Low-Latency Privacy-Preserving Deep Learning Design via Secure MPC

Ke Lin[1], Yasir Glani[1] and Ping Luo[1,*]

[1] Tsinghua University, 30 Shuangqing Rd., Haidian District, Beijing, 100084, China

Abstract

Secure multi-party computation (MPC) facilitates privacy-preserving computation between multiple parties without leaking private information. While most secure deep learning techniques utilize MPC operations to achieve feasible privacy-preserving machine learning on downstream tasks, the overhead of the computation and communication still hampers their practical application. This work proposes a low-latency secret-sharing-based MPC design that reduces unnecessary communication rounds during the execution of MPC protocols. We also present a method for improving the computation of commonly used nonlinear functions in deep learning by integrating multivariate multiplication and coalescing different packets into one to maximize network utilization. Our experimental results indicate that our method is effective in a variety of settings, with a speedup in communication latency of $10 \sim 20\%$.

Keywords

Multi-party computation, deep learning, privacy-preserving

1. Introduction

Secure multi-party computation (MPC) [1, 2] enables parties to compute securely over their private data without revealing the data to each other. Secure MPC offers privacy-preserving property, which makes it suitable for most privacy-sensitive domains, such as medical research and finance. Upon the development of deep learning techniques, the ability to capture important information from large datasets of neural models raises concerns regarding the surveillance of individuals [3]. In this case, the prospects of secure MPC demonstrate its application to secure machine learning and deep learning. While MPC-based deep learning frameworks have achieved significant performance in general scenarios, most works suffer from the limitations caused by 1. *network communication* due to the nature of exchanging intermediate information during MPC execution, 2. *excessive computation* introduced by complex MPC protocols. Since the computation of MPC protocols is largely determined by their sophisticated design, optimizing the protocol itself would seem to be difficult and infeasible. Thus, some studies [4] are concerned with improving the communication stage of MPC protocols to make them more practical. In this paper, we present an approach to reduce the communication latency of the MPC protocol through optimized multivariate multiplication.

In general, privacy-preserving deep learning frameworks usually adopt secret-sharing-based techniques to avoid extensive computational overheads [5, 6, 7, 8]. Consequently, secret-sharing-based methods require multiple exchanges of intermediate results to achieve collaborative MPC operations. As these MPC techniques are based on linear computations, such as addition and multiplication, modern deep learning techniques that inherently rely on linear algebra benefit significantly from them. Considering the heavy dependency on linear operations, our research aims to reduce unnecessary communication rounds following [9] during the execution of MPC protocols.

Our main contributions are as follows:

- We propose a low-latency secret-sharing-based method for computing multivariate multiplications and univariate polynomials using network communication that is efficient and reduces unnecessary communication rounds on the fly.
- We improve the computation of nonlinear functions by integrating the proposed multivariate multiplication and coalescing different packets into one single packet to maximize network utilization.
- We conducted experiments to evaluate the effectiveness of our method in the context of models with varying sizes, networks with different latency and bandwidth, the accuracy of downstream classification tasks, and the number of participants involved. The results indicate an overall improvement of $10 \sim 20\%$ in communication latency.

2. Background

2.1. Arithmetic Secret Sharing Based Scheme

Our setup is primarily focused on arithmetic operations, so we represent all inputs and intermediate results in terms of linear secret sharing between n parties, especially in the context of additive secret-sharing schemes.

Apart from the general (n, t)-Shamir secret sharing scheme [10], which relies on the degree-t polynomials over n parties, we adopt the simple arithmetic secret sharing scheme based on $(n, 0)$-Shamir secret sharing. In other words, we share a scalar value $x \in \mathbb{Z}/Q\mathbb{Z}$ across n parties $\mathcal{P}$, where $\mathbb{Z}/Q\mathbb{Z}$ denotes a ring with Q elements, following the notations of [5]. The sharing of x is defined as $[x] = \{[x]_p\}_{p \in \mathcal{P}}$, where $[x]_p$ is the party p's share of x. The ground-truth value x could be reconstructed from the sum of the shares of each party, i.e. $x = \sum_{p \in \mathcal{P}} [x]_p$.

When parties wish to share a value x, they generate a pseudorandom zero-share that sums to 0. The party that possesses the value adds x to their share in secret. To represent floating-point numbers, we adopt a fixed-point encoding to encode any floating-point number x_F into a fixed-point representation, x. Alternatively, we consider that each x is the result of multiplying a floating-point number x_F by a scaling factor $B = 2^L$ and rounding to the nearest integer,

The IJCAI-2024 AISafety Workshop

* Corresponding author.

leonard.keilin@gmail.com (K. Lin); yasirglani@gmail.com (Y. Glani); luop@tsinghua.edu.cn (P. Luo)

0009-0002-5376-7881 (K. Lin); 0000-0003-0060-4771 (Y. Glani); 0000-0001-6171-3811 (P. Luo)

i.e. $x = \lfloor Bx_F \rceil$. Here L is the precision of the fixed-point encoding. To decode a ground-truth floating-point value x_F from x, we compute as follows: $x_F \approx x/B$.

