MORSE-STF: A Privacy Preserving Computation System

Qizhi Zhang, Yuan Zhao, Lichun Li, JiaoFu Zhang, Qichao Zhang, Yashun Zhou, Dong Yin, Sijun Tan, Shan Yin
{qizhi.zqz, caesar.zy, lichun.llc, zhangjiaofu.jf, qichao.zqc, yashun.zys, yindong.yd, tansijun.tsj, yinshan.yys}@antgroup.com

Abstract—Privacy-preserving machine learning has become a popular area of research due to the increasing concern over data privacy. One way to achieve privacy-preserving machine learning is to use secure multi-party computation, where multiple distrusting parties can perform computations on data without revealing the data itself. We present Secure-TF, a privacy-preserving machine learning framework based on MPC. Our framework is able to support widely-used machine learning models such as logistic regression, fully-connected neural network, and convolutional neural network. We propose novel cryptographic protocols that have lower round complexity and less communication for computing sigmoid, ReLU, conv2D and there derivatives. All are central building blocks for modern machine learning models. With our more efficient protocols, our system is able to outperform previous state-of-the-art privacy-preserving machine learning framework in the WAN setting.

I. INTRODUCTION

Machine learning applications are "data-hungry", as more data usually leads to better concrete performance of machine learning models. The growing amount of data required to train a machine learning model also raises significant privacy concerns. For example, multiple organizations may want to jointly train a machine learning models by aggregating their data, but want to keep their private data to the other parties for privacy reasons. Secure multi-party computation (MPC) is a cryptographic primitives that can achieve the above objective. In secure MPC, parties can jointly obtain the output of the models without revealing their input data to the other parties. Recently, there has been a active line of research in combining machine learning with MPC to achieve privacy-preserving machine learning (PPML). For examples, SecureML [MZ17], SecureNN [WGC19], ABY3 [MR18], ABY 2.0 [PSSY21], CrypTFlow [KKC+20], CrypTFlow 2 [RRK+20], TFE [DMD+18], Rosetta [CHS+20], CrypTen [KVV+20] and Falcon [WBT+21]. These prior work has made significant performance improvement of the MPC protocols for machine learning operations. However, many of these prior work achieve desirable concrete performances in the LAN network setting, which has high network bandwidth (∼1.2GB/s) and low network delay (∼2ms). However, such network conditions are rarely available in practice. In practical scenarios, organizations usually have their servers located in different regions, and the network condition is restricted to the WAN setting with low bandwidth (∼100Mbps) and high delay (∼10-60ms). In the WAN setting, communication between parties is the major performance bottleneck. We therefore need to develop more efficient MPC protocols with lower communication and fewer communication rounds in order to address the performance bottleneck.

We present our system: Secure Tensorflow (STF), a MPC based privacy-preserving machine learning framework that achieves state of the art performance in the WAN network setting. Central to our performance improvement is the design of more efficient protocols for the convolution, relu, and sigmoid operations that are essential building blocks of today’s machine learning models.

Our contribution. The main contribution of this work can be summarized as follows:

- We abstract out the computation of linear operations (e.g. scalar multiplication, matrix multiplication, convolution) as computing a bilinear map, include scale multiplication, matrix multiplication, and convolution. We also show that the back propagation of convolution operations is bilinear, and design protocols that are 57.4% more communication efficient than prior state of the art (Table III.1).
- We propose a novel protocol to compute the sigmoid function. We approximate sigmoid using Fourier Series. Compared to the popular method that approximate sigmoid using the Chebyshev polynomial, our protocol are 80% and 52% more efficient in terms of communication and round complexity (Table A.4). Our protocol is also more efficient than methods based on piecewise linear function and Newton-Raphson approximation.
- We present novel protocols for most significantly bit (MSB) extraction, which is the major building block for private comparison and computing non-linear operation such as ReLU. Based on our protocols, we implement three different protocols for ReLU with round complexity from \(O(1)\), \(O(\log n)\), \(O(n)\) and communication complexity that decrease progressively. Each of these protocols are suited for a particular network setting in terms of network bandwidth and delay, which we investigate in our experiments. (Table IV.3)
- We build our system on top of Tensorflow, a widely-used machine learning framework. We show that our system is better than TFE [DMD+18], Rosetta [CHS+20], SecureNN [WGC19] and CrypTen [KVV+20] in terms of training speed and the stability using numeric experiments. (Table IV.3 and Table IV.2).

Our STF support the train and inference of logistic regression (LR), fully-connected neural network (FC), and convolutional...
neural network (CNN). We compare the function of our STF and other popular tool of PPML in the left 5 columns in the Table [1].

We compare our protocols for sigmoid, DReLU, Conv2D and the back propagation of Conv2D to other popular tool of PPML in the right 4 columns in the Table [1].

II. FRAMEWORK OF STF

The framework of STF is like the following figure 2. In the bottom of the framework, the model Random is builded. We realize a secure deterministic random bit generator (DRBG), which conforms to the CTR_DRBG standardized in NIST Special Publication 800-90A. We use AES-NI to implement AES and generate multiple random number streams simultaneously in one DRBG to fill up the pipelines of the AES-NI execution units.

Under the Random model, three basic class is constructed, i.e., SharedTensorBase, PrivateTensorBase and SharedPairBase.

The model Basic realized three basic class i.e. SharedTensor, PrivateTensor and SharedPair and the operators for these classes.

The class SharedTensor means the class of the Tensors over \( \mathbb{Z}_n \), where \( n \) is a number of type int64. The class PrivateTensor means the class of the Tensors which is private data belong to owner, over \( \mathbb{Z}_n \) or fixed-point numbers with a public fixed point. The class SharedPair means the class of the Tensors which is hold as additive secret shares on workerL and workerR, over \( \mathbb{Z}_n \) or fixed-point numbers with a public fixed point.

We first realize the class SharedTensorBase, PrivateTensorBase, SharedPairBase. The main differential between the classes with -Base and classes without -Base is that the classes with -Base reload the operators +, -, *, >, \( \geq \), <, \( \leq \) and etc.. The class SharedTensor and SharedPairBase with reloading the operators +, -, *, >, \( \geq \), <, \( \leq \) and etc..

