Journal of Cryptographic Engineering (2025) 15:2
https://doi.org/10.1007/s13389-024-00365-1

RESEARCH ARTICLE

MLFormer: a high performance MPC linear inference framework for transformers

Siqi Liu[1,6] · Zhusen Liu[2] · Donglong Chen[1,3] · Wangchen Dai[4] · Lu Zhou[5] · Zhe Liu[3] · Ray C. C. Cheung[6] · Çetin Kaya Koç[5,7]

Received: 18 January 2024 / Accepted: 21 October 2024

Abstract

Transformer-based models are widely used in natural language processing tasks, and their application has been further extended to computer vision as well. In their usage, data security has become a crucial concern when deploying deep learning services on cloud platforms. To address these security concerns, Multi-party computation (MPC) is employed to prevent data and model leakage during the inference process. However, Transformer model introduces several challenges for MPC computation, including the time overhead of the Softmax (normalized exponential) function, the accuracy issue caused by the "dynamic range" of approximated division and exponential, and the high memory overhead when processing long sequences. To overcome these challenges, we propose MLformer, an MPC-based inference framework for transformer models based on Crypten Knott et al. (Adv Neural Inf Process Syst 34: 4961–4973, 2021), a secure machine learning framework suggested by Facebook AI Research group, in the semi-honest adversary model. In this framework, we replace the softmax attention with linear attention, which has linear time and memory complexity with input length. The modification eliminates the softmax function entirely, resulting in lower time and memory overhead. To ensure the accuracy of linear attention, we propose the scaled linear attention to address the dynamic range issue caused by the MPC division used and a new approximate division function is proposed to reduce the computational time of the attention block. Furthermore, to improve the efficiency and accuracy of MPC exponential and reciprocal which are commonly used in transformer model, we propose a novel MPC exponential protocol and first integrate the efficient reciprocal protocol Bar-Ilan and Beaver (in Proceedings of the 8th annual ACM symposium on principles of distributed computing, pp. 201–209, 1989) to our framework. Additionally, we optimize the computation of causal linear attention, which is utilized in private inference of auto-regression tasks, using our novel CUDA kernel functions. All the proceeding optimizations contribute to the construction of a more accurate and efficient framework. The experimental results demonstrate that our framework achieves comparable accuracy with reduced inference time and GPU memory overhead compared to the original transformer model. The speedup reaches 78.79% compared to traditional private transformer with input length of 1024 patches.

Keywords Multi-party computation · Linear transformer · Private inference · Parallel processing · GPU

✉ Donglong Chen
donglongchen@uic.edu.cn

✉ Wangchen Dai
w.dai@my.cityu.edu.hk

Siqi Liu
siqiliua@163.com

[1] Guangdong Provincial Key Laboratory of IRADS, BNU-HKBU United International College, Zhuhai 519000, China

[2] Hangzhou Innovation Institute of Beihang University, Hangzhou 311121, China

[3] Zhejiang Lab, Hangzhou 310000, China

[4] Sun Yat-sen University, Shenzhen 518107, China

[5] Nanjing University of Aeronautics and Astronautics, Nanjing 210000, China

[6] City University of Hong Kong, Hong Kong 310000, China

[7] Iğdır University, Turkey, and University of California Santa Barbara, Santa Barbara, USA

Published online: 19 November 2024

1 Introduction

Since the transformer was first proposed by Vaswani et al. [3] for neural machine translation, it has been broadly applied in Natural Language Processing (NLP) and computer vision tasks [4, 5]. With the emergence of cloud services providing powerful resources and offering benefits such as availability and cost-effectiveness, deep learning models, including the transformer-based models, are increasingly migrated to the cloud. The computing paradigm works as follows: first, the company shares its trained model with the cloud servers while the user provides individual data; subsequently, the cloud server processes the inference of the trained model on the user-provided data. However, this computing paradigm raises concerns about data security: parties may be curious, malicious or corrupted, resulting in the unauthorized disclosure of confidential models and data. For instance, ChatGPT, the most popular artificial intelligent application recently, supported by GPT−3.5 (Generative Pre-training Transformer [6]) takes plaintext inputs and generates human-like answers for users. The text inputs in user requests can potentially leak confidential information, which might lead to the identification of the individual user. Therefore, addressing the security issue is essential for cloud services of transformer-based models.

Researchers have proposed Privacy-Preserving Machine Learning (PPML) to address the security issues raised by machine learning applications. Various technologies have been used to implement PPML, including trusted execution environments (TEE) [7], homomorphic encryption (HE) [8], and secure multi-party computation (MPC) [9]. TEEs rely on trusted hardware to prevent physical attacks, provided that the hardware implementation is free of any bugs. MPC and HE provide strong security guarantees based on modern cryptography, but HE typically incurs more performance overhead than MPC. MPC allows collaborating parties to conduct computations securely without revealing their secret data to one another within a secure cryptographic framework. In addition, even if some parties are compromised, security is not entirely broken due to the protection provided by MPC protocols. This paper specifically focuses on the application of MPC to protect transformer-based models.

Previous research on MPC-based private inference has mainly concentrated on Convolutional Neural Networks (CNNs) that are primarily deployed in computer vision tasks. For instance, SecureML [10] employed a mixed protocol consisting of Arithmetic, Boolean, and Yao sharing [11] to implement two-party PPML. CrypTFlow [12] enabled private inferences for large datasets such as ImageNet using large-scale models like ResNet family [13] and VGG-16 [14]. However, the field of private inference for Transformer-based models has not been extensively studied. In a recent study [15], the private inference of transformer was processed using CrypTen [1], a framework that integrates secure MPC primitives with tensor computations and modular networks in machine learning. CrypTen is designed for all types of neural network structures, rather than being tailored specifically for transformers. The structure between CNNs and Transformer-based models is significantly different, and therefore, Transformer models present new challenges for MPC-based private inference. These challenges include the softmax function's time overhead, the dynamic range of approximated reciprocal and exponential, and the memory overhead, particularly when dealing with long input sequences.

The primary challenge of implementing MPC-based private inference for Transformer models is the significant runtime overhead incurred by the softmax function. While runtime overhead of non-linear functions like ReLU can be significant when CNNs are implemented in MPC, the softmax function accounts for an even larger portion, around half of the overall inference runtime when transformers are implemented in MPC, according to [15]. The repeated use of the softmax function at each layer of a Transformer model further compounds the performance penalty. Additionally, to ensure numerical stability, the softmax function commonly needs to compute the max, a computationally expensive operation in MPC.

In addition to the runtime overhead, another challenge of MPC-based inference for Transformer models is the dynamic range issue of reciprocal and exponential in MPC. Reciprocal and exponential are both significant and indispensable computations in Transformer. However, the complex non-linear functions like reciprocal and exponential are often approximated as polynomial functions in MPC. These approximations introduce "dynamic range", where the accuracy is acceptable only when inputs fall inside a particular range. Substantial numerical errors will occur if the inputs are outside this range, leading to a potential loss of model accuracy.

Another issue that Transformer models have to address is the high memory overhead when processing long sequences. This issue is mainly caused by the use of softmax-attention, whose memory complexity is quadratic with the input size(i.e. $O(N^2)$, where N refers to the input length). Similar to the time overhead issue, the repeated use of the softmax function at each layer contributes to the increase of the memory overhead. The quadratic complexity of the memory overhead introduces a significant impediment to processing long sequences, even for plaintexts.

To overcome the challenges discussed above, researchers have proposed various efficient transformers. For instance, Linformer [16] suggested to shorten sequences before computing self-attention by projecting the keys and values and Nystromformer [17] approximated softmax attention utilizing the Nystrom method [18]. Though reducing the time and

memory overhead of processing long input, both methods still rely on the softmax function to compute self-attention.

In this paper, an MPC-based inference framework named MLformer based on CrypTen [1] is built for privacy-preserving inference of transformer-based models. In the framework, we suggest using linear attention [19] of linear time and memory complexity with input length as a replacement for softmax attention. Without softmax function, linear attention requires no computation of max in MPC, resulting in lower runtime overhead. We also propose scaled linear attention to handle the accuracy issue caused by the dynamic range of MPC division used in linear attention. At the same time, we design a new approximate division function working with scaling to reduce the computational time of linear attention. Furthermore, we are the first to implement the reciprocal protocol proposed by BarIlan and Beaver [2] on an MPC-based deep learning framework. And we also introduce a novel protocol for MPC exponential. Both the new protocols for reciprocal and exponential show satisfactory performance in accuracy and time overhead, so as to improve the accuracy of the model result and reduce the time overhead of the inference process. We additionally optimize and accelerate the computation of causal linear attention, which is utilized in private inference of autoregression tasks, using our novel CUDA kernel functions. All the preceding optimizations contribute to the construction of a more accurate and faster framework.

The following contributions are made in this paper:

- An MPC-based inference framework, named MLformer, is proposed for transformer-based models with linear attention instead of softmax attention, resulting in linear time and memory complexity with input length.
- We propose “scaled linear attention" to address the dynamic range issue of private division used in the attention block. Meanwhile, an approximate function for private division is proposed by linear regression working with scaling, which achieves a 6.02× speedup.
- We propose a novel protocol for MPC exponential and implement an efficient protocol for MPC reciprocal, both of which demonstrate excellent performance in terms of accuracy, dynamic range, and time overhead. Our novel exponential protocol of 4 iterations achieves a speedup of 12.61% and the efficient reciprocal achieves a 43.05× speedup.
- The computational efficiency of causal linear attention in private inference is improved by utilizing our novel CUDA kernel functions, achieving a speedup of 94.48%. The experiments on ImageNet demonstrate that our framework works faster than previous private transformers with less GPU memory and it has comparable accuracy with original transformer on long sequence inference tasks. The speedup reaches 78.79% compared to the traditional private transformer with input length of 1024 patches.

2 Related work

There is a substantial corpus of previous studies that have focused on MPC-based privacy-preserving machine learning. For example, several frameworks such as SecureML [10], Chameleon [20], CrypTFlow [12], CryptGPU [21] and CrypTen [1] have been proposed for private training and inference based on MPC. A mixed protocol consisting of arithmetic, Boolean, and Yao sharing [11] was used by SecureML [10] to make PPML more practical and efficient in a two-server framework. Chameleon [20] introduced a semi-trusted third party for generating “multiplication triples". CrypTFlow [12] enabled private inference for large datasets such as ImageNet using large-scale models like ResNet family [13] and VGG-16 [14]. In our work, we utilize CrypTen as the base MPC framework. CrypTen [1] is a framework that integrates secure MPC primitives such as beaver triples, arithmetic and binary sharing with tensor computations and modular networks in machine learning based on Pytorch. CryptGPU [21] is a framework based on CrypTen that focuses on 3-party inference using 2-out-of-3 replicated sharing. CrypTFlow, CryptGPU, and CrypTen all enable the implementation of GPU on inference. Dong et al. [22] introduced FLEX BNN, which leverages flexible and small bit-width conversion approach to accelerate the private inference for Binary Neural Network. Optimizations for private inference have also been suggested by ParSecureML [23], and the optimized comparison protocol [24].

However, limited research has been conducted on private inference for transformer models. Resende et al. [25] focused on the text classification task in NLP and developed a Naive Bayes Classifier based on MPC. Feng et al. [26] developed SecureNLP, a framework based on MPC mainly for recurrent neural network (RNN)-based sequence-to-sequence model for neural machine translation tasks. Meanwhile, Wang et al. [15] attempted to protect transformer-based models using CrypTen framework. The-X [27] provided a method of homomorphic encryption (HE) for private inference of transformer-based models. In this work, we focus on the use of MPC for private inference of transformers and aim to identify solutions for the challenges they present.