2.2. Arithmetic Secret Sharing Based MPC

It is noteworthy that arithmetic secret shares are homomorphic and can be used to implement secure MPC, especially in the context of linear computation in most cases.

Addition. The sum of two secret shared values $[x]$ and $[y]$ could be directly computed as $[z] = [x] + [y]$, where each party $p \in \mathcal{P}$ computes $[z]_p = [x]_p + [y]_p$ without multi-party communications.

Multiplication. Two secret shared values $[x]$ and $[y]$ are multiplied using a random Beaver triple [11] generated by the Trusted Third Party (TTP): a triplet $([a], [b], [ab])$. It should be noted that the Beaver triple could be shared in advance by each party. The parties first calculate $[\epsilon] = [x] - [a]$ and $[\delta] = [y] - [b]$. In this way, the $[\epsilon]$ and $[\delta]$ are then revealed to all parties (denoted as $\mathbf{Reveal}(\cdot)$) without compromising information since the ground-truth values a, b remain unknown to each party except for the TTP. The final results could be computed as $[xy] = [c] + \epsilon[b] + [a]\delta + \epsilon\delta$. Algorithm 1 illustrates the multiplication using Beaver triples.

Linear functions. It is possible to implement functions that consist of linear operations by combining additions and multiplications. Common operations in deep learning, such as element-wise product and convolution, are allowed in a linear paradigm.

Nonlinear functions. Due to the inherent infeasibility of nonlinear functions in the standard arithmetic secret-sharing scheme, most works use approximation methods to simulate the outcome of nonlinear functions. In particular, Taylor Expansion, Newton-Rhapson, and Householder methods are commonly used to approximate nonlinear functions using only linear operations. All reciprocal functions, exponential functions, loss functions, kernel functions, and other useful functions in deep learning are calculated this way, for example.

Algorithm 1 Beaver Multiplication $\mathbf{Mul}([x], [y])$

```
Input: Secret-shared inputs $[x], [y]$, Beaver triple $([a], [b], [ab])$.
Output: $[xy]$.
▷ Compute masked values
$[\epsilon] \leftarrow [x - a] = [x] - [a]$
$[\delta] \leftarrow [y - b] = [y] - [b]$
▷ Reveal $\epsilon$ and $\delta$ through one-round communications
$\epsilon \leftarrow \mathbf{Reveal}([\epsilon])$
$\delta \leftarrow \mathbf{Reveal}([\delta])$
return $\epsilon\delta + \epsilon[b] + [a]\delta + [ab]$
```

2.3. Notations

This section summarizes the notations used throughout this work. We denote $[x]$ as a secret sharing of x. $\mathbf{Reveal}([x])$ means that the ground-truth value x is revealed to every party involved in the computation through one-round communications. Since most linear operations are also applicable to element-wise operations and matrix operations, x can also represent a vector, matrix, or even a tensor if there is no confusion and ambiguity.

3. Related Work

To achieve communication-efficient MPC, various approaches have been developed to optimize the communication rounds and the throughput of communication. Ishai and Kushilevitz [12] proposes a new representation of polynomials for round-efficient secure computation, dividing high-degree polynomials into multiple low-degree polynomials that are easy to solve. Mohassel and Franklin [13] performs operations directly on polynomials, such as polynomial multiplication and division. Dachman-Soled et al. [14] improves the evaluation of multivariate polynomials with different variables being held as private inputs by each party. Then, Lu et al. [4] proposes an efficient method for evaluating high-degree polynomials with arbitrary numbers of variables. While the current research has focused on improving the calculation of polynomials, our study aims to develop a communication-efficient and effective MPC system for use in modern deep learning frameworks by leveraging arithmetic tuples computation from Krips et al. [9]. This system is not confined to only computing polynomials within finite rings, as seen in previous studies.

In recent years, several deep learning frameworks that preserve privacy have emerged to enable the secure inference of neural network models. Wagh et al. [8] implements a maliciously secure 3-party MPC protocol from SecureNN [15] and ABY^3 [16]. Knott et al. [5] provides flexible machine-learning APIs with a rich set of functions for secure deep learning. Li et al. [7] presents a fast and performant MPC Transformer inference framework designed to be privacy-preserving. Our low-latency linear MPC implementation is built on top of Knott et al.'s `CrypTen` framework and provides a significant improvement in communication latency.

4. Methodology

4.1. Multivariate Multiplication

Since Beaver triples illustrate how to multiply two variables with pre-shared triplets, a classic multiplication between multiple variables, such as $[xyz]$, requires several rounds of binary multiplication, i.e. $\mathbf{Mul}(\mathbf{Mul}([x], [y]), [z])$. This naive implementation, however, introduces additional communication rounds during the on-the-fly $\mathbf{Reveal}$ process. In general, a n-ary multiplication requires $n - 1$ rounds of communication.