The model ml is constructed based on the model Basic. There two sub-model of ml, namely, LogisticRegression and nn. The model LogisticRegression realized logistic regression model in two case: a. the features lie in one Party. and b. the features lie in two Parties.

The sub-model nn realize Neural Network framework based on the model Basic. There two sub-model of nn, namely, the model Layer and the model Network in the model nn.

The model Layer realize the layers Input, Dense, ReLU, CrossEntropyWithSigmoid, CrossEntropyWithSoftmax, Conv2D, AveragePooling2D, and etc. For every Layer, forward and back propagation is realized.

The model Network realize a framework of Neural Network. It allow user construct a Neural Network by "add Layers".

III. PROTOCOLS

In order to realize the layers Dense, ReLU, CrossEntropyWithSigmoid, CrossEntropyWithSoftmax, Conv2D security, we use some new protocols. We introduce the main protocols using in STF as follows:

A. Bilinear map

Let \( A, B, C \) be there Abelian group, a map \( f : A \times B \rightarrow C \) is called bilinear if

\[
\begin{align*}
f(a_1 + ka_2, b) &= kf(a_1, b) + f(a_2, b) \\
f(a, b_1 + kb_2) &= f(a, b_1) + kf(a, b_2)
\end{align*}
\]

for any \( a_1, a_2 \in A, b_1, b_2 \in B \) and \( k \in \mathbb{Z} \). (In fact, it is equivalent to

\[
\begin{align*}
f(a_1 + a_2, b) &= f(a_1, b) + f(a_2, b) \\
f(a, b_1 + b_2) &= f(a, b_1) + f(a, b_2)
\end{align*}
\]

for any \( a_1, a_2 \in A, b_1, b_2 \in B \).)

We give some examples for bilinear map:
Example III.1. Let $A := \mathbb{Z}_N$, $A^n$ is the Abelian group consists of all the arrays of $n$ dimension on $A$. Then the multiplication operator
\[
A^n \times A^n \rightarrow A^n
\]
\[
(a^i)_{i=1}^n , (b^i)_{i=1}^n \mapsto (a_i b_i)_{i=1}^n
\]
is a bilinear map.

Example III.2. Let $A := \mathbb{Z}_N$, $M_{m,n}(A), M_{n,p}(A)$ be the Abelian groups consists of all the $m \times n$ matrices, $n \times p$ matrices and $m \times p$ matrices on $A$ respectively. Then the matrix multiplication operator
\[
M_{m,n}(A) \times M_{n,p}(A) \rightarrow M_{m,p}(A)
\]
\[
P, Q \mapsto PQ
\]
is a bilinear map.

Example III.3. Let $A := \mathbb{Z}_N$. Let $M_{m,n}(A), M_{r,s}(A), M_{m',n'}(A)$ be the Abelian groups consists of all the $m \times n$ matrices, $r \times s$ matrices, $m' \times n'$ matrices on $A$. Then the two-dimension convolution operator
\[
\text{Conv2D} : M_{m,n}(A) \times M_{r,s}(A) \rightarrow M_{m',n'}(A)
\]
\[
\text{input} \quad \text{filter} \mapsto \text{output}
\]
is a bilinear map, where
\[
\text{output}_{i,j} = \sum_{\delta_i, \delta_j} \text{input}_{s_1+i+\delta_i, s_2+j+\delta_j} \ast \text{filter}_{\delta_i, \delta_j}
\]
$s_1$ and $s_2$ are the strides.

Example III.4. Let $A := \mathbb{Z}_N$. Let $\otimes_{B,m,n,C} A \times \otimes_{C,r,s,D} A \rightarrow \otimes_{B,m',n',D} A$ be the Abelian groups consists of all the 4-dimension tensors of shape $[B, m, n, C], [C, r, s, D], [B, m', n', D]$ over $A$. Then the two-dimension convolution operator $\text{Conv2D}$
\[
\otimes_{B,m,n,C} A \times \otimes_{C,r,s,D} A \rightarrow \otimes_{B,m',n',D} A
\]
\[
\text{input} \quad \text{filter} \mapsto \text{output}
\]
is a bilinear map, where
\[
\text{output}_{b,i,j,k} = \sum_{\delta_i, \delta_j, \delta_k} \text{input}_{b, s_1+i+\delta_i, s_2+j+\delta_j, q} \ast \text{filter}_{\delta_i, \delta_j, q, k}
\]
$s_1$ and $s_2$ are the strides.
For any fixed \( i \), we have
\[
\frac{\partial \text{loss}}{\partial x_i} = \sum_k \frac{\partial \text{loss}}{\partial z_k} \frac{\partial z_k}{\partial x_i} = \sum_k \sum_j p_{k,j} \frac{\partial \text{loss}}{\partial z_k} y_j
\]
where
\[
p_{k,j} = a_{ij}^k
\]
Hence \( \frac{\partial \text{loss}}{\partial x_i} \) is bilinear of \( (\frac{\partial \text{loss}}{\partial z_k})_k \) and \( y \).

It is similar to the proof of (a).

According to the Theorem \( \text{III.5} \) under the symbols defined in Example \( \text{III.2} \) the back propagation to input
\[
\text{bi} : \otimes_{B,m',n',D} A \times \otimes_{C,r,s,D} A \rightarrow \otimes_{B,m,n,C} A
\]
and the back propagation to filter
\[
\text{bf} : \otimes_{B,m',n',D} A \times \otimes_{B,m,n,C} A \rightarrow \otimes_{C,r,s,D} A
\]
are bilinear. Therefore we have our protocol \( \Pi_{\text{Conv2D}} \) for compute the back propagation to input and the protocol \( \Pi_{\text{Conv2Df}} \) for compute the back propagation to filter in Algorithm \( \text{6} \) and Algorithm \( \text{7} \).

We compare the communication of our protocols \( \Pi_{\text{Conv2D}}, \Pi_{\text{Conv2Df}}, \Pi_{\text{Conv2Dif}} \) to other workes in Table \( \text{III.1} \).

