2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P) | 978-1-6654-6512-0/23/$31.00 ©2023 IEEE | DOI: 10.1109/EuroSP57164.2023.00031

Privformer: Privacy-preserving Transformer with MPC

1st Yoshimasa Akimoto
University of Tsukuba
Tsukuba, Ibaraki, Japan
yoshimasa@mdl.cs.tsukuba.ac.jp

2nd Kazuto Fukuchi
University of Tsukuba, RIKEN AIP
Tsukuba, Ibaraki, Japan
fukuchi@cs.tsukuba.ac.jp

3rd Youhei Akimoto
University of Tsukuba, RIKEN AIP
Tsukuba, Ibaraki, Japan
akimoto@cs.tsukuba.ac.jp

4th Jun Sakuma
Tokyo Institute of Technology, RIKEN AIP
Meguro, Tokyo, Japan
sakuma@c.titech.ac.jp

***Abstract*—The Transformer is a deep learning architecture that processes sequence data. The Transformer attains the state-of-the-art in several tasks of sequence data analysis, and its variants, such as BERT and GPT-3, are used as a defacto-standard for solving general tasks in natural language processing (NLP). This work presents a 3-party multiparty computation (MPC) protocol for secure inference of the Transfomer in the honest majority setting. The attention layer is the most time-consuming part when implementing an MPC protocol for the Transformer with existing building blocks. The attention mechanism is a core component of the Transformer that captures and exploits complex dependencies among elements in the input sequences. The attention mechanism invokes the exponentiation function $O(S^2)$ times, which becomes a major bottleneck when implementing the Transformer with existing MPC primitives. To deal with this, we employ the Performer [11], a variant of the Transformer where the sigmoid function that invokes the exponentiation function is replaced with the ReLU function, a more MPC-friendly nonlinear function. Also, by introducing a kernel-based approximation of the attention matrix with random orthogonal matrices, we show that the attention layer can be processed with $O(S)$ times calls of the ReLU function. We investigate the efficiency of the proposed method by an end-to-end implementation of the Transformer with 3-party MPC. Experimental evaluation shows that, for translating a sequence where the output sequence length is 64, the entire computation time takes about 19 minutes in the LAN environment.**

1. Introduction

The Transformer [49] has been commonly used in machine learning for natural language processing (NLP) tasks such as machine translation [51], text summarization [33], question answering [8]. The Transformer was initially applied to NLP tasks that require exploiting complex dependencies among elements in the input sequences (e.g., dependency parsing for determination of grammatical structures). Subsequently, it has been applied to visual question answering (VQA) [10], image classification [18], image generation [56], and protein fold prediction (Alpha fold 2) [24]. More information on the Transformer and its variants can be found in [48].

The Transformer trained on a large and general dataset (e.g., general text corpus) is called a pre-trained model (e.g., BERT, GPT-3) and is known to deal with various tasks. When applying a pre-trained model to a specific task, the model is finetuned by transfer learning using a small amount of training data specifically prepared for that task. By combining pre-training and transfer learning, highly accurate recognition performance can be achieved even with small amounts of training data in a reasonable computation time.

One difficulty in the use of Transformer-based models is their huge model size. For example, GPT-3 is a sentence generation language model which contains about 175 billion parameters. Thus, when using Transformer-based models, the model is often deployed on a cloud server rather than local computing resources. At the time of use, the information flow follows the API model; users send input data to the server, which processes it with the model and returns only the results to the users.

From the users' perspective, data privacy is one of the main issues in cloud-based model deployment. In the API model, users must submit all data to the cloud server. For example, consider cloud-based machine translation of a document. Sending the entire document to a cloud server would not be acceptable if a document is highly confidential. Also, simultaneous interpretation of conversations is becoming possible with state-of-the-art machine translation technology. However, for the same reason, a cloud-based API model is not realistic when confidentiality protection is mandatory.

Privacy concerns can be resolved by distributing the entire model to users and performing all computations on the user side. However, recent state-of-the-art Transformer-based models are extremely large in scale and require GPUs for processing. Preparing such a large-scale computing environment on the user side is also unrealistic. In addition, the model is often finetuned for a specific task at great effort and cost. Such a model is a valuable information asset, and the distribution of such models to users might not be acceptable.

Several architectures to process sequence data have been developed for the last decade, such as recurrent

DOI 10.1109/EuroSP57164.2023.00031

neural networks (RNN), LSTM, and Seq2Seq. Currently, pre-trained Transformer variants such as BERT (Bidirectional Encoder Representations from the Transformers) [17], and GPT (Generative Pre-trained Transformers)-3 [6] are recognized as the state-of-the-art for a large number of NLP tasks and recognized used as a defacto-standard. Considering the situations discussed above, this work aims to develop a 3-party MPC protocol for secure inference of the Transfomer in the honest majority setting, which allows utilization of the Transformer deployed in a cloud server through API, preserving both model and user privacy.

1.1. Related Work

Privacy protection in deep learning inference and training has made significant progress in the past five years. Most of those studies rely on one or a combination of privacy-preserving computation techniques, such as homomorphic encryption [46], [22], secret sharing [32], [44], [13] and combination of them [45], [23], [35]. Privacy-preserving inference and training of convolutional neural networks (CNNs) have been investigated intensively, and it is being demonstrated that privacy-preserving computation of CNN with image-net scale data is feasible by CrypTFlow [26], CrypTFlow 2 [43], CryptGPU [47], [39], to name a few. The survey paper [43] summarizes recent advances in privacy-preserving computation techniques for deep learning.

The application of privacy-preserving computation techniques to deep learning models for sequence analysis (e.g., natural language processing, speech recognition, time series analysis) is still fewer than those for image recognition. However, several works investigated privacy-preserving computation of deep neural networks for sequence analysis, such as recurrent neural networks (RNN), LSTM, Seq2Seq, and Transformer-based models.

SIRNN [42] proposed a 2-party protocol for RNN to process inference with sequence data. They introduced communication-efficient protocols to process non-linear functions that are costly to process with multi-party computation (MPC), such as exponential function, sigmoid function, $\tanh$, and square root inverse, using lookup tables and mixed-bit width. The protocol comprises a hybrid of secret sharing and oblivious transfer and guarantees semi-honest security. Feng et al. proposed n-party protocols using both additive and multiplicative secret sharing to compute sigmoid function and $\tanh$, which are typical nonlinear functions used in neural networks [19]. Using them, Feng et al. proposed PrivLSTM and PrivSeq2Seq, which securely compute inference with LSTM and Seq2Seq in the semi-honest adversary model.

Wang et al. proposed a n-party MPC protocol to realize an inference with an approximation of the Transformer in the semi-honest adversary model using additive and binary secret sharing [53]. This work focused on the fact that the evaluation of the softmax function is dominant in the entire computation of the Transformer; they proposed to reduce the evaluation time of the softmax function by using Linformer [52] and Nystromformer [55], which approximately compute the Transformer. The goal and methodology of this work are close to ours; however, there is a significant difference in the following two points. First, Nystromformer considers the encoder part of the Transformer only, and the decoder part that involves the masked attention is not considered. Masked attention is a necessary component to deal with complex tasks, such as machine translation and sentence generation. As we will discuss in Section 5.2 intentively, special consideration is needed to apply a mask matrix to the attention matrix with MPC. Second, the framework assumes the semi-honest adversary model, and security against malicious adversaries is not supported. Our protocols work in the honest majority setting.

Chen et al. presented a protocol, THE-X, for secure inference with the Transformer using homomorphic encryption [9]. Chen et al. introduced an approximation of the Transformer with polynomial activation functions by finetuning. Given a pretrained Transformer model, non-polynomial functions, such as Gaussian error linear units(GELU), softmax, and layer normalization, are replaced with polynomial substitutes, and the entire model is finetuned so that its inference performance is improved. Their experimental evaluation assessed the predictive performance while its computation time and communication bandwidth were not reported. Lee et al. also proposed privacy-preserving embeddings based on homomorphic (CKKS) encryption and demonstrated its performance by text classification on the encryption of embeddings by BERT [30]. The framework supports efficient GPU implementation of the CKKS encryption scheme; however, only the downstream classification (logistic regression) part is secured by homomorphic encryption, and the entire sequence encoding part is not secured.

Apart from the privacy-preserving computation of the Transformer, there have been a few studies on model privacy and input privacy. Lang et al. and Coavoux et al. investigated the risk of information leak from latent representations obtained by the Transformer encoder and presented defense methods using adversarial training [12], [29]. Lu et al. revealed the risk of inverting gradients of the Transformer using gradient information collected in the process of federated learning [34]. Qu et al. investigated the same risk of leakage of private inputs and its countermeasure using differential privacy [41]. The goal of these studies is orthogonal to ours, and these studies employ different security models.

1.2. Our Contribution

The Transformer encoder layer is useful for some tasks, such as classification. However, when the Transformer is required to work as a generative model, such as machine translation and sentence generation, the decoder part containing the masked attention is necessary. In this study, we develop MPC protocols for end-to-end computation of inference with the full Transformer model, including the encoder layer and decoder layer.

The contribution of this work is summarized as follows. We design a 3-party MPC protocol for secure inference of Transfomer in the honest majority setting. MPC protocols in the honest majority setting have been pioneered by Araki et al. [3] and Furukawa et al. [20], and then applied to inference and training of machine learning models in ABY3 [16], SecureNN [50], and FALCON [31].

MPC protocols introduced by FALCON realize privacy-preserving computations commonly used for deep neural networks, such as matrix multiplication, rectified linear unit (ReLU), derivative of ReLU (DReLU), and batch normalization. Some of them are designed using a combination of techniques used in ABY3 and SecureNN. Our protocol also heavily relies on protocols presented in these works.

Our technical contributions are three-fold:

1) an MPC protocol for attention with $O(S)$ calls of ReLU function in $O(1)$ rounds,
2) two MPC protocols for masked attention; one requires $O(S^2)$ time in $O(1)$ rounds, and the other achieves $O(S)$ time complexity while running in $O(S)$ rounds,
3) a novel MPC protocol for square root inverse.

First, we present an efficient MPC protocol, ReLU attention, to compute the attention matrix, which is a core component contained in the encoder layer of the Transformer. Implementation of the attention layer can be realized by combining existing protocols naively, whereas such a naive composition results in inefficiency for the following reasons. With preliminary investigation, we reveal that the evaluation of exponentiation in the sigmoid function of the attention layer is the bottleneck of the entire computation (Section 4). More precisely, given an input sequence of length S, the exponentiation function on MPC is invoked $O(S^2)$ times in the attention layer. To deal with this, we introduce two techniques: (1) we replace the sigmoid function in the layer with the ReLU function (ReLU attention, Section 5), and (2) we approximate the Transformer with the Performer [11], which is a kernel-based generalization of the attention. Since the ReLU function is more MPC friendly, replacing sigmoid with ReLU improves computational efficiency. Also, the approximation of ReLU attention with random orthogonal matrices by the Performer allows processing the attention layer with only $O(S)$ calls of the ReLU function.

Second, we experimentally investigate the efficiency of two MPC protocols to process the masked attention layer, another core component contained in the decoder layer of the Transformer. Different from the MPC protocol for (non-masked) ReLU attention, MPC for masked attention requires $O(S^2)$ calls of ReLU even with approximation (ReLU attention$_{QK}$ in Section 5.2.1). This stems from the fact that the mask matrix needs to be applied to the $S \times S$ attention matrix irrespective of approximation. One remedy is to apply the mask to attention vectors sequentially so that MPC does not need to deal with the entire attention matrix. With this modification, we can reduce the invocation of MPC for ReLU to $O(S)$ times; instead, the round complexity increases to $O(S)$ due to sequential masking (ReLU attention$_{VK}$ in Section 5.2.2). Considering this difference in the round and communication complexity, the superiority of the two protocols depends on the communication environment. Through experimental evaluation, we reveal the computational efficiency of these strategies for masked attention.

Third, we introduce a novel MPC protocol for square root inverse. Square root inverse is contained in the batch normalization layer or layer normalization layer and commonly appears not only for the Transformer, but also for various neural networks, and has been investigated in SIRNN and FALCON. To the best of our knowledge, only FALCON [31] investigated MPC for square-root inverse in the honest majority setting. One limitation of this protocol is that it approximately estimates $\log_2 b$ where b is the input and shares it among parties in the form of cleartext, which can leak information (variance of signals) about private input sequences. We propose an MPC protocol for square root inverse that does not leak any intermediate values and is secure in the honest majority setting.

The manuscript is organized as follows. Section 2 introduces the necessary building blocks used for our proposed method. Section 3 defines our problem and the threat model. Section 4 discusses problems when realizing the Transformer by combining existing MPC protocols. Section 5 proposes an MPC protocol for ReLU attention and masked ReLU attention. Section 6 introduces an MPC protocol for square root inverse. In Section 7, we construct Privformer, an end-to-end MPC protocol to compute the Transformer using proposed building blocks. Section 8 discusses the security of the proposed protocols. Section 9 demonstrates the efficiency of the protocols experimentally, and Section 10 concludes our work.

2. Preliminaries

2.1. Secure Multiparty Computation

Secret sharing. Let P_0, P_1, P_2 be parties participating in computation. The party after P_i and the party before P_i is denoted by P_{i+1} and P_{i-1}, respectively where P_{-1} stands for P_2 and P_3 stands for P_0. Let $x \in \mathbb{Z}_m$ be a secret value. We consider 2-out-of-3 replicated secret sharing (RSS). In what follows, $\langle x \rangle_m$ indicates that P_1 holds (x_1, x_2), P_2 holds (x_2, x_3), and P_3 holds (x_3, x_1) where $x \equiv x_1 + x_2 + x_3 \pmod{m}$ holds. x can be recovered with information possessed by any two out of the three parties. When x_1, x_2, x_3 are chosen uniformly at random on $\mathbb{Z}_m$ with satisfying $x \equiv x_1 + x_2 + x_3 \pmod{m}$, a single corruption does not leak anything about x.