To reduce the communication rounds involved in the multivariate multiplications, the basic binary Beaver triple is extended into a general n-ary Beaver triple. This results in only *one round* of communication required throughout the entire process.

Assume the n inputs could be represented as $\{[x_i]\}_{i=1}^n$. The precomputation and preshared information required by the extended protocol is $\{\mathcal{A}_i\}_{i=1}^n$. Here $\mathcal{A}_1 := \{[a_j]\}_{j=1}^n$ is defined as the set of n auxiliary shared values used to blind the the inputs $\{[x_i]\}_{i=1}^n$, which is also similar to the Beaver's idea. Then $\mathcal{A}_i (i \geq 2)$ is defined as the set of shared degree-i cross-terms consisting of the variables in $\mathcal{A}_1$. For example, $\mathcal{A}_2$ could be defined as $\mathcal{A}_2 := \{[a_i a_j] \mid i \neq j \wedge 1 \leq i, j \leq n\}$, and $\mathcal{A}_3 := \{[a_i a_j a_k] \mid i \neq j \neq k \wedge 1 \leq i, j, k \leq n\}$, and so on. Similar to the construction of $[\epsilon]$ and $[\delta]$ in Section 2.2, we define the difference between the inputs and the masks as $[\delta_i] := [x_i] - [a_i]$. The secret-shared $[\delta_i]$ is then made public across all parties without leakage to the ground-truth value of both $[x_i]$ and $[a_i]$. The improvement of our method originates from the following equation:

$$\begin{aligned}\prod_{i=1}^{n} x_i &= \prod_{i=1}^{n}(\delta_i + a_i)\\ &= \prod \delta_i + \sum_i \delta_i \frac{\prod a_m}{a_i} + \sum_{i,j,i\neq j} \delta_i \delta_j \frac{\prod a_m}{a_i a_j}\\ &\quad + \cdots + \prod a_i.\end{aligned} \tag{1}$$

Here we informally use the fractional representation, such as $\frac{\prod a_m}{a_i}$, to denote the products of all the terms *except for* certain ones. Note that this fractional form does not involve any actual division. Also, each secret-shared term of $[\frac{\prod a_m}{a_i \dots a_j}]$ could be found in the auxiliary sets $\{\mathcal{A}_i\}_{i=1}^n$, which is preshared across all parties.

Adaptation of Equation 1 in secret-sharing scheme is as follows:

$$\begin{aligned}[\prod_{i=1}^{n} x_i] &= \prod \delta_i + \sum_i \delta_i [\frac{\prod a_m}{a_i}] + \sum_{i,j,i\neq j} \delta_i \delta_j [\frac{\prod a_m}{a_i a_j}]\\ &\quad + \cdots + [\prod a_i].\end{aligned} \tag{2}$$

Since Equation 2 is linear to the secret-sharing terms, all communications could be conducted in parallel, i.e. in a single round of communications. In this case, we could simply reveal all the secret-sharing terms in $\{\mathcal{A}_i\}_{i=1}^n$ and compute the sharing of final results in constant complexity. The protocol is formally described in Algorithm 2.

Algorithm 2 Multivariate Beaver Multiplication of n inputs $\mathbf{Mul}([x_1], [x_2], \dots, [x_n])$

```
Input: Secret-shared inputs $\{[x_i]\}_{i=1}^n$, auxiliary sets $\{\mathcal{A}_i\}_{i=1}^n$.
Output: $[\prod x_i]$.
▷ Compute masked values
for $i \in [1, n]$ do
    $[\delta_i] \leftarrow [x_i - a_i] = [x_i] - [a_i]$
end for
▷ Reveal $\delta_i$ through one-round communications
parallel for $i \in [1, n]$ do
    $\delta_i \leftarrow \mathbf{Reveal}([\delta_i])$
end parallel for
▷ Compute results using preshared $\{\mathcal{A}_i\}_{i=1}^n$
return $\prod \delta_i + \sum_i \delta_i [\frac{\prod a_m}{a_i}] + \sum_{i,j,i\neq j} \delta_i \delta_j [\frac{\prod a_m}{a_i a_j}] + \cdots + [\prod a_i]$
```

The total rounds of communications are indeed reduced from $n-1$ to constant 1 when a regular n-ary multiplication is performed, but the overall size of communication data increases from linear to exponential. In a naïve implementation, the data size of n-ary multiplication is only $3(n-1)$ for a total transmission of $n-1$ Beaver triples. As opposed to a multivariate implementation, it is $2^n - 1$ to transmit the auxiliary sets $\{\mathcal{A}_i\}_{i=1}^n$. Therefore, in practice, there is a trade-off between communication latency and throughput.