### C. Sigmoid

In the early work for training a Logistic Regression using MPC, the sigmoid is approximated by a polynomial. A most common approximate is the Chebyshev polynomial of degree 9, i.e.
\[
\text{sigmoid}(x) \approx a_0 + a_1 x + a_3 x^3 + a_5 x^5 + a_7 x^7 + a_9 x^9
\]
where
\[
\begin{align*}
a_0 &= 0.5, \\
a_1 &= 0.2159198015, \\
a_3 &= -0.0082176259, \\
a_5 &= 0.0001825597, \\
a_7 &= -0.0000018848, \\
a_9 &= 0.000000072
\end{align*}
\]

We show the graph of the Chebyshev polynomial approximated to sigmoid in Figure 3 and 4. It easy to see that, the Chebyshev polynomial is closed to sigmoid in the domain \((-8, 8)\), but is grow up quickly in other domain, that lead to exploding gradient.

When we train a Logistic Regression model, we use the following trick genrally:

a. To use normalized features, i.e the features of samples is normalized s.t average value is 0 and standard deviation is 1.

b. Initialize the coefficient of LR model using i.i.d Gauss distribution with \( \mu = 0, \sigma = \frac{1}{\sqrt{N}} \), where \( N \) is the number of columns in the table of features.

The probability of \( \sum w_i x_i > 3 \) is very low. But once the there is a singular sample such that \( \sum w_i x_i > 8 \), it easy to lead to exploding gradient.

In ABY3 [MR18], ABY 2.0 [PSSY21], SecureML [MZ17], the sigmoid is approximated by an piecewise linear function.

\[
\text{sigmoid}(x) \approx \begin{cases} 
0, & x < -1/2 \\
x + 1/2, & -1/2 \leq x < 1/2 \\
1, & x \geq 1/2
\end{cases}
\]

But it require to run the privacy compare protocols that lead to a big communication and round complexity.

In our work, we use the Fourier Series to approximate the sigmoid.

\[
\text{sigmoid}(x) \approx a_0 + a_1 \sin \frac{2\pi x}{32} + a_2 \sin \frac{4\pi x}{32} + a_3 \sin \frac{6\pi x}{32} + a_4 \sin \frac{8\pi x}{32} + a_5 \sin \frac{10\pi x}{32}
\]

where
\[
\begin{align*}
a_0 &= 0.5, \\
a_1 &= 0.61727893, \\
a_2 &= -0.02416704, \\
a_3 &= 0.16933091, \\
a_4 &= -0.04526986, \\
a_5 &= 0.08159136
\end{align*}
\]

It is closed to sigmoid in the domain \((-8, 8)\) basically, and lead to exploding gradient never. (See Figure 5). Its complexity of communication and round is low also.

For a positive integral number \( m < n \), in order to compute \( \sin \frac{2k\pi x}{2^n} \) for \( x \in \mathbb{Z}/2^n \mathbb{Z} \) using MPC, we use the principle
where $k$ can be integral scale or vector. The protocol is described in Algorithm 8.

**Complexity of Communication.**

The offline communication of protocol $\Pi_{Sin}^K$ is $(4K + 2)n$ bits. The communication can be further reduced to $2Km$ bits using PRF. The online communication of protocol $\Pi_{Sin}^K$ is $2n$ bits, and in one round. Hence, the total communication of protocol $2(K + 1)$ bits.

In order to compute Sigmoid, we just need to compute the inner value of $\sin \frac{2k\pi x}{2m}$ and $(a_1, \cdots, a_K)$ and plus 0.5. Hence its complexity of communication is same as $\Pi_{Sin}^K$.

We give the complexity of communication and round of $\Pi_{Sin}^K$ and $\Pi_{Sigmoid}$ in Table A.4.

**D. MSB (Most Significant Bit)**

In STF, there are two protocols to compute MSB. They are the MSB protocol $\Pi_{MSB}^{linear}$ for linear round and the MSB protocol of logarithm round $\Pi_{MSB}^{log}$.

Let $c_i$ is the carry from the $i - 1$ bit to the $i$ bit, then we have

1. $c_0 = 0$,
2. $c_{i+1} = (x_i^L + x_i^R)c_i + x_i^L x_i^R$ for $i = 0, 2, \cdots n - 2$, where the arithmetic operators are the operators in finite field (Galois field) $F_2$.
3. $MSB = x_{n-1}^L + x_{n-1}^R + c_{n-1}$.

We can compute MSB through computing $\{c_i\}_{i=1}^{n-1}$ step by step. In order to decrease the communication, we rewrite the property a as

$$b_i = (x_i^L + c_i)(x_i^R + c_i) - c_i$$

The protocol is described in Algorithm 9.

**Complexity of Communication and round of $\Pi_{MSB}^{linear}$**

There is only step 2 need communication. When $i = 1$, Step 2 is a half multiplication, which communication is 2 bits in 1 round in online phase and 1 bit in offline phase. When $i = 2, \cdots n - 1$, Step 2 is a full multiplication, which communication is 4 bits in 1 round in online phase and 1 bit in offline phase. Hence the communication of Protocol 9 is $2 + 4(n - 2) = 4n - 6$ bits in $n - 2$ rounds in online phase and $n - 1$ bits in offline phase.

Moreover, we can rewrite the formula b. as the following format:

$$P_i = \begin{pmatrix} c_{i+1} & \vdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{pmatrix} = \begin{pmatrix} (x_i^L + x_i^R) & x_i^L x_i^R \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{pmatrix} P_{n-2} \cdots P_0 \begin{pmatrix} c_0 \\ \vdots \\ 1 \end{pmatrix}$$

Therefore we have

$$P_i = \begin{pmatrix} (c_{i+1}) & \vdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{pmatrix} = \begin{pmatrix} (x_i^L) & x_i^L x_i^R \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{pmatrix}$$

We can compute $P_{n-2} \cdots P_0$ along a binary tree as follows:

$$Q_0 \xrightarrow{\text{null}} Q_{10} \xrightarrow{Q_{10}} Q_{01} \xrightarrow{Q_{01}} Q_{00}$$

$$Q = \begin{cases} P_{int(s)} & \text{if } \text{len}(s) < n - 1 \\ I_2 & \text{otherwise} \end{cases}$$

where $I_2$ is the identity matrix of size 2.