In this study, we use $L = 2^{2^\ell}$ where $\ell = 5$ or 2 for the modulo L following FALCON [31]. Also, in order to treat real numbers on $\mathbb{Z}_L$, $x \in \mathbb{R}$ is converted to a fixed-point representation $\hat{x} = \lfloor x \cdot 2^{\mathrm{FP}} \rfloor \mod L$ where the precision is set as $\mathrm{FP} = 13$ bits.

We introduce secure multi-party computation (MPC) for basic algebraic operations and several functions customized for deep learning that work with secret sharing. All protocols work with three parties. The linear operation, multiplication, reconstruction, matrix multiplication, and select shares are commonly known building blocks for secure multi-party computation [31], [36], [50]. For ReLU (rectified linear unit) and DReLU (the derivative of ReLU), which are functions customized for deep learning computation, we employ building blocks introduced by FALCON [31]. Since previous studies have shown that these building blocks work for deep learning inference with practical size models, we use these building blocks in our method.

In this manuscript, we describe a protocol for MPC as a mapping from 2-out-of-3 RSSs to 2-out-of-3 RSSs. Let f be a mapping $f : X_1 \times X_2 \times \ldots \times X_N \mapsto Y_1 \times Y_2 \times$

$\ldots \times Y_M$. Then, for an input $x_1 \in X_1, \ldots, x_N \in X_N$, and output $y_1 \in Y_1, \ldots, y_M \in Y_M$ where $(y_1, \ldots, y_M) = f(x_1, \ldots, x_N)$, an MPC protocol to compute f is denoted by $\Pi_f(\langle x_1 \rangle_L, \ldots, \langle x_N \rangle_L) \rightarrow (\langle y_1 \rangle_L, \ldots, \langle y_M \rangle_L)$.

Linear operation. Addition and subtraction can be computed locally. Suppose three parties hold two shares $\langle x \rangle_L$ and $\langle y \rangle_L$. Then, computing $x_i + y_i \mod L$ and $x_{i+1} + y_{i+1} \mod L$ locally by P_i where $i \in \{0, 1, 2\}$ yields $\langle x \pm y \rangle_L$. Multiplication by a constant can also be computed locally. Suppose c is a public constant, and each party holds $\langle x \rangle_L$. Then, by computing $cx_i \mod L$ and $cx_{i+1} \mod L$ locally by P_i where $i = 0, 1, 2$, each party obtains $\langle cx \rangle_L$.

Multiplication $\Pi_{\text{Mult}}(\langle x \rangle_L, \langle y \rangle_L) \rightarrow \langle xy \rangle_L$. Π_{Mult} is a protocol to compute the multiplication of two distinct shares. Suppose three parties hold two shares $\langle x \rangle_L$ and $\langle y \rangle_L$ where x and y are represented by the fixed point representation with precision bit FP. Assuming the result of multiplication can be represented with the fixed point representation of FP bits, three parties obtain shares of $\langle xy \rangle_L$ by executing Π_{Mult} with these inputs. After multiplication, the truncation protocol [20], [36] needs to be applied to the result of multiplication to keep the consistency of the fixed-point precision. MPC for neural networks involves a large number of matrix multiplications, and the choice of multiplication scheme has a significant impact on the overall efficiency. Detailed discussion on this point is shown in Appendix A.

Reconstruction $\Pi_{\text{Reconst}}(\langle x \rangle_L) \rightarrow x$. Π_{Reconst} is a protocol to take all shares as input and reconstruct the value represented by the share. When three parties hold shares of $\langle x \rangle_L$, all parties obtain x by executing Π_{Reconst}.

Matrix Multiplication $\Pi_{\text{MatMul}}(\langle X \rangle_L, \langle Y \rangle_L) \rightarrow \langle XY \rangle_L$. Π_{MatMul} is a protocol to compute the multiplication of two distinct shares of matrices. Suppose three parties hold shares of two matrices $\langle X \rangle_L \in \mathbb{Z}_L^{P \times Q}$ and $\langle Y \rangle_L \in \mathbb{Z}_L^{Q \times R}$. Then, by execution of Π_{MatMul}, three parties obtain shares of $\langle XY \rangle_L \in \mathbb{Z}_L^{P \times R}$. This protocol also requires keeping the consistency of the scale after execution. We remark that the communication complexity of this protocol depends on the size of the output matrix $P \times Q$ only, not on the size of the input matrices $P \times Q$ and $Q \times R$.

Select Shares $\Pi_{\text{Select}}(\langle x_0 \rangle_L, \langle x_1 \rangle_L, \langle b \rangle_2) \rightarrow \langle x_b \rangle_L$. Π_{Select} takes shares of two values, $\langle x_0 \rangle_L, \langle x_1 \rangle_L$ and a share of a bit, $\langle b \rangle_2$, as input, and returns a share specified by the bit, $\langle x_b \rangle_L$, to parties.

ReLU $\Pi_{\text{ReLU}}(\langle x \rangle_L) \rightarrow \langle \text{ReLU}(x) \rangle_L$. The Rectified Linear Unit (ReLU) is an activation function that is commonly used for deep neural networks

$$\text{ReLU}(x) = \begin{cases} x \text{ if } x > 0, \\ 0 \text{ o.w..} \end{cases} \tag{1}$$

Π_{ReLU} is a protocol that applies ReLU to an input share. When three parties hold a share $\langle x \rangle_L$, three parties obtain a share $\langle ReLU(x) \rangle_L$ by executing Π_{ReLU}.

DReLU $\Pi_{\text{DReLU}}(\langle x \rangle_L) \rightarrow \langle \text{DReLU}(x) \rangle_L$. Π_{DReLU} is a protocol that computes the derivative of the Rectified Linear Unit (ReLU)[1],

$$\text{DReLU}(x) = \begin{cases} 0 \text{ if } x < 0, \\ 1 \text{ if } x \geq 0. \end{cases} \tag{2}$$

Given share $\langle x \rangle_L$, $\Pi_{\text{DReLU}}(\langle x \rangle_L)$ returns a share $\langle \text{DReLU}(x) \rangle_L$.

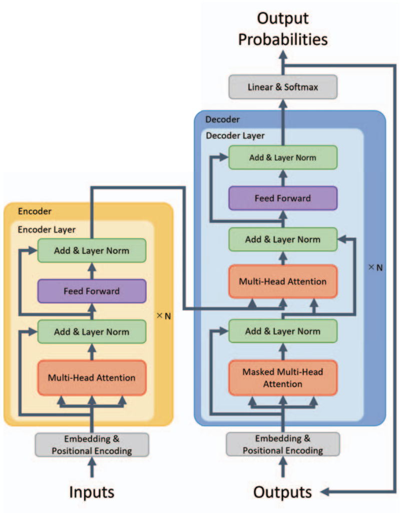


Figure 1. Schematic diagram of the Transformer model.

2.2. Transformer

In the following, we denote the ith row vector of matrix A by A_i throughout the paper. The Transformer [49] consists of two different types of neural networks, the encoder and decoder. Given an input sequence (e.g., a sequence of English words), the encoder outputs a sequence of latent representations (Fig. 1, left). The decoder takes two inputs, (1) the sequence of latent representations provided by the encoder and (2) all previous outputs of the decoder itself. With these inputs, the decoder outputs a probability vector over output elements (e.g., a probability vector over Spanish words) (Fig.1, right). In order to obtain a full output sequence, the decoder iterates the above computation until it outputs the termination signal.

Let X be an embedding matrix of an input sequence and Y be an embedding matrix of the output sequence. Then, the process of the Transformer is abstractly formulated by $\text{Transformer}(W, X) = Y$ where W represents all the model parameters of the Transformer. We remark that the input and output sequence dimensions can be different.

A schematic diagram of the Transformer is shown in Figure 1. The encoder and decoder are composed of N layers of the encoder and decoder layer, respectively. In the following, the computation process required for each sublayer is introduced. An explanation of the embedding and positional encoding is shown in Appendix B since it is not considered in our protocol.

Multi-Head Attention. The attention mechanism takes as input a sequence of latent representation in the

[1] DReLU(0) is not defined. In our implementation, 1 is returned when $x = 0$ for calculation stability.

form of matrices $Q, K, V \in \mathbb{R}^{S\times d}$, and it outputs the same form of output matrix $\mathrm{MultiHeadAttention}(Q, K, V) \in \mathbb{R}^{S\times d2}$. Given three matrices Q (query), K (key), and V (value) in the latent representation, the attention is evaluated by,

$$\mathrm{Attention}(Q, K, V) = \mathrm{Softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V \tag{3}$$

where $\mathrm{Softmax}(\cdot)$ applies the softmax function to each row vector of the matrix[3]. Intuitively, the (i, j) element of QK^T (attention weight matrix) represents the degree of relevance between the ith row vector in the query and the jth row vector in the key. The ith attention vector, that is, the ith row vector of $\mathrm{Attention}(Q, K, V)_i$, represents a latent representation of the ith element weighted by the corresponding attention weights.

Multi-head attention is a straightforward extension of the attention mechanism of Eq. 3. We consider h different attention mechanisms with weight matrices $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{d\times d_m}$ for $i = 1, \ldots, h$,

$$\mathrm{head}_i = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{5}$$

where $\mathrm{head}_i \in \mathbb{R}^{S\times d_m}$. Then, multi-head attention combines these attention matrices with weight matrix $W^o \in \mathbb{R}^{hd\times d_m}$ as

$$\mathrm{MHA}(Q, K, V, \Omega) = \mathrm{Concat}(\mathrm{head}_1, ..., \mathrm{head}_h)W^O \tag{6}$$

where $\mathrm{MHA}(Q, K, V, \Omega) \in \mathbb{R}^{S\times d_m}$.

Masked multi-head attention is a variant of multi-head attention, which is used in the decoder layer only. In multi-head attention in the encoding layer, attention is evaluated using attention weights with any pair of elements in the input sequence. On the other hand, in the decoding process at the decoder, the output sequence is not yet complete. So, attention in the decoding layer needs to be evaluated only with pairs of elements that the decoder has already generated. For example, in the translation task, the decoder takes as input a word sequence after translation. The word sequence of the translated words must be generated from the front, and when predicting the i-th translated word, only information up to the $i-1$th word should be used. For this reason, the lower triangle matrix is multiplied by the attention matrix to hide the information associated with untranslated words. Softmax attention applies the softmax function to each element of the attention matrix. So, a huge negative value is added to the attention weights corresponding to elements that have not yet been generated:

$$\mathrm{MaskedAttention}(Q_i, K_i, V_i) = \mathrm{Softmax}\left(M_i + \frac{Q_i K_i^T}{\sqrt{d_k}}\right)V_i \tag{7}$$

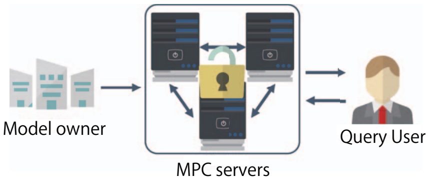


Figure 2. Protocol overview.

where M_i is the mask matrix, which forms an upper triangle matrix. Elements with a non-zero value in M are determined, reflecting the decoding progress.

Add & Layer Normalization. Input to layer normalization is a summation of two matrices, $X' + X'' = X \in \mathbb{R}^{S\times d_m}$. Here, $X' \in \mathbb{R}^{S\times d_m}$ is the output of the previous layer (multi-head attention or feed-forward network), and X'' is the input of the previous layer $X' \in \mathbb{R}^{S\times d_m}$ obtained through skip connection. Let $X_{i,k}$ denotes the (i, k) element of X and $\boldsymbol{x}_i \in \mathbb{R}^{d_m}$ denotes the ith row vector of X. First, layer normalization transforms latent representation vectors given X by

$$\mu_i = \frac{1}{d_m}\sum_{k=1}^{d_m} X_{i,k}, \sigma_i^2 = \frac{1}{d_m}\sum_{k=1}^{d_m}(X_{i,k} - \mu_i)^2, \tag{8}$$

$$\hat{X}_{i,k} = \frac{X_{i,k} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} \tag{9}$$

where ϵ is a small positive constant to stabilize the calculation. Then, layer normalization outputs

$$\mathrm{LN}_{\gamma,\beta}(\hat{\boldsymbol{x}}_i) = \boldsymbol{\gamma} \odot \hat{\boldsymbol{x}}_i + \boldsymbol{\beta} \tag{10}$$

where $\odot$ denotes the Hadamard product, and $\gamma \in \mathbb{R}^{d_m}$ and $\boldsymbol{\beta} \in \mathbb{R}^{d_m}$ are model parameters. Details of layer normalization can be found in Xiong et al.'s work [54].

Feed-Forward Network. Let $X \in \mathbb{R}^{S\times d_m}$ be an input and $\boldsymbol{x}_i$ denote the ith row vector of X. Then, the fully-connected layer applies (1) an Affin transformation with $(W^{F1}, \boldsymbol{b}^{F1})$ to each row vector of X, $\boldsymbol{x}_i$, (2) activation with ReLU, (3) another Affing transformation with $(W^{F2}, \boldsymbol{b}^{F2})$

$$\mathrm{FFN}(\boldsymbol{x}_i) = \mathrm{ReLU}(W^{F1}\boldsymbol{x}_i + \boldsymbol{b}^{F1})W^{F2} + \boldsymbol{b}^{F2}, \tag{11}$$

and outputs the resulting matrix, which has the same size as the input.

