SecInfer: Secure and Efficient Model Inference on Vertically Partitioned Data

Haotian Deng[†], Hongwei Li[†], Hanxiao Chen(Corresponding author)[†], Meng Hao[‡],
Pengzhi Xing[†], Jia Hu[†], Rui Zhang[†], Wenbo Jiang[†]
[†]School of Computer Science and Engineering, University of Electronic Science and Technology of China, China
[‡]School of Computing and Information Systems, Singapore Management University, Singapore

***Abstract*—Deep learning models have achieved unprecedented success in various domains, such as healthcare and finance. However, deploying model inference in real-world applications, where data is distributed among multiple entities, poses significant privacy concerns. Existing secure model inference work has limitations in computational overhead and scalability, especially when dealing with complex models and multiple parties with vertically partitioned data.**

In this work, we design and implement an efficient and scalable secure inference framework for vertically partitioned data, supporting execution with a large number of parties. Our work considers a semi-honest setting with all-but-one corruptions. The core of our framework is a series of secure and efficient protocols for complex non-linear functions of the model inference, such as ReLU and Maxpool. These protocols are designed based on secure multi-party computation preliminaries, significantly enhancing efficiency while maintaining rigorous security guarantees. We conduct comprehensive experiments to evaluate the performance of our framework. Experimental results show that SecInfer **substantially improves the communication and computation performance of secure naive inference works by up to $3.71\times$ and $3.42\times$, respectively.**

***Index Terms*—Secure inference, Vertically partitioned data, Secure multi-party computation, Multiple parties, Dishonest-majority**

I. Introduction

Deep learning models have been subject to enormous uptake in the recent past and achieved unprecedented success in wide areas such as healthcare [1] and finance [2]. One of the main processes of this implementation is model inference, referring to using a trained deep learning model to make predictions or decisions. In some settings, data is vertically partitioned when several organizations own different attributes of information for the same set of entities. For example, when assessing the risk of a potential agreement, an insurance company requires confidential loans, health and identity information held by banks, medical institutions and the government, respectively. This has led to the deployment of multi-party model inference on *vertically* partitioned data, where a service provider deploys its deep learning models on the cloud and allows users to upload their input to obtain the inference result.

However, these inferences present many security concerns over privacy [3]. On the one hand, the exact parameters of a well-trained model may be sensitive and regularly related to business secrets or other intellectual property [4]. For another, model inference is fueled by data. In many cases, the involved data is usually privacy-sensitive [5]. For example, it can be personal data, like income tax, medical details, biometric traits, etc., subject to privacy concerns and data protection laws. Also, non-personal data can be sensitive, e.g., involving corporate deposits or military forces [6].

This scenario can be seen as a *client–server* model [7], [8]. The client, who holds the input data and requests the predication service, may be concerned about the privacy of the input data as it could contain some sensitive information; the server, who has trained this model on private data, may wish to monetize the model and thus is not willing to make the model parameters available for use. This creates a fundamental tension between preserving client data privacy and protecting the server's intellectual property.

Several cryptographic techniques have been developed to address the above challenge, such as Homomorphic Encryption (HE) and Secure Multi-Party Computation (MPC). HE allows computations on encrypted data without revealing the plaintext [9], [10] and has been applied to model inference [11]. However, HE suffers from high computational overhead and limited operation support, making it impractical for complex models or real-time applications [12], [13]. Moreover, circuit depth is restricted by HE scheme parameters, limiting applicability to deep networks [14]. MPC [15], [16] has also advanced significantly and is used for privacy-sensitive inferences [17], [18]. General-purpose MPC protocols for model inference are often designed for two-party settings and do not scale well to multi-party scenarios. This poses challenges for collaborative inference, as communication and computation costs grow quickly with the number of participants [19], [20].

In this work, we design and implement an efficient and scalable secure inference framework on vertically partitioned data, supporting running with a large number of parties. Our work considers the semi-honest setting with all-but-one corruptions. Specifically, model inference runs crossly by linear layers and non-linear layers[1]. We focus on the non-linear layer of model inference, which contributes to most part of the execution costs [5], [17]. Compared to secure basic model inference, SecInfer

[1]Linear layers, such as fully connected and convolutional layers, perform transformations that are linear in nature, i.e., addition and multiplication operations. Non-linear layers introduce non-linearity through activation functions (e.g., ReLU) or Maxpool operations, enabling the model to learn complex patterns.