For example, in the case $s = 8$, $d = 3$ we have $\text{int}(110) = 6, \text{int}(111) = 7$ and $Q_{110} = P_6, Q_{111} = I_2$.

For a binary string $s$ of length $d(s) < d$ (include the case $s = \text{null}$ where $d(s) = 0$), let $s0$ be the binary string of length $d(s) + 1$ consists of $s$ and $0$, $s1$ be the binary string of length $d(s) + 1$ consists of $s$ and 1.

we define $Q_s$ recursively as follows:

$$Q_s := \begin{cases} P_{int(s)} & \text{if } \text{len}(s) < n - 1 \\ Q_{s1}Q_{s0} & \text{if } \text{len}(s) = d \end{cases}$$

Then we can compute $Q$ recursively using protocol $\Pi_{MSB}^{log}$ described in Algorithm 10.

\[\text{TABLE III.1}\]
\[\text{COMPARE TO RECENT WORKS}\]
Complexity of Communication and round of $\Pi^{\log}_{MSB}$

For any binary string $s$ of length $d(s) = s$, in order to compute $Q_s = P_{\text{int}}(s)$, it just need to compute $x_i^L + x_i^R$ and $x_i^L - x_i^R$ is need not communication, $x_i^L - x_i^R$ is a half multiplication, that need 1 bit in offline and 2 bits in one online round. Hence the total communication to compute $Q_s$ for $d(s) = d$ is $n-1$ bits in offline and $2(n-1)$ bits in one online round.

For any binary string $s$ of length $d(s) < s$, in order to compute $Q_s = Q_{s1} Q_{s0}$, we need not do anything for the case $Q_{s1} = 1$ or $Q_{s0} = 1$. In the case $Q_{s1} \neq 1$ and $Q_{s0} \neq 1$, we use the principle that

$$Q_s = \begin{pmatrix} a_1 & a_0 \\ b_1 & b_0 \end{pmatrix}$$

for $Q_{s1} = \begin{pmatrix} a_1 \\ b_1 \end{pmatrix}$, $Q_{s0} = \begin{pmatrix} a_0 \\ b_0 \end{pmatrix}$.

We need just compute the first line of $Q_s$, which equal to $a_1 (a_0, b_0) + (0, b_1)$. Hence the communication is 2 bits in offline phase, and 6 bits in 1 online round. Hence the total communication to compute all the $Q_s$ for $d(s) = d' < d$ less than $2^d' - 2$ in offline phase and $2^d' - 6$ in 1 online round.

Hence the total communication of protocol $\Pi^{\log}_{MSB}$ in offline phase less than

$$n - 1 + \sum_{i=0}^{d-1} 2^i \cdot 2 \leq 3(n-3)$$

The total communication of protocol $\Pi^{\log}_{MSB}$ in online phase less than

$$2(n-1) + \sum_{i=0}^{d-1} 2^i \cdot 6 \leq 8(n-1)$$

in $d+1$ rounds.

E. DReLU

In STF, three MPC protocol to compute DReLU is realized. They are the DReLU protocol $\Pi^{\text{linear}}_{DReLU}$ of linear round, the DReLU protocol $\Pi^{\text{linear}}_{DReLU}$ of logarithm round, and the DReLU protocol $\Pi^{\text{constant}}_{DReLU}$ of constant round. We introduce the protocol $\Pi^{\text{linear}}_{DReLU}$ and $\Pi^{\text{constant}}_{DReLU}$ in Algorithm 11 and 12.

For the protocol $\Pi^{\text{constant}}_{DReLU}$, refer to [ZZL+21] please.

We collect the communication complexity of the three protocol for DReLU in the Table A.3.

We can see the numeric experiments using different protocol of DReLU in Section [V-C] The experiments show that, in different case, different protocol of DReLU should be used. Specially, when the delay is high and the dimension of the input of ReLU layer is small, the protocol $\Pi^{\text{constant}}_{DReLU}$ is recommend; when the delay is low and the dimension of the input of ReLU layer is big, the protocol $\Pi^{\text{linear}}_{DReLU}$ is recommend; the protocol $\Pi^{\text{log}}_{DReLU}$ is recommend in otherwise.

F. ReLU and its Back-propagation

ReLU and its back propagation is computed as following

$$\text{ReLU}(x) = \begin{cases} 
  x & \text{if } \text{DReLU}(x) = 1 \\
  0 & \text{otherwise}
\end{cases}$$

In order to compute ReLU and its back propagation for shares, we use the Obvious selection protocol $\Pi^{N,\text{special}}_{\text{O-select}}$ in [ZZL+21]. Figure 11.

G. Inverse and Softmax

We use Newton method to compute the approximated inverse of a shares. The recurrence formula is

$$x_{n+1} := 2x_n - x_n^2$$

We use $\text{ReLU}(x_i) / \sum_i \text{ReLU}(x_i)$ to approximate Softmax.

IV. Experiments

We compare our STF with to TFE [DMD+18], Rosetta [CHS+20], SecureNN [WGC19] and CrypTen [KVH+20]. We run our experiments on three servers, each equipped with a Intel(R) Xeon(R) Platinum 8269CY 2.50GHz CPU and 7.715GB memory. Our network bandwidth is set to 100MB/s. We adjust the delay of our network to different levels according to the specific setting of our experiments.

A. Experiments for binary classification problem

For the binary classification problem, we evaluate the performance of logistic regression and fully-connected neural network on two datasets. The first dataset is xindai10, which contains 8429 examples in the training set and 3614 examples in the testing set. The dataset xindai10 is subsamples from the dataset GiveMeSomeCredit [xin]. Each example has 10 features. The dataset xindai10 is subsamples from the dataset GiveMeSomeCredit [xin].

The second dataset we evaluate is xindai291, which contains 37529 examples in the training set and 12509 examples in the testing set. Each example has a feature dimension of 291. We set our network delay to 10ms, 30ms, 60ms respectively and evaluate the performance of our system on each of these settings. We compare our system to other three PPML framework that supports logistic regression (TFE [DMD+18], Rosetta [CHS+20] and CrypTen [KVH+20]) and present our result in Table [V.1]. We do not compare with Falcon [WTB+21] because of some security problem with Falcon (See [A]). We set the batch size to 128 in all of these experiments.