4.2. Univariate Polynomials

The formal form of univariate polynomials is defined as $P(x) = \sum_{i=0}^n b_i x^i$, where b_i refers to the coefficients of the degree-i term. The use of univariate polynomials enables efficient evaluation and manipulation of polynomial expressions. According to Damgård et al. [17], we can compute all required $[x^i]$ in parallel using multivariate multiplications, then multiply them with corresponding plaintext coefficients. Despite its benefits, this trick has the disadvantage of exponentially increasing the size of transmitted data, which becomes unbearable when the exponent exceeds 5.

This method can be implemented in practice by computing a tuple of base terms and then multiplying the tuple by a certain term iteratively, as in the exponentiating by squaring method or the fast modulo algorithm. In other words, a tuple $\boldsymbol{g} = (1, x, \dots, x^{m-1})$ of size $\|\boldsymbol{g}\| = m$ could be multiplied by $x^{\|\boldsymbol{g}\|}$ repeatedly to iterate all the x^i terms. The overview of the implementation of univariate polynomials is described in Algorithm 3. Note that $\boldsymbol{b}_{s:e}$ is the subvector of $\boldsymbol{b}$ from position s to e.

Algorithm 3 Univariate Polynomial $\mathbf{Poly}([x], \boldsymbol{b})$

```
Input: Secret-shared input $[x]$, coefficients $\boldsymbol{b} = (b_0, b_1, \dots, b_n)$, base terms size $\|\boldsymbol{g}\|$.
Output: $\sum_{i=0}^n b_i [x^i]$.
▷ Construct base terms
parallel for $i \in [1, \|\boldsymbol{g}\|]$ do
    $[x^i] \leftarrow \mathbf{Mul}([x], \dots, [x])$   ▷ multiplied by $[x]$ of $i$ times
end parallel for
$\boldsymbol{g} \leftarrow (1, [x], \dots, [x^{\|\boldsymbol{g}\|-1}])$
▷ Iteratively exponentiating
$t \leftarrow 0$
for $i \in [0, \lfloor \frac{n}{\|\boldsymbol{g}\|} \rfloor - 1]$ do
    s $\leftarrow i \cdot \|\boldsymbol{g}\|$
    e $\leftarrow (i+1) \cdot \|\boldsymbol{g}\|$
    $t \leftarrow t + \boldsymbol{b}_{s:e} \cdot \boldsymbol{g}$
    ▷ Vectorized Beaver Multiplication
    $\boldsymbol{g} \leftarrow [x^{\|\boldsymbol{g}\|}] \cdot \boldsymbol{g}$
end for
return $t$
```

4.3. Nonlinear Approximations

In this section, we take one step further to optimize the commonly used nonlinear functions by leveraging the property of parallelization of our proposed multivariate multiplication.

Exponentiation. Since exponential functions grow in geometrical speed, approximations based on series expansion generally suffer from a significant reduction in accuracy since we do not know the exact value of the input. Consequently, we resort to the naive iterative approximation, which is capable of utilizing multivariate multiplication effectively:

$$e^{[x]} = \lim_{n \to \infty} \left(1 + \frac{[x]}{d^n}\right)^{d^n}.$$

During each iteration, the d-th power of the previous result is calculated. With increasing iteration rounds n, the answer would become closer to the actual results.

Logarithm. The calculation of logarithms relies on the higher-order iterative methods for a better convergence, i.e. Householder methods on $y = \ln x$:

$$[h_n] = 1 - [x]e^{-[y_n]}$$
$$[y_{n+1}] = [y_n] - \sum_{k=1}^{\infty} \frac{1}{k}[h_n^k]$$

Note that the implementation of logarithm is the combination of exponentiation and univariate polynomials. The degree of the polynomials determines the precision of the output.

Reciprocal. The reciprocal function $y = \frac{1}{x}$ is calculated using the Newton-Raphson method with an initial guess y_0:

$$\begin{aligned}[y_{n+1}] &= [y_n](2 - [x][y_n]) \\ &= 2[y_n] - [x][y_n][y_n]\end{aligned}$$

Trigonometry. Trigonometric functions could be treated as the special case of exponentiation with $d = 2$. The sine and cosine functions are calculated in the field of complex numbers:

$$[\sin x] = \mathrm{Im}([e^{ix}])$$
$$[\cos x] = \mathrm{Re}([e^{ix}])$$

Using the above-mentioned nonlinear functions, we can calculate most of the existing loss functions in deep learning, such as the `sigmoid`, `tanh`, and cross-entropy functions. Various other common nonlinear functions, such as the `softmax` function and kernel function, can also be calculated using exponential and reciprocal functions.

4.4. Communication Coalescing

The key to achieving low-latency secret-sharing computation is to reduce the total number of rounds of communications among different parties. While we introduce a latency-friendly implementation of basic math operations, other kinds of communications, such as precision checking, still require an additional but *independent* communication round.

In general, the communication involved in multiple math operations could be abstracted as a communication graph, or strictly, as a communication tree. Accordingly, we observe some independent communications that do not affect downstream results can be deferred and combined into one single round of communication. This process is referred to as *communication coalescing*, and it eliminates unnecessary rounds of communication and improves the utilization of network bandwidth.