979-8-3315-0521-9/25/$31.00 ©2025 IEEE

significantly reduces the execution time (i.e. $2.38 \sim 3.71\times$) communication cost (i.e. $3.36 \sim 3.42\times$). In summary, our contributions are as follows:

1) **Multi-party secure inference framework for vertically partitioned data:** We propose a novel framework that enables multiple parties to perform secure model inference on vertically partitioned datasets collaboratively. The framework is designed to be highly scalable, supporting a large number of participating entities due to communication linear in the number of parties.
2) **Secure MPC protocols for non-linear functions:** The core of our framework is a series of efficient MPC protocols specifically designed for complex non-linear mathematical functions commonly used in deep learning models, such as ReLU and Maxpool. These protocols optimize the computation and communication overhead of the non-linear layers, which are the primary contributors to execution costs in secure inference. Our protocols achieve this efficiency while maintaining rigorous security guarantees in the semi-honest model with all-but-one corruptions.
3) **Implementation and comparison:** We conduct the comprehensive implementation of our protocols to evaluate the performance of our framework. Experimental results show that SecInfer substantially improves the communication and computation performance of secure naive inference works by up to $3.71\times$ and $3.42\times$, respectively.

We begin with the details of some useful preliminaries in Section II. In Section III, we provide the threat model and high-level overview of SecInfer. Section IV presents our nonlinear protocols. Section V reports the experimental results and then concludes this work in Section VI.

II. Preliminaries

This section provides an overview of the necessary notations, secret-sharing schemes, and conversion techniques used in our protocol.

A. Notation

We denote the number of parties as N and the i-th party as P_i, where $i \in [N]$ and $[N]$ denotes the set of integers $\{1, ..., N\}$. We use $\langle x \rangle = (\langle x \rangle_1, ..., \langle x \rangle_N)$ to denote a value shared among N parties, where the subscript i denotes the i-th share of x that is held by party P_i. We denote a vector (or bit-string) as a lower-case bold letter or an upper-case letter, such as $\boldsymbol{x}$ and T. For $\boldsymbol{x}$, the i-th element of $\boldsymbol{x}$ is denoted as $\boldsymbol{x}[i]$, where $i \in [\ell]$ and ℓ denotes the length of $\boldsymbol{x}$. We use bold upper-case letters (e.g., $\boldsymbol{X}$) to represent matrices. $\boldsymbol{X}[i, \cdot]$ and $\boldsymbol{X}[\cdot, j]$ respectively denote the i-th row and j-th column of $\boldsymbol{X}$. We use $[\![x]\!]$ to represent a HE ciphertext on x.

B. Fixed-Point Representation

Following prior works [21], we encode a real number $\hat{x} \in \mathbb{R}$ as a field element $x \in \mathbb{F}_p$ using a fixed-point representation. This representation in $\mathbb{F}_p$ is parameterized by a scale variable s, which defines the fractional bit length. We introduce two mappings for conversion between real numbers and their field representation:

- **R2F:** $\mathbb{R} \to \mathbb{F}_p$. The mapping from reals to the field is given by $\mathrm{R2F}(x, p, s) = \lfloor x \cdot 2^s \rfloor \bmod p$.
- **F2R:** $\mathbb{F}_p \to \mathbb{R}$. The mapping from the field to reals is $\mathrm{F2R}(x, p, s) = (x - c \cdot p)/2^s$, where the operations are over $\mathbb{R}$ and $c = 1\{x > (p-1)/2\}$.

For $s = 0$, the conversions simplify to those between signed integers and their field representation. Thus, $\mathbb{F}_p$ can encode signed integers in the range $\left[-\frac{p-1}{2}, \frac{p-1}{2}\right]$.

C. Secret Sharing

Throughout this work, we use n-out-of-n additive secret sharing schemes over two finite fields [5], [7], [22]. The two specific finite fields that we consider are $\mathbb{F}_p$ and $\mathbb{F}_2$, where p denotes a prime. We typically refer to shares over $\mathbb{F}_p$ as arithmetic shares and shares over $\mathbb{F}_2$ as Boolean shares, which are represented by $\langle x \rangle^A$ and $\langle x \rangle^B$, respectively. Shares are generated among N parties by sampling random finite field elements $\langle x \rangle_1^S, ..., \langle x \rangle_N^S$, where $S \in \{A, B\}$ indicates the type of shares. The only constraint of the shares is that $x = \sum_{i=1}^N \langle x \rangle_i^S \bmod p$, where $\sum$ denotes the summary of addition in the finite field. The addition operation of elements in $\mathbb{F}_2$ corresponds to the XOR operation among them. In this case, we can just write $x = \bigoplus_{i=1}^N \langle x \rangle_i^B$. Additive secret sharings support addition and local mult with constants due to its linear homomorphism.