Several previous studies have attempted to optimize the time and memory complexity of transformers through alternative implementations. For instance, Child et al. [28] proposed the sparse factorization of the attention matrix to reduce complexity from quadratic to $O(N\sqrt{N})$. More recently, Kitaev et al. introduced Reformer [29], which uses locality-sensitive hashing (LSH) to reduce complexity to $O(N \log N)$. In their method, fewer dot products are com-

puted under the constraint that the keys should be kept the same as the queries. Consequently, Reformer cannot be utilized to handle tasks where the keys and the queries are different. Linformer [16] reduces the time and memory overhead by using low-rank assumption to shorten value and key. Nystromformer [17] makes use of the Nystrom method to approximate softmax attention, which divides attention into three parts. Both Linformer and Nystromformer employ the softmax function when computing attentions, which is expensive in MPC. Linear transformer proposed in study [19] reduces the time and memory complexity to O(N) using linear attention to approximate softmax attention, which eliminates the use of softmax function. By not using the softmax function, the linear transformer gains an advantage in processing long sequences compared to other alternative transformers.

3 Background

3.1 Transformer-based model

Transformers, introduced by Vaswani et al. [3] at Google Brain in 2017, are widely utilized in Natural Language Processing (NLP) and computer vision (CV). Transformers adopt the self-attention mechanism, which assigns different weights to input components based on their importance. Compared to recurrent neural networks(RNNs), transformers perform better parallelism, allowing for each component of the input to be processed in parallel, instead of in sequential. This advantage enables the training of large datasets, leading to the development of pre-trained systems like BERT [30] (Bidirectional Encoder Representations from Transformers) and GPT (Generative Pre-trained Transformer [6]).

Other transformer-based models include XLM [31] and Vision Transformer (ViT) [4] employed in image classification tasks. The model architectures of XLM and ViT are shown in Fig. 1a and b, respectively. XLM consists of an input embedding layer, followed by several layers of Transformers. ViT is comprised of a fully-connected classifier on the top, multiple transformer layers in the middle, and normalization layers at the beginning. Though input processing layers may differ among models, Transformer layers share a common structure. Figure 1c depicts the computation decomposition of a Transformer Encoder, which is comprised of two main compute building blocks: the multi-head attention block and the feed-forward block. Figure 1d presents a detailed breakdown of the computations involved in a multi-head attention layer. Figure 1e illustrates the comprehensive calculations underlying Scaled dot-Product attention.

In many transformer models, the inputs are sentences that can be broken down into multiple words. Specifically, for Vision Transformer (ViT) models, the inputs are typically images that can be segmented into multiple patches, similar to how a sentence is divided into words. Throughout this paper, we consistently refer to each element of an input sequence as a "token". To provide position information, a position vector is added to the token vector. We use x to represent the input and x_i to denote the token vector. Assuming the number of tokens is N, x is composed of N x_i that are packed together. As shown in Fig. 1d, the inputs of the multi-head attention are Q ("queries"), K ("keys"), and V ("values"), which are derived from x. They are computed as Eq. 1 shows:

$$Q = xW_Q \tag{1a}$$
$$K = xW_K \tag{1b}$$
$$V = xW_V \tag{1c}$$

In the Eq. 1a–c, Q, K, and V are calculated by performing three linear transformations on x with the weight matrices W_Q, W_K, and W_V, respectively.

"Scaled Dot-Product Attention" can be described as follows:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \tag{2}$$

where d_k represents the dimension of keys. First, the dot products of the queries Q with all keys K are calculated and then it is scaled by d_k. Next, a softmax function is applied to the dot product to obtain the weights on the values. Note that the softmax function is applied row-wise to QK^T. Finally, attention is computed by multiplying values V and its weights.

The feed-forward block in Fig. 1c contains a two-layer linear operation that can be described as follows:

$$FF(x_{attn}) = Linear(Activation(Linear(x_{attn}))), \tag{3}$$

where ReLU is the activation function commonly used.

3.2 Secure multi-party computation

Multi-party Computation(MPC) is a foundation of cryptography and a significant technology for privacy-preserving applications. The implementations of secure multiparty computation typically involve one or more of the following techniques: secret-sharing, garbled circuits, and oblivious transfer. In this work, we make use of secret-sharing approaches to achieve MPC, and will focus on introducing MPC protocols that are based on secret-sharing in this subsection.

Arithmetic sharing and binary sharing are two primary secret sharing formats that are widely used in MPC for sharing an operand x. $[x]_i$ and $\langle x\rangle_i$ refer to the shares of x in arithmetic and binary sharing, respectively. Assuming a sce-

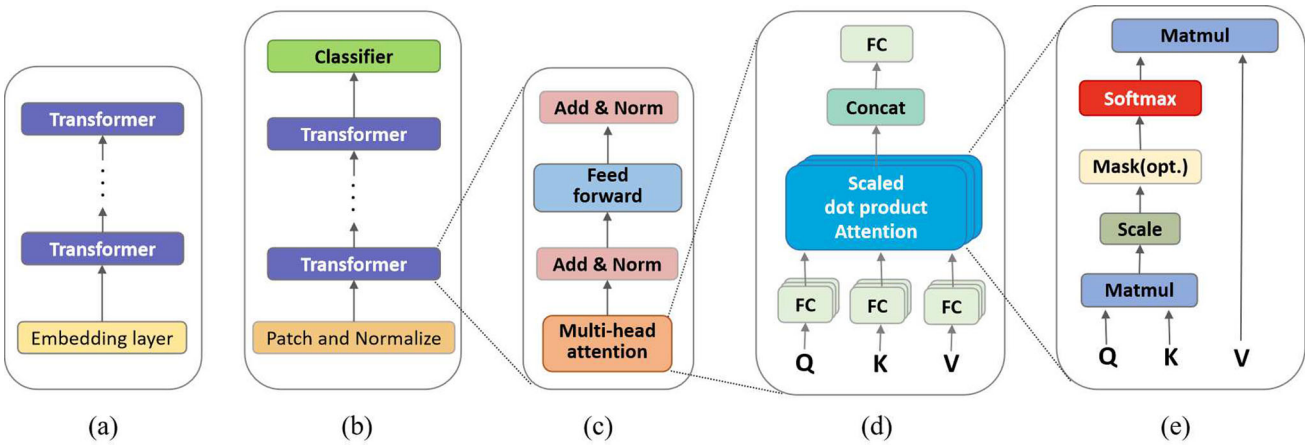

Fig. 1 Computation decomposition of Transformer-based model: **a** structure of XLM; **b** structure of a ViT; **c** structure of a transformer; **d** multi-headed attention; **e** scaled dot-product attention

nario of two-party computation, the share and reconstruct stages can be demonstrated as follows:

Share(x)

- Arithmetic sharing: x can be divided to two additive secret shares $[x]_1$ and $[x]_2$, where $[x]_1 = r$ and $[x]_2 = x - r$. The random variable r is sampled from a uniform distribution.
- Binary sharing: x can be divided to two binary secret shares $\langle x\rangle_1$ and $\langle x\rangle_2$, where $\langle x\rangle_1 = r$ and $\langle x\rangle_2 = x \oplus r$. The random variable r is sampled from a uniform distribution as well.

Reconstruct$([x]_i/\ \langle x\rangle_i,\ i = 1, 2)$

- Arithmetic sharing: x can be reconstructed by adding $x = [x]_1 + [x]_2$
- Binary sharing: x can be reconstructed by an XOR computation: $x = \langle x\rangle_1 \oplus \langle x\rangle_2$

We mainly consider an "Outsourcing Inference" scenario: when the pre-trained ML(machine learning) models are deployed on n servers, the client can securely share their feature inputs across these servers. The servers are able to jointly and secretly compute the result of inference and return it to the client. Under this scenario, the MPC protocol enables a client to confidentially share its operands (a vector, matrix, etc.) among various "untrusted parties" (MPC servers) without revealing any information about the original operands. In secret sharing, each MPC server receives its unique secret share, which is the only data viewable and manipulable by the server. Once the local MPC computations are completed, the MPC servers provide the outcomes to the client. To obtain the final plaintext result, the client combines the results from various MPC servers using operations like summation or XOR.

Figure 2 depicts an example of a two-server secure computation in outsourcing inference where the client intends to compute $z = xy$. Both x and y are in arithmetic sharing format. Firstly, the client distributes the relevant secret shares $[x]_i$ and $[y]_i$ $(i = 1, 2)$ to the MPC servers. Next, the servers conduct local computations on their individual shares. Finally, the client receives the outputs from $Server1$ and $Server2$, which can be combined to obtain the result z. During the local computation, the servers can communicate with each other with the protection of MPC protocols.

For operations such as matrix multiplications and convolutions that are typically used in modern machine learning models, arithmetic secret sharing is especially well-suited. On the other hand, for other commonly used functions such as rectified linear units (ReLU), binary secret sharing is required for evaluation. The homomorphic properties of arithmetic and binary secret sharing make them useful for secure computation implementation.

3.3 CrypTen

CrypTen is an MPC framework designed for the private inference and training of machine learning models. The framework utilizes both arithmetic and binary secret sharing, as well as the conversion between them. Private addition and multiplication are the fundamental building blocks for computations in CrypTen. Linear and non-linear functions are implemented in different manners.

Linear functions: In CrypTen, linear operations such as matrix products, dot products, and convolutions are computed by breaking them down into combinations of private additions and multiplications.

Non-linear functions: CrypTen employs standard approximations that only require private additions and multiplications to construct non-linear functions. It employs a limit

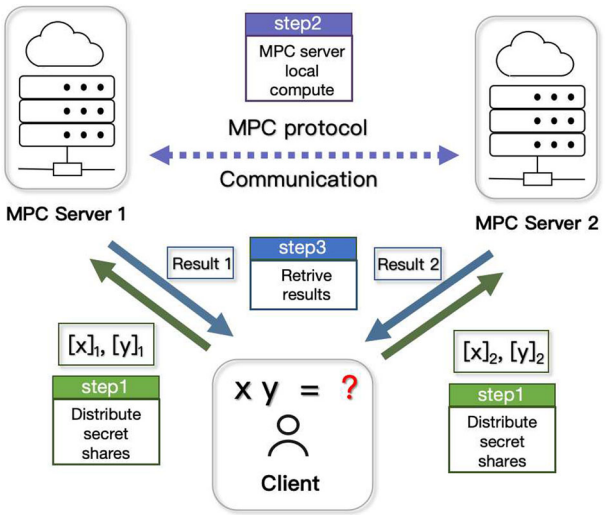

Fig. 2 Two server example of secure computation in outsourcing inference

approximation to evaluate exponential, Householder iterations to evaluate logarithms, and Newton-Rhapson iteration to evaluate reciprocal. These techniques enable CrypTen to implement fundamental machine learning model functions such as the sigmoid, softmax, and logistic-loss functions, along with their corresponding gradients.

The MPC computations are demonstrated in detail in the following subsections.

3.3.1 MPC addition

In an arithmetic secret sharing operation to add two values, x and y, $[z] = [x]+[y]$, each party p has their respective shares of $[x]$ and $[y]$: $[x]_p$ and $[y]_p$. Subsequently, each participant $p \in P$ performs local summation: $[z]_p = [x]_p + [y]_p$ to obtain the shares of the sum z.