Linear & Softmax. Input to this layer is a matrix of latent representation, $X \in \mathbb{R}^{S\times d_m}$. For each row vector $\boldsymbol{x}_i$ of X, a linear transformation $\boldsymbol{z}_i = \boldsymbol{W}\boldsymbol{x}_i$ is applied where $\boldsymbol{W} \in \mathbb{R}^{d_m\times V}$ and V is the vocabulary size of the embedding. Then, the softmax function is applied as $\boldsymbol{y}_i = \mathrm{softmax}(\boldsymbol{z}_i)$. Here, $\boldsymbol{y}_i \in [0, 1]^V$ denotes the vector of word occurrence probabilities. The output word sequence is obtained by choosing the word having the highest probability for each element in the sequence.

3. Problem Formulation

3.1. Problem Definition

We consider three types of parties, model owner, query user, and multi-party computation (MPC) servers (Fig.

[2] Q, K, and V are taken from embedding vectors while its treatment depends on the application. For unsupervised language models, Q, K, and V are taken from the embedding of the input sequence. For machine translation, Q is taken from the embedding of the target (output) sequence, and K, V are from the embedding of the source (input) sequence.

[3] Given a row vector $\boldsymbol{x}$, the ith element of the output vector of the softmax function is

$$\mathrm{Softmax}(x)_i = \frac{e^{x_i}}{\sum_{k=1}^{d} e^{x_k}}. \tag{4}$$

2). We suppose a model owner owns a pretrained model parameter W of the Transformer. W is private and cannot be revealed to any other parties, while the model owner allows anyone to evaluate $Y = \mathrm{Transformer}(W, X)$ for any input sequence X. A query user owns an embedding matrix X associated with an input sequence T. X is private and cannot be revealed to any other parties while the query user wishes to compute $Y = \mathrm{Transformer}(W, X)$ privately and obtain Y with keeping Y private to anyone else. Also, we suppose three MPC servers jointly help to compute $Y = \mathrm{Transformer}(W, X)$ upon request from the model owner and the query user. The MPC servers do not own any private input and behave as servers that provide a secure multi-party computation service for inference with the Transformer.

In this work, we suppose the Transformer model W is pre-trained, and inference with the pretrained Transformer model is considered. Model training for the Transformer with MPC is outside the scope of this study. Then, the model of privacy-preserving Transformer inference we consider in this study is stated as follows. The model owner provides the three MPC servers with secret shares of the model parameter W; the query user also provides the three MPC servers with secret shares of input X. Here, we suppose all computations of MPC servers run on 2-out- of-3 RSS. The three MPC servers then jointly perform MPC to compute $Y = \mathrm{Transformer}(W, X)$, which will be presented later. After protocol execution, each MPC server obtains two elements of secret shares of Y, which are given to the query user so that it can reconstruct Y.

3.2. Threat Model

We assume honest parties are the majority among the three parties in the above problem setting. This honest majority setting has been widely employed in privacy-preserving machine learning in the MPC setting [31], [36], [37], [50]. In our three-server setting, honest majority means that two out of the three MPC servers are honest adversaries. An honest adversary is a party that follows the prescribed protocol honestly and does not collude mutually to gather collected information. At most, one MPC server may act as a semi-honest adversary or malicious adversary. A semi-honest adversary is a party that does not deviate from the protocol and collaboratively gathers information. A malicious adversary is a party that may use any attack strategy and thus arbitrarily deviates from the protocol specification.

Suppose a particular institution uses MPC to provide privacy-preserving services. Then, if we can ensure that (1) the number of external attackers that intrudes into the system is kept equal or less than 1, and (2) collusion among server administrators can be avoided, the honest majority setting is useful to secure the service. In addition, in the honest majority setting, any information about X (resp. W) is not leaked even with collusion of a single MPC server and the model owner (resp. query user).

We do not protect our protocol against denial-of-service (DoS) attacks in which a party refuses to cooperate. For such adversaries, we simply abort the computation. We assume that the three parties communicate through a shared point-to-point communication channel. The three parties are also assumed to have pair-wise shared seeds, which are private to parties not contained in the pair. The seeds are utilized for a pseudo-random number generator (PRNG). AES is used as the PRNG in our implementation.

4. Computing the Transformer with Known MPC Building Blocks

This section discusses the difficulties in the implementation of MPC protocol for Transfomer with existing building blocks. We remark that analysis on embedding, the feed-forward network layer, and the linear and softmax layer are shown in Appendix C because these layers can be realized with a naive composition of existing protocols.

Multi-Head Attention. Multi-head attention contains two computations, (1) softmax function and (2) multiplication of large matrices. Technically, it is possible to execute multi-head attention with MPC by approximating nonlinear functions with polynomial functions. However, the execution time cannot be practical when using existing building blocks due to the following reasons.

In the multi-head attention, the $S \times S$ sized attention matrix is first obtained with matrix multiplication. The communication complexity of matrix product with MPC depends on its output matrix size, and the communication complexity of this part is $O(S^2)$. Then, the sigmoid function is applied to each row vector in the matrix, which invokes the exponentiation function $O(S^2)$ times.

When computing the Transformer with MPC using primitives in Section 2.1, all steps other than this layer invoke the exponentiation function $O(S)$ times. If we can avoid matrix multiplication that outputs $S \times S$ size matrix and reduces invocation of exponentiation function to $O(S)$, the entire computation time of protocol execution would apparently improve. We consider this problem in Section 5.

Layer Normalization. Layer normalization requires the computation of $\frac{1}{\sqrt{x}}$, which cannot be processed by naive application of building blocks in Section 2.1. To evaluate the square-root inverse, FALCON [31] introduces an MPC protocol that obtains an approximation solution using the iterative Newton method. This protocol requires linear operations only. One concern of this protocol is that it reveals some information on the private input in the middle of the protocol (see Section 6 in detail). We present a novel MPC protocol for square-root inverse that does not leak any intermediate values in Section 6.

5. ReLU Attention

5.1. MPC for ReLU Attention

The previous section discussed that the softmax attention requires $O(S^2)$ times calls of MPC for exponentiation. We introduce a generalized attention mechanism to deal with the difficulties that arise in the computation of softmax attention on MPC. This mechanism has been introduced as a building block of the Performer [11], originally proposed as a generalization of the Transformer. Softmax attention evaluates the similarity matrix (attention weight matrix) using the softmax function. Choromanski et al. proposed to generalize the similarity matrix so that the attention mechanism can employ an arbitrarily defined similarity using the kernel theory [11]. By choosing the kernel appropriately, the Performer can reduce the computation complexity of inference without sacrificing the predictive performance.

Our idea is to take advantage of the generality of the similarity matrix to make the computation of this part more MPC friendly. Particularly, we introduce ReLU attention. When evaluating nonlinear functions on MPC, it is usually substituted to polynomial approximation, which introduces approximation error, and its computation can be costly when the polynomial dimension is large. To avoid this, we use the ReLU function as nonlinear activation because it can be evaluated exactly with multiplication and logical OR only. Since ReLU attention consists of these MPC-friendly operations only, it is expected to run in reasonable computation time. Also, the attention weight matrix of the ReLU attention asymptotically approaches the original attention weight matrix of the Transformer in the limit of the feature map dimension (explained later). So, the predictive performance is guaranteed to be close to the Transformer when a sufficiently large feature map is used [11].

ReLU attention. The softmax attention of Eq. 5 evaluates the similarity between the key and query by the normalized dot product of a key and vector, followed by the application of the softmax function. Given query $Q \in \mathbb{R}^{S\times d}$, key $K \in \mathbb{R}^{S\times d}$, and value $V \in \mathbb{R}^{S\times d}$, ReLU attention of the Performer approximates the similarity matrix using the Gaussian kernel based on a random feature map as follows:

$$\begin{gathered}\mathrm{ReLUAttention}(Q,K,V) = cQ'(K'^TV) \\ \text{where } Q' = \mathrm{ReLU}(Q\Omega) \text{ and } K' = \mathrm{ReLU}(K\Omega).\end{gathered} \tag{12}$$

Here, $\Omega \in \mathbb{R}^{d\times r}$ is a random projection matrix, and r denotes the dimension of the space after projection by random projection matrix Ω. Each row Ω_i is sampled from

$$\Omega_i \sim \mathcal{N}(0, \boldsymbol{I}_d) \tag{13}$$

and orthogonalized by the Gram-Schmidt method. c is a public constant for normalization. Originally, c is determined to be dependent on inputs. However, since treating c as a constant does not significantly impact the final predictive performance [11], we treat c as a constant.

Introducing a kernel with ReLU and random feature maps can prevent the computation of exponentiation. The attention weight matrix of ReLU attention can be computed with matrix product and ReLU only. Since both operations can be immediately computed with existing building blocks efficiently, employing ReLU attention is a great advantage in MPC implementation.

Protocol. ReLU attention with MPC can be immediately obtained following the computation of Eq. 12. Alg.1 is an MPC protocol to compute ReLU Attention with three parties, P_0, P_1 and P_2. Three parties possess shares of $Q, K, V \in \mathbb{Z}_L^{S\times d}$ and $\Omega \in \mathbb{Z}_L^{d\times r}$. c is possessed by the three parties in cleartext because c is a non-private data-independent constant. The query user provides shares of Q, K, and V with the MPC servers. Also, the model owner provides shares of random orthogonal matrix Ω with the MPC servers.

Algorithm 1 ReLU Attention Π_{ReLUAtt} :

```
Require: Each $P_j$ holds $\langle Q\rangle_L, \langle K\rangle_L, \langle V\rangle_L \in \mathbb{Z}_L^{S\times d}$,
    $\langle \Omega\rangle_L \in \mathbb{Z}_L^{d\times r}$. Public constant $c \in \mathbb{R}$ for normalization.
Ensure: Each $P_j$ gets $\langle \mathrm{ReLUAtt}(Q,K,V)\rangle_L \in \mathbb{Z}_L^{S\times d}$
 1: $\langle Q\Omega\rangle_L \leftarrow \Pi_{\mathrm{MatMul}}(\langle Q\rangle_L, \langle\Omega\rangle_L)$
 2: $\langle K\Omega\rangle_L \leftarrow \Pi_{\mathrm{MatMul}}(\langle K\rangle_L, \langle\Omega\rangle_L)$
 3: $\langle Q'\rangle_L \leftarrow \Pi_{\mathrm{ReLU}}(\langle Q\Omega\rangle_L)$
 4: $\langle K'\rangle_L \leftarrow \Pi_{\mathrm{ReLU}}(\langle K\Omega\rangle_L)$
 5: $\langle K'^TV\rangle_L \leftarrow \Pi_{\mathrm{MatMul}}(\langle K'\rangle_L^T, \langle V\rangle_L)$
 6: $\langle Q'K'^TV\rangle_L \leftarrow \Pi_{\mathrm{MatMul}}(\langle Q'\rangle_L, \langle K'^TV\rangle_L)$
 7: $\langle \mathrm{ReLUAttention}(Q,K,V)\rangle_L = c \cdot \langle Q'K'^TV\rangle_L$
```

In Eq. 12, matrix product $Q\Omega, K\Omega, K'^TV$ and $Q'(K'^TV)$ can be computed using Π_{MatMul}. ReLU of $K' = \mathrm{ReLU}(K\Omega)$ and $Q' = \mathrm{ReLU}(QK\Omega)$ can be computed using Π_{ReLU}. After execution of Π_{ReLUAtt}, each party obtains shares of $\mathrm{ReLUAttention}(Q,K,V) \in \mathbb{Z}_L^{S\times d}$. Thus, the computation of Eq. 12 on MPC can be realized using the existing building blocks.

Complexity analysis. In actual protocol execution, the execution of MPC for non-polynomial functions heavily affects the overall efficiency. Thus, we evaluate the number of execution times of MPC for exponentiation and ReLU function. Also, we evaluate the communication and round complexity. In the following complexity analysis, we regard the dimension of the latent representation d as a constant. We thus consider the complexity with respect to the sequence length S and the dimension of the random feature map r.

We compare the complexity of MPC for ReLU attention in Eq. 12 with MPC for softmax attention in Eq. 3. Recall that the communication complexity of matrix products on MPC depends on the output matrix size. In the evaluation of softmax attention on MPC, $QK^T \in \mathbb{R}^{S\times S}$ needs to be evaluated, and its communication complexity is $O(S^2)$ (Table 1, line 2). In ReLU attention, $Q\Omega$ and $K\Omega$ are computed first, which results in $S \times r$ matrices, and its communication complexity is $O(rS)$. Next, ReLU attention computes $K'^TV \in \mathbb{R}^{r\times d}$ and then $Q(K'^TV) \in \mathbb{R}^{S\times d}$, whose communication complexity is $O(r)$ and $O(S)$, respectively (with ignoring d). Thus, the overall communication complexity is reduced to $O(rS)$ by introducing ReLU attention (Table 1, line 3).

Softmax attention invokes the exponentiation function for $O(S^2)$ times, while ReLU attention invokes ReLU function $O(S)$ times in steps 3 and 4. For each input element, all computations are independent. So, the round complexity of the entire protocol is $O(S)$.

With the complexity analysis above, MPC for ReLU attention is expected to be more efficient than MPC for softmax attention, particularly when the input sequence S is long. We will show a detailed efficiency analysis of ReLU attention with experimental results in Section 9.

5.2. MPC for Masked ReLU Attention

In this subsection, we propose two protocols for masked ReLU attention; one requires $O(S^2)$ time in $O(1)$ rounds, and the other achieves $O(S)$ time complexity while running in $O(S)$ rounds.