D. Lookup Table

We use an efficient multi-party lookup-table protocol proposed by GYKW [23], supporting scalable conversions. Specifically, given a table T of size 2^ℓ, P_1 encrypts and permutes T using a bit-string $\boldsymbol{r}_1 \in \{0,1\}^\ell$ and sends it to P_2. Each subsequent party P_i permutes the table with a new bit-string $\boldsymbol{r}_i$ until P_n finally obtains $[\![T_n]\!]$. Each party holds a Boolean sharing $\langle \boldsymbol{r} \rangle_i^B = \boldsymbol{r}_i$.

Consider that each party holds a shared index $\langle \boldsymbol{j} \rangle^B$. The parties compute $\langle \boldsymbol{j} \oplus \boldsymbol{r} \rangle^B$ and reconstruct it for P_n, who retrieves $[\![T_n[\boldsymbol{j} \oplus \boldsymbol{r}]]\!] = [\![T[\boldsymbol{j}]]\!]$. This value is then converted from a Boolean to an arithmetic sharing, forming the foundational step for the subsequent conversion process.

E. Boolean and Arithmetic Conversion

Model inference typically employs arithmetic operations for its computations. Due to Boolean sharing performing better in the non-linear layers, we utilize conversion based on the lookup table technique to support highly efficient and scalable non-linear protocols.

Boolean-to-arithmetic (B2A) conversion. Efficient Boolean-to-arithmetic conversion relies on interpreting the problem as a public table lookup with a private index. In particular, assume that each party holds a Boolean sharing $\langle x \rangle^B$ of message x and a public table of size 2 with 0 and 1 in $\mathbb{F}_p$. To run the protocol of this conversion efficiently, it incorporates state-of-the-art optimization on HE schemes like

Functionality $\mathcal{F}_{\mathsf{Conv}}$

This functionality operates over a finite field $\mathbb{F}_p$ (resp., $\mathbb{F}_2$) for arithmetic (resp., Boolean) secret-shared values, and interacts with parties $P_1, \ldots, P_n$.

From Boolean to Arithmetic: Upon receiving $(\mathsf{B2A}, \mathsf{id}, \mathsf{id}')$ from all parties where $(\mathsf{id}, \mathsf{bool})$ is present in memory, retrieve $(\mathsf{id}, \mathsf{bool}, x)$ and store $(\mathsf{id}', \mathsf{arith}, x)$.

From Arithmetic to Boolean: Upon receiving $(\mathsf{A2B}, \mathsf{id}, \mathsf{id}_0, \ldots, \mathsf{id}_{\ell-1})$ from all parties where $(\mathsf{id}, \mathsf{arith})$ is present in memory and $\ell = \lceil p \rceil$, retrieve $(\mathsf{id}, \mathsf{arith}, x)$ and store $(\mathsf{id}_i, \mathsf{bool}, x_i)$ for $i \in [0, \ell - 1]$, where $x = \sum_{i=0}^{\ell-1} x_i \cdot 2^i \mod p$.

Figure 1: Functionality for the black box of conversions.

independently permuting the encrypted entries within each table that are all packed into the same ciphertext efficiently.

Arithmetic-to-Boolean (A2B) conversion. This multi-party lookup-table protocol for arithmetic-to-Boolean conversion builds on similar ideas to the former one but introduces additional complexities. By viewing it as a table lookup, it generates random Boolean shares and then transforms these into arithmetic shares. The protocol uses efficient preprocessing to mask lookup tables and reduces communication complexity by leveraging homomorphic encryption for linear communication efficiency. The conversion includes checking if the value is within a predefined range using secure comparison and adjusting if necessary to ensure correctness.

The conversion is packaged as a black box, shown in Figure 1 for our protocols calling in the following work.

III. Secure Model Inference

This section details the threat model and provides a high-level overview of SecInfer.

A. Threat Model

The security of SecInfer is provably provided against a semi-honest probabilistic polynomial time adversary $\mathcal{A}$. In this setting, there is a set of n mutually-distrusting parties, namely $\mathcal{P} = \{P_1, ..., P_n\}$ where $n \geq 2$, with each party P_i having some private input x_i. The model in our work is secure against passively corrupted (semi-honest, also called *honest-but-curious*) adversaries that follow the protocol specification but try to infer additional information about the other parties' private input. Given a *publicly-known* n-ary function f, the goal is to let every party learn the output $y \stackrel{def}{=} f(x_1, \ldots, x_n)$, such that the adversary does not learn any additional information, beyond what can be learnt from y and the inputs of the corrupt parties.