5. Security Analysis

The correctness of the multivariate multiplication is trivial based on the observation in Equation 1 and 2. As univariate polynomials are implemented using the same method as the extended fast modulo algorithm, their effectiveness could also be demonstrated by the correctness and security of multivariate multiplication. Coalescing mechanisms only alter the order of communication rounds without modifying the payload, which is also reliable and secure.

Multivariate computations are similarly secure as traditional Beaver multiplications under semi-honest conditions. It is intuitively obvious that since a_i is chosen at random by TTP, the $\delta_i = x_i - a_i$ value is indistinguishable from a random number. Consequently, the disclosure of $[\delta_i]$ does not reveal any critical information regarding x_i. This assumption holds even if multiple parties, except for the TTP, collude.

To clarify the security of multivariate multiplication formally, we denote $[x]_p$ as the secret share of x for party $p \in \mathcal{P}$. The global equations of the multivariate system are as follows:

$$\begin{aligned}\sum_{p\in\mathcal{P}}[x_i]_p &= x_i \\ \sum_{p\in\mathcal{P}}[a_i]_p &= a_i \\ \sum_{p\in\mathcal{P}}[a_i a_j]_p &= a_i a_j \\ &\cdots\cdots \\ \sum_{p\in\mathcal{P}}[a_1 \dots a_n] &= a_1 \dots a_n \\ [x_i]_p - [a_i]_p &= [\delta_i]_p\end{aligned} \tag{3}$$

with known $[\delta_i]_p$ for every $p \in \mathcal{P}$ to each party. From each party's view, these $2^n + 2n - 1$ equations have $\Theta(2^n\|\mathcal{P}\|)$ unknown variables. This indicates the difficulty in determining the exact value of x_i, as shown in [18].

A party's `view` represents all the values it can obtain during its execution. Then the following theorem holds:

Theorem 1. *Let $\{x_i'\}$ and $\{x_i''\}$ be random values. The distribution of the view of each party is identical when $x_i = x_i'$ or $x_i = x_i''$.*

This guarantees the security of multivariate multiplication by ensuring the indistinguishability between the random distribution and the view's distribution.

6. Experiments

6.1. Experimental Setup

As part of our proposed methodology, we use `CrypTen` [5] as the basic MPC deep learning framework, which has already provided naïve implementations of secret-sharing-based computations. In most of our experiments, we use 3-party MPC on CPUs. Additionally, we allow a maximum of 4-ary multiplication as stated in Section 4.1, and we set $d = 3$ for exponentiation and $k = 8$ for logarithm as described in Section 4.3.

To measure the performance, we perform several experiments with deep learning models with different sizes: (a) Linear Support Vector Classification (LinearSVC) with L2 penalty; (b) LeNet [19] with shallow convolutional and linear layers along with ReLU activation functions; (c) ResNet-18 model [20] with multiple convolutional, linear, pooling, and activation layers; (d) Transformer Encoder model [21] with a single multi-head attention layer and BatchNorm [22] in place of LayerNorm [23]. We employ several datasets for classification tasks with appropriate adaption to specific models, including MNIST [19], CIFAR-10 [24], ImageNet[25], and Sentiment140 [26] datasets.

Table 1
Communication Latency and Data with Different Network Settings. Latency and data are measured in milliseconds and MiBs.

Model	Dataset	t_{comp}	Data Size		t_{comm} (N_{low})		t_{comm} (N_{med})		t_{comm} (N_{high})	
			Naïve	Ours	Naïve	Ours	Naïve	Ours	Naïve	Ours
LinearSVC	MNIST	0.032	0.022	0.024	0.082	0.059	0.604	0.568	4.339	4.026
LeNet	CIFAR-10	0.900	38.562	41.647	1.373	1.182	7.673	6.983	53.010	47.868
ResNet-18	ImageNet	110.447	11571.294	12612.711	219.541	185.293	1681.685	1407.570	~11 760	~9 870
Transformer	Sentiment140	7.824	771.787	893.421	12.190	9.715	78.624	61.877	559.645	421.413

Each of our experiments is conducted in a simulated multi-node environment using Docker. TTP is conducted in an independent environment separate from the normal parties. To manually simulate different network environments concerning bandwidth and latency, we utilize the `docker-tc` tool to adjust the docker network settings accordingly.

6.2. Metrics

To provide a comprehensive evaluation of our proposed method, we adopt metrics from a variety of perspectives.

- t_{comp}: The computational time cost for evaluating a single data sample in one round.
- t_{comm}: The time cost of communication associated with evaluating a single data sample in one round.
- **Size of Transmission Data**: The size of transmitted network packets when evaluating a single data sample in one round.
- **Accuracy**: The classification accuracy when evaluated on a particular dataset.