Algorithm 2 Masked ReLU Attention$_{\text{QK}}$ $\Pi_{\text{MReLUAtt}_{\text{QK}}}$:

```
Require: Each $P_j$ holds $\langle Q\rangle_L, \langle K\rangle_L, \langle V\rangle_L \in \mathbb{Z}_L^{S\times d}$, $\langle\Omega\rangle_L \in \mathbb{Z}_L^{d\times r}$. Public constant $M_{\text{tril}} \in \mathbb{Z}_L^{S\times S}$ for mask and $c \in \mathbb{R}$ for normalization.
Ensure: Each $P_j$ gets $\langle \text{MaskedReLUAttention}_{\text{QK}}(Q,K,V)\rangle_L \in \mathbb{Z}_L^{S\times d}$
1: $\langle Q\Omega\rangle_L \leftarrow \Pi_{\text{MatMul}}(\langle Q\rangle_L, \langle\Omega\rangle_L)$
2: $\langle K\Omega\rangle_L \leftarrow \Pi_{\text{MatMul}}(\langle K\rangle_L, \langle\Omega\rangle_L)$
3: $\langle Q'\rangle_L \leftarrow \Pi_{\text{ReLU}}(\langle Q\Omega\rangle_L)$
4: $\langle K'\rangle_L \leftarrow \Pi_{\text{ReLU}}(\langle K\Omega\rangle_L)$
5: $\langle Q'K'^T\rangle_L \leftarrow \Pi_{\text{MatMul}}(\langle Q'\rangle_L, \langle K'\rangle_L^T)$
6: $\langle \text{tril}(Q'K'^T)\rangle_L = M_{\text{tril}} \odot \langle Q'K'^T\rangle_L$
7: $\langle \text{tril}(Q'K'^T)V\rangle_L \leftarrow \Pi_{\text{MatMul}}(\langle \text{tril}(Q'K'^T)\rangle_L, \langle V\rangle_L)$
8: $\langle \text{MaskedReLUAttention}(Q,K,V)\rangle_L = c \cdot \langle \text{tril}(Q'K'^T)V\rangle_L$
```

5.2.1. Masked ReLU Attention in $O(1)$ Rounds.

In softmax attention, a mask matrix M is applied to QK^T as softmax$((M + QK^T/\sqrt{d})_i)$ in Eq. 7. The ReLU attention analog for masked attention is defined straightforwardly by

$$\begin{aligned} \text{MReLUAtt}(Q,K,V) &= c \cdot \text{tril}(Q'(K')^T)V \quad (14)\\ &= c(M_{\text{tril}} \odot Q'(K')^T)V \\ \text{where } Q' = \text{ReLU}(Q\Omega) &\text{ and } K' = \text{ReLU}(K\Omega). \end{aligned}$$

Here, tril() is the function of extracting the lower triangular part of the matrix and filling the remaining elements with zero. M_{tril} is a mask matrix with 1 in the lower triangular part and 0 in the remaining elements. $\odot$ denotes Hadamard (element-wise) product.

Protocol. We design an MPC for masked ReLU attention using Eq. 14 in Alg. 2. We suppose P_0, P_1 and P_2 possess shares of $Q, K, V \in \mathbb{Z}_L^{S\times d}, \Omega \in \mathbb{Z}_L^{d\times r}$, $G \in \{0\}^{r\times d}$, and constant c. In this protocol, the query user prepares shares of Q, K, V and provides them with MPC servers. Also, the model owner (query user) needs to prepare shares of a random orthogonal matrix Ω and provide them with the MPC servers. Mask matrix M_{tril} is public because it depends only on the index of the element in the sequence the protocol is processing. After the execution of the protocol, three parties obtain shares of masked ReLU attention in $\mathbb{Z}_L^{S\times d}$.

Eq. 14 can be computed by matrix product and ReLU only, and its realization with MPC is immediately obtained as Alg. 2. Since $Q'^T K'$ is computed first in this algorithm, we call this MaskedReLUAttention$_{\text{QK}}$.

Complexity analysis. Complexity analysis of masked softmax attention and masked ReLU attention$_{QK}$ is summarized in Table 1, line 4 and line 5, respectively. In both protocols, the matrix product that outputs the greatest sized matrix is $Q'K'^T \in \mathbb{R}^{S\times S}$ and its communication complexity is $O(S^2)$. This increase in communication complexity is due to the change in the order of matrix multiplication. In ReLU attention without the mask, $K'^T V$ is computed first. By doing so, we can avoid computation with $S \times S$ sized matrix. In contrast, masked ReLU attention needs to have $Q'K'^T$ first to apply the mask.

Algorithm 3 Masked ReLU Attention$_{\text{VK}}$ $\Pi_{\text{MReLUAtt}_{VK}}$:

```
Require: Each $P_j$ holds $\langle Q\rangle_L, \langle K\rangle_L, \langle V\rangle_L \in \mathbb{Z}_L^{S\times d}$, $\langle\Omega\rangle_L \in \mathbb{Z}_L^{d\times r}$ where $\Omega$ is random orthogonal, $\langle G\rangle_L$ where $G \in \{0\}^{d\times r}$, and constant $c \in \mathbb{R}$ for normalization.
Ensure: Each $P_j$ gets $\langle \text{MReLUAtt}_{\text{Naive}}(Q,K,V)\rangle_L \in \mathbb{Z}_L^{S\times d}$
1: $\langle Q\Omega\rangle_L \leftarrow \Pi_{\text{MatMul}}(\langle Q\rangle_L, \langle\Omega\rangle_L)$
2: $\langle K\Omega\rangle_L \leftarrow \Pi_{\text{MatMul}}(\langle K\rangle_L, \langle\Omega\rangle_L)$
3: $\langle Q'\rangle_L \leftarrow \Pi_{\text{ReLU}}(\langle Q\Omega\rangle_L)$
4: $\langle K'\rangle_L \leftarrow \Pi_{\text{ReLU}}(\langle K\Omega\rangle_L)$
5: for i = $\{1, ..., S\}$ do
6:     $\langle G_i\rangle_L \leftarrow \Pi_{\text{MatMul}}(\langle K_i'\rangle_L^T, \langle V_i\rangle_L)$
7:     $\langle G\rangle_L$ = $\langle G\rangle_L$ + $\langle G_i\rangle_L$.
8:     $c\langle \text{tril}(Q'K'^T)V_i\rangle_L \leftarrow c \cdot \Pi_{\text{MatMul}}(\langle Q_i'\rangle_L, \langle G\rangle_L)$
9: end for
10: Each $P_j$ concatenates $c\langle \text{tril}(Q'K'^T)V_1\rangle_L, ..., c\langle \text{tril}(Q'K'^)TV_S\rangle_L$ and outputs the resulting matrix of shares as $\langle \text{MReLUAtt}(Q,K,V)\rangle_L$
```

Masked softmax attention invokes exponentiation function for $O(S^2)$ times while masked ReLU attention$_{QK}$ invokes ReLU function $O(S)$ times in steps 3 and 4. For each input element, all computations are independent. So, the round complexity of the entire protocol is $O(S)$.

5.2.2. Masked ReLU Attention in $O(S)$ Time.

Masked ReLU Attention$_{\text{QK}}$ needs to compute QK^T, which causes $O(S^2)$ time/communication complexity. To avoid the quadratic dependence on S, we introduce an iterative approach to attain the $O(S)$ time/communication protocol for masked ReLU attention. In Eq. 14, the application of mask is realized by taking the lower triangular part of the matrix. This masking can be reformulated by iterative product of row vectors of K' and V' by

$$\begin{aligned} [\text{tril}(Q'(K')^T)V]_i &= Q_i' G_i \\ &\text{where } G_j = {K_j'}^T V_j \in \mathbb{R}^{r\times d}, G_i = \sum_{j=1}^{i} G_j. \end{aligned} \quad (15)$$

Here, we have the ith row of $\text{tril}(Q'(K')^T)V$ as the result. The entire masked matrix is obtained by computing Eq. 15 for $i = 1, \ldots, S$ and concatenating the resulting row vectors into a matrix.

Protocol. We design an MPC for masked ReLU attention using Eq. 15 in Alg. 3. Since $V^T K'$ is computed first in this algorithm, we call this protocol MaskedReLUAttention$_{\text{VK}}$. We suppose P_0, P_1 and P_2 possess shares of $Q, K, V \in \mathbb{Z}_L^{S\times d}, \Omega \in \mathbb{Z}_L^{d\times r}$, $G \in \{0\}^{r\times d}$, and constant c. After the execution of the protocol, three parties obtain shares of masked ReLU attention in $\mathbb{Z}_L^{S\times d}$.

$\text{ReLU}(Q\Omega), \text{ReLU}(K\Omega)$ are computed in Steps 1-4 in the same manner as Alg.1. Steps 6-14 use the matrix sum and matrix product to compute shares of $\text{tril}(Q'K'^T)V$ for each row following Eq. 15. In step 15, the results are concatenated into a matrix.We remark that the reconstructed

TABLE 1. Comparison of the number of nonlinear function calls, communication complexity, and round complexity of softmax attention and ReLU attention.

	Nonliner	Comm.	Round
Softmax Attention	$O(S^2)$	$O(S^2)$	$O(1)$
ReLU Attention	$O(S)$	$O(rS)$	$O(1)$
Masked Softmax Attention	$O(S^2)$	$O(S^2)$	$O(1)$
Masked ReLU Attention$_{\mathrm{QK}}$	$O(S)$	$O(S^2)$	$O(1)$
Masked ReLU Attention$_{\mathrm{VK}}$	$O(S)$	$O(rS)$	$O(S)$

outputs of $\Pi_{\mathrm{MaskedReLUAttention}_{VK}}$ are equivalent to those of $\Pi_{\mathrm{MaskedReLUAttention}_{QK}}$ when given the same inputs.

Complexity analysis. The complexity analysis of this protocol is summarized in Table 1, line 6. Steps 1-4 are the same as maskedReLUattention$_{QK}$ and its communication complexity is $O(rS)$. The size of matrix G_i computed in step 6 is $r \times d$ and its communication complexity is $O(r)$ when regarding d as constant. The size of matrix $Q'K'^TV_i$ computed in step 8 is $1 \times r$, and its communication complexity is $O(r)$, too. These steps are repeated for S times, so the communication complexity of the entire protocol $O(rS)$.

Masked ReLU attention$_{VK}$ invokes ReLU function $O(S)$ times in steps 3 and 4. Also, computation in the for-loop (steps 6-8) is dependent and requires one round for each execution. So the round complexity is $O(S)$.

MaskedReLUattention$_{QK}$ has $O(S^2)$ communication complexity with constant rounds. On the other hand, MaskedReLUattention$_{VK}$ runs in $O(S)$ communication complexity while it requires $O(S)$ rounds. Thus, which protocol is more efficient depends on the computing environment. We will give a detailed experimental comparison of these protocols in Section 9.

6. Square-root Inverse

6.1. Square-root Inverse in FALCON

FALCON [31] proposed an MPC protocol for batch normalization among three parties. In the protocol, three parties take shares of b as input and compute shares of $\frac{1}{\sqrt{b}}$ in the following way.

1) Given shares of b as input, find α such that $2^\alpha \leq b < 2^{\alpha+1}$ in the cleartext and share it among the three parties by using protocol Π_{pow} (See Section 3.7 in Li et al.'s work [31])
2) Find the initial value for the Newton iteration as $x_0 \leftarrow 2^{-\lceil \alpha/2 \rceil}$ in the cleartext. Then, shares of x_0 are distributed among the three parties.
3) Given shares of x_0 and b as input, repeat the following Newton iterations (four times)
$$x_{i+1} \leftarrow \frac{x_i}{2}(3 - bx_i^2) \tag{16}$$
4) Output shares of x_{i+1} as an approximation of $\frac{1}{\sqrt{b}}$

In the first step, α is computed in the form of shares of a ℓ-digit binary number, $\alpha[i], i \in \{\ell - 1, ..., 0\}$. Then, the protocol reconstructs shares of each bit $\langle \alpha[i] \rangle_2$ and computes $\alpha = \sum_i 2^{\alpha[i]}$ in the cleartext to find the initial value for the Newton iteration. Since α is a good approximation (lower bound) of private input b, it is preferable to avoid revealing α in the cleartext.