B. Experiments for multi-classification problem

For the multi-class classification problem, we evaluate fully-connected neural network (FC) and convolutional neural network (CNN) on the MNIST dataset. We set the network bandwidth to 100MB/s and network delay to 60ms. We use 128 as our batch size and train our neural network for 10 epochs. We consider network A and network D described in SecureNN [WGC19] for performance comparison. We compare our system with SecureNN [DMD+18] and CrypTen [KVH+20] that supports these networks and present the result in the following
The speed of our STF is faster than SecureNN 282% and CrypTen 50.70% speedup on these networks than prior state-of-the-art.

The results demonstrate that our system achieves a 177% to 198% speedup on these networks.

We do not compare against TFE [DMD+18] and use the protocol such that the training speed is maximal.

### C. Select protocol of DReLU in different cases

Since we present three DReLU protocols with tradeoffs in communication and round complexities. We evaluate the performance of each of these protocols under different network delay. (See Table IV.3.) Our experiments show that in different delay and different network structure, different protocol for DReLU should be used. We recommend user to train 20 batch and compute the training speed as

\[
\text{training speed} = \frac{\text{end time of 20 batch} - \text{end time of 10 batch}}{10}
\]

and use the protocol such that the training speed is maximal.

**Table IV.2**

| dataset | Network | delay | \( \Pi_{DReLU} \) | train time | speed up |
|---------|---------|-------|-----------------|------------|---------|
| xindai10 | 32.32 | 5ms | 67s | 46s | 208s |
| xindai10 | 32.32 | 10ms | 113s | 52s | 208s |
| xindai10 | 32.32 | 30ms | 306s | 94s | 224s |
| xindai10 | 32.32 | 60ms | 606s | 165s | 252s |
| xindai291 | 7.7 | 5ms | 220s | 33s | 694s |
| xindai291 | 7.7 | 10ms | 445s | 133s | 205s |
| xindai291 | 7.7 | 30ms | 1307s | 308s | 253s |
| xindai291 | 7.7 | 60ms | 2549s | 612s | 492s |
| xindai291 | 32.32 | 5ms | 234s | 123s | 928s |
| xindai291 | 32.32 | 10ms | 456s | 134s | 928s |
| xindai291 | 32.32 | 30ms | 1332s | 296s | 979s |
| xindai291 | 32.32 | 60ms | 2657s | 640s | 1284s |
| Mnist network A | 60ms | 3996s | 790s | 2527s |
| Mnist network B | 60ms | 19125s | 26109s | - |
| Mnist network C | 60ms | 23766s | 33764s | - |
| Mnist network D | 60ms | 4620s | 3230s | 25913s |

\* Bandwidth=100Mbps, delay=60 ms, batchsize=128. The training time of one batch is computed as the (training time of 10 batches)/10.

\* For the network A and network D we use the option delay=middle, for the network B and network C we use the option delay=low.

**Table IV.3**

**V. RELATED WORK**

**ABY3** [MR18] is a private preserving computing system with three parties. In ABY3, sigmoid is approximated by a piecewise linear function. DReLU is realized by a protocol with communication \( O(n) \) and rounds \( O(\log n) \).

**ABY2.0** [PSSY21] is a private preserving computing system with two parties. The sigmoid is approximated by a piecewise linear function. DReLU is realized by a protocol with communication \( O(n) \) and rounds \( O(\log n) \).

In ABY3 and ABY2.0, the private preserving training and inference of full-connected neural network is realized, but that of convolution neural network is not realized. SecureNN [WGC19] is a privacy-preserving computing system in the three-party semi-honest setting. In SecureNN it implements its convolutional layer using matrix multiplication. Its DReLU protocol is has \( O(n \log n) \) communication and \( O(1) \) computation.
rounds. SecureNN does not support sigmoid function, and it therefore does not support logistic regression.

TF Encrypted (TFE) \cite{MD+18} is a privacy-preserving machine learning system based on Tensorflow. TFE uses SecureNN as its MPC backend, but modifies some of SecureNN’s protocols. For example, TFE supports sigmoid, which is approximated by Chebyshev polynomial. DReLU is implemented using a similar protocol as SecureNN. TFE supports secure inference of full-connected neural network and convolution neural network, but does not support secure training.

Rosetta \cite{CHS+20} is a privacy-preserving machine learning framework based on Tensorflow. Rosetta also adapts protocols from SecureNN, but does not support convolutional layer. It currently supports secure inference and training of full-connected neural network. Its sigmoid function is approximated by Chebyshev polynomial.

CrypTFlow \cite{KRC+20} is a privacy-preserving computing system in the three-party semi-honest setting. CrypTFlow’s ReLU protocol is improved over the constant round protocol from SecureNN. In CrypTFlow sigmoid is not realized, DReLU is realized by an improvement.

CrypTFlow2 \cite{RRK+20} is a private preserving computing system with two parties. In CrypTFlow2 sigmoid is not realized, DReLU is realized by a protocol with communication $O(n)$ and rounds $O(\log n)$.

In CrypTFlow and CrypTFlow2, Conv2D is realized by matrix multiplication where same mask is used for same elements, but the back propagations of conv2D is not realized. The private preserving inference of full-connected neural network and convolution neural network is realized, but the training for full-connected neural network and convolution neural network is not realized. The operator sigmoid is not realized also.

CrypTen \cite{KVH+20} is a private preserving computing system based on PyTorch. CrypTen realize the approximated sigmoid by Newton-Raphson iterations, DReLU is realized by a protocol with communication $O(n)$ and rounds $O(\log n)$. It use the bilinear property of conv2D to realize the MPC protocol of conv2D, and use the bilinear property of conv_transpose2d to realize the MPC protocol of the back propagation to input of Conv2D. For the back propagation to filter of conv2D, a protocol of redundant communication is designed and used.