6.3. Latency & Throughput

To assess the efficiency of our proposed method, we simulate networks with different network latencies: (a) network N_{low} with 0.1ms latency, (b) network N_{med} with 5ms latency, (c) and network N_{high} with 40ms latency. All of these networks have a bandwidth of 1Gbps. Our simulated multi-node settings include 3 nodes with an additional TTP by default.

As shown in Table 1, the computation cost of each model is negligible in medium and high latency network settings in comparison to the communication cost. Therefore, we will focus only on the communication costs associated with our proposed method.

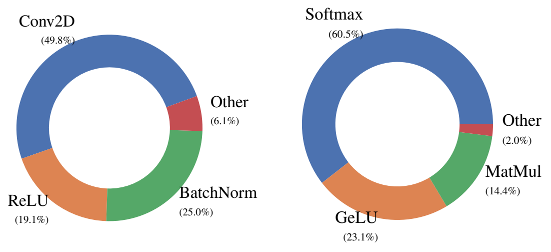
Compared to the naïve method implemented by `CrypTen`, our method illustrated in Section 4.1 remains close since it does not introduce a substantial amount of additional communication payload if the maximum number of input variables is set appropriately. For instance, a 3-ary or 4-ary multiplication would not produce a significant increase in the total size of communications.

It is noteworthy that our proposed method reduces the communication cost in every network setting as compared to the naïve implementation of MPC. Overall, we achieve an improvement of $10 \sim 20\%$, which shows significant enhancement in the performance of high-latency environments for practical purposes.

Furthermore, we observe that our proposed method behaves differently with neural models with different architectures. Figure 1 illustrates the communication occupation percentage of ResNet basic blocks and Attention blocks. As can be seen, the attention mechanism is constrained by its communication bottleneck in `Softmax` operation, while CNN is constrained by its communication via convolutional operations. Considering that our method makes an improved optimization for nonlinear functions, attention-based models show a significant improvement in latency, with almost a 25% improvement. Additionally, this explains the limited improvement of only $8 \sim 15\%$ in traditional machine-learning models and CNN-based models.

Table 2
Classification Accuracy using Different Methods.

Model	Dataset	Accuracy (%)		
		Origin	Naïve	Ours
LinearSVC	MNIST	100.00	100.00	100.00
LeNet	CIFAR-10	100.00	100.00	100.00
ResNet-18	ImageNet	69.30	61.58	60.10
Transformer	Sentiment140	59.87	58.55	57.74



(a) ResNet Basic Block (b) Attention Block

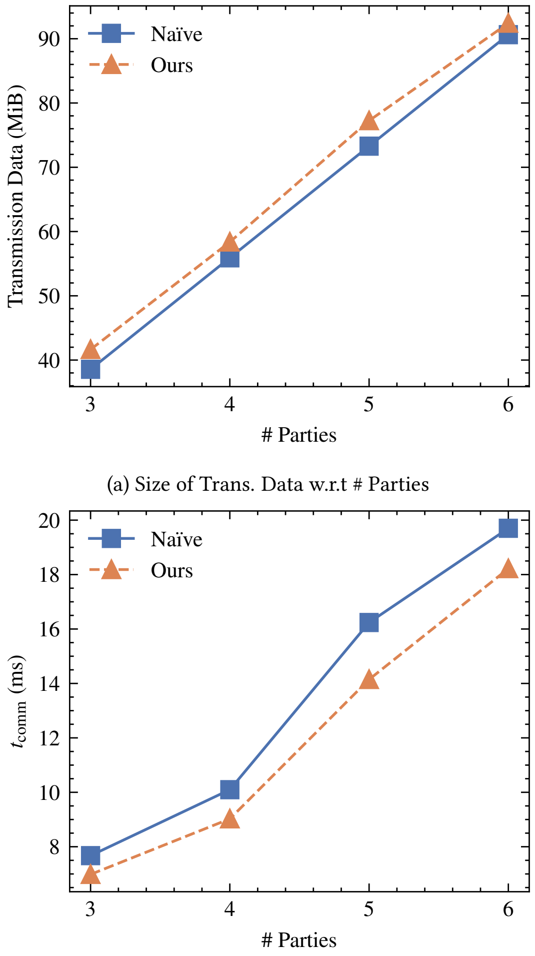
Figure 1: Communication percentage of different models.

6.4. Evaluation

In this section, we examine the side effects and factors associated with the basic settings, such as the downstream tasks' accuracy, the number of parties involved, and the trade-off between network latency and bandwidth.

To evaluate the drop in accuracy, we compare our method with both the original baseline and the naïve implementation without a low-latency design. Figure 2 shows that, in relatively small scenarios, both the naïve implementation and our methods are capable of achieving perfect performance as the baselines. Nevertheless, both MPC-based implementations obtain lower accuracy in complex scenarios than the baseline, while our methods perform slightly worse than the naïve implementation. We hypothesize that the multivariate multiplication introduces additional precision requirements, which in turn reduces accuracy.