B. Overview of SecInfer

SecInfer provides an efficient framework for secure inference on vertically partitioned data, as shown in Figure 2, where data is privately held by multiple entities (like banks, hospitals, and campuses). At the core of our frame are non-linear protocols. When securely implementing the protocols

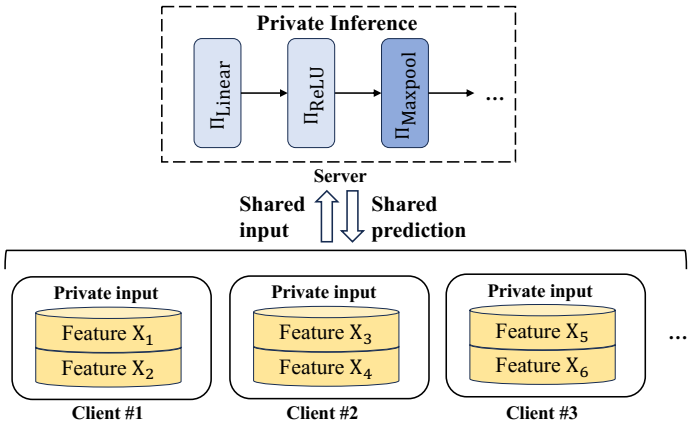

Figure 2: Overview of secure inference in SecInfer

in the non-linear layer of our framework SecInfer, the clients begin with arithmetic shares of the input to the layer and end with arithmetic shares of the layer output after the protocol. Though the main operations are done in Boolean sharing type, SecInfer puts arithmetic and Boolean conversion at the beginning and end of the layers. This allows us to stitch protocols for proper layers sequentially to obtain a complete secure inference scheme. The semi-honest security will follow trivially from the sequential composability of individual sub-protocols.

IV. Scalable Multi-Party Non-linear Protocol

In this section, we present a series of efficient multi-party protocols for the non-linear layers of the deep learning models. In Figure 3, we propose a protocol basic protocol with high scalability. An optimized protocol is shown in Figure 4, which reduces some redundancy and extra costs in both computation and communication. Based on the above optimization, our Maxpool protocol in Figure 6 is much more efficient than the naive approach.

A. Basic ReLU

In Figure 3, we describe our basic protocol for ReLU that takes as input arithmetic shares of x and returns arithmetic shares of y. Note that the scale of data processing through the linear layer is to be doubled to $2k$, which needs recovering to k to ensure the precision of the most significant bits. We first propose a naive approach to achieve the scale recovery and ReLU function.

All parties first invoke $\mathcal{F}_{\mathsf{A2B}}$ detailed in Figure 1 for conversion at the beginning, converting the input into Boolean shares (step 1). In step 2, all parties compute $\langle t \rangle^B \leftarrow \mathbf{1}\{\langle x \rangle^B > \frac{p-1}{2}\}$ to get the plaintext x's true sign bit, obtaining 1 if $x < 0$ and 0 otherwise. Each party then converts the value presented over the finite field to its original value, achieved by subtracting p through the Boolean subtract operation if the sharing is negative (step 3).

To keep the scale fixed in s, in step 4, each party locally shifts s bit of the value to the right, filling the empty significant s bits with the true sign bit $\langle t \rangle^B$ while dropping the least significant s bits.

With the true sign bit obtained above, they can easily put the negative sharing to zero, which just needs to be multiplied by $\neg\langle t\rangle^B$. At the end of this protocol, all parties call $\mathcal{F}_{\mathsf{B2A}}$ for Boolean-to-arithmetic conversion to obtain their arithmetic shares (steps 5&6).

Protocol $\Pi_{\mathsf{basicReLU}}$

Inputs: All parties hold an arithmetic sharing $\langle x\rangle^A$, and a module number p.

Processing:

1) All parties call $\mathcal{F}_{\mathsf{A2B}}$ on input $\langle x\rangle^A$ to obtain $\langle x\rangle^B$.
2) All parties compute $\langle t\rangle^B \leftarrow \mathbf{1}\{\langle x\rangle^B > \frac{p-1}{2}\}$ to get the true sign bit of each sharing.
3) All parties compute $\langle x\rangle^B \leftarrow \langle x\rangle^B - \langle t\rangle^B \cdot p$ to convert the value representing in $\mathbb{F}_p$ to the true value.
4) All parties shift the most significant $\ell - s$ bits of $\langle x\rangle^B$ to the right, and fill the most significant s bits with its true sign bit $\langle t\rangle^B$, obtaining $\langle y\rangle^B \leftarrow \langle t\rangle^B || \ldots || \langle t\rangle^B || \langle x_{\ell-1}\rangle^B || \ldots || \langle x_s\rangle^B$.
5) All parties compute $\langle y\rangle^B \leftarrow \langle y\rangle^B \cdot \neg\langle t\rangle^B$ to set the negative value with the sharing of 0.
6) All parties call $\mathcal{F}_{\mathsf{B2A}}$ on input $\langle y\rangle^B$ to obtain $\langle y\rangle^A$.

Figure 3: Basic secure ReLU function.