Falcon \cite{WTB+21} is a private preserving computing system with three parties. In Falcon, sigmoid is not realized, DReLU is realized by a protocol with communication $O(n)$ and rounds $O(\log n)$. It realize Conv2D and its back propagations using matrix multiplication. The inference and training for both full-connected neural network and convolution neural network is realized. But in Falcon, there is a security issue (See Appendix A).

VI. CONCLUSION

We propose a Privacy Preserving Computation System STF. Based on this system, we construct privacy preserving Logistic Regression, Full-connected Neural Network and Convolution Neural Network. The performance of our STF is better than SecureNN \cite{WGC19}, TFE \cite{MD+18}, Rosetta \cite{CHS+20}, CrypTen \cite{KVH+20} in training speed and stability the state of the art in the normal hardware and WAN.
REFERENCES

[CHS+20] Yuanfeng Chen, Gaofeng Huang, Junjie Shi, Xiang Xie, and Yilin Yan. Rosetta: A Privacy-Preserving Framework Based on TensorFlow. https://github.com/LatticeX-Foundation/Rosetta, 2020.

[DMD+18] Morten Dahl, Jason Mancuso, Yann Dupis, Ben Decoste, Morgan Giraud, Ian Livingstone, Justin Patrquin, and Gavin Uhma. Private machine learning in tensorflow using secure computation. CoRR, abs/1810.08130, 2018.

[KRC+20] Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. CrypTFlow: Secure TensorFlow inference. In 2020 IEEE Symposium on Security and Privacy, pages 336–353. IEEE Computer Society Press, May 2020.

[KDH+20] B. Knott, S. Venkataraman, A.Y. Hannum, S. Sengupta, M. Ibrahim, and J.P. van der Maaten. Crypters: Secure multi-party computation meets machine learning. In Proceedings of the NeurIPS Workshop on Privacy-Preserving Machine Learning, 2020.

[MR18] Payman Mohassel and Peter Rindal. ABY³: A mixed protocol framework for machine learning. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, ACM CCS 2018, pages 35–52. ACM Press, October 2018.

[MZ17] Payman Mohassel and Yupeng Zhang. SecureML: A system for scalable privacy-preserving machine learning. In 2017 IEEE Symposium on Security and Privacy, pages 19–38. IEEE Computer Society Press, May 2017.

[PSSY21] Arpita Patra, Thomas Schneider, Ajith Suresh, and Hossein Yalame. ABY2.0: Improved mixed-protocol secure two-party computation. In 30. USENIX Security Symposium (USENIX Security’21), Virtual Event, August 11-13, 2021. USENIX. To appear. Full version: https://ia.cr/2020/1225

[RRK+20] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. CryptFlow2: Practical 2-party secure inference. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, ACM CCS 20, pages 325–342. ACM Press, November 2020.

[WGC19] Sameer Wagh, Divya Gupta, and Nishanth Chandran. SecureNN: 3-party secure training for neural network training. PoPETs, 2019(3):26–49, July 2019.

[WTC+21] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. FALCON: Honest-Mostly Maliciously Secure Framework for Private Deep Learning. 2021. Give me some credit.

[xin] Qizhi Zhang, Bingsheng Zhang, Lichun Li, Shan Yin, and Juanjuan Sun. Secure computation for g-module and its applications. Cryptology ePrint Archive, Report 2021/857, 2021. https://ia.cr/2021/857

APPENDIX

A. Security issue in Falcon

There is a security issue in Falcon [WTB+21]: In the protocol $\Pi_{BM}^{f,pp}(P_1, P_2, P_3)$ (Algorithm 6 in page 9) in [WTB+21], $P_1, P_2, P_3$ get the $\alpha$ such that $2^\alpha \leq b < 2^{\alpha+1}$. There is a lot of the leakage of the information of $b$.

B. Protocols of Bilinear map

Algorithm 1 Protocol $\Pi_{BM}^{f,pp}$

Input: $P_0$ hold $a \in A$, $P_1$ hold $b \in B$.
Output: $P_0, P_1$ get the shares of $f(a, b)$ over $C$.

1. $P_2$ generates random $\tilde{a} \in A$, $b \in B$, $\tilde{c}_0 \in C$, and compute $\tilde{c}_1 := f(\tilde{a}, b) - \tilde{c}_0$;
2. $P_2$ sends $\tilde{a}, \tilde{c}_0$ to $P_0$ and sends $\tilde{b}, \tilde{c}_1$ to $P_1$;
3. $P_0$ computes $\delta a := a - \tilde{a}$ and send to $P_1$;
4. $P_1$ computes $d b := b - \tilde{b}$ and send to $P_2$;
5. $P_0$ computes $c_0 := f(\tilde{a}, \delta b) + \tilde{c}_0$, $P_1$ computes $f(\delta a, b) + \tilde{c}_1$; return $(c_0, c_1)$.

Proof of correctness of $\Pi_{BM}^{f,pp}$:

\[
\begin{align*}
\text{Input: } & f(a, b) = f(a - \tilde{a}, b) + f(\tilde{a}, b - \tilde{b}) + f(\tilde{a}, \tilde{b}) \\
\text{Output: } & (f(\delta a, b) + \tilde{c}_1) + (f(\tilde{a}, \delta b) + \tilde{c}_0) \\
\end{align*}
\]

Algorithm 2 Protocol $\Pi_{BM}^{f,ps}$

Input: $P_0$ hold $a \in A$, $P_1, P_2$ hold the shares of $b = b_0 + b_1 \in B$.
Output: $P_0, P_1$ get the shares of $f(a, b)$ over $C$.

1. $P_0, P_1$ call the protocol $\Pi_{BM}^{f,pp}$ to compute $f(a, b_1)$ and get the shares $(u_0, u_1)$
2. $P_0$ computes $v = f(a, b_0)$ return $(u_0 + v, u_1)$.

It is easy to see the correctness of the protocol $\Pi_{BM}^{f,ps}$.

Algorithm 3 Protocol $\Pi_{BM}^{f,ss}$

Input: $P_0, P_1$ hold the shares of $a = a_0 + a_1 \in A, b = b_0 + b_1 \in B$.
Output: $P_0, P_1$ get the shares of $f(a, b)$ over $C$.