The throughput and latency of MPC-based methods are



(a) Size of Trans. Data w.r.t # Parties

(b) Latency w.r.t # Parties

Figure 2: Transimission data and latency of naïve and our proposed methods when a different number of parties are involved. The experiment is conducted using the LeNet model on CIFAR-10 using a medium latency network.

also affected by the number of parties involved in the computation. From Figure 2, it can be seen that the communication data size of both methods increases linearly as the number of parties involved increases. There is, however, a tendency for the latency to be worse when there are more parties involved.

Moreover, Figure 3 illustrates how network bandwidth affects communication costs. When sufficient bandwidth is available, our method can still optimize the network latency. It is important to note, however, that when the bandwidth becomes the bottleneck, our method would not be any more effective in reducing the overall costs of communication. This indicates that bandwidth remains an important factor in a multi-node MPC setting, especially as the number of nodes in use grows.

7. Discussion

Since the proposed multivariate multiplication is based on a finite ring, it is likely to have precision issues that lead to incorrect results. Fortunately, a loss in precision would not significantly affect the overall performance of deep learning, since the loss could be interpreted as random noise and distortion in the input data.

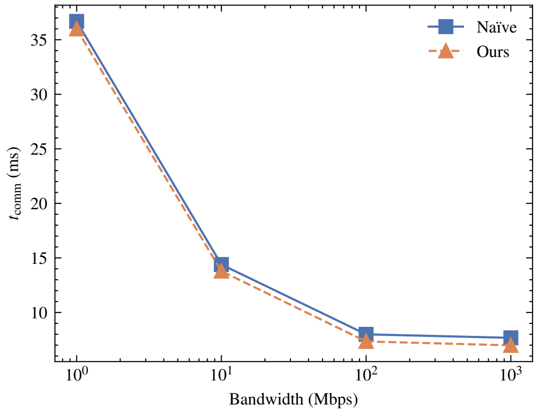


Figure 3: Latency of naïve and our proposed methods concerning different network bandwidth. The experiment is conducted using the LeNet model on CIFAR-10 using a medium latency network.

Moreover, our proposed method is only applicable to functions that are based on linear MPC operation. To avoid heavy communications, a modern MPC-based deep learning framework would also involve other protocols, such as Homomorphic Encryption [27], Garbled Circuit [28], and Function Secret Sharing[29]. Though these works may have less communication, our approach could still be seamlessly integrated with the current secret-sharing framework and achieve a latency improvement of ~20% without adding excessive computational workload.

8. Conclusion

This study proposes a secret-sharing-based MPC method for enhancing the linear computation required in deep learning through increased communication utilization. By utilizing the multivariate multiplication and communication coalescing mechanisms, we can reduce the number of unnecessary communication rounds during the execution of both linear and nonlinear deep learning functions. In our experiments, we demonstrate that our proposed methods achieve an overall improvement in latency of $10 \sim 20\%$ when compared to the naïve MPC implementation. Additionally, it indicates that throughput and downstream task performance are comparable to naïve implementations, which demonstrate the method's validity and efficiency. We hope that this work will inspire future improvements in privacy-preserving deep learning techniques and lead to more practical MPC applications.

Acknowledgments

This work is supported by the National Key R&D Program of China under grant (No. 2022YFB2703001).

References

[1] I. Damgård, V. Pastro, N. Smart, S. Zakarias, Multiparty computation from somewhat homomorphic encryption, in: R. Safavi-Naini, R. Canetti (Eds.), CRYPTO 2012, Springer Berlin Heidelberg,

Berlin, Heidelberg, 2012, pp. 643–662. doi:10.1007/978-3-642-32009-5_38.

[2] M. Keller, Mp-spdz: A versatile framework for multi-party computation, in: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, CCS '20, Association for Computing Machinery, New York, NY, USA, 2020, p. 1575–1590. doi:10.1145/3372297.3417872.

[3] X. Liu, L. Xie, Y. Wang, J. Zou, J. Xiong, Z. Ying, A. V. Vasilakos, Privacy and security issues in deep learning: A survey, IEEE Access 9 (2021) 4566–4593. doi:10.1109/ACCESS.2020.3045078.

[4] D. Lu, A. Yu, A. Kate, H. Maji, Polymath: Low-latency mpc via secure polynomial evaluations and its applications, Proceedings on Privacy Enhancing Technologies 2022 (2021). doi:10.2478/popets-2022-0020.

[5] B. Knott, S. Venkataraman, A. Hannun, S. Sengupta, M. Ibrahim, L. van der Maaten, Crypten: Secure multi-party computation meets machine learning, NIPS 34 (2021) 4961–4973. doi:10.48550/arXiv.2109.00984.

[6] S. Tan, B. Knott, Y. Tian, D. J. Wu, CryptGPU: Fast privacy-preserving machine learning on the gpu, in: IEEE S&P, 2021. doi:10.1109/SP40001.2021.00098.