3.3.2 MPC multiplication

Beaver triples [32] are used to implement the multiplication of two values x and y in arithmetic sharing: $[z] = [x][y]$. The triples are produced offline by a trusted third party (TTP) or a trusted first party (TFP). By utilizing TFP, the triples are generated by one of the parties participating in the MPC process through additive homomorphic encryption or other techniques, eliminating the need for a third party. Although CrypTen provides both TFP and TTP methods, currently only TFP is feasible in practice. Values of a Beaver triple (([a], [b], [c])) in arithmetic sharing must satisfy the property of $c = ab$. a and b are randomly generated from the ring Z/QZ. After obtaining the beaver triple, the parties calculate $[\epsilon] = [x] - [a]$ and $[\delta] = [y] - [b]$. $[\epsilon]$ and $[\delta]$ are then decrypted as ϵ and δ and transmitted to all the parties in one communication round. No information can be revealed as a and b were drawn at random. Thereafter, each party can compute $[x][y]$ by computing $[c]+\epsilon[b]+[a]\delta+\epsilon\delta$. Correctness of the private multiplication can be verified using:

$$\begin{aligned}[c] + \epsilon[b] + [a]\delta + \epsilon\delta &= [a][b] + [x][b] - [a][b] \\ &\quad + [y][a] - [b][a] \\ &\quad + ([x] - [a])([y] - [b]) \\ &= [x][y]\end{aligned}$$

3.3.3 MPC square

To calculate the square $[x^2]$, a beaver pair ([a], [b]) provides by the TFP is used by the parties. Values of the pair must satisfy the property of $b = a^2$. The parties then compute $[\epsilon] = [x] - [a]$ and next reveal $[\epsilon]$ as $\epsilon = x - a$. The result is then obtained via $[x^2] = [b]+2\epsilon[a]+\epsilon^2$. The correctness can be verified using:

$$\begin{aligned}[b] + 2\epsilon[a] + \epsilon^2 &= [a]^2 + 2([x] - [a])[a] + ([x] - [a])^2 \\ &= [x^2]\end{aligned}$$

3.3.4 MPC comparisons

The comparison of two additive shared values x and y: $[x < y]$, can be computed by obtaining $[d] = [x] - [y]$ and comparing the result to zero: $[d < 0]$. $[d]$ in additive sharing is then converted into $\langle d \rangle$ in binary sharing. The sign bit of $\langle d \rangle$ can be fetched by shifting it to the LSB: $\langle s \rangle = sgn(\langle d \rangle)$. Subsequently, the sign bit $\langle s \rangle$ is converted from binary share to additive share to obtain $[x < y]$. Other comparison results can be computed based on $x < y$. The conversions between additive and binary sharing are used in comparisons.

3.3.5 MPC ReLU

In plaintext, ReLU can be defined as $ReLU(x) = x \times (x > 0)$. It can be computed by the variation $ReLU([x]) = [x][x > 0]$. To compute the variation, $x > 0$ is calculated first and then the multiplication of $[x]$ and $[x > 0]$ is executed. Comparison is used when computing ReLU, which makes the operation expensive in MPC.

3.3.6 MPC exponential

CrypTen leverages limit approximation to approximate exponential, which is the key operation of softmax function. The approximation can be described as follows:

$$e^x = \lim_{n \to \infty} \left(1 + \frac{x}{2^n}\right)^{2^n} \tag{4}$$

where n represents the number of approximation iterations. Due to the faster growth rate of exponential functions compared to polynomials, polynomial approximations like the Taylor Series are not utilized.

3.3.7 MPC reciprocal

Similar to the exponential function, the reciprocal is also computed using an approximation function. CrypTen computes reciprocals using the Newton–Raphson iteration as following described:

$$\frac{1}{x} = \lim_{n\to\infty} y_n = y_{n-1}\left(2 - x y_{n-1}\right) \tag{5}$$

where $y_0(x) = 3e^{0.5} - x + 0.003$ and the default number of iterations is 10.

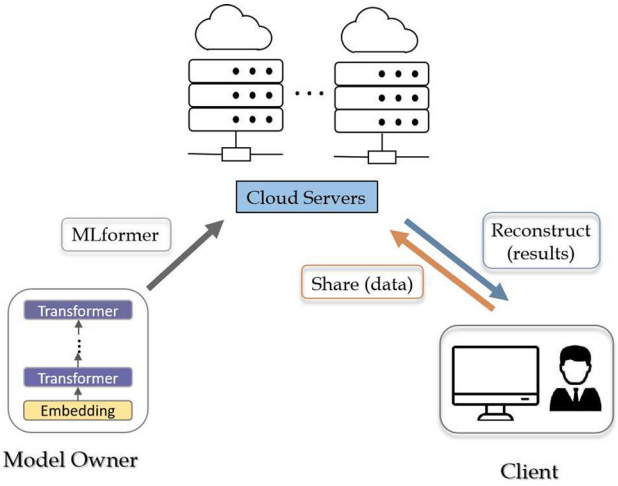

Fig. 3 An illustration of the private transformer inference through our proposed MLformer framework

4 Framework description

In order to address the aforementioned challenges, we have developed an MPC-based framework named MLformer for private inference of transformer-based models. Based on CrypTen [1], MLformer utilizes the MPC protocols which have been introduced in Section 3.3, which have been proven to be secure in study [1]. In our framework MLformer, we implement linear attention [19] as a replacement for the standard softmax attention, which bestows the framework linear time and memory complexity with input length. To overcome the dynamic range problem for MPC division used in the attention block, we introduce scaled linear attention and a novel approximation division function, which ensures the accuracy of linear attention and reduces the time overhead. Furthermore, an efficient reciprocal protocol is utilized for faster and more accurate reciprocal in MPC used in the whole framework. Meanwhile, to improve the accuracy and efficiency of exponential used in our framework, we propose a novel protocol for it as well. Given the crucial role of reciprocal and exponential computations in transformer inference, these advancements significantly accelerate inference speed and yield more precise results. Collectively, these components constitute our framework and enable it to perform better than previous works in terms of speed and accuracy. Each component will be sequentially elucidated in this section.

Similar to previous privacy-preserving inference frameworks CrypTen [1] and CryptGPU [21], MLformer follows a semi-honest model, in which the servers observe the protocol honestly but curiously. Assuming that there are a total of n parties, a maximum of t parties ($t \leq n-1$) may collude. As illustrated in Fig. 3, to perform private Transformer inference on client data, cloud servers first acquire the transformer model from the model owner and convert it to the MPC mode, which takes shared data as input, through our framework MLformer. The client subsequently shares his data with the cloud servers through secret sharing, which ensures that the servers have no visibility into the client's private data. Following this, the cloud servers can provide private inference services for the client on the shared data and next return the results to the client. Ultimately, the plaintext result can only be reconstructed by the client.

4.1 Linear transformer

The linear transformer, originally proposed in [19], is constructed by using a feature map-based dot product attention mechanism in place of the traditional softmax attention used in the standard transformer [3]. This modification yields lower time and memory overhead when processing long sequence inputs.

4.1.1 Linearized attention

Equation 6 utilizes softmax attention which computes a similarity score by taking the exponential of the dot product between a query and a key. Regarding the i-th row of a matrix as a vector, the attention Eq. 2 can be generalized as follows,

$$A_i = \frac{\sum_{j=1}^{N} sim\left(Q_i, K_j\right) V_j}{\sum_{j=1}^{N} sim\left(Q_i, K_j\right)} \tag{6}$$

Assuming that the input is a sentence and N denotes the number of word vectors of the input. $sim(\cdot)$ is the similarity function that computes the similarity score. Equation 6 can be equivalent to Eq. 2 by replacing $sim(\cdot)$ with $sim\,(q,k) = exp(\frac{q^T k}{\sqrt{d_k}})$. Obeying the definition of Eq. 6, other attention

mechanisms can be described. The only condition that needs to be satisfied is to ensure that $sim(\cdot)$ is non-negative.

With the feature kernel $\phi(x)$, Eq. 6 can be rewritten as follows:

$$A_i = \frac{\sum_{j=1}^{N} \phi\left(Q_i\right)^T \phi\left(K_j\right) V_j^T}{\sum_{j=1}^{N} \phi\left(Q_i\right)^T \phi\left(K_j\right)} \tag{7}$$

Utilizing the property of matrix multiplication, it can be simplified as:

$$A_i = \frac{\phi\left(Q_i\right)^T \sum_{j=1}^{N} \phi\left(K_j\right) V_j^T}{\phi\left(Q_i\right)^T \sum_{j=1}^{N} \phi\left(K_j\right)} \tag{8}$$

Using the packed Q, K, and V, we can write it in vectorized form as follows:

$$A_i = \frac{\phi(Q)\left(\phi(K)^T V\right)}{\phi(Q)\left(\sum_j^N \phi(K_j)\right)^T} \tag{9}$$

"A" refers to the attention matrix and the feature map $\phi(\cdot)$ above is employed to the matrices Q and K in row wise.

4.1.2 Feature Maps

In a linear transformer, a feature map that produces a positive similarity score is utilized:

$$\phi(x) = elu(x) + 1 \tag{10}$$

where $elu(\cdot)$ refers to exponential linear unit [33], which is an activation function. It can prevent the gradients from vanishing by ensuring that the score does not become zero when x is negative.

Employing the feature map defined by Eq. 10 to the attention function defined by Eq. 9, linear attention has a complexity of $O(NDM)$ of multiplications and additions, where N represents the number of input word vectors, D represents the dimensionality of the vectors, and M represents the number of attention heads. Thus linear attention has a linear time complexity with input sequence length.

4.1.3 Causal linear attention

The high parallelism of the transformer architecture makes it highly efficient for autoregressive tasks. In tasks such as autoregressive text generation, the transformer masks the attention computation, such that a position can only be influenced by the positions ahead of it. For example, in the decoder stage of translation tasks, the i-th word can be predicted leveraging the information of the j-th word only if $j \leq i$. The linear transformer provides causal linear attention to deal with autoregressive tasks. Causal linear attention can be defined by modifying Eq. 6 as follows:

$$A_i = \frac{\sum_{j=1}^{i} sim\left(Q_i, K_j\right) V_j}{\sum_{j=1}^{i} sim\left(Q_i, K_j\right)} \tag{11}$$

The masked attention mechanism can also be linearized, and can be expressed in the following manner:

$$A_i = \frac{\phi\left(Q_i\right)^T \sum_{j=1}^{i} \phi\left(K_j\right) V_j^T}{\phi\left(Q_i\right)^T \sum_{j=1}^{i} \phi\left(K_j\right)} \tag{12}$$

By introducing $S_i = \sum_{j=1}^{i} \phi\left(K_j\right) V_j^T$ and $Z_i = \sum_{j=1}^{i} \phi\left(K_j\right)$, Eq. 12 can be simplified as follows:

$$A_i = \frac{\phi\left(Q_i\right)^T S_i}{\phi\left(Q_i\right)^T Z_i} \tag{13}$$

The computation time for S_i and Z_i can be constant if we leverage S_{i-1} and Z_{i-1} that have been previously computed. As a result, causal linear attention has linear computational complexity with sequence length as well.

4.2 Utilizing linear attention

To address the issue of high time and memory overhead when processing long sequences, linear attention is employed instead of softmax attention. Algorithm 1 and Algorithm 2 illustrate the process for computing Scaled Dot Product Attention (softmax attention) and Linear Attention, respectively.

Algorithm 1 Scaled Dot Product Attention [3].