1. $P_2$ generates random $\tilde{a}_0, \tilde{b}_1 \in A, b_0, b_1 \in B, \tilde{c}_0 \in C$, and compute $\tilde{a} := a_0 + \tilde{a}_1, b := b_0 + b_1$ and $\tilde{c}_1 := f(\tilde{a}, b) - \tilde{c}_0$;
2. $P_2$ sends $\tilde{a}_0, \tilde{b}_1, \tilde{c}_0$ to $P_0$ and sends $\tilde{a}_1, b_1$ and $\tilde{c}_1$ to $P_1$;
3. $P_0$ computes $\delta a_0 := a_0 - \tilde{a}_0, \delta b_0 := b_0 - b_0$ and $\delta b := b_0 + b_1$;
4. $P_1$ computes $\delta a_1 := a_1 - \tilde{a}_1, \delta b_1 := b_1 - b_1$;
5. $P_0, P_1$ exchange $\delta a_0, \delta a_1, \delta b_0, \delta b_1$, and compute $\delta a := \delta a_0 + \delta a_1$ and $\delta b := \delta b_0 + \delta b_1$;
6. $P_0$ computes $c_0 := f(\delta a_0, b_0) + f(\tilde{a}_0, \delta b) + \tilde{c}_0$, $P_1$ computes $c_1 := f(\delta a_1, b_1) + f(\tilde{a}_1, \delta b) + \tilde{c}_1$; return $(c_0, c_1)$.

Proof of correctness of $\Pi_{BM}^{f,ss}$:

\[
\begin{align*}
\text{Input: } & f(a, b) = f(a - \tilde{a}, b) + f(\tilde{a}, b - \tilde{b}) + f(\tilde{a}, \tilde{b}) \\
\text{Output: } & (f(\delta a, b_0) + f(\tilde{a}_0, \delta b) + \tilde{c}_0 + \tilde{c}_1) + (f(\delta a_1, b_1) + f(\tilde{a}_1, \delta b) + \tilde{c}_1) \\
\end{align*}
\]

The communication complexity of the protocols $\Pi_{BM}^{f,pp}, \Pi_{BM}^{f,ps}, \Pi_{BM}^{f,ss}$ in following Table A.1.
C. Protocol of conv2D and Back Propagation of Conv2D

We give our protocols for conv2D and Back Propagation of Conv2D in Algorithm 4, Algorithm 5, Algorithm 6 and Algorithm 7.

Algorithm 4 Protocol $\Pi_{\text{conv2D}}^a$

Input: $P_0, P_1$ hold the shares of input over $\otimes_{B,m,n,C} A$, and the shares of filter over $\otimes_{C,r,s,D} A$.

\[
\partial_{\text{loss}} = \partial_{\text{output}} + \partial_{\text{input}} = \otimes_{B,m,n,C} A, \text{ filter} = filter_0 + filter_1 \in \otimes_{C,r,s,D} A.
\]

Output: $P_0, P_1$ get the shares of output over $\otimes_{B,m',n',D} A$.

1: $P_0, P_1$ call the protocol $\Pi_{\text{BM}}^a$ to compute output.

Algorithm 5 Protocol $\Pi_{\text{conv2D}}^b$

Input: $P_0, P_1$ hold the shares of input over $\otimes_{B,m,n,C} A$, and the shares of filter over $\otimes_{C,r,s,D} A$.

\[
\partial_{\text{loss}} = \partial_{\text{output}} + \partial_{\text{input}} = \otimes_{B,m,n,C} A, \text{ filter} = filter_0 + filter_1 \in \otimes_{C,r,s,D} A.
\]

Output: $P_0, P_1$ get the shares of output over $\otimes_{B,m',n',D} A$.

1: $P_0, P_1$ call the protocol $\Pi_{\text{BM}}^b$ to compute output.

D. Protocol of sin

We introduce our protocol to compute $\sin \frac{2kx}{2^m}$ in Algorithm 8.

We give the complexity of communication and round of $\Pi_{\text{sin}}^K$ and $\Pi_{\text{Sigmoid}}$ in Table A.4.

Algorithm 7 Protocol $\Pi_{\text{Con2D}}^c$

Input: $P_0, P_1$ hold the shares of $\partial_{\text{loss}}$ over $\otimes_{B,m',n',D} A$, $input_{0} + input_{1} \in \otimes_{B,m,n,C} A$.

Output: $P_0, P_1$ get the shares of $\partial_{\text{output}}$ over $\otimes_{B,m,n,C} A$.

1: $P_0, P_1$ call the protocol $\Pi_{\text{BM}}^c$ to compute $\partial_{\text{output}}$.

Algorithm 8 Protocol $\Pi_{\text{sin}}^K$

Input: $P_0, P_1$ hold shares of $x$ over $\frac{1}{2^m}Z/2^nZ$, and a common vector $k \in Z^K$.

Output: $P_0, P_1$ get the shares of $\sin \frac{2kx}{2^m}$ over $\left(\frac{1}{2^m}Z/2^nZ\right)^K$.

1: $P_2$ generates random $\tilde{x}_L, \tilde{x}_R \in \frac{1}{2^m}Z/2^nZ$, $u_L, u_R \in \left(\frac{1}{2^m}Z/2^nZ\right)^K$, and compute $\tilde{x} := \tilde{x}_L + \tilde{x}_R, v_L = \left[\sin \frac{2kx}{2^m}\right] - u_L, v_R = \left[\cos \frac{2kx}{2^m}\right] - v_L \in \left(\frac{1}{2^m}Z/2^nZ\right)^K$.

2: $P_2$ sends $\tilde{x}_L, u_L, v_L$ to $P_0$ and sends $\tilde{x}_R, u_R, v_R$ to $P_1$.

3: $P_0$ computes $\delta_{L} := x_L - \tilde{x}_L$ and send to $P_1$.

4: $P_1$ computes $\delta_{R} := x_R - \tilde{x}_R$ and send to $P_0$.

5: $P_0, P_1$ reconstruct $\tilde{x} := \delta_{L} + \delta_{R} \in \frac{1}{2^m}Z/2^nZ$, and compute $s := \sin \frac{2kx}{2^m} \in \left(\frac{1}{2^m}Z/2^nZ\right)^K, c := \cos \frac{2kx}{2^m} \in \left(\frac{1}{2^m}Z/2^nZ\right)^K$ respectively.