B. Optimized ReLU

We propose an optimized ReLU protocol shown in Figure 4. In Figure 3, a secure ReLU protocol has been shown. However, due to the necessity of sequential high-cost operations of the subtraction operation under Boolean shares, it produces high costs in both execution time and communication. To reduce the redundancy and cost mentioned above, our optimized protocol cuts off this expensive operation to achieve the same goal of recovering the scale and ReLU operation.

The optimized protocol makes rearrangements to the execution order of the steps. Specifically, since recovering to the scale is complicated over a finite field while easy to operate for a real number $\hat{x} \in \mathbb{R}$, we run $\langle x\rangle^B \leftarrow \langle x\rangle^B \cdot \neg\langle t\rangle^B$ in advance to ensure all numbers are non-negative (step 3). Thus, in step 4, all parties can directly use simple truncation $\langle y\rangle^B \leftarrow \langle x_{\ell-1}\rangle^B || \ldots || \langle x_s\rangle^B$ on the sharing. Finally, they compute arithmetic shares of y using a call to $\mathcal{F}_{\mathsf{B2A}}$.

Protocol Π_{ReLU}

Inputs: All parties hold an arithmetic sharing $\langle x\rangle^A$, and a module number p.

Processing:

1) All parties call $\mathcal{F}_{\mathsf{A2B}}$ on input $\langle x\rangle^A$ to obtain $\langle x\rangle^B$.
2) All parties compute $\langle t\rangle^B \leftarrow \mathbf{1}\{\langle x\rangle^B > \frac{p-1}{2}\}$ to get the true sign bit of each sharing.
3) All parties compute $\langle x\rangle^B \leftarrow \langle x\rangle^B \cdot \neg\langle t\rangle^B$.
4) All parties set $\langle y\rangle^B \leftarrow \langle x_{\ell-1}\rangle^B || \ldots || \langle x_s\rangle^B$.
5) All parties call $\mathcal{F}_{\mathsf{B2A}}$ on input $\langle y\rangle^B$ to obtain $\langle y\rangle^A$.

Figure 4: Optimized secure ReLU function.

C. Maxpool

We further apply the above optimization technique within the Π_{Maxpool} protocol. The detailed implementation of this protocol is illustrated in Figure 6. In the secure Maxpool protocol, each party provides matrices in an arithmetic sharing format and extracts each kernel-sized window into a linear form. Subsequently, the parties group each column into a batch, effectively reducing the communication overhead (step 1).

Functionality $\mathcal{F}_{\mathsf{Non-linear}}$

This functionality has all of the same features as $\mathcal{F}_{\mathsf{Conv}}$ with the following additional commands.

ReLU: Upon receiving $(\mathsf{ReLU}, \mathsf{id}, \mathsf{id}')$ from all parties where $(\mathsf{id}, \mathsf{bool})$ is present in memory, retrieve $(\mathsf{id}, \mathsf{bool}, x)$ and store $(\mathsf{id}', \mathsf{bool}, y)$, where y is equal to x if $x > 0$ and 0 otherwise.

Maxpool: Upon receiving $(\mathsf{Maxpool}, \mathsf{id}_1, \ldots, \mathsf{id}_n, \mathsf{id})$ from all parties where $(\mathsf{id}_i, \mathsf{arith})$ for $i \in [1, n]$ are present in memory and n is the kernel size, retrieve $(\mathsf{id}_i, \mathsf{arith}, x_i)$ for each $i \in [1, n]$ and store $(\mathsf{id}, \mathsf{bool}, \max(\{x_i\}))$.

Figure 5: Functionality for the black box of Non-linear operations.

Protocol Π_{Maxpool}

Inputs: All parties hold an arithmetic sharing $\langle \boldsymbol{X}\rangle^A$ of size $n \times n$, window size $m \times m$, stride size s, and a modulus number p.

Processing:

1) Extract each window of size $m \times m$ from $\langle \boldsymbol{X}\rangle^A$ with stride s and flatten it into a row vector. Concatenate all row vectors to form a new matrix $\langle \boldsymbol{Y}\rangle^A$ of size $t^2 \times m^2$, where $t = \lfloor \frac{n-m}{s} \rfloor + 1$.
2) Initialize $\langle T_{\mathsf{max}}\rangle^A \leftarrow \langle \boldsymbol{Y}[\cdot, 0]\rangle^A$.
3) For each column j from 1 to $m^2 - 1$:
   a) Compute: $\langle T_{\mathsf{sub}}\rangle^A \leftarrow \langle T_{\mathsf{max}}\rangle^A - \langle \boldsymbol{Y}[\cdot, j]\rangle^A$.
   b) ReLU: $\langle T_{\mathsf{relu}}\rangle^A \leftarrow \mathcal{F}_{\mathsf{ReLU}}(\langle T_{\mathsf{sub}}\rangle^A, p)$.
   c) Update: $\langle T_{\mathsf{max}}\rangle^A \leftarrow \langle T_{\mathsf{relu}}\rangle^A + \langle \boldsymbol{Y}[\cdot, j]\rangle^A$.
4) After processing all columns, $\langle T_{\mathsf{max}}\rangle^A$ contains the maximum value for each row of $\langle \boldsymbol{Y}\rangle^A$.
5) Reshape $\langle T_{\mathsf{max}}\rangle^A$ into a matrix of size $t \times t$ for the final output.