Input: queries $Q(l \times d_q)$, keys $K(l \times d_k)$, values $V(l \times d_k)$ from Equation 1, where $d_q = d_k$
Output: $Attention(l \times d_k)$
1: $R_1 \leftarrow QK^T$
2: $R_2 \leftarrow R_1/\sqrt{d_k}$
3: $R_3 \leftarrow softmax(R_2)$
4: $Attention \leftarrow R_3 V$

Algorithm 2 Linear Attention [19].

Input: queries $Q(l \times d_q)$, keys $K(l \times d_k)$, values $V(l \times d_k)$ from Equation 1, where $d_q = d_k$
Output: $Attention(l \times d_k)$
1: Compute $\phi(Q)$ and $\phi(K)$
2: $S \leftarrow \phi(K)^T V$
3: $R_1 \leftarrow \phi(Q)S$
4: $Z \leftarrow \sum_{j=1}^{N} \phi\left(K_j\right)$
5: $R_2 \leftarrow \phi(Q)Z^T$
6: $Attention \leftarrow R_1/R_2$

As shown in Algorithms 1 and 2, the inputs of both attentions, denoted as Q, K, and V, share the same row dimension of l, which represents the number of token vectors equal to the input sequence length. Linear attention, unlike scaled dot-product attention, does not compute the costly softmax function; instead, it requires the computation of the feature map function. However, the input matrix of softmax function, R_2 in Algorithm 1, is of dimension $l \times l$, whereas inputs of the feature map function, Q and K in Algorithm 2, are of the same dimension $l \times d_k$. Thus, the softmax function has a quadratic computational complexity with the input length, while the feature map function has a linear computational complexity instead. It should be noted that in softmax function and feature map function, operations, such as exponentials, require to be computed based on multiplications and additions. Therefore, the total cost of multiplications and additions scales as $O(N^2)$ for softmax attention, where N refers to the sequence length. It is the same for the memory requirements, since the full attention matrix must be retained to facilitate the computation of gradients related to queries, keys and values. However, for linear attention, the computational cost of multiplications and additions scales linear with the input length, $O(N)$. This also holds for memory requirements since $S = \sum_{j=1}^{N} \phi\left(K_j\right)^T V_j$ and $Z = \sum_{j=1}^{N} \phi\left(K_j\right)$ can be computed only once and be reused for every query. Thus our framework utilizing linear attention works faster than that with softmax attention when the input is long sequence.

4.3 Scaled linear attention

The MPC reciprocal is approximated using the Newton–Raphson iteration in CrypTen [1]. Nevertheless, this method of approximation has a range of accuracy mentioned in [15]. Only within a specified range does this approximated method produce results of negligible error. If the input is beyond this range, the approximated outcome will have a significant error, rendering the model output useless. As the number of iterations increases, the accuracy range also expands, but it's impossible to increase the iteration number infinitely because it will increase the computational overhead at the same time.

The percentage of error[1] for MPC reciprocal is displayed in Fig. 4. There is a great fluctuation in the errors of the values, where only the reciprocal errors of $x \in (0, 150]$ can be limited to under 20%. For $x > 250$, the errors may even become infinite. The division is used in the last step of linear attention computation, see R_1/R_2 in Algorithm 2. However, R_2 may become very large due to the summation of K_j at step 4 of Algorithm 2, leading to substantial error of the reciprocal of R_2. This error can have an adverse effect on the linear attention result and influence the result of inference in a negative way.

[1] $abs(MPC - Actual)/Actual$

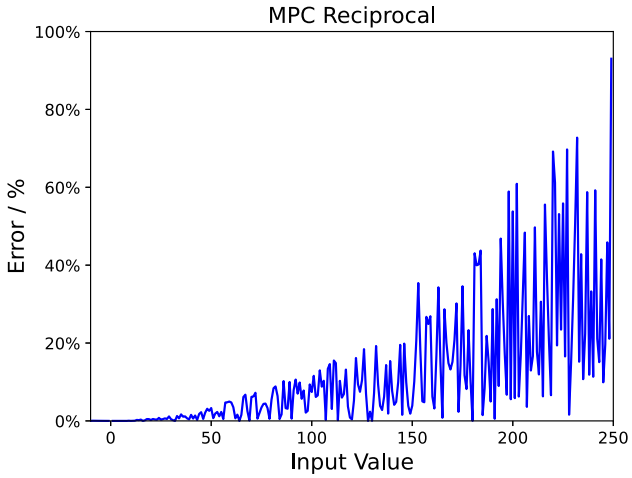

Fig. 4 Error percentage of Newton–Raphson iteration approximated reciprocal in MPC (Default iterations: 10)

To address the accuracy issue of the approximated reciprocal used in linear attention, we propose scaled linear attention, which is demonstrated by Algorithm 3. In the scaled linear attention, the featured Q and K are scaled by $\sqrt{d_k}$, respectively. The succeeding computations remain the same as linear attention in Algorithm 2. Scaling the featured Q and K ensures that R_2, whose reciprocal needs to be calculated, falls within the accuracy range of the approximated reciprocal. The scaling of featured Q and K does not affect the result of attention since the final step of Algorithm 3, R_1/R_2, will ultimately eliminate the influence. Thus the scaled linear attention can ensure the accuracy of the inference result of transformer using linear attention.

Algorithm 3 Scaled Linear Attention.

Input: queries $Q(l \times d_q)$, keys $K(l \times d_k)$, values $V(l \times d_k)$ from Equation 1, where $d_q = d_k$
Output: $Attention(l \times d_k)$
1: Compute $\phi(Q)$ and $\phi(K)$
2: $Q_s \leftarrow \phi(Q)/\sqrt{d_k}$
3: $K_s \leftarrow \phi(K)/\sqrt{d_k}$
4: $S \leftarrow K_s V$
5: $R_1 \leftarrow Q_s S$
6: $Z \leftarrow \sum_{j=1}^{N} \phi\left(K_j\right)$
7: $R_2 \leftarrow Q_s Z^T$
8: $Attention \leftarrow R_1/R_2$

4.4 Novel division approximated function

Different from Newton's method, we suggest to approximate reciprocal by polynomial functions inspired by [34]. We utilize linear regression to generate a polynomial of a given

degree(we set 3), which will be the approximation of reciprocal. We develop a single-layer linear neural network to carry out the linear regression.

$$R(x) = nn.linear(x) \tag{14}$$

We randomly generate input values in [0.3, 2] and utilize their reciprocal results as the mean squared error (MSE) targets. After training for 20,000 epochs with a learning rate of 0.001, the polynomial approximation is as follows:

$$\frac{1}{x} = 4.6043 - 6.6698(x) + 3.7314\left(x^2\right) - 0.7028\left(x^3\right) \tag{15}$$

This approximation yields an L1 error of 0.006 for $x \in (0.3, 2)$. It can be utilized in scaled linear attention since the scaling method guarantees that the range of x is satisfied. On the other hand, since Q_s and Z in Algorithm 3 are always positive, R_2, the input of our reciprocal approximation function is also ensured to be positive. Newton's iteration method (default iteration number of 10) uses multiple MPC multiplications to compute reciprocal, while our approximated method employs less multiplications so as to work faster than it. Thus this new approximated function reduce the run time of linear attention, which speed up the whole inference process as well.

4.5 Implementing the efficient reciprocal protocol

To address the accuracy issue brought by approximated division, we have endeavored to find a protocol that can calculate the reciprocal more accurately and efficiently. The protocol proposed by Bar-Ilan and Beaver [2] computes reciprocal using a single MPC multiplication rather than requiring multiple multiplications in Newton's iteration method. We are the first to integrate the protocol into MPC frameworks designed for deep learning. This reciprocal protocol (we name as "Efficient Reciprocal") can be described as Algorithm 4.

Algorithm 4 Efficient Reciprocal [2].

Input: N MPC servers; each party has $[X]_i$, where X can be a matrix or a value

Output: $Y = X^{-1}$ (X^{-1} is the multiplicative inverse of X)

1: Generate a random group element Z (by a trusted third party or trusted first party)
2: Share the random Z to parties: each party has $[Z]_i$
3: **for** *parties* 1 to n **do**
4: Compute $[W]_i = [XZ]_i$ by beaver triples
5: Broadcast $[W]_i$ to retrieve $W = XZ$
6: Compute W^{-1} secretly and locally (W^{-1} is the multiplicative inverse of W)
7: $[Y]_i \leftarrow [Z]_i W^{-1}$
8: **end for**

The following equation can demonstrate the correctness of the protocol:

$$[Y] = [Z]W^{-1} = [Z](ZX)^{-1} = \frac{[Z]}{Z}X^{-1} = [X^{-1}]$$

Different from Newton's iteration approximated method, this reciprocal protocol needs only one MPC multiplication and one round of communication among parties to obtain the share of X^{-1}. Eliminating the approximated function, the efficient reciprocal protocol can compute the reciprocal of x accurately. We set the range of random Z as $(-2^{38}, 2^{38})$ when the ring size is 2^{64} to make a trade-off between the accuracy range and security guarantee. This protocol is proven to be secure in study [2] and other MPC primitives used are from CrypTen [1]. The efficient reciprocal protocol can be implemented in all the operations where the secret reciprocal is computed. By incorporating this protocol, the private inference can experience enhancements in both speed and accuracy.

4.6 Proposed MPC exponential protocol

Under a ring size of 2^{64} and a fixed point precision of 16 bits, the CrypTen framework is capable of generating an exponential of x only when $e^x \in (1/2^{16}, 2^{47})$, with x ranging from (-10,31). Thus we focus on the exponential computation of $x \in (-10, 31)$. CrypTen uses the limit approximation to compute exponential, which has the accuracy issue caused by "dynamic range" similar to reciprocal. Figure 5 shows the error percentage of the MPC exponential computed by limit approximation with the default iteration number of 8. The exponential errors are limited to under 20% only when $x \in (-9, 10)$ and the errors are even larger than 100% when x is close to 30.

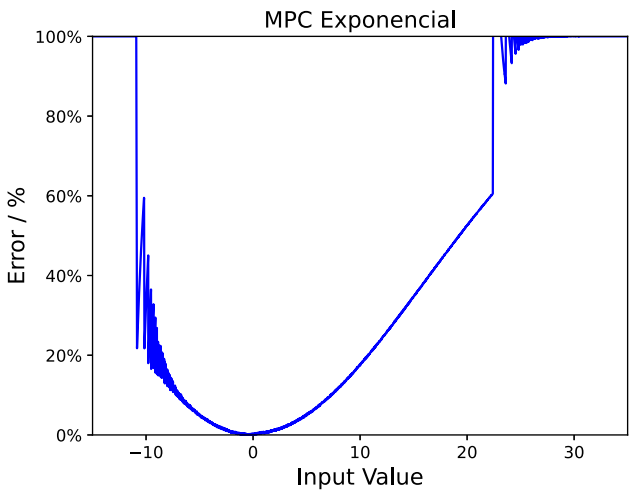

Fig. 5 Error percentage of limitation approximated exponential in MPC (Default iterations: 8)

To address the accuracy issue of MPC exponential, we have proposed a novel protocol for computing the exponential of value x in MPC. First, a random value $z \in (-4, 4)$ is generated and shared with parties. Having the share $[x]$, each party then locally computes $[x - z] = [x] - [z]$. $[x - z]$ is next revealed to all the parties and its exponential $e^{(x-z)}$ can be computed in plaintext by each party. Following this, each party computes exponential of $[z]$ using the alternative limit approximation proposed by us. Finally $[e^x]$ can be obtained by multiplying $[e^z]$ and $e^{(x-z)}$. The alternative limit approximation can be described as follows:

$$e^x = \lim_{n\to\infty} \left(1 + \frac{x}{2^n} + \frac{1}{2}\left(\frac{x}{2^n}\right)^2\right)^{2^n} \tag{16}$$

Instead of using the first two terms of the exponential Taylor polynomial, as the limit approximation used by CrypTen (Eq. 4) does, we make use of the first three terms of the polynomial. By employing more polynomial terms, the number of iterations required for achieving the same level of accuracy can be reduced, leading to less computation time.