Algorithm 4 Sqrt Inverse $\Pi_{\mathrm{SqrtInverse}}$:

```
Require: Each $P_j$ holds $\langle b \cdot 2^{\mathrm{FP}} \rangle_L$. Public constant $2^{\mathrm{FP}}$ and $2^{\frac{\mathrm{FP}}{2}} \cdot 2^{\mathrm{FP}}$
Ensure: Each $P_j$ gets $\langle \frac{1}{\sqrt{b}} \cdot 2^{\mathrm{FP}} \rangle_L$    ▷ Input:
 1: $\langle t \rangle_L \leftarrow \langle 1 \rangle_L$.
 2: for $i = \ell - 1, ..., 1, 0$ do    ▷ Calculate $\langle \alpha[0] \rangle_2, ..., \langle \alpha[\ell - 1] \rangle_2$ such that $2^\alpha \leq b \cdot 2^{\mathrm{FP}} < 2^{\alpha+1}$
 3:     $\langle \alpha[i] \rangle_2 \leftarrow \Pi_{\mathrm{DReLU}}(\langle b \cdot 2^{\mathrm{FP}} \rangle_L - 2^{2^i} \langle t \rangle_L)$
 4:     $\langle t \rangle_L \leftarrow \Pi_{\mathrm{Select}}(\langle t \rangle_L, 2^{2^i} \langle t \rangle_L, \langle \alpha[i] \rangle_2)$
 5: end for
 6: $\langle \beta \rangle_L \leftarrow 2^{\frac{\mathrm{FP}}{2}} \cdot 2^{\mathrm{FP}} \langle 1 \rangle_L$
 7: for $i = 0, 1, ..., \ell - 1$ do ▷ Calculate $\frac{1}{\sqrt{2^\alpha}} \cdot 2^{\frac{\mathrm{FP}}{2}} \cdot 2^{\mathrm{FP}}$ as $\beta$
 8:     $\langle \beta' \rangle_L \leftarrow \Pi_{\mathrm{Mult}}(\langle \beta \rangle_L, 2^{-2^{i-1}} \cdot 2^{\mathrm{FP}} \langle 1 \rangle_L)$
 9:     $\langle \beta \rangle_L \leftarrow \Pi_{\mathrm{Select}}(\langle \beta \rangle_L, \langle \beta' \rangle_L, \langle \alpha[i] \rangle_2)$
10: end for
11: $\langle x_0 \rangle_L \leftarrow \langle \beta \rangle_L$.
12: for $i = 0, ..., 3$ do ▷ Calculate the approximate value of $\frac{1}{\sqrt{b}} \cdot 2^{\mathrm{FP}}$ using Newton's method
13:     $\langle x_i^2 \rangle_L \leftarrow \Pi_{\mathrm{Mult}}(\langle x_i \rangle_L, \langle x_i \rangle_L)$
14:     $\langle bx_i^2 \cdot 2^{\mathrm{FP}} \rangle_L \leftarrow \Pi_{\mathrm{Mult}}(\langle b \cdot 2^{\mathrm{FP}} \rangle_L, \langle x_i^2 \rangle_L)$
15:     $\langle x_{i+1} \rangle_L \leftarrow \Pi_{\mathrm{Mult}}(\frac{\langle x_i \rangle_L}{2}, 3 \cdot 2^{\mathrm{FP}} \langle 1 \rangle_L - \langle bx_i^2 \cdot 2^{\mathrm{FP}} \rangle_L)$
16: end for
17: $\langle \frac{1}{\sqrt{b}} \cdot 2^{\mathrm{FP}} \rangle_L \leftarrow \langle x_4 \rangle_L$
```

6.2. Secure Square-root Inverse on MPC

In this subsection, we introduce Alg. 4, which is an MPC protocol that computes shares of $\frac{1}{\sqrt{b}}$ without revealing α. P_0, P_1, P_2 takes shares of $b \cdot 2^{\mathrm{FP}} \in \mathbb{Z}_L$ as input and obtains shares of $\frac{1}{\sqrt{b}} \cdot 2^{\mathrm{FP}} \in \mathbb{Z}_L$.

Alg. 4 proceeds as follows. $\langle 1 \rangle_L$ in step 1 is given as $(0, 0, 1)$ deterministically. Given shares of b, the for-loop (steps 2-5) finds shares of α such that $2^\alpha \leq b{\cdot}2^{\mathrm{FP}} < 2^{\alpha+1}$. Here, 2^{FP} is multiplied to treat b as an FP-bit fixed-point number. Using a similar idea used in Π_{Pow} of FALCON, we can find $\langle \alpha[i] \rangle_2, i = 0, \ldots \ell - 1$ using Π_{DReLU}. We remark that, in this protocol, $\langle \alpha[i] \rangle_2$ are not reconstructed and treated as shares in the following computation.

The for-loop (steps 7-10) converts shares of α into shares of $1/\sqrt{2^\alpha}$, which is used as an initial value for the Newton iteration. Precicely, we find shares of $\beta = \frac{1}{\sqrt{2^\alpha}} \cdot 2^{\frac{\mathrm{FP}}{2}} \cdot 2^{\mathrm{FP}}$ to treat the value as a fixed point number. $\frac{\mathrm{FP}}{2}$ is a factor to fill in the gap that arises in the subsequent calculation. Given α in the binary digit representation, $2^{-\alpha/2}$ can be obtained as $2^{-\alpha/2} = \prod_{\ell=0}^{\ell-1} \alpha[i] \cdot 2^{-2^{i-1}}$. To compute this on MPC, we first set $\langle \beta \rangle_L = \langle 2^{\frac{\mathrm{FP}}{2}} \cdot 2^{\mathrm{FP}} \rangle_L$ as the initial value (step 6). Then, for each element in $\{\langle \alpha[0] \rangle_2, ..., \langle \alpha[\ell - 1] \rangle_2\}$, $\langle \beta \rangle_L$ is unchanged when $\alpha[i] = 0$; otherwise multiply $2^{-2^{i-1}}$ to $\langle \beta \rangle_L$ selectively (step 8-9). Here, protocol Select Shares Π_{Select} is used for selective multiplication by $\alpha[i]$. After processing all elements, we have $\langle \beta \rangle_L = \frac{1}{\sqrt{2^\alpha}} \cdot 2^{\frac{\mathrm{FP}}{2}} \cdot 2^{\mathrm{FP}}$, which works as the initial value for the Newton iteration.

In the for-loop (steps 12-16), using $\langle \beta \rangle_L$ as the initial value of the Newton iteration, Eq. 16 is iterated on MPC four times. Finally, shares $\langle x_4 \rangle_L = \langle \frac{1}{\sqrt{b}} \cdot 2^{\mathrm{FP}} \rangle_L$ are obtained.

7. Privformer

This section presents the overall protocol for the Transformer using protocols for ReLU Attention and Layer Normalization proposed in the previous sections. Protocol for the feed-forward network is described in Appendix since it can be constructed with existing building blocks only.

7.1. Multi-Head Attention

P_0, P_1, P_2 takes shares of $Q, K, V \in \mathbb{Z}_L^{S\times d_m}$, model parameters $W_i^Q, W_i^K, W_i^V \in \mathbb{Z}_L^{d_m\times d}$ for $1 \leq i \leq H$, and $W^O \in \mathbb{Z}_L^{d_m\times d_m}$ as input, and outputs $\langle \mathrm{MHA}(Q, K, V, \Omega) \in \mathbb{Z}_L^{S\times d_m}\rangle$ after execution of the protocol.

$QW_i^Q, KW_i^K, VW_i^V, \mathrm{Concat}(\mathrm{head}_1, ..., \mathrm{head}_H)W^O$ can be computed using Π_{MatMul}. Each head_i is computed using either Alg.1 when used in the encoder or Alg. 2 or Alg. 3 when used in the decoder. The concatenation of $\langle \mathrm{head}_i\rangle$ in Step 9 can be computed locally. The entire protocol is described in Appendix D.1.

Masked multi-head attention Π_{MMHA} can be processed by replacing $\Pi_{\mathrm{ReLUAtt}}(\langle QW_i^Q\rangle_L, \langle KW_i^K\rangle_L, \langle VW_i^V\rangle_L)$ with $\Pi_{\mathrm{MReLUAtt}}(\langle QW_i^Q\rangle_L, \langle KW_i^K\rangle_L, \langle VW_i^V\rangle_L)$.

7.2. Layer Normalization

Alg. 5 is a protocol for computing layer normalization with three parties. P_0, P_1, and P_2 take shares of $X \in \mathbb{Z}_L^{S\times d_m}$ and $\beta, \gamma, \epsilon \in \mathbb{Z}_L^{d_m}$ as input and outputs shares of $\mathrm{LN}_{\gamma,\beta}(X)$ after execution of the protocol.

Computation of $\langle \mu_i\rangle_L$ requires addition and multiplication by a constant, which can be computed locally (step 2). $\tilde{X}_{i,k}$ is computed as the difference between input $X_{i,k}$ and mean μ_i first (step 4), and then Π_{mult} is used to compute its square (step 5). $\langle \sigma_i^2\rangle$ is obtained by summation over i, followed by division by a constant, which can be computed locally, too (step 7).

For numerical stability of the square root inverse in the next step, we add a small positive constant value ϵ to σ_i^2 (step 8), denoted by $\langle b_i\rangle$. Normalization (step 11) is attained by multiplying $\tilde{X}_{i,k}$ with $\langle \frac{1}{\sqrt{b_i}}\rangle$, which is obtained by using $\Pi_{\mathrm{InverseSqrt}}$ introduced in Section 6 with $\langle b_i\rangle$ (step 9). Eq. 10 is computed immediately by Π_{mult} and linear operations (steps 13,14).

7.3. Putting Everything Together

Protocols for the encoder and decoder are realized by combining protocols for multi-head attention, feed-forward network, and layer normalization as building blocks. The encoder is composed of N layers of encoder Layers. In a single encoder Layer, sub-layers are processed in the order of MHA, LN, FFN, and LN. Each sub-layer can be calculated by using $\Pi_{\mathrm{MHA}}, \Pi_{\mathrm{FFN}}$, and Π_{LN}, respectively. The entire protocol of the encoder is described in Appendix D.3. The decoder is also composed of N layers of decoder layers. In a single decoder Layer, sub-layers are processed in the order of MaskedMHA, LN, MHA, LN, FFN, and LN. Each sub-layer can be calculated by using $\Pi_{\mathrm{MHA}}, \Pi_{\mathrm{FFN}}$, and Π_{LN}. The entire protocol of the encoder is described in Appendix D.4.

Algorithm 5 Layer Normalization Π_{LN}

```
Require: Each $P_j$ holds $\langle X\rangle_L \in \mathbb{Z}_L^{S\times d_m}, \langle\beta\rangle_L, \langle\gamma\rangle_L \in \mathbb{Z}_L^{d_m}$. Public constant $\epsilon$.
Ensure: Each $P_j$ gets $\langle \mathrm{LN}_{\gamma,\beta}(X)\rangle_L$
 1: for $i = \{1, ..., S\}$ do
 2:     $\langle\mu_i\rangle_L = \frac{1}{d_m}\sum_{k=1}^{d_m}\langle X_{i,k}\rangle_L$
 3:     for $k = \{1, ..., d_m\}$ do
 4:         $\langle\tilde{X}_{i,k}\rangle = \langle X_{i,k}\rangle_L - \langle\mu_i\rangle_L$
 5:         $\langle\tilde{X}_{i,k}^2\rangle \leftarrow \Pi_{\mathrm{Mult}}(\langle\tilde{X}_{i,k}\rangle_L, \langle\tilde{X}_{i,k}\rangle)$
 6:     end for
 7:     $\langle\sigma_i^2\rangle_L = \frac{1}{d_m}\sum_{k=1}^{d_m}\langle\tilde{X}_{i,k}^2\rangle_L$
 8:     $\langle b_i\rangle_L = \langle\sigma_i^2\rangle_L + \langle\epsilon\rangle_L$.
 9:     $\langle\frac{1}{\sqrt{b_i}}\rangle_L \leftarrow \Pi_{\mathrm{SqrtInverse}}(\langle b_i\rangle_L)$
10:     for $k = \{1, ..., d_m\}$ do
11:         $\langle\hat{X}_{i,k}\rangle \leftarrow \Pi_{\mathrm{Mult}}(\langle\tilde{X}_{i,k}\rangle_L, \langle\frac{1}{\sqrt{b_i}}\rangle)$
12:     end for
13:     $\langle\gamma\hat{X}_i\rangle \leftarrow \Pi_{\mathrm{Mult}}(\langle\gamma\rangle_L, \langle\hat{X}_i\rangle)$
14:     $\langle \mathrm{LN}_{\gamma,\beta}(X_i)\rangle_L \leftarrow \langle\gamma\hat{X}_i\rangle_L + \langle\beta\rangle_L$
15: end for
```

Recall that embedding of input sequences and extraction of output sequences from embedding matrices are outside the scope of this study and performed locally at the query user's side in our construction. The entire computation of Privformer is summarized in the list in Appendix D.5.

8. Security Analysis

The security analysis of the protocol in the presence of malicious adversaries commonly follows ideal/real simulation paradigm [7] [21]. The security of a protocol is proved by comparing what an adversary can do in a real execution to what the adversary can do in the ideal execution. Protocol execution in the ideal world involves a trusted third party, which obtains the input from each party, computes the prescribed functionality honestly, and returns the output to each party. Informally, we say that the protocol is secure if, for every adversary in the real world, there exists a simulator in the ideal world that can produce communication transcripts, including inputs and outputs that are indistinguishable from those in the real execution.

Theorem 1.2 in Kushilevitz et al.'s work [27] states that every protocol that is perfectly secure in the stand-alone model and has a straight-line black-box simulator is secure under concurrent general composition. Informally speaking, perfect security requires that the distribution of inputs and outputs of the adversary and participating parties in the real and ideal executions are exactly the same. In the stand-alone model, only a single protocol execution takes place. A black box simulator is a universal ideal world adversary that interacts with a real adversary only through oracle access. Furthermore, we say a black-box simulator is straight-line if it does not rewind.

Theorem 1. *[Theorem 1.2 in Kushilevitz et al., 2009] Every protocol that is perfectly secure in the stand-alone* *model, and has a straight-line black-box simulator, is secure under concurrent general composition.*

The methodology for security analysis of our protocols basically follows that of FALCON [31], which relies on this theorem.

The methodology for security analysis of our protocols basically follows that of FALCON [31], which relies on this theorem. Protocols Π_{ReLUAtt}, $\Pi_{\mathrm{MReLUAtt}_{QK}}$, $\Pi_{\mathrm{MReLUAtt}_{VK}}$,$\Pi_{\mathrm{SqrtInv}}$, Π_{MHA},Π_{LN}, Π_{FFN}, Π_{Encoder}, and Π_{Decoder} are designed with sequential/concurrent execution of local computation (e.g., data oblivious concatenation), linear operations, and security protocols which are perfectly secure in the stand-alone model, and has a straight-line black-box simulator (see Table 4 for the detailed description for inputs and outputs of every entity and invoked subprotocols). Thus, when information theoretically correlated randomness (e.g., Section 2.2 in Araki et al.'s work [3]) is used in these protocols, the application of concurrent general composition by Theorem 1 proves the perfect security of the protocols presented in this study.