Figure 6: Secure Maxpool function.

Consider the elements in the first column, denoted as $\langle T_{\mathsf{max}}\rangle^A \leftarrow \langle \boldsymbol{Y}[\cdot, 0]\rangle^A$. In step 3, we compute the difference between the first and second columns under arithmetic sharings, yielding $\langle T_{\mathsf{sub}}\rangle^A \leftarrow \langle T_{\mathsf{max}}\rangle^A - \langle \boldsymbol{Y}[\cdot, 1]\rangle^A$. Based on the result of this subtraction, we invoke $\mathcal{F}_{\mathsf{ReLU}}$, which sets all negative values to zero while keeping the non-negative values unchanged. Afterward, we add back the subtrahend, $\langle \boldsymbol{Y}[\cdot, 1]\rangle^A$. Specifically, for each row, let us assume that the shares of the first and second columns are $\langle x\rangle^A$ and $\langle y\rangle^A$, respectively, and the result of the operation for this row is $\langle z\rangle^A$. The output value in this row is $\langle x\rangle^A$ if $\langle x\rangle^A > \langle y\rangle^A$,

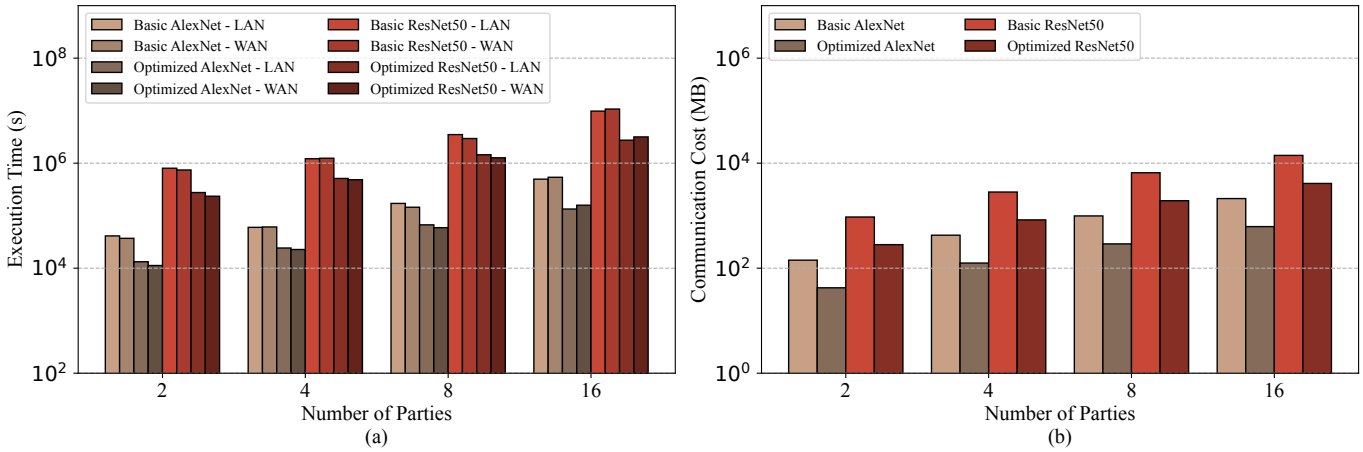

Figure 7: **Performance evaluation for the non-linear protocols of our framework in end-to-end inference.** For the execution time (resp. the communication cost) vs. the number of parties in model inference, i.e. AlexNet and ResNet50.

and $\langle y \rangle^A$ otherwise. We repeat this process for all columns, ensuring that all the entries in $\langle T_{\mathsf{max}} \rangle^B$ represent the maximum value for each row.

To generate the final output of the Maxpool layer, each party reshapes the resulting vector back into a matrix of dimensions $\lfloor \frac{n-m}{s} \rfloor + 1$, according to the shape of the window and the stride size.