Algorithm 5 Exponential.

Input: N MPC servers; each party has $[x]_i$
Output: $Y = e^x$
1: Generate a random value $z \in (-4, 4)$ and $m = z/2^n$ (by trust third party or trust first party)
2: Share z and m to parties: each party has $[z]_i$ and $[m]_i$
3: **for** $parties1$ to n **do**
4: $[x - z]_i \leftarrow [x]_i - [z]_i$
5: Broadcast $[x - z]_i$ to retrieve $x - z$
6: Compute e^{x-z} locally
7: $[e^m]_i \leftarrow 1 + [m]_i + \frac{1}{2}\prod_{\text{Square}}([m]_i)$
8: $[e^z]_i \leftarrow \left(\prod_{\text{Square}}([e^m]_i)\right)^n$
9: $[Y]_i \leftarrow [e^z]_i e^{x-z}$
10: **end for**

The correctness can be described as the following:

$$[Y] = [e^z]e^{x-z} = \frac{[e^z]}{e^z}e^x = [e^x]$$

Algorithm 5 demonstrates our protocol for MPC exponential in detail, where $\prod_{\text{Square}}(\cdot)$ refers to the MPC Square protocol provided by CrypTen. In Algorithm 5, the random z whose exponential needs to be computed secretly is generated from a small range of (-4,4), leading to a smaller n, the number of iterations, as well. It should be noted that this range is large enough to allow for exponential of $x \in (-10, 30)$ to be computed securely using our exponential protocol. Through experiments, we have determined that time-efficient exponential with satisfied accuracy can be achieved with an iteration number of 4. Since this exponential protocol will be utilized for feature map computation, wherein only the exponential of $x < 0$ needs to be computed, we set the default iteration number as 4 in our framework. By utilizing our exponential protocol, both the accuracy and speed of the private inference of transformer can be improved.

5 GPU acceleration

To optimize the computation of causal linear attention, which is utilized in private inference of autoregression tasks, we propose several novel CUDA kernel functions including Matrix multiplication (Matmul), Cumulative summation (Cumsum), and Causal product. Among them, the CUDA function for Matrix multiplication could be implemented not only in causal linear attention but on every instance within the framework where it is employed. Meanwhile, the CUDA functions for Cumulative summation and Causal product are utilized particularly in causal linear attention. All these accelerations help speed up the private inference of transformer based on our framework when using GPU.

CrypTen is a framework based on PyTorch, which allows off-loading computations to the GPU. On the GPU, functions such as matrix multiplication can be optimized using highly-optimized implementations from CUDA libraries such as cuDNN and cuBLAS. However, these libraries are primarily designed to perform computations over floating-point numbers and are inconsistent with the integer types needed for executing MPC computations. To address this issue, CrypTen utilizes the method described in CryptGPU [21]. Noting that integers a and b belong to $\mathbb{Z} \cap \lceil -2^{26}, 2^{26} \rceil$, the computation of their product ab can be carried out via floating-point representations of 64-bit and can still recover an accurate result for the integers. CrypTen divides every 64-bit variable into four components: $a = a_0 + 2^{16}a_1 + 2^{32}a_2 + 2^{64}a_3$, wherein each a_i refers to a 16-bit integer component. The 64-bit integer product ab is computed by summing 10 pairwise products of their 16-bit components. This method is utilized for matrix multiplications and convolutions as well. However, the process of splitting and recovering results in extra time overhead. Our CUDA kernel functions for matrix multiplication and causal product are implemented over integer data type, thus eliminating the need for the splitting and recovery process and saving additional time overhead.

5.1 Matrix multiplication acceleration

To accelerate the computation of 3-dimensional matrix multiplication, we have designed a CUDA kernel function that improves the parallelism by organizing the threads and reduces the IO time by utilizing shared memory. Threads on the GPU are organized by $(grid, block, thread)$ as illustrated in Fig. 6. A kernel function can apply to a grid that contains multiple blocks, where each block is made up of

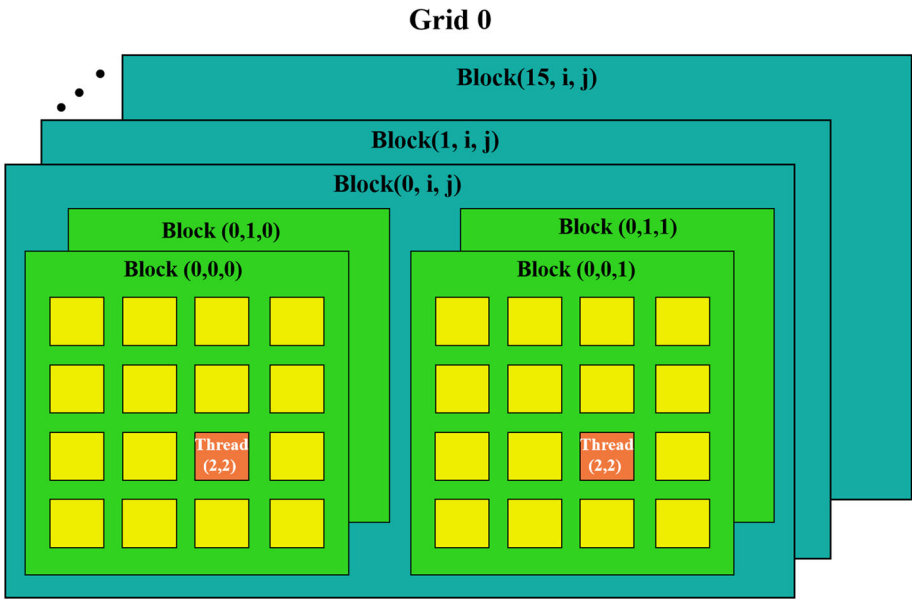

Fig. 6 An overview of the threads organization: one grid of dim (16, 2, 2) containing blocks of dim (4, 4)

several threads. The global memory of the GPU is where kernel functions read inputs and write results. On the other hand, shared memory on the GPU is different from global memory because it can be easily accessed by all threads within a block, resulting in lower data transmission time cost.

To illustrate, we present an example of a CUDA kernel function for matrix multiplication of K and V. Assuming that all the sizes of K, V, and the result matrix S are of (16, 8, 8). The shape (16, 8, 8) is merely an example, and in practice, the dimensions can be of any size. We first apply one grid of size (16, 2, 2) which contains 64 blocks. Each block is of size (4, 4) and contains 16 threads. Shared memory Ks and Vs of size (4, 4) are applied by each block to store values of K and V, respectively. Every thread in the block first reads values of K and V on global memory to the shared memory Ks and Vs. Next, every thread of the block uses the values of Ks and Vs to compute the matrix multiplication result.

More specifically, as Fig. 7 shows, $K[0][i][j]$ and $V[0][i][j]$ are divided to 4 blocks respectively as Kb_0, Kb_1, Kb_2, Kb_3 and Vb_0, Vb_1, Vb_2, Vb_3. To compute the result of $S[0][6][6]$, threads in Block(0, 1, 1) first read values of Kb_2 and Vb_2 to shared memory Ks and Vs. For instance, thread(2, 2) reads $K[0][6][2]$ and $V[0][2][6]$ to $Ks[2][2]$ and $Vs[2][2]$, respectively. Once all the threads complete their fetching, the thread(2, 2) employs the values of $Ks(row2)$ and $Vs(col2)$ to calculate the first half of the result $S[0][6][6]$. The second half is computed by following the identical steps by threads of Block(0, 1, 1) using values of Kb_3 and Vb_3. Finally, thread(2, 2) writes the sum of the two halves to $S[0][6][6]$ in global memory.

Overall, to accelerate 3D matrix multiplication, we've developed a CUDA kernel function that enhances parallelism and reduces IO time using shared memory and efficient thread organization. Moreover, the kernel function is highly adaptable, capable of handling inputs of various shapes.

5.2 Cumulative summation acceleration

Similar to matrix multiplication, shared memory is also utilized to optimize the summation of K_j. Assuming that the dimensions of K and the resultant matrix R are (8, 8); namely, the shape (8, 8) is merely an example, and the dimensions can be of any size in practice. Then the block size is set as (2, 8), and the grid size is (4, 1). Each block uses shared memory of size (2, 8) to store values of K. As Fig. 8 illustrates, every thread within the block first fetches values of K to store in Ks. Then every thread of the block cumulatively adds the elements in Ks of the same row to compute the sum of K_j. More specifically, to compute $R[1][1]$, threads in Block(0, 0) reads values of Kb_0 to Ks. For instance, thread(1, 1) of the Block(0, 0) reads $K[1][0]$ to $Ks[1][0]$. Once the reading is completed, thread(1, 1) then adds $Ks[1][0]$ and $Ks[1][1]$ to obtain the result of $R[1][1]$. Finally, the result is written to R in global memory.

5.3 Causal product acceleration

The Linear Transformer [19] sequentially computes causal products of queries duo to the cumulative operations used when computing S_i and Z_i. This results in linear time and space complexity, but limits the parallelism of causal product computation. To address this issue, we propose CUDA kernel functions that compute causal product of each query in parallel. To obtain the causal product of a given query Q_i(where $i < N$), two steps are involved: first, computing $S_i = \sum_{j=1}^{i} \phi\left(K_j\right) V_j^T$, and second, computing $C_i =$

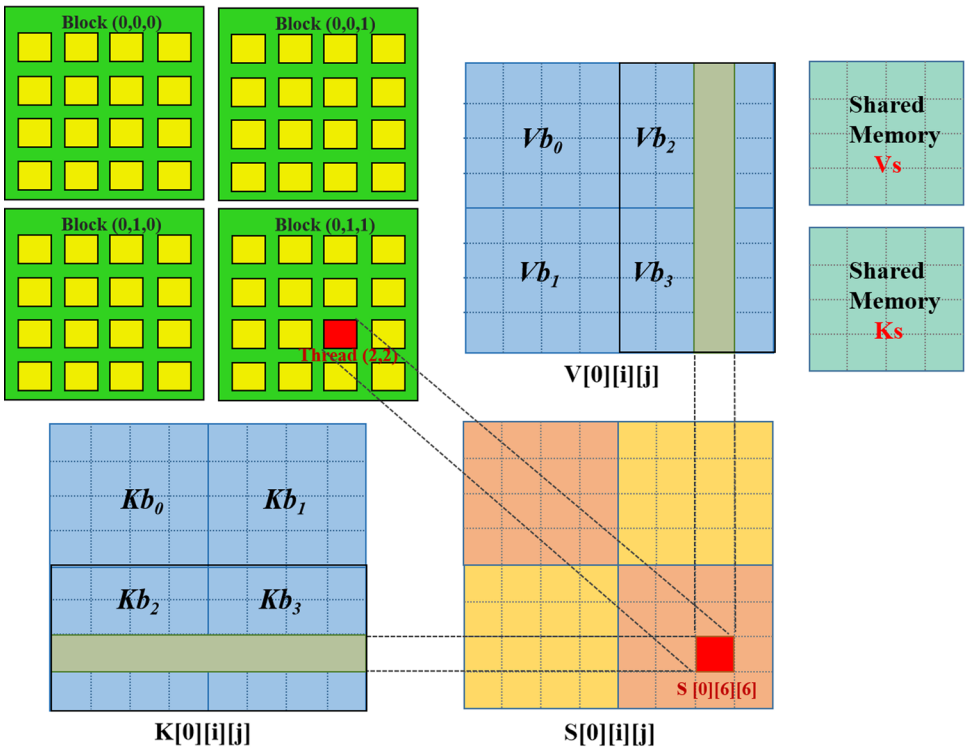

Fig. 7 The thread organization of our kernel function for 3-dimensional Matrix Multiplication