One limitation of using information theoretically correlated randomness is that it doubles the communication per AND gate. Since the Privformer requires a large number of multiplication, communication is a major bottleneck. For practicality, we introduced computational correlated randomness using a pseudo-random number generator (PRNG) for our implementation. We remark that experimental evaluations of other major MPC neural networks (ABY3, SecureNN, FALCON, etc.) also used PRNGs for correlated randomness.

Since multiplication and matrix multiplication is considered to consume a large portion of communication bandwidth in our protocol construction, we can expect constant times reduction of communication bandwidth by introducing PRNG. Unfortunately, the concurrent general composition may not be applicable when using PRNGs because of a pathological counterexample shown in Section 4 in Kushilevitz et al.'s work [28]; however, except for such an artificial case, composition with PRNG can be regarded as secure practically. Such "hypothetical" security is accepted in practice, including security proofs in the random oracle model.

We conjecture that the security of composition when using PRNG can be proved by the hybrid argument, where each pseudo-random number generation is replaced with ideal calls to the corresponding ideal functionality, where rewindings are necessary for the simulation of the composed protocols. Whether this can be proved or whether such proof is necessary for practical use is left to future consideration. In this study, we did not pursue this point because it is far from the main subject. If provable security is required, we can use information theoretically correlated randomness with accepting a doubling of the communication volume. With this consideration, the performance of the proposed protocols is evaluated using PRNG.

9. Experiments

9.1. Experimental setup

This section evaluates the proposed protocols' computation time and communication bandwidth. We implemented each of the proposed protocols on top of the FALCON framework [1]. The experiments were conducted on Amazon EC2 c4.8x large instances using two experimental settings, LAN and WAN. Each instance has OS Ubuntu 18.04 LTS, CPU 2.9 GHz Intel Xeon E5-2666 v3 processor, and RAM 64GB.

LAN Three c4.8x large instances in the same region. The average bandwidth was 4.93 Gbits/s, and the average ping response time was 1.17 ms.

WAN Deploy the same instances as in the LAN configuration in different Ohio, Tokyo, and London regions. The average bandwidth was 97.4 Mbits/s, and the average ping response time was 141.67 ms.

Parameters were set as $L = 2^{2^\ell}, \ell = 5, p = 67$, fixed-point representation with 13-bit precision. The computation time and the communication bandwidth required for protocol execution were measured, assuming all parties behave as semi-honest adversaries. The average of 10 runs is used as the execution time.

Input to the Privformer is an $S \times d_m$ embedding matrix where each element is drawn from the uniform distribution within $[-10.0, 10.0]$, multiplied by 2^{FP} (FP: precision of fixed-point numbers = 13), and rounded to the nearest integer. We set $d_m = 512$ for all experiments, which is a commonly used hyperparameter for embedding. Also, we set $S = 32, 64, \ldots, 1028$ for experiments in Sec. 9.2 and Sec. 9.3 where S corresponds to the length of the input sequence. A sequence with length 1024 is sufficiently long to handle the typical use cases of the Transformer, such as sentiment analysis from sentences in natural language, machine translation, text summarization, and so on. The input embedding matrix we used for experiments corresponds to random sequences, not real sentences. Considering that the computation time and communication do not vary depending on the content of the input sequence, it is sufficient for performance evaluation of MPC as long as the scale of the data is comparable with data for real applications.

9.2. Unmasked Softmax/ReLU Attention

In Section 5.1, we proposed a protocol for (unmasked) ReLU attention that has $O(rS)$ communication complexity. Compared to original (unmasked) softmax attention, whose computation and communication complexities are $O(S^2)$, the proposed protocol's computation time and communication bandwidth are expected to be reduced, particularly when S is large. In addition, unlike softmax attention, ReLU Attention employs the ReLU function for activation, which is more MPC-friendly. This section compares the computation time and the communication bandwidth of protocols of the two unmasked attention mechanisms.

For ReLU attention, the dimensionality of latent representation was set as $d_m = 64$, and the dimensionality of the output space of the random projection matrix Ω was set as $r = d_m \ln d_m$ following the recommendation [11].

For softmax attention, we approximate the exponentiation function to the polynomial obtained by Chebyshev polynomial $\mathrm{Exp}(x) = 1 + x + 0.5x^2 + 0.1665x^3 + 0.0438x^4$, which was employed in Li et al.'s work [31]. For both unmasked attention mechanisms, the inverse was approximately computed by the Newton iteration [31]. Figure 3 shows MPC's computation time and communication bandwidth for the unmasked softmax/ReLU attention mechanisms in LAN and WAN environments when the length of input sequence S was varied as $32, 64, \ldots, 1024$.

First, the computation time of softmax attention shows a quadratic increase while that of ReLU Attention increases linearly in both LAN and WAN environments (Figure 3 (a), (b)). According to the results, when S is less than 128, the computation time of the ReLU attention is almost the same as or less than the softmax attention. This is because the evaluation of ReLU on MPC is more efficient than exponentiation in softmax. When S becomes greater than 128, the computation time of ReLU attention is significantly shorter than softmax attention in both LAN and WAN environments. This gap arises from the difference in the communication complexity and the number of nonlinear function calls.

The same results were obtained for the communication bandwidth. The communication bandwidth increases quadratically for softmax attention and linearly for ReLU attention (Figure 3 (c)). The communication bandwidth for ReLU attention is smaller than softmax attention for $S = 128, \ldots, 1024$. The number of rounds is constant for both softmax Attention and ReLU Attention, but the number of rounds required for ReLU attention is about half of the rounds for softmax attention (Figure 3 (d)). This is because the circuit depth required to evaluate ReLU is shallower than exponentiation. When the sequence length is 1024, the execution time of ReLU attention is about 9.41 times faster in the LAN environment and about 8.25 times faster in the WAN environment than softmax attention. The communication bandwidth is reduced by a factor of 11.67. These results confirm the significant reduction of computation time and communication bandwidth of the proposed ReLU Attention.

TABLE 2. COMPUTATION TIME (S) AND COMMUNICATION TRAFFIC (MB) OF EACH LAYER IN THE TRANSFORMER.

	Time [sec]		Comm	Rounds
	LAN	WAN	[MB]	
Multi-Head Att.	0.611	10.944	48.361	160
Masked Multi-Head Att.	0.613	12.424	47.231	192
Feed-Forward Network	1.671	6.669	19.136	38
Layer Normalization	0.077	10.753	1.64	278
Encoder	14.141	253.164	424.668	4524
Decoder	17.768	402.85	717.894	7344

9.3. Masked Softmax/ReLU Attention

In Section 5.2, we proposed protocols for two different types of masked ReLU attention. One is masked ReLU attention in $O(1)$ rounds (masked ReLU Attention$_{QK}$), and the other is masked ReLU attention in $O(S)$ round (masked ReLU Attention$_{VK}$).

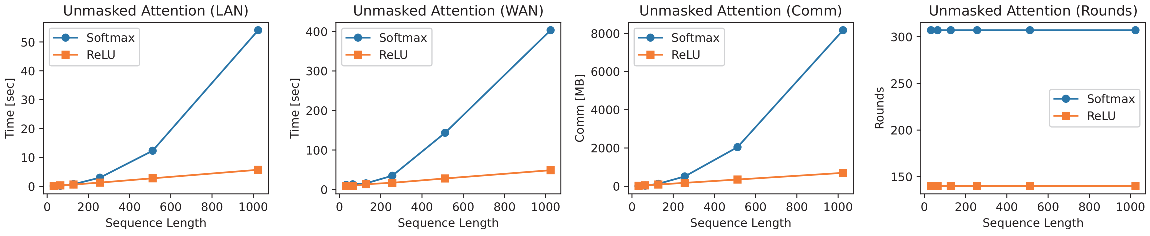
Masked ReLU Attention$_{QK}$ is a protocol that applies a mask matrix to the unmasked version of ReLU attention proposed in Section 5.1. Because of the multiplication by the mask matrix, it cannot achieve linear communication complexity unfortunately, but it runs in constant rounds as well as the unmasked version. Masked ReLU attention$_{QK}$ has the same communication and round complexity as the masked softmax attention. However, it employs ReLU as the activation function, which is more MPC-friendly. Also, the number of evaluations of nonlinear functions by masked ReLU attention is less than that by masked softmax attention. So the computation time is expected to be reduced. Masked ReLU attention$_{VK}$ is a protocol that achieves linear time and linear communication bandwidth by allowing $O(S)$ rounds. This protocol has improved communication complexities. So it is expected to reduce the computation time and communication bandwidth compared to the masked softmax attention. On the other hand, computation in $O(S)$ rounds can be inefficient when the input sequence length S is large.

Whether Masked ReLU attention$_{QK}$ or Masked ReLU attention$_{VK}$ is more useful depends on the input sequence length, communication environment, and communication overhead. This section compares the computation time, communication bandwidth, and rounds of the three masked attention mechanisms.

The results are shown in Figure4. First, we compared the masked softmax attention and masked ReLU attention$_{VK}$. The computation time of masked softmax attention increased quadratically in both LAN and WAN environments. On the other hand, the computation time of masked ReLU attention$_{\mathrm{VK}}$ increased linearly (Figure 4(a), (b)). This is consistent with the results of Table 1. Unfortunately, unlike unmasked attention, the actual computation time of masked ReLU attention$_{\mathrm{VK}}$ was greater than masked softmax attention when $S = 32, \ldots, 512$ in the LAN environment and $S = 32, \ldots, 1024$ in the WAN environment. This can be partially attributed to the fact that masked ReLU attention$_{\mathrm{VK}}$ requires $O(S)$ rounds (Figure 4(d)).

Next, we compared masked softmax attention and masked ReLU attention$_{\mathrm{QK}}$. The computation time for masked softmax attention and masked ReLU attention$_{\mathrm{QK}}$ increased quadratically in both LAN/WAN environments (Figure 4(a), (b)). This is consistent with the results of Table 1. The computation time for masked ReLU attention$_{\mathrm{QK}}$ was shorter than masked softmax attention for $S = 128, \ldots, 1024$ in the LAN environment and $S = 32, \ldots, 1024$ in the WAN environment. Also, at $S = 128, \ldots, 1024$, the communication bandwidth for masked ReLU attention$_{\mathrm{QK}}$ was smaller than that for masked softmax attention. This improvement is considered to be due to the fact that the nonlinear operation used for softmax attention is replaced by ReLU, whose computation time and communication bandwidth are shorter than those of exponential and inverse operations. For sequence length $S = 1024$, masked ReLU attention$_{\mathrm{QK}}$ was about four times faster than masked softmax attention in the LAN environment and about 4.5 times faster in the WAN environment; the communication bandwidth was reduced to about one-ninth.

From the above results, masked softmax attention is the fastest when the sequence length S is short (e.g., $S \leq 64$) in the LAN environment, while Masked ReLU Attention$_{\mathrm{QK}}$ achieves the best performance in other cases. It should be noted that masked ReLU attention$_{\mathrm{VK}}$ was the slowest in the range of sequence lengths tested in these



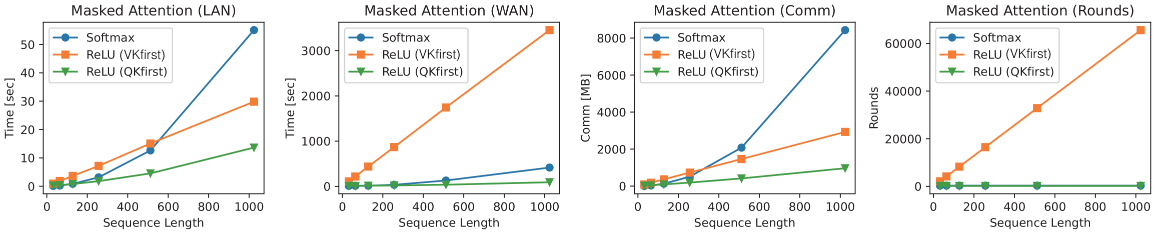
(a) Comp. time (sec., LAN)

(a) Comp. time (sec., WAN)

(c) Comm. bandwidth (MB)

(d) # Rounds

Figure 3. (a) Computation time (s) in LAN, (b) Computation time (s) in WAN, (c) communication bandwidth (MB), and (d) the number of rounds required to complete the protocol of the unmasked Softmax and ReLU attention mechanism when the length of input sequence S was varied as $32, 64, \ldots, 1024$. The horizontal axis represents the length of the input sequence.



(a) Comp. time (sec., LAN)

(a) Comp. time (sec., WAN)

(c) Comm. bandwidth (MB)

(d) # Rounds

Figure 4. (a) Computation time in LAN (s), (b) computation time in WAN (s), (c) communication bandwidth (MB), and (d) the number of rounds required to complete the protocol of the masked Softmax and ReLU attention (with VK first and QK first) mechanism in LAN and WAN environments when the length of input sequence S was varied as $32, 64, \ldots, 1024$. The horizontal axis represents the length of the input sequence.

experiments, but we remark that it would achieve the best performance when the sequence lengths are longer.

9.4. Computation Time and Communication Bandwidth of Layers

The layers that configure the Transformer are multi-head attention, masked multi-head attention [4], feed-forward network, and layer normalization. We implemented an encoder and decoder with these layers and measured the computation time (s) and the communication bandwidth (MB) in the LAN and WAN environments. As model hyperparameters, we used $d_m = 512$ for the number of dimensions of the latent representation, $d_{ff} = 2048$ for the dimension of the middle layer in the feed-forward network, $h = 8$ for the number of heads, and $r = 256$ as the space dimensionality after projection by the random matrix Ω. These parameter settings follow the performance evaluation of the Performer [11]. Results are evaluated for input sequences of length $S = 64$ with batch size $B = 1$.

