 707–721.
[40] Bita Darvish Rouhani, M. Sadegh Riazi, and Farinaz Koushanfar. 2017. DeepSecure: Scalable provably-secure deep learning. *arXiv preprint arXiv:1705.08963* (2017).
[41] Amartya Sanyal, Matt Kusner, Adria Gascon, and Varun Kanade. 2018. TAPAS: Tricks to accelerate (encrypted) prediction as a service. In *International Conference on Machine Learning*. PMLR, 4490–4499.
[42] Wei Tang, Gang Hua, and Liang Wang. 2017. How to train a compact binary neural network with high accuracy?. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 31.
[43] The HIPAA Privacy Rule. (n.d.). https://www.hhs.gov/hipaa/for-professionals/privacy/index.html.
[44] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. 2016. EMP-toolkit: Efficient MultiParty computation toolkit. https://github.com/emp-toolkit. (2016).
[45] Sophia Yakoubov. 2017. A gentle introduction to Yao's Garbled Circuits. (2017).
[46] Andrew Yao. 1986. How to generate and exchange secrets. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*.
[47] Jiahui Yu and Thomas S. Huang. 2019. Universally slimmable networks and improved training techniques. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 1803–1811.
[48] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. 2018. Slimmable neural networks. *arXiv preprint arXiv:1812.08928* (2018).
[49] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. 2016. DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160* (2016).

Received 31 October 2022; revised 28 April 2023; accepted 25 June 2023