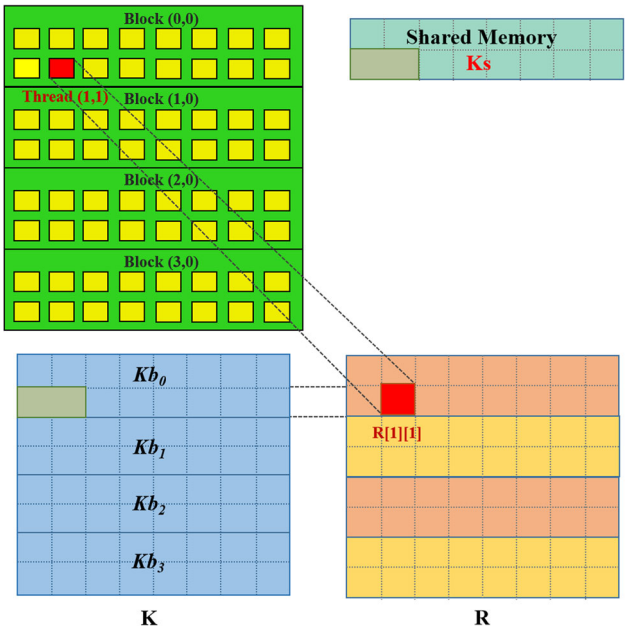

Fig. 8 The thread organization of our kernel function for the cumulative summation of K_j

$\phi(Q_i)S_i$. We propose optimized CUDA kernel functions for both steps, where each S_i and each C_i are computed in parallel, respectively. To simplify the description, $\phi(Q)$ and $\phi(K)$ are represented by Q and K. Assuming a size of (8, 8) for Q, K, and V, namely, the shape (8, 8) is merely an example to illustrate, and the dimensions can be of any size when implementing the kernel functions in practice.

5.3.1 CUDA Kernel function for computing S_i

The first step is to compute S_i. As illustrated in Fig. 9a, the grid is of size (2, 2) containing 4 blocks and the block is of size (8, 4, 4) containing 128 threads. The size of the result matrix KV is (8, 8, 8). In each block, every thread first reads the values of K and V into shared memory Ks of size (8, 4, 1) and Vs of size (8, 1, 4), respectively. The threads then compute the product of the corresponding values in Ks and Vs and store the results on KVs, which is the shared memory of size (8, 4, 4) applied by each block. As Fig. 9a shows, the outcome of $k_j^T v_j$ is stored in $KVs[j]$. Each $k_j^T v_j$ is computed in parallel through these steps.

More specifically, to compute the value of $KV_0[0][2][2]$, the threads in Block(0,0) first read values of Kb_0 and Kb_1 to Ks and Vb_0 and Vb_1 to Vs, respectively. For example, the thread(0,2,2) reads $K[2][0]$ and $V[0][2]$ to $Ks[0][2][0]$ and $Vs[0][0][2]$. After the read is completed by all the threads, the thread(0,2,2) can multiply values in $Ks[0][2][0]$ and $Vs[0][0][2]$ to get $KVs[0][2][2]$. Then KV_0 is computed by cumulatively summing KVs on dimension 0. Each block computes KV_i in parallel. Finally, the S_i is stored in $KV[i]$.

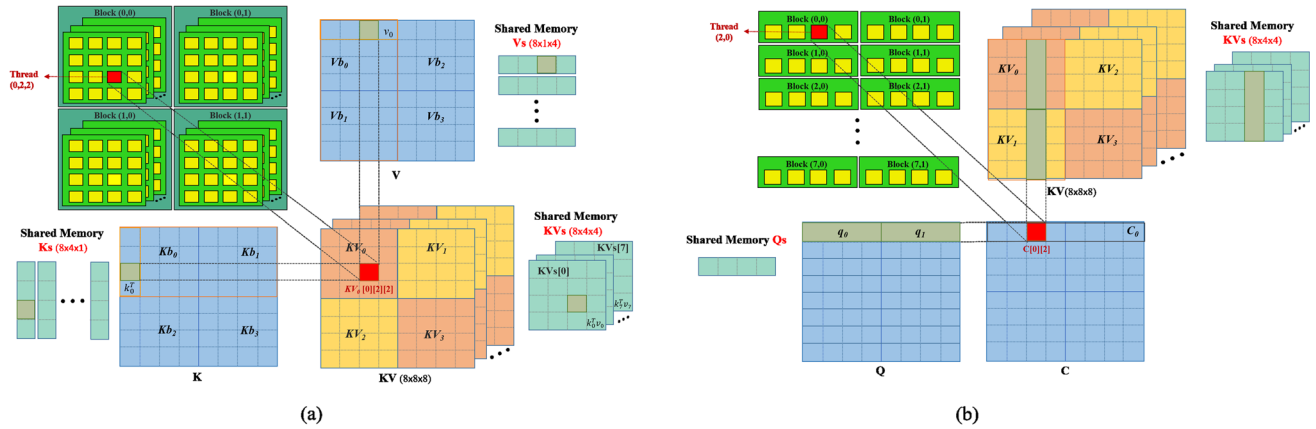

Fig. 9 The thread organization of our kernel function for computing causal product. **a** computation of S_i. **b** computation of C_i

5.3.2 CUDA Kernel function for computing C_i

After the computation of S_i is completed, C_i is computed next. As Fig. 9b shows, the applied grid is of size (8, 2), consisting of 16 blocks with a block size of (1, 4). Firstly, every thread in a block reads the values of Q and KV into shared memory Qs of size (1, 4) and KVs of size (4, 4), respectively. Then the threads proceed to compute C_i using the values of Qs and KVs. The results are stored in matrix C of size (8, 8). For instance, as Fig. 9b illustrates, the thread(0, 2) of Block(0, 0) is designated to compute $C[0][2]$. After the threads in Block(0, 0) have read the values of q_0 to Qs and the values of KV_0 to KVs, thread(0, 2) calculates the first half of $C[0][2]$ by multiplying vector Qs by the 3rd column vector of KVs. Subsequently, the other half of $C[0][2]$ is computed by a thread(0, 2) using values of q_1 and KV_1 through the same steps. In these steps, C_i(row vector of C) can be computed in parallel by blocks.

With our proposed CUDA kernel functions, it is now feasible to compute each S_i and each C_i in parallel for each query, thus to compute the causal product of each query in parallel and reduce the time overhead during inference.

6 Experimental evaluation

6.1 Experimental setup

To evaluate our framework when processing inputs of different sequence lengths, we conduct an image classification task. For this task, we utilize the ViT [4] model, which is a transformer-based neural network consisting of multiple transformer encoder layers. And we use a small portion of the ImageNet dataset, a large-scale visual recognition dataset containing more than a million training images as the training dataset. Each example in ImageNet is an RGB image center-cropped to dimensions of 224 × 224. In the training and inference process of ViT, images are sliced into sequential patches. For instance, if each patch is of dimension 14 × 14, then a 224 × 224 image can be sliced into a sequence with 16 × 16 patches. Thus altering the size of the input image changes the input length, while the patch size remains unchanged.

To evaluate our framework, we compare it with the original transformer, linformer, and nystromformer, which are all implemented on the MPC framework CrypTen [1]. We provide three versions of our framework, version 1 (MLformer1) with linear attention but original division and exponential, version 2 (MLformer2) with linear attention, our division approximation and our exponential protocol(4 iterations), version 3 (MLformer3) with linear attention, efficient reciprocal protocol and our exponential protocol(4 iterations). CrypTen Softmax refers to the private ViT model with softmax attention based on CrypTen, CrypTen Linformer refers to its usage of linformer, and CrypTen Nystrom refers to the model employing Nystromformer.

All experiments were conducted on a server equipped with an Nvidia GeForce RTX 3070, running Ubuntu 20.04 LTS.

6.2 Benchmarks for private inference

6.2.1 Inference time

Table 1 presents the MPC inference time of 1-layer ViTs conducted on different frameworks with inputs of varying sequence lengths. The training dataset utilized in our experiments is derived from a tiny subset of ImageNet, comprising 13,000 images from 10 different classes. The input images for inference are also drawn from this subset (2 images). The different frameworks used in our experiments are all configured with identical settings: a batch size of 2, training epochs of 5, learning rate of 0.001, and a patch size of 14 × 14. We varies the lengths of the input sequence by changing the image size.

Table 1 1-layer ViTs' private inference times of different frameworks

Framework	Seq Length: 100		Seq length: 256		Seq Length: 625		Seq length: 1024	
	Time/s	Speedup/%	Time/s	Speedup/%	Time/s	Speedup/%	Time/s	Speedup/%
CrypTen Softmax [1]	0.4238	–	0.4880	–	0.7923	–	1.386	–
CrypTen Nystrom [17]	0.6192	−46.10	0.6293	−28.94	0.6621	16.43	0.7005	49.45
CrypTen linformer [16]	0.4312	−1.729	0.4404	9.768	0.4909	38.04	0.5063	63.46
This work (MLformer1)	0.4135	2.449	0.4394	9.958	0.4740	40.18	0.5025	63.73
This work (MLformer2)	0.3571	15.74	0.3603	26.18	0.4138	47.78	0.4247	69.35
This work (MLformer3)	0.2022	**52.30**	0.2220	**54.50**	0.2517	**68.22**	0.2938	**78.79**

Bold indicates the best results of our methods in different evaluations

Table 1 shows that the private inference of traditional transformer based on Crypten (Crypten Softmax) works the slowest on input length of 625 and 1024, all the other models outperform it. On the other hand, our framework, which incorporates efficient reciprocal and our novel exponential (MLformer3), presents the fastest performance, achieving a speedup of 68.22% and 78.79% compared to Crypten Softmax on sequence lengths of 625 and 1024, respectively. Even our framework with original division and exponential (MLformer1) works faster than models using Linformer and Nystromformer. Frameworks with Linformer and Nystromformer have no advantage on processing short length input (such as length of 100), running slower than Crypten Softmax.

Among the three MLformer versions we provided, MLformer3 performs the best, MLformer2 s and MLformer1 the slowest. MLformer2 outperforms MLformer1 due to our new division approximation and new protocol for exponential, which reduces the time overhead for MPC division used in linear attention and the MPC exponential used in the whole framework. On the other hand, MLformer3 performs better than MLformer2 because we implement the efficient reciprocal protocol to compute divisions of the whole framework, which speeds up the inference process.

In addition, compared to CrypTen Softmax, the speedup rates of our MLformer of all three versions increase as the sequence length grows. Figure 10 illustrates how the run time of CrypTen Softmax and MLformer1 scales with increasing sequence length when the model employs a single layer. The run time of CrypTen Softmax with softmax attention grows quadratically, while that of our framework MLformer1 with linear attention grows linearly, providing an advantage for it to deal with tasks of long input.