V. Evaluation

A. Experiment Setup

We implement our protocols with C++. Our work is based on GYKW [23] and MOTION [24], while GYKW is for sharing type conversion, i.e., A2B and B2A, and MOTION is for non-linear operations. Our experiments focus on the online phase, as the offline part can be precomputed ahead. Following the existing work [23], we evaluate our experiment in both LAN and WAN settings. To make our results reproducible, our implementation in this paper has been made public[2]. The experiment considers two settings with up to 16 parties, which are described as follows:

1) Under LAN: The network bandwidth is 1 Gbps with 0.1 ms latency.
2) Under WAN: The network bandwidth is 200 Mbps with 100 ms latency.

All experiments are performed on Intel(R) Xeon(R) Gold 5218R CPU @ 2.10GHz, 80 cores, 251 GiB of memory. Unless otherwise specified, the parameters for threshold homomorphic encryption in the public-key setting achieve the 128-bit security level, the plaintext prime p is equal to $2^{32}-2^{30}+1$, the length of the ciphertext prime q is more than 530 bits, and the number of slots $N = 65536$. We compare the performance of our optimized protocols with that of commonly used counterparts. All results are evaluated using the same number of multiple threads. We present the average results across 10 runs for each experiment.

[2]https://github.com/haotian-deng/SecInfer

Protocol	Setting	Number of Parties			
		2	4	8	16
ReLU (basic)	Comm.	94	281	654	1400
	LAN	45.60	69.43	200.13	558.56
	WAN	42.33	70.95	168.37	611.61
ReLU (optimized)	Comm.	28	83	192	409
	LAN	15.79	29.33	82.99	156.74
	WAN	13.45	27.72	72.62	180.32
Maxpool (basic)	Comm.	169	506	1177	2520
	LAN	94.15	121.38	336.39	1088.27
	WAN	77.85	121.66	289.17	1153.57
Maxpool (optimized)	Comm.	50	149	345	737
	LAN	24.47	42.20	105.86	261.14
	WAN	20.48	38.19	97.45	335.49

Table I: **Performance of non-linear protocols.** Communication costs estimation in bytes and running time in milliseconds (ms) for running the basic and optimized non-linear protocols in the LAN and WAN settings.

B. Microbenchmarks

Evaluation of ReLU Protocol. Table I presents the execution time and communication costs for both the basic and optimized protocols, considering variations in the number of parties and network environments (LAN or WAN as described in Section V-A). We evaluate scenarios involving up to 16 parties (with a $2\times$ parameter increase). In the LAN setting, our optimized protocol Π_{ReLU} achieves a performance improvement of approximately $2.37 \sim 3.57\times$ in execution time compared to the basic protocol $\Pi_{\mathsf{basicReLU}}$, while under WAN, the improvement ranges from $2.32 \sim 3.39\times$. Additionally, communication costs are reduced by approximately $3.36 \sim 3.42\times$ compared to the basic version.

Evaluation of Maxpool Protocol. We further compare the optimized Maxpool protocol to the basic version. In LAN settings, the optimized protocol achieves an execution time reduction of approximately $2.88 \sim 4.17\times$ compared to the non-optimized version, while also outperforming the basic version by at least $3.38 \sim 3.42\times$ in terms of communication

efficiency. Under WAN conditions, the execution time cost shows an improvement of approximately $2.97 \sim 3.80\times$.

C. End-to-end Inference Evaluation

In this section, we evaluate the end-to-end performance of our framework utilizing two different models, i.e., AlexNet and ResNet50.

Execution Time. Figure 7(a) presents the execution times for model inference (AlexNet and ResNet50) across varying numbers of parties, up to a maximum of 16. Inferences conducted using the basic protocols are labeled as *basic* AlexNet (or *basic* ResNet50), while those employing optimized protocols are denoted as *optimized* AlexNet (or *optimized* ResNet50). This comparison illustrates the performance gains of our optimized MPC-based inference relative to the baseline non-optimized versions.

The results indicate a substantial reduction in inference times for both AlexNet and ResNet50 when using the optimized protocols. Specifically, the optimized protocol reduces execution time by approximately $2.48 \sim 3.71\times$ for AlexNet and $2.38 \sim 3.58\times$ for ResNet50 in the LAN setting. Similar improvements are observed under WAN, with reductions of $2.45 \sim 3.40\times$ for AlexNet and $2.33 \sim 3.39\times$ for ResNet50.

Communication Cost. Figure 7(b) demonstrates that both AlexNet and ResNet50 achieve considerable reductions in communication costs when utilizing the optimized protocols. Specifically, the optimized versions reduce communication overhead by approximately $3.36 \sim 3.42\times$ compared to the basic protocols. The communication cost scales linearly with the number of parties, highlighting the efficiency of our approach in larger distributed environments.