The results are summarized in Table 2. The feed-forward network spent the largest computation time among the layers configuring the Transformer in the LAN environment. The process of the feed-forward network layer consists of two matrix multiplications and one ReLU. The design of this layer is simple, whereas the size of the matrix used in the computation is large as $d_{ff} \times d_m = 2048 \times 512$, which causes a large amount of computation time. On the other hand, in the WAN environment, the computation time of the feed-forward network is the shortest among the layers. Instead, the masked multi-head attention took the longest computation time. This is because the overhead caused by the communication traffic and many rounds is more significant in the WAN environment.

The encoder consists of four layers: multi-head attention, layer normalization, feed-forward network, and layer normalization. In the LAN environment, the encoder required 14 seconds of computation time and 425 MB of communication bandwidth for an input sequence of 64 lengths. In comparison, the decoder required 18 seconds of computation time and 717 MB of communication bandwidth. This indicates that it takes 14 seconds in the LAN environment to process tasks that require the encoder only, such as topic classification and sentiment analysis from natural language sentences, when the length of the input sequence contains 64 words.

Inference with the full Transformer model requires the execution of the encoder once and the decoder S_o times where S_o corresponds to the length of the output sequence. In the LAN environment, when the output sequence length is 64, the encoder required 14 seconds, and the decoder required $18 \times 64 = 1152$ (s) of computation time with 4.64 GB of communication bandwidth. We remark that computation at the query user side (e.g., word embedding) is ignorable. This indicates that it takes about 19 minutes in a LAN environment to process tasks that require both the encoder and decoder, such as automatic translation and text summarization, when the length of the output sentence contains 64 words. In the WAN environment, for an input/output sequence with a length of 64, the

[4] Considering the experimental results in Section 9.3, we employed masked ReLU attention$_{QK}$ for masked multi-head attention for this experiment

computation time required for processing the encoder was 253 (s), and that by the decoder was $403 \times 64 = 25792$ (s). This indicates that it takes about 7.3 hours to process the same task. Improvement of the computation and communication complexity and computation time for long output sequences is an interesting direction for future work.

10. Conclusion

In this study, we developed a 3-party MPC that can evaluate the inference of the Transformer. Our implementation takes about 20 minutes to infer a 64-word sentence in a LAN environment. Further performance improvements are needed for applications that require real-time operations, such as machine translation. However, it is considered useful for tasks that do not require real-time processing, such as genome data analysis and program code translation. Further speed-up by using parallel processing with GPUs is a promising future direction.

Acknowledgement

The authors would like to thank J. Furukawa for important suggestions for security analysis. We are grateful to the anonymous reviewers for their useful comments. This work was partly supported by JSPS KAKENHI Grant Numbers JP23H00483, JP22H00519, JP22H00521, and JST CREST Grant Number JPMJCR21D3.

References

[1] https://github.com/snwagh/falcon-public.

[2] Toshinori Araki, Assi Barak, Jun Furukawa, Tamar Lichter, Yehuda Lindell, Ariel Nof, Kazuma Ohara, Adi Watzman, and Or Weinstein. Optimized honest-majority mpc for malicious adversaries —breaking the 1 billion-gate per second barrier. In 2017 IEEE Symposium on Security and Privacy (S&P), pages 843–862. IEEE, 2017.

[3] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pages 805–817, 2016.

[4] Mikel Artetxe, Gorka Labaka, Eneko Agirre, and Kyunghyun Cho. Unsupervised neural machine translation. arXiv preprint arXiv:1710.11041, 2017.

[5] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Advances in Cryptology—CRYPTO ’ 91: Proceedings 11, pages 420–432. Springer, 1992.

[6] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. Advances in neural information processing systems, 33:1877–1901, 2020.

[7] Ran Canetti. Security and composition of multiparty cryptographic protocols. Journal of CRYPTOLOGY, 13(1):143–202, 2000.

[8] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions. arXiv preprint arXiv:1704.00051, 2017.

[9] Tianyu Chen, Hangbo Bao, Shaohan Huang, Li Dong, Binxing Jiao, Daxin Jiang, Haoyi Zhou, and Jianxin Li. The-x: Privacy-preserving transformer inference with homomorphic encryption. arXiv preprint arXiv:2206.00216, 2022.

[10] Yen-Chun Chen, Linjie Li, Licheng Yu, Ahmed El Kholy, Faisal Ahmed, Zhe Gan, Yu Cheng, and Jingjing Liu. Uniter: Universal image-text representation learning. In European conference on computer vision, pages 104–120. Springer, 2020.

[11] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. arXiv preprint arXiv:2009.14794, 2020.

[12] Maximin Coavoux, Shashi Narayan, and Shay B Cohen. Privacy-preserving neural representations of text. arXiv preprint arXiv:1808.09408, 2018.

[13] Anders Dalskov, Daniel Escudero, and Marcel Keller. Secure evaluation of quantized neural networks. Proceedings on Privacy Enhancing Technologies, 2020(4):355–375, 2020.

[14] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P Smart. Practical covertly secure mpc for dishonest majority–or: breaking the spdz limits. In Computer Security–ESORICS 2013: 18th European Symposium on Research in Computer Security, Egham, UK, September 9-13, 2013. Proceedings 18, pages 1–18. Springer, 2013.

[15] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Advances in Cryptology–CRYPTO 2012: 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings, pages 643–662. Springer, 2012.

[16] Daniel Demmler, Thomas Schneider, and Michael Zohner. Aby-a framework for efficient mixed-protocol secure two-party computation. In NDSS, 2015.

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805, 2018.

[18] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. arXiv preprint arXiv:2010.11929, 2020.

[19] Qi Feng, Debiao He, Zhe Liu, Huaqun Wang, and Kim-Kwang Raymond Choo. Securenlp: A system for multi-party privacy-preserving natural language processing. IEEE Transactions on Information Forensics and Security, 15:3709–3721, 2020.

[20] Jun Furukawa, Yehuda Lindell, Ariel Nof, and Or Weinstein. High-throughput secure three-party computation for malicious adversaries and an honest majority. In Annual international conference on the theory and applications of cryptographic techniques, pages 225–255. Springer, 2017.

[21] Oded Goldreich. Foundations of cryptography: volume 2, basic applications. Cambridge university press, 2009.

[22] Zhicong Huang, Wen-jie Lu, Cheng Hong, and Jiansheng Ding. Cheetah: Lean and fast secure two-party deep neural network inference. IACR Cryptol. ePrint Arch., 2022:207, 2022.

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

[24] Jeremy Kahn. Lessons from deepmind ’ s breakthrough in protein-folding a.i. In Fortune, 2020.

[25] Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: making spdz great again. In Advances in Cryptology–EUROCRYPT 2018: 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29-May 3, 2018 Proceedings, Part III, pages 158–189. Springer, 2018.

[26] Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow: Secure tensorflow inference. In 2020 IEEE Symposium on Security and Privacy (SP), pages 336–353. IEEE, 2020.

[27] Eyal Kushilevitz, Yehuda Lindell, and Tal Rabin. Information-theoretically secure protocols and security under composition. In Proceedings of the thirty-eighth annual ACM Symposium on Theory of Computing, pages 109–118, 2006.

[28] Eyal Kushilevitz, Yehuda Lindell, and Tal Rabin. Information-theoretically secure protocols and security under composition. SIAM Journal on Computing, 39(5):2090–2112, 2010.

[29] Jiaqi Lang, Linjing Li, Weiyun Chen, and Daniel Zeng. Privacy protection in transformer-based neural network. In 2019 IEEE International Conference on Intelligence and Security Informatics (ISI), pages 182–184. IEEE, 2019.

[30] Garam Lee, Minsoo Kim, Jai Hyun Park, Seung-won Hwang, and Jung Hee Cheon. Privacy-preserving text classification on bert embeddings with homomorphic encryption. arXiv preprint arXiv:2210.02574, 2022.

[31] Shaohua Li, Kaiping Xue, Bin Zhu, Chenkai Ding, Xindi Gao, David Wei, and Tao Wan. Falcon: A fourier transform based approach for fast and secure convolutional neural network predictions. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 8705–8714, 2020.

[32] Jian Liu, Mika Juuti, Yao Lu, and Nadarajah Asokan. Oblivious neural network predictions via minionn transformations. In Proceedings of the 2017 ACM SIGSAC conference on computer and communications security, pages 619–631, 2017.

[33] Yang Liu and Mirella Lapata. Text summarization with pretrained encoders. arXiv preprint arXiv:1908.08345, 2019.

[34] Jiahao Lu, Xi Sheryl Zhang, Tianli Zhao, Xiangyu He, and Jian Cheng. April: Finding the achilles' heel on privacy for vision transformers. arXiv preprint arXiv:2112.14087, 2021.

[35] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A cryptographic inference service for neural networks. In 29th USENIX Security Symposium (USENIX Security 20), pages 2505–2522, 2020.

[36] Payman Mohassel and Peter Rindal. Aby3: A mixed protocol framework for machine learning. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pages 35–52, 2018.

[37] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In 2017 IEEE symposium on security and privacy (SP), pages 19–38. IEEE, 2017.

[38] Masato Neishi, Jin Sakuma, Satoshi Tohda, Shonosuke Ishiwatari, Naoki Yoshinaga, and Masashi Toyoda. A bag of useful tricks for practical neural machine translation: Embedding layer initialization and large batch size. In Proceedings of the 4th Workshop on Asian Translation (WAT2017), pages 99–109, 2017.

[39] Lucien KL Ng and Sherman SM Chow. Gforce: Gpu-friendly oblivious and rapid neural network inference. In USENIX Security Symposium, pages 2147–2164, 2021.

[40] Ye Qi, Devendra Singh Sachan, Matthieu Felix, Sarguna Janani Padmanabhan, and Graham Neubig. When and why are pre-trained word embeddings useful for neural machine translation? arXiv preprint arXiv:1804.06323, 2018.

[41] Chen Qu, Weize Kong, Liu Yang, Mingyang Zhang, Michael Bendersky, and Marc Najork. Natural language understanding with privacy-preserving bert. In Proceedings of the 30th ACM International Conference on Information & Knowledge Management, pages 1488–1497, 2021.

[42] Deevashwer Rathee, Mayank Rathee, Rahul Kranti Kiran Goli, Divya Gupta, Rahul Sharma, Nishanth Chandran, and Aseem Rastogi. Sirnn: A math library for secure rnn inference. arXiv preprint arXiv:2105.04236, 2021.

[43] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow2: Practical 2-party secure inference. In Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, pages 325–342, 2020.

[44] M Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin Lauter, and Farinaz Koushanfar. Xonn:xnor-based oblivious deep neural network inference. In 28th USENIX Security Symposium (USENIX Security 19), pages 1501–1518, 2019.

[45] M Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M Songhori, Thomas Schneider, and Farinaz Koushanfar. Chameleon: A hybrid secure computation framework for machine learning applications. In Proceedings of the 2018 on Asia Conference on Computer and Communications Security, pages 707–721, 2018.

[46] Amartya Sanyal, Matt Kusner, Adria Gascon, and Varun Kanade. Tapas: Tricks to accelerate (encrypted) prediction as a service. In International Conference on Machine Learning, pages 4490–4499. PMLR, 2018.

[47] Sijun Tan, Brian Knott, Yuan Tian, and David J Wu. Cryptgpu: Fast privacy-preserving machine learning on the gpu. In 2021 IEEE Symposium on Security and Privacy (SP), pages 1021–1038. IEEE, 2021.

[48] Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. ACM Computing Surveys (CSUR), 2020.

[49] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Advances in neural information processing systems, pages 5998–6008, 2017.

[50] Sameer Wagh, Divya Gupta, and Nishanth Chandran. Securenn: 3-party secure computation for neural network training. Proc. Priv. Enhancing Technol., 2019(3):26–49, 2019.

[51] Qiang Wang, Bei Li, Tong Xiao, Jingbo Zhu, Changliang Li, Derek F Wong, and Lidia S Chao. Learning deep transformer models for machine translation. arXiv preprint arXiv:1906.01787, 2019.

[52] Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. arXiv preprint arXiv:2006.04768, 2020.

[53] Yongqin Wang, Edward Suh, Wenjie Xiong, Brian Knott, Benjamin Lefaudeux, Murali Annavaram, and Hsien-Hsin Lee. Characterizing and improving mpc-based private inference for transformer-based models. In NeurIPS 2021 Workshop Privacy in Machine Learning, 2021.

[54] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. On layer normalization in the transformer architecture. In International Conference on Machine Learning, pages 10524–10533. PMLR, 2020.

[55] Yunyang Xiong, Zhanpeng Zeng, Rudrasis Chakraborty, Mingxing Tan, Glenn Fung, Yin Li, and Vikas Singh. Nyströmformer: A nyström-based algorithm for approximating self-attention. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 35, pages 14138–14148, 2021.

[56] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks. In International conference on machine learning, pages 7354–7363. PMLR, 2019.

A. Choice of Multiplication Scheme

MPC for neural networks involves many matrix multiplications, and the choice of multiplication scheme in MPC significantly impacts the overall efficiency. In MPC implementations for neural networks based on secret sharing, Araki et al.'s method [3] or Beaver triples [5] of mainly used.

The method of Araki et al. uses replicated secret sharing among three parties. This method requires 2 bits of communication per multiplication at $N = 2$ when information-theoretic security is necessary. When computational security is sufficient, it requires only 1 bit of communication per multiplication, which is extremely efficient, while it requires sharing a seed for pseudo random number generation between the two participants at the time of initialization. ABY3 [36], FALCON [31], and others are MPC implementations for the neural network that follows this scheme. Privformer also relies on this scheme.

Beaver triples (or multiplication triples) are triples (a, b, c) such that $a \cdot b = c$ holds. For MPCs employing linear secret sharing, a single multiplication can be realized at the cost of two openings and a local computation by consuming a single Beaver triple. Beaver triples can be generated independently of the input. For this characteristic, the protocol execution can be evaluated with two phases: the offline phase for generating Beaver triples and the online phase for protocol execution. In other words, Beaver triples can keep the online phase low-cost in exchange for the high-cost offline phase. SPDZ [14], [15], [25] is an MPC framework that employs Beaver triples for multiplication; SecureML [37] is an MPC implementation for machine learning, including neural networks, that employs Beaver triples.