6: $P_0$ compute $w_L := sv_L + cu_L \in \left(\frac{1}{2^m}Z/2^nZ\right)^K, P_1$ compute $w_R := sv_R + cu_R \in \left(\frac{1}{2^m}Z/2^nZ\right)^K$.

7: Return $(w_L, w_R)$.

E. Protocol of MSB

We give the Protocol $\Pi_{\text{MSB}}^\text{linear}$ and $\Pi_{\text{MSB}}^\text{log}$ in Algorithm 9 and Algorithm 10.

Algorithm 9 Protocol $\Pi_{\text{MSB}}^\text{linear}$

Input: $P_0, P_1$ hold shares of $x$ over $\frac{1}{2^m}Z/2^nZ$.

Output: $P_0, P_1$ get shares of $\text{MSB}(x)$ over $\frac{1}{2^m}Z/2^nZ$.

1: $P_0$ let $c_0 = 0, P_1$ let $c_0 = 0$.

2: for $i = 1$ to $n$ do

3: $P_0, P_1$ compute $c_i := (x_{i-1} + c_{i-1})((x_R - c_{i-1}) - c_{i-1}) - c_{i-1}$.

4: end for

5: $P_0, P_1$ compute $m = x_{n-1} + x_R - c_{n-1}$.

6: Return $m$.

Algorithm 10 Protocol $\Pi_{\text{MSB}}^\text{log}$

Input: $P_0, P_1$ hold shares of $x$ over $\frac{1}{2^m}Z/2^nZ$.

Output: $P_0, P_1$ get shares of $\text{MSB}(x)$ over $\frac{1}{2^m}Z/2^nZ$.

1: $P_0, P_1$ let $Q_s := P_{\text{int}}(s)$ for all binary string $s$ of length $d(s) = d$;

2: for $i = 4$ to $n$ do

3: $P_0, P_1$ compute $Q_s$ for all binary string $s$ of length $d(s) = i$.

4: end for

5: Return $Q$.

F. Protocols of DReLU

We introduce the protocol $\Pi_{\text{DReLU}}^\text{linear}$ and $\Pi_{\text{DReLU}}^\text{log}$ in Algorithm 11 and 12.

We collect the communication complexity of the three protocols for DReLU in the Table A.3.
Table A.2

Communication Complexity of $\Pi_{\text{Conv2D}}$, $\Pi_{\text{Conv2D}b}$, $\Pi_{\text{Conv2D}f}$

| Protocol | offline com. | online comm. | online round | total comm. |
|----------|--------------|--------------|--------------|-------------|
| $\Pi_{\text{Conv2D}}$ | $(m'n'BD) \log |A|$ | $(mnBC + rsCD) \log |A|$ | 1 | $(mnBC + rsCD + m'n'BD) \log |A|$ |
| $\Pi_{\text{Conv2D}b}$ | $(m'n'BD) \log |A|$ | $2(mnBC + rsCD) \log |A|$ | 1 | $(2mnBC + 2rsCD + m'n'BD) \log |A|$ |
| $\Pi_{\text{Conv2D}f}$ | $(mnBC) \log |A|$ | $2(m'n'BD + rsCD) \log |A|$ | 1 | $(mnBC + 2rsCD + 2m'n'BD) \log |A|$ |
| $\Pi_{\text{Conv2D}f}$ | $(rsCD) \log |A|$ | $2(mnBC + m'n'BD) \log |A|$ | 1 | $(2mnBC + rsCD + 2m'n'BD) \log |A|$ |

Table A.3

Communication Complexity of $\Pi_{\text{DReLU}}$

| Protocol | offline com. | online comm. | online round | total comm. |
|----------|--------------|--------------|--------------|-------------|
| $\Pi_{\text{linear}}$ | $n-1$ | $4n-6$ | $n-1$ | $5n-7$ |
| $\Pi_{\text{log}}$ | $3(n-1)$ | $8(n-1)$ | $\lceil \log(n-1) \rceil + 1$ | $11(n-1)$ |
| $\Pi_{\text{const}}$ | $(2n - 1) \log p + 1$ | $2n \log p + 3n + \log n - 2$ | 4 | $(4n - 1) \log p + 3n + \log n - 1$ |

* Here $p$ is a prime number with $p \geq n + 2$.

Table A.4

Communication Complexity of $\Pi_{\text{Sigmoid}}$

| Protocol | offline com. | online com. | online round | total comm. |
|----------|--------------|-------------|--------------|-------------|
| $\Pi_{\text{poly}}$ | $20n$ | $5n$ | 5 | $25n$ |
| $\Pi_{\text{KSin}}$ | $2Kn$ | $2n$ | 1 | $2(K+1)n$ |
| $\Pi_{\text{fourier}}$ | $10n$ | $2n$ | 1 | $12n$ |

Algorithm 11 Protocol $\Pi_{\text{DReLU}}$  

**Input:** $P_0, P_1$ hold shares of $x$ over $\mathbb{Z}/2^n\mathbb{Z}$  
**Output:** $P_0, P_1$ get shares of $\text{DReLU}(x)$ over $\mathbb{Z}/2^n\mathbb{Z}$  
1: $P_0, P_1$ run the protocol $\Pi_{\text{linear}}$ to get the share of $z = \text{MSB}(x)$  
2: $P_0, P_1$ compute the share of $w = 1 - z$  
3: Return $w$

Algorithm 12 Protocol $\Pi_{\text{DReLU}}$

**Input:** $P_0, P_1$ hold shares of $x$ over $\mathbb{Z}/2^n\mathbb{Z}$  
**Output:** $P_0, P_1$ get shares of $\text{DReLU}(x)$ over $\mathbb{Z}/2^n\mathbb{Z}$  
1: $P_0, P_1$ run the protocol $\Pi_{\text{log}}$ to get the share of $z = \text{MSB}(x)$  
2: $P_0, P_1$ compute the share of $w = 1 - z$  
3: Return $w$