VI. Conclusion

In this work, we presented SecInfer, an efficient and scalable secure inference framework for vertically partitioned data, addressing real-world scenarios involving multiple independent entities. Our framework is optimized for multi-party settings and focuses on the high-overhead non-linear layers of model inference, significantly reducing both computation and communication costs. Our optimized protocols achieved up to $3.71\times$ faster execution and $3.42\times$ lower communication costs compared to existing methods, making secure inference more practical for applications that require data privacy and distributed ownership. In future work, we aim to further optimize SecInfer by enhancing protocol efficiency and improving resilience against stronger adversaries, including malicious settings.

Acknowledgment

This work is supported by the National Key R&D Program of China under Grant 2022YFB3103500, the National Natural Science Foundation of China under Grant 62402087 and 62020106013, the Chengdu Science and Technology Program under Grant 2023-XT00-00002-GX, the Fundamental Research Funds for Chinese Central Universities under Grant ZYGX2020ZB027 and Y030232063003002, the China Postdoctoral Science Foundation under Grant BX20230060, BX20240053 and 2024M760356, Natural Science Foundation of Sichuan under Grant 2025ZNSFSC1490.

References

[1] E. Choi, M. T. Bahadori, J. Sun, J. Kulas, A. Schuetz, and W. Stewart, "Retain: An interpretable predictive model for healthcare using reverse time attention mechanism," in *Proceedings of NeurIPS*, 2016.

[2] J. Wang, Y. Zhang, K. Tang, J. Wu, and Z. Xiong, "Alphastock: A buying-winners-and-selling-losers investment strategy using interpretable deep reinforcement attention networks," in *Proceedings of ACM SIGKDD*, 2019.

[3] N. Carlini, F. Tramer, E. Wallace, M. Jagielski, A. Herbert-Voss, K. Lee, A. Roberts, T. Brown, D. Song, U. Erlingsson *et al.*, "Extracting training data from large language models," in *Proceedings of USENIX Security*, 2021.

[4] N. Agrawal, A. Shahin Shamsabadi, M. J. Kusner, and A. Gascón, "Quotient: Two-party secure neural network training and prediction," in *Proceedings of ACM CCS*, 2019.

[5] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *Proceedings of USENIX Security*, 2018.

[6] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings *et al.*, "Advances and open problems in federated learning," *Foundations and trends® in machine learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[7] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "Delphi: A cryptographic inference service for neural networks," in *Proceedings of USENIX Security*, 2020.

[8] D. Rathee, M. Rathee, N. Kumar, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "Cryptflow2: Practical 2-party secure inference," in *Proceedings of ACM CCS*, 2020.

[9] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of STOC*, 2009.

[10] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," Cryptology ePrint Archive, 2012.

[11] H. Choi, S. S. Woo, and H. Kim, "Blind-touch: Homomorphic encryption-based distributed neural network inference for privacy-preserving fingerprint authentication," in *Proceedings of AAAI*, 2024.

[12] F. Boemer, A. Costache, R. Cammarota, and C. Wierzynski, "ngraph-he2: A high-throughput framework for neural network inference on encrypted data," in *Proceedings of ACM CCS*, 2019.

[13] P. Xie, M. Bilenko, T. Finley, R. Gilad-Bachrach, K. Lauter, and M. Naehrig, "Crypto-nets: Neural networks over encrypted data," *arXiv preprint arXiv:1412.6181*, 2014.

[14] H. Chabanne, A. De Wargny, J. Milgram, C. Morel, and E. Prouff, "Privacy-preserving classification on deep neural network," *Cryptology ePrint Archive*, 2017.

[15] A. C.-C. Yao, "How to generate and exchange secrets," in *Proceedings of FOCS*, 1986.

[16] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game, or a completeness theorem for protocols with honest majority," in *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, 2019, pp. 307–328.

[17] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *Proceedings of IEEE S&P*, 2017.

[18] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via minionn transformations," in *Proceedings of ACM CCS*, 2017.

[19] M. Keller, "Mp-spdz: A versatile framework for multi-party computation," in *Proceedings of ACM CCS*, 2020.

[20] A. Patra and A. Suresh, "Blaze: blazing fast privacy-preserving machine learning," *arXiv preprint arXiv:2005.09042*, 2020.

[21] M. Hao, H. Chen, H. Li, C. Weng, Y. Zhang, H. Yang, and T. Zhang, "Scalable zero-knowledge proofs for non-linear functions in machine learning," in *Proceedings of USENIX Security*, 2024.

[22] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, p. 612–613, Nov. 1979.

[23] R. Garg, K. Yang, J. Katz, and X. Wang, "Scalable mixed-mode mpc," in *Proceedings of IEEE S&P*, 2024.

[24] L. Braun, D. Demmler, T. Schneider, and O. Tkachenko, "Motion–a framework for mixed-protocol multi-party computation," *ACM Transactions on Privacy and Security*, vol. 25, no. 2, pp. 1–35, 2022.