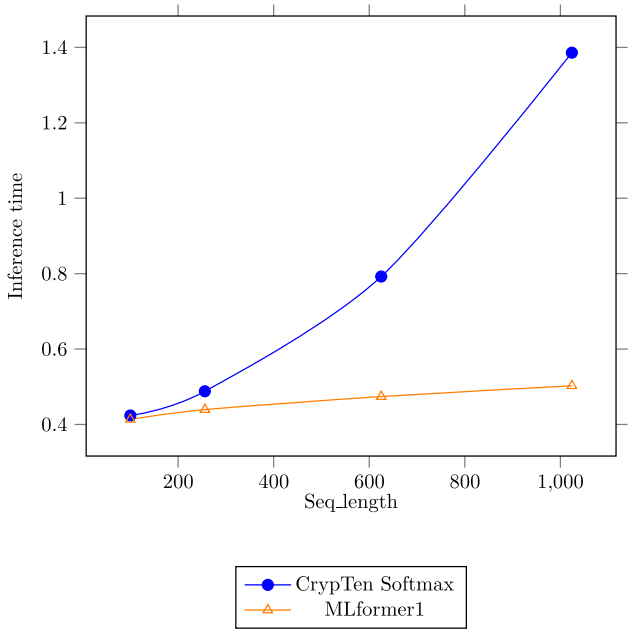

Fig. 10 1-layer ViTs' private inference time of CrypTen Softmax and MLformer1

Table 2 Time cost of MPC reciprocal

Operation	Time cost/ms	Speedup
CrypTen reciprocal [1]	27.85	1×
Our approximated reciprocal	4.627	6.02×
The efficient reciprocal [2]	0.6468	43.05×

6.2.2 MPC reciprocal

We compare our new division approximation and the efficient reciprocal protocol [2] with the original reciprocal method used in Crypten. Crypten employs the Newton's iteration approach for reciprocal approximation with the default iteration number of 10, while our reciprocal approximation is generated by linear regression. The reciprocal is computed accurately using the efficient reciprocal protocol instead of approximating the result. Table 2 shows the time cost of different methods to compute reciprocal of encrypted input. Our new reciprocal approximation achieves a 6.02× speedup compared to the Crypten reciprocal. Meanwhile, the efficient reciprocal protocol works far faster than the two approximated methods, achieving a 43.05× spee-dup compared to the Crypten reciprocal.

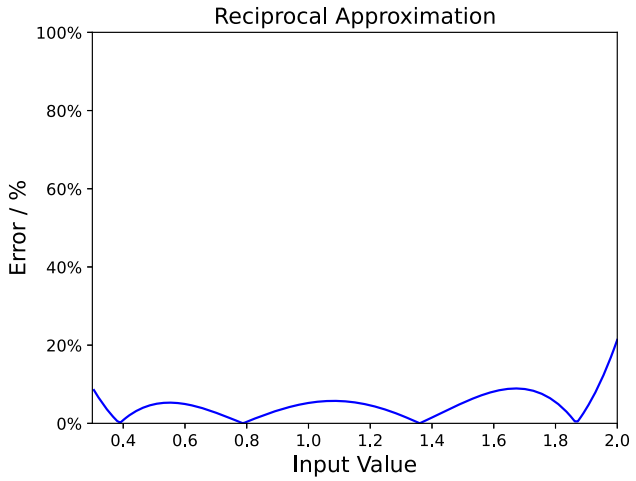

Fig. 11 Error percentage of MPC reciprocal approximated by linear regression

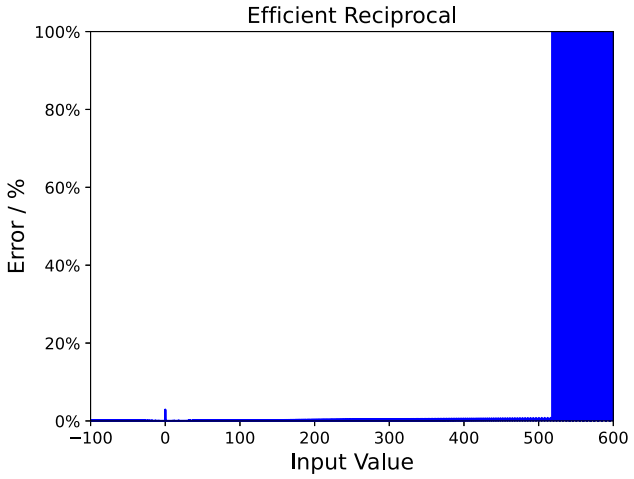

Fig. 12 Error percentage of efficient reciprocal proposed by Bar-Ilan and Beaver [2]

Figure 11 shows the percentage of error of our new MPC reciprocal approximation to actual reciprocal for $x \in (0.3, 2)$. The error for x is predominantly under 20%, thus resulting in negligible impact on inference accuracy, according to the analysis presented in part 5 of this subsection. This new approximation is implemented on the last step of scaled linear attention, specifically as algorithm 3 presents. The last step can be rewritten as $r * (1/z)$, where $1/z$ can be calculated utilizing our new approximation, so as to reduce the computational time of linear attention block.

Figure 12 shows the percentage of error of the efficient reciprocal protocol to actual reciprocal for $x \in (-100, 600)$. The error percentage of the efficient reciprocal is close to zero for $x \in (-100, 500)$, which is far lower than the Crypten reciprocal approximated by Newton's iteration. Additionally, the input range where the reciprocal is computed accurately of the efficient reciprocal is larger than the Crypten reciprocal. Thus, the efficient reciprocal outperforms the Newton's approximated reciprocal both in runtime and accuracy. This protocol can be implemented to all the operations where the secret reciprocal is computed. It is able to speed up the private inference process and enhance the accuracy of the inference.

6.2.3 MPC exponential

We compare our exponential (iteration number 4) with Crypten exponential. Table 3 shows the time costs of these two methods when they are implemented directly on encrypted values. Our exponential achieves 12.61% compared to the Crypten exponential.

Figures 5 and 13 present the error percentage to actual exponential of Crypten exponential and our exponential protocol, respectively. For Crypten exponential, the errors are limited to under 20% only when $x \in (-9, 10)$ and the errors are even larger than 100% when x is close to 30. However, the error percentage of our exponential is under 10% and 30% for $x \in (-6, 30)$ and $x \in (-10, -6)$, respectively. Obviously, our exponential protocol can compute the exponential of $x \in (-10, 30)$ more accurately than the Crypten exponential. Thus, our exponential of 4 iterations outperforms the default Crypten exponential both in runtime and accuracy.

Table 3 Time cost of MPC exponential

Operation	Time cost/ms	Speedup(1)/%
CrypTen exponential [1]	4.187	–
Our exponential	3.660	**12.61**

Bold indicates the best results of our methods in different evaluations

This exponential protocol is used in the feature map computation, where exponential of $x < 0$ needs to be computed only. It can accelerate the private inference process and improve the inference accuracy. On the other hand, for those negative inputs of feature map whose exponential are too small to be represented in the ring, their exponentials computed by our protocol is 0, which has no influence on the subsequent computations.

6.2.4 Memory overhead

Table 4 shows the GPU memory overhead of 1-layer ViTs of different frameworks. It is obviously that our framework MLformer3 requires significantly less memory than Crypten Softmax which uses softmax attention. Moreover, the memory saving rate increases as the sequence length grows. On the other hand, the memory overhead scales with growing input lengths in both frameworks, the scale of Crypten Softmax is evidently larger than that of our framework. This results from the quadratic memory complexity of softmax function with input length. Given the considerations of both time and

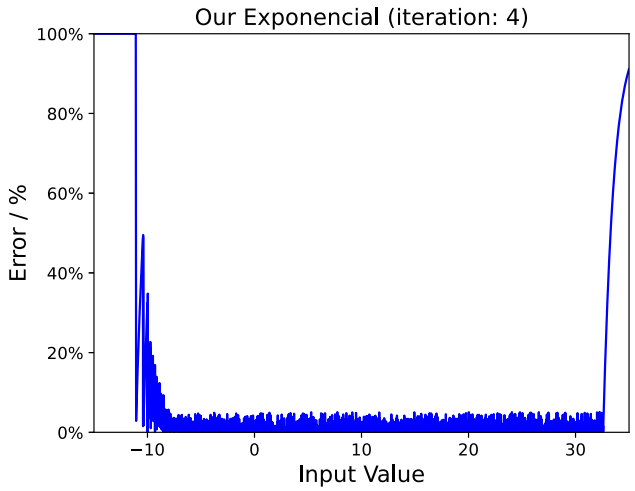

Fig. 13 Error percentage of MPC exponential using our protocol (iteration number:4)

memory overhead, linear attention is better suited for long sequence inference.

6.2.5 Inference accuracy

We conduct experiments on the Fashion_Mnist dataset [35] to analyze the effect of linear attention and new reciprocal and exponential methods on transformer model accuracy. We trained two distinct transformer models, one using softmax attention with actual division (traditional transformer) and the other employing linear attention with the efficient reciprocal and our exponential protocol (MLformer3). The two models are trained under identical conditions, with a layer depth of 2, training epoch of 10, and learning rate of 0.001. As Table 5 shows, the accuracy of the two transformer models on test images is similar, with traditional transformer achieving an accuracy of 83.66% and MLformer3 achieving 83.67%. The experiments demonstrate how our framework MLformer3 can perform as accurately as the traditional transformer model, while requiring less time overhead.

6.3 Benchmarks of GPU acceleration

We have proposed CUDA kernel functions for Matmul (matrix multiplication), Cumsum (cumulative summation), and Causal Product and integrated them into our framework. For private inference implemented on our framework, the model is initially converted into an ONNX model, consisting of multiple computing nodes. We also create modules that map our CUDA kernel functions with the nodes of ONNX in our MPC framework. To evaluate our kernel functions, we have implemented the operations directly on arithmetic encrypted matrices.

Table 5 Test accuracy of different frameworks

Framework	Test accuracy/%
Traditional transformer [3]	83.66
This work(MLformer3)	83.67

6.3.1 Matrix multiplication

In Table 6, it is evident that the use of our CUDA kernel function significantly enhances the performance of matrix multiplication across various matrix sizes. Specifically, when considering two input matrices, (A_{enc} and B_{enc}), each with dimensions (16, 8, 8), our approach achieves a speedup of approximately 69.52% compared to the method without our kernel, as referenced from [1]. Additionally, notable speedups are observed for other matrix dimensions, such as (8, 8, 8) × (8, 8, 8) and (16, 8, 64) × (16, 64, 8), where speedups of 68.69% and 69.09% are achieved, respectively, demonstrating the robustness of our kernel function in improving computational efficiency across a range of matrix sizes. The Matmul operation is frequently used in the server's local computations, particularly in computing MPC multiplication using beaver triples. Thus, implements of our CUDA function for Matrix multiplication can greatly contribute to the acceleration of the inference.