[7] D. Li, H. Wang, R. Shao, H. Guo, E. Xing, H. Zhang, Mpcformer: fast, performant and private transformer inference with mpc, in: The 11th International Conference on Learning Representations, 2023. doi:10.48550/arXiv.2211.01452.

[8] S. Wagh, S. Tople, F. Benhamouda, E. Kushilevitz, P. Mittal, T. Rabin, Falcon: Honest-majority maliciously secure framework for private deep learning, Proceedings on Privacy Enhancing Technologies 2021 (2020) 188 – 208. doi:10.2478/popets-2021-0011.

[9] T. Krips, R. Küsters, P. Reisert, M. Rivinius, Arithmetic tuples for mpc, Cryptology ePrint Archive (2022). URL: https://eprint.iacr.org/2022/667.

[10] A. Shamir, How to share a secret, Commun. ACM 22 (1979) 612–613. doi:10.1145/359168.359176.

[11] D. Beaver, Efficient multiparty protocols using circuit randomization, in: J. Feigenbaum (Ed.), CRYPTO 1991, Springer Berlin Heidelberg, Berlin, Heidelberg, 1992, pp. 420–432. doi:10.1007/3-540-46766-1_34.

[12] Y. Ishai, E. Kushilevitz, Randomizing polynomials: A new representation with applications to round-efficient secure computation, in: Proceedings 41st Annual Symposium on Foundations of Computer Science, 2000, pp. 294–304. doi:10.1109/SFCS.2000.892118.

[13] P. Mohassel, M. Franklin, Efficient polynomial operations in the shared-coefficients setting, in: M. Yung, Y. Dodis, A. Kiayias, T. Malkin (Eds.), PKC 2006, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 44–57. doi:10.1007/11745853_4.

[14] D. Dachman-Soled, T. Malkin, M. Raykova, M. Yung, Secure efficient multiparty computing of multivariate polynomials and applications, in: J. Lopez, G. Tsudik (Eds.), Applied Cryptography and Network Security, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 130–146. doi:10.1007/978-3-642-21554-4_8.

[15] S. Wagh, D. Gupta, N. Chandran, Securenn: Efficient and private neural network training, Cryptology ePrint Archive (2018). URL: https://eprint.iacr.org/2018/442.

[16] P. Mohassel, P. Rindal, Aby3: A mixed protocol framework for machine learning, in: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS '18, Association for Computing Machinery, New York, NY, USA, 2018, p. 35–52. doi:10.1145/3243734.3243760.

[17] I. Damgård, M. Fitzi, E. Kiltz, J. B. Nielsen, T. Toft, Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation, in: S. Halevi, T. Rabin (Eds.), Theory of Cryptography, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 285–304. doi:10.1007/11681878_15.

[18] G. Couteau, A note on the communication complexity of multiparty computation in the correlated randomness model, in: EUROCRYPT 2019, Springer-Verlag, Berlin, Heidelberg, 2019, p. 473–503. doi:10.1007/978-3-030-17656-3_17.

[19] Y. Lecun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (1998) 2278–2324. doi:10.1109/5.726791.

[20] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: CVPR, 2016, pp. 770–778. doi:10.1109/CVPR.2016.90.

[21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, I. Polosukhin, Attention is all you need, in: I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), NIPS, volume 30, Curran Associates, Inc., 2017. doi:10.5555/3295222.3295349.

[22] S. Ioffe, C. Szegedy, Batch normalization: accelerating deep network training by reducing internal covariate shift, in: Proc. 32nd Int. Conf. Machine Learning - Volume 37, ICML'15, JMLR.org, 2015, p. 448–456. doi:10.5555/3045118.3045167.

[23] J. L. Ba, J. R. Kiros, G. E. Hinton, Layer normalization, 2016. arXiv:1607.06450.

[24] A. Krizhevsky, G. Hinton, et al., Learning multiple layers of features from tiny images (2009).

[25] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, Imagenet: A large-scale hierarchical image database, in: CVPR, 2009, pp. 248–255. doi:10.1109/CVPR.2009.5206848.

[26] A. Go, R. Bhayani, L. Huang, Twitter sentiment classification using distant supervision, CS224N project report, Stanford 1 (2009) 2009.

[27] C. Gentry, Fully homomorphic encryption using ideal lattices, in: Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing, STOC '09, Association for Computing Machinery, New York, NY, USA, 2009, p. 169–178. doi:10.1145/1536414.1536440.

[28] A. C. Yao, Protocols for secure computations, in: 23rd Annual Symposium on Foundations of Computer Science (sfcs 1982), 1982, pp. 160–164. doi:10.1109/SFCS.1982.38.

[29] E. Boyle, N. Gilboa, Y. Ishai, Function secret sharing, in: E. Oswald, M. Fischlin (Eds.), EUROCRYPT 2015, Springer Berlin Heidelberg, Berlin, Heidelberg, 2015, pp. 337–367. doi:10.1007/978-3-662-46803-6_12.