Although it is difficult to compare the two schemes in a strict sense because of differences in the security model and applicable situations, for example, [2] shows that the method using Beaver triples is more efficient in terms of online performance, while the method using replicated secret sharing is more efficient in terms of the entire (online and offline) processing cost including pre-computation. Since the Transformer requires a lot of multiplications, the multiplication needs to be as fast as possible. For this reason, our design choice was replicated secret sharing.

B. Embedding and Positional Encoding

The Transformer takes as input a sequence of words (e.g., a sentence of a natural language, source code, DNA sequence). Let $T = \{t_1, ..., t_S\}$ be an input word sequence with sequence length S where $t_i \in \mathcal{T}$ is a symbol (e.g., a word in natural language, function name in a programming language, genetic code, etc.). Each word t_i is converted into a word embedding vector $\boldsymbol{x}_i \in \mathbb{R}^{d_m}$ with embedding dimension d_m by a process called embedding, resulting in a word vector sequence $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_S\} \in \mathbb{R}^{S \times d_m}$. Next, a positional encoding vector $\boldsymbol{v}_i \in \mathbb{R}^{d_m}$ representing the position of each word in the sequence is appended to each word embedding vector in order to reflect its position in the embedding vector as $\boldsymbol{x}'_i = \boldsymbol{x}_i + \boldsymbol{v}_i$. Finally, the embedding vectors are trasformed into matrix $X' = (\boldsymbol{x}'_1, ..., \boldsymbol{x}'_S) \in \mathbb{R}^{S \times d_m}$, which is given to the Transformer as input.

C. Computing the Transformer with Known MPC Building Blocks

This section discusses the implementation of MPC protocols for the Transfomer with existing building blocks.

Embedding & Positional Encoding. The embedding is often processed by publicly shared pre-trained embedding function [4], [38], [40], [49]. Also, positional encoding vectors depend only on the word position and can be computed analytically without using private inputs (see, [49], Sec 3.5). Thus, the model owner has no incentive to keep the parameters private. Considering this, we suppose that query users transform input sequence T into word embedding matrix X' locally at the query user side, and MPC takes shares of the word embedding matrix as input in our construction. For this reason, MPC of embedding & positional encoding is not discussed in this study.

Feed-Forward Network. The feed-forward network contains linear computation for fully-connected (FC) layers and element-wise ReLU to the output of the FC layer. These can be computed immediately by using MPC for matrix multiplication and ReLU, and its communication complexity is in $O(S)$ with ignoring factor d_m.

Linear & Softmax. Similar to Embedding & Positional Encoding, computation in this layer is processed by a commonly known pre-trained embedding function. For this reason, we suppose that query users transform the embedding matrix into the output sequence locally, and computation in this layer is not subject to processing by MPC.

D. Subprotocols for Privformer

D.1. Multi-Head Attention

Alg. 6 is a protocol for computing multi-head ReLU attention with three parties, where attention in Eq. 5 is replaced by ReLU attention.

Algorithm 6 Multi-Head ReLU Attention Π_{MHA} :

```
Require: Each $P_j$ holds $\langle Q \rangle_L, \langle K \rangle_L, \langle V \rangle_L \in \mathbb{Z}_L^{S \times d_m}$, $\langle \Omega \rangle_L \in \mathbb{Z}_L^{d \times r}$, $\langle W_i^Q \rangle_L, \langle W_i^K \rangle_L, \langle W_i^V \rangle_L \in \mathbb{Z}_L^{d_m \times d}$, $\langle W^O \rangle_L \in \mathbb{Z}_L^{d_m \times d_m}$ for $i \in \{1, ..., H\}$.
Ensure: Each $P_j$ gets $\langle \mathrm{MHA}(Q, K, V, \Omega) \rangle_L \in \mathbb{Z}_L^{S \times d_m}$
for $i$ = $\{1, ..., H\}$ do
    $\langle QW_i^Q \rangle_L \leftarrow \Pi_{\mathrm{MatMul}}(\langle Q \rangle_L, \langle W_i^Q \rangle_L)$
    $\langle KW_i^K \rangle_L \leftarrow \Pi_{\mathrm{MatMul}}(\langle K \rangle_L, \langle W_i^K \rangle_L)$
    $\langle VW_i^V \rangle_L \leftarrow \Pi_{\mathrm{MatMul}}(\langle V \rangle_L, \langle W_i^V \rangle_L)$
    $\langle \mathrm{head}_i \rangle_L \leftarrow \Pi_{\mathrm{ReLUAtt}}(\langle QW_i^Q \rangle_L, \langle KW_i^K \rangle_L, \langle VW_i^V \rangle_L)$
end for
Each $P_j$ concatenates $\langle \mathrm{head}_1 \rangle_L, ..., \langle \mathrm{head}_H \rangle_L$ to make $\langle QK^TV \rangle_L \in \mathbb{Z}_L^{S \times d_m}$
$\langle \mathrm{MHA}(Q, K, V, \Omega) \rangle_L \leftarrow \Pi_{\mathrm{MatMul}}(\langle QK^TV \rangle_L, \langle W^O \rangle_L)$
```

D.2. Feed-Forward Network

The feed-forward network can be processed by computing Eq. 11 using Π_{mutmal} and Π_{ReLU}. Alg. 7 is a three-party protocol to compute the feed-forward network of Eq. 11. P_0, P_1, P_2 take a share of $X \in \mathbb{Z}_L^{S \times d_m}$ as input and output $\langle \mathrm{FFN}(X) \rangle_L \in \mathbb{Z}_L^{S \times d_m}$. Let $B^{F1} = (\boldsymbol{b}_1^{F1}, \ldots, \boldsymbol{b}_S^{F1})^T$ and $B^{F2} = (\boldsymbol{b}_1^{F2}, \ldots, \boldsymbol{b}_S^{F2})^T$. Then, the application of the feed-forward network to each row in X is realized by

$$\mathrm{FFN}(X) = (\mathrm{ReLU}(XW^{F1} + B^{F1}))W^{F1} + B^{F1}. \quad (17)$$

Since the computation of the feed-forward network consists of linear operations and ReLU only, its realization on MPC is straightforward.

D.3. Encoder

Alg. 8 is a three-party protocol for the encoder with self-attention (i.e., $Q = X, K = X, V = X$). P_0, P_1, and P_2 take a share of $X \in \mathbb{Z}_L^{S_{in} \times d_m}$ as input and output $\langle \mathrm{encoder}(X) \rangle_L \in \mathbb{Z}_L^{S_{in} \times d_m}$.

TABLE 3. Computation time (s) and communication bandwidth (MB) of the encoder and decoder when changing the batch size.

Batch Size	Encoder			Decoder		
	Time (s)		Comm	Time (s)		Comm
	Runtime	Amortized	(MB)	Runtime	Amortized	(MB)
1	14.141	14.141	424.668	17.768	17.768	717.894
2	27.082	13.541	849.336	33.6901	16.845	1435.79
4	53.801	13.452	1698.67	59.682	14.921	2871.57
8	107.758	13.469	3397.34	134.69	16.836	5743.12

Algorithm 7 Feed Forward Network Π_{FFN}

```
Require: Each $P_j$ holds $\langle X\rangle_L \in \mathbb{Z}_L^{S\times d_m}$, $\langle W^{F1}\rangle_L \in \mathbb{Z}_L^{d_m\times d_{\text{ff}}}$, $\langle B^{F1}\rangle_L \in \mathbb{Z}_L^{S\times d_{\text{ff}}}$, $\langle W^{F2}\rangle_L \in \mathbb{Z}_L^{d_{\text{ff}}\times d_m}$, $\langle B^{F2}\rangle_L \in \mathbb{Z}_L^{S\times d_m}$.
Ensure: Each $P_j$ gets $\langle \text{FFN}(X)\rangle_L$
$\langle XW^{F1}\rangle_L \leftarrow \Pi_{\text{MatMul}}(\langle X\rangle_L, \langle W^{F1}\rangle_L)$
$\langle X'\rangle_L \leftarrow \Pi_{\text{ReLU}}(\langle XW^{F1}\rangle_L + \langle B^{F1}\rangle_L)$
$\langle X'W^{F2}\rangle_L \leftarrow \Pi_{\text{MatMul}}(\langle X'\rangle_L, \langle W^{F2}\rangle_L)$
$\langle \text{FFN}(X)\rangle_L \leftarrow \langle X'W^{F2}\rangle_L + \langle B^{F2}\rangle_L$.
```

Algorithm 8 Encoder Π_{Encoder}

```
Require: See Table 4, line 9, columns 2 and 6 .
Ensure: Each $P_j$ gets $\langle \text{Encoder}(Q,K,V)\rangle_L \in \mathbb{Z}_L^{S_{in}\times d_m}$
for i = 1, ... , N do
    $\langle \text{MHA}_i\rangle_L \leftarrow \Pi_{\text{MHA}}(\langle Q\rangle_L, \langle K\rangle_L, \langle V\rangle_L)$
    $\langle \text{LN1}_i\rangle_L \leftarrow \Pi_{\text{LN}}(\langle \text{MHA}_i\rangle_L)$
    $\langle \text{FFN}_i\rangle_L \leftarrow \Pi_{\text{FFN}}(\langle \text{LN1}_i\rangle_L)$
    $\langle \text{LN2}_i\rangle_L \leftarrow \Pi_{\text{LN}}(\langle \text{FFN}_i\rangle_L)$
    $\langle Q\rangle_L, \langle K\rangle_L, \langle V\rangle_L \leftarrow \langle \text{LN2}_i\rangle_L$
end for
$\langle \text{Encoder}(Q,K,V)\rangle_L \leftarrow \langle \text{LN2}_N\rangle$
```

D.4. Decoder

Alg.9 is a three-party protocol for the decoder with self-attention. P_0, P_1, and P_2 take shares of $X \in \mathbb{Z}_L^{S_{out}\times d_m}$ and shares of encoder output $E \in \mathbb{Z}_L^{S_{in}\times d_m}$ as input, and output $\langle \text{Decoder}(X)\rangle_L \in \mathbb{Z}_L^{S_{out}\times d_m}$.

Algorithm 9 Decoder $\Pi_{\text{Decoder}}(P_0, P_1, P_2)$:

```
Require: See Table 4, line 10, column 2,5 and 6.
Ensure: Each $P_j$ gets $\langle \text{Decoder}(Q,K,V)\rangle_L \in \mathbb{Z}_L^{S_{out}\times d_m}$
for i = 1, ... , N do
    $\langle \text{MMHA}_i\rangle_L \leftarrow \Pi_{\text{MHA}}(\langle Q\rangle_L, \langle K\rangle_L, \langle V\rangle_L)$
    $\langle \text{LN1}_i\rangle_L \leftarrow \Pi_{\text{LN}}(\langle \text{MaskedMHA}_i\rangle_L)$
    $\langle \text{MHA}_i\rangle_L \leftarrow \Pi_{\text{MHA}}(\langle \text{LN1}_i\rangle_L, \langle K_E\rangle_L, \langle V_E\rangle_L)$
    $\langle \text{LN2}_i\rangle_L \leftarrow \Pi_{\text{LN}}(\langle \text{MHA}_i\rangle_L)$
    $\langle \text{FFN}_i\rangle_L \leftarrow \Pi_{\text{FFN}}(\langle \text{LN2}_i\rangle_L)$
    $\langle \text{LN3}_i\rangle_L \leftarrow \Pi_{\text{LN}}(\langle \text{FFN}_i\rangle_L)$
    $\langle Q\rangle_L, \langle K\rangle_L, \langle V\rangle_L \leftarrow \langle \text{LN3}_i\rangle_L$ (ToDO: correct?)
end for
$\langle \text{Decoder}(Q,K,V)\rangle_L \leftarrow \langle \text{LN3}_N\rangle$
```

D.5. Entire computation of Privformer

.

The entire computation of Privformer is summarized in the following list.

1) The model owner sends shares of the model parameters to the computation servers
2) The query user transforms his/her input token sentence into a word vector sequence $X_E \in \mathbb{R}^{S_{in}\times d_m}$ by Embedding + Positional Encoding
3) The query user transforms X_E into shares $\langle X_E\rangle_L$ and distributes them to the computation servers P_0, P_1, P_2
4) The computation servers run Π_{encoder} with taking $\langle X_E\rangle_L$ and shares of model parameters as input, and outputs shares of the embedding matrix $\langle E\rangle_L$ by the encoder.
5) The query user transforms the output token sequence provided by the decoder so far into word vector sequence $X_D \in \mathbb{R}^{S_{out}\times d_m}$ by Embedding + Positional Encoding
6) The query user transforms X_D into shares $\langle X_D\rangle_L$ and distributes them to the computation servers P_0, P_1, P_2
7) The computation servers run Π_{decoder} with taking $\langle X_D\rangle_L, \langle E\rangle_L$, and shares of model parameters as input, and obtains the output $\langle D\rangle_L$ of the decoder.
8) The computation servers send $\langle D\rangle_L$ to the query user.
9) The query user reconstructs $\langle D\rangle_L$, obtains an output token by Linear+Softmax, and adds the token to the end of the output sequence obtained so far.
10) If the termination signal is output by the decoder, output tokens have been decoded so far, and decoding is terminated. Otherwise, jump to step 5

E. Effect of Batch Processing

We varied the batch size and evaluated the change in the computation time of Transfer per input sequence. The execution time of the encoder and decoder were measured in the LAN environment by changing the batch size as $1, 2, 4