6.3.2 Cumulative summation

When computing Z_i in causal linear attention, the cumulative summation of K is calculated. Table 7 demonstrates the performance improvement achieved through the use of our kernel function for the cumulative sum operation across different matrix sizes. Specifically, when the input matrix (K) has dimensions of (8, 16, 16), our kernel function enhances

Table 4 1-layer ViTs' GPU memory overhead of different frameworks

Seq length/$patches$	CrypTen Softmax [1]/MB	This Work(MLformer3)/MB	Save rate/%
100	1997	1997	0
256	2369	2071	**12.58**
625	4393	2247	**48.85**
1024	6917	2321	**66.44**

Bold indicates the best results of our methods in different evaluations

Table 6 MPC time cost of matrix multiplication using our Kernel function

Matrices size	Without our Kernel [1]/ms	Using our Kernel/ms	Speedup/%
(8,8,8)×(8,8,8)	3.309	1.036	**68.69**
(16,8,8)×(16,8,8)	3.435	1.047	**69.52**
(16,8,64)×(16,64,8)	3.568	1.103	**69.09**

Bold indicates the best results of our methods in different evaluations

Table 7 MPC time cost of cumulative sum using our Kernel functions

Matrix size	Without our Kernel [1]/ms	Using our Kernel/ms	Speedup/%
(4,8,8)	0.3509	0.2977	**15.18**
(8,8,8)	0.3512	0.3007	**14.39**
(8,16,16)	0.3609	0.3049	**15.53**

Bold indicates the best results of our methods in different evaluations

computational efficiency with a speedup of approximately 15.53% over the original cumulative sum function, as referenced from [1]. Furthermore, significant performance gains are consistently observed across other tested matrix sizes, such as (4, 8, 8) and (8, 8, 8), with speedups of 15.18% and 14.39%, respectively, underscoring the effectiveness of our kernel function in optimizing the cumulative sum operation. Thus, this CUDA function can speed up the computation of causal linear attention, which is important for Transformer inference of autoregression tasks.

6.3.3 Causal product

In linear transformer [19], the causal product was computed through a "for loop" where S_i is computed by S_{i-1}, resulting in a linear time complexity while sacrificing the parallel computation of queries' causal products. Our CUDA kernel functions for causal product make it possible to compute causal products for queries in parallel during inference. According to Table 8, our kernel function, when applied to encrypted tensors (Q), (K), and (V) each with dimensions of (8, 8), exhibits a substantial performance improvement over the method detailed in [19], achieving a remarkable speedup of 94.48%. The substantial acceleration of Causal product can greatly speed up the causal linear attention, a vital component for private inference of autoregression tasks based on Transformer.

6.3.4 Autoreggressive task: image generation

To evaluate the impact of GPU acceleration on the total inference time for autoregressive tasks, we conducted an experiment using an image generation task. We utilized a trained Transformer model with causal masking attention from the study by Katharopoulos et al. [19] and performed private inference on the MNIST dataset. Our goal was to measure the total inference time required to generate a single image. We used a batch size of 1 and run the private inferences on our GPU device, an Nvidia GeForce RTX 3070. As Table 9 shows, the inference time without our kernel is 15.86 s. In contrast, using our kernel functions reduces the inference time significantly to 6.28 s, achieving a notable speedup of 60.40%. This demonstrates the significant advantage of our optimized kernel functions in enhancing the computational efficiency of private inference in Transformers for autoregressive tasks.

Table 9 MPC time cost of image generation task using our Kernel functions

Methods	Inference time/s	Speedup/%
Without our Kernel [19]	15.86	–
Using our Kernel	6.28	**60.40**

Bold indicates the best results of our methods in different evaluations

7 Conclusions

We propose an MPC based framework MLformer for private inference of transformer based models. To overcome the security and efficiency challenges brought by Transformer when implementing MPC protocols, we implement an alternative attention and propose several novel protocols. First, we suggest replacing the softmax attention mechanism with

Table 8 MPC time cost of causal product using our Kernel functions

Operation	Without our Kernel [1]/ms	Using our Kernel/ms	Speedup/%
Causal Product	37.49	2.069	**94.48**

Bold indicates the best results of our methods in different evaluations

linear attention, which eliminates the use of softmax function and has a linear time and memory complexity with input length. By removing the maximum operation, which is expensive in MPC, the time overhead is reduced. Second, to address the dynamic range issue for MPC division used in linear attention, we propose scaled linear attention where inputs are scaled with no influence on the output. Additionally, we introduce a new method for approximating MPC division using linear regression, which reduces the run time of attention block. Furthermore, to improve the accuracy and efficiency for MPC reciprocal and exponential, we implement an efficient reciprocal protocol and propose a novel exponential protocol. Lastly, we integrate our proposed CUDA kernel functions to the framework to accelerate the computations of attention blocks. Overall, The implements of the linear attention and our proposed protocols improve the efficiency of the framework while ensuring its accuracy, making it more efficient when handling long input sequences. Finally, our framework achieves considerable speedups compared to previous MPC based frameworks for Transformer while maintaining similar accuracy performance with traditional Transformer.

Acknowledgements This work is partially supported by the National Natural Science Foundation of China (62002023, 62002239, 62372417, 62132008, 62071222, U22B2030, U20A20176), Guangdong Provincial Key Laboratory IRADS (2022B1212010006, R0400001-22), Guangdong Province General Universities Key Field Project (New Generation Information Technology) (2023ZDZX1033), Zhejiang Lab open research project (No. K2022PD0AB03), the Natural Science Foundation of Jiangsu Province (BK20220075) and the Fok Ying-Tong Education Foundation for Young Teachers in the Higher Education Institutions of China (No. 20193218210004).

Author Contributions (1) Siqi Liu made substantial contributions to the conception, design, implementation of the work; (2) Siqi Liu, Zhusen Liu, and Donglong Chen analyzed the restults and wrote the main manuscript; (3) Wangchen Dai, Lu Zhou, Zhe Liu, Ray C. C. Cheung, and Çetin Kaya Koç gave sufficient guidance on the paper revision; (4) All authors reviewed the manuscript.

Data Availibility Statement No datasets were generated or analysed during the current study.

Declarations

Conflict of interest The authors declare no conflict of interest.

References

1. Knott, B., Venkataraman, S., Hannun, A., Sengupta, S., Ibrahim, M., Maaten, L.: Crypten: Secure multi-party computation meets machine learning. Adv. Neural. Inf. Process. Syst. **34**, 4961–4973 (2021)
2. Bar-Ilan, J., Beaver, D.: Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In: Proceedings of the 8th Annual ACM Symposium on Principles of Distributed Computing, pp. 201–209 (1989)
3. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. Adv. Neural Inf. Process. Syst. **30** (2017)
4. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., Houlsby, N.: An image is worth 16x16 words: transformers for image recognition at scale. arXiv:2010.11929, [cs.CV] (2020)
5. Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., Guo, B.: Swin transformer: Hierarchical vision transformer using shifted windows. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 10012–10022 (2021)
6. Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al.: Improving language understanding by generative pre-training. OpenAI (2018)
7. Hashemi, H., Wang, Y., Annavaram, M.: DarKnight: a data privacy scheme for training and inference of deep neural networks. arXiv:2006.01300, [cs.CR] (2020)
8. Sun, X., Zhang, P., Liu, J.K., Yu, J., Xie, W.: Private machine learning classification based on fully homomorphic encryption. IEEE Trans. Emerg. Top. Comput. **8**(2), 352–364 (2018)
9. Lindell, Y.: Secure multiparty computation. Commun. ACM **64**(1), 86–96 (2020)
10. Mohassel, P., Zhang, Y.: Secureml: A system for scalable privacy-preserving machine learning. In: 2017 IEEE Symposium on Security and Privacy, pp. 19–38 (2017). IEEE
11. Yao, A.C.: Protocols for secure computations. In: 23rd Annual Symposium on Foundations of Computer Science (sfcs 1982), pp. 160–164 (1982). IEEE
12. Kumar, N., Rathee, M., Chandran, N., Gupta, D., Rastogi, A., Sharma, R.: Cryptflow: Secure tensorflow inference. In: 2020 IEEE Symposium on Security and Privacy, pp. 336–353 (2020). IEEE
13. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
14. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv:1409.1556, [cs.CV] (2015)
15. Wang, Y., Suh, G.E., Xiong, W., Lefaudeux, B., Knott, B., Annavaram, M., Lee, H.-H.S.: Characterization of MPC-based private inference for transformer-based models. In: 2022 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pp. 187–197 (2022). IEEE
16. Wang, S., Li, B.Z., Khabsa, M., Fang, H., Ma, H.: Linformer: Self-attention with linear complexity. arXiv:2006.04768, [cs.LG] (2020)
17. Xiong, Y., Zeng, Z., Chakraborty, R., Tan, M., Fung, G., Li, Y., Singh, V.: Nyströmformer: A nyström-based algorithm for approximating self-attention. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, pp. 14138–14148 (2021)
18. Baker, C.T., Taylor, R.: The numerical treatment of integral equations. J. Appl. Mech. **46**(4), 969 (1979)
19. Katharopoulos, A., Vyas, A., Pappas, N., Fleuret, F.: Transformers are RNNs: Fast autoregressive transformers with linear attention. In: International Conference on Machine Learning, pp. 5156–5165 (2020). PMLR
20. Riazi, M.S., Weinert, C., Tkachenko, O., Songhori, E.M., Schneider, T., Koushanfar, F.: Chameleon: A hybrid secure computation framework for machine learning applications. In: Proceedings of the 2018 on Asia Conference on Computer and Communications Security, pp. 707–721 (2018)
21. Tan, S., Knott, B., Tian, Y., Wu, D.J.: CryptGPU: Fast privacy-preserving machine learning on the GPU. In: 2021 IEEE Symposium on Security and Privacy (SP), pp. 1021–1038 (2021). IEEE

22. Dong, Y., Chen, X., Song, X., Li, K.: FlexBNN: fast private binary neural network inference with flexible bit-width. IEEE Trans. Inf. Forens. Secur. (2023)
23. Zhang, F., Chen, Z., Zhang, C., Zhou, A.C., Zhai, J., Du, X.: An efficient parallel secure machine learning framework on GPUs. IEEE Trans. Parallel Distrib. Syst. **32**(9), 2262–2276 (2021)
24. Sutradhar, K., Om, H.: A privacy-preserving comparison protocol. IEEE Trans. Comput. **72**(6), 1815–1821 (2023). https://doi.org/10.1109/TC.2022.3215640
25. Resende, A., Railsback, D., Dowsley, R., Nascimento, A.C., Aranha, D.F.: Fast privacy-preserving text classification based on secure multiparty computation. IEEE Trans. Inf. Forensics Secur. **17**, 428–442 (2022)
26. Feng, Q., He, D., Liu, Z., Wang, H., Choo, K.-K.R.: SecureNLP: A system for multi-party privacy-preserving natural language processing. IEEE Trans. Inf. Forensics Secur. **15**, 3709–3721 (2020)
27. Chen, T., Bao, H., Huang, S., Dong, L., Jiao, B., Jiang, D., Zhou, H., Li, J., Wei, F.: THE-X: privacy-preserving transformer inference with homomorphic encryption. arXiv:2206.00216, [cs.CR] (2022)
28. Child, R., Gray, S., Radford, A., Sutskever, I.: Generating long sequences with sparse transformers. arXiv:1904.10509, [cs.LG] (2019)
29. Kitaev, N., Kaiser, Levskaya, A.: Reformer: the efficient transformer. arXiv:2001.04451, [cs.LG] (2020)
30. Kenton, J.D., Chang, M.-W., Kenton, L., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of NAACL-HLT, vol. 1, p. 2 (2019)
31. Conneau, A., Lample, G.: Cross-lingual language model pretraining. Adv. Neural Inf. Process. Syst. **32** (2019)
32. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: Advances in Cryptology—CRYPTO'91: Proceedings 11, pp. 420–432 (1992). Springer
33. Clevert, D.-A., Unterthiner, T., Hochreiter, S.: Fast and accurate deep network learning by exponential linear units (ELUs). arXiv:1511.07289, [cs.LG] (2016)
34. Watson, J.-L., Wagh, S., Popa, R.A.: Piranha: A GPU platform for secure computation. In: 31st USENIX Security Symposium, pp. 827–844 (2022)
35. Xiao, H., Rasul, K., Vollgraf, R.: Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. arXiv:1708.07747, [cs.LG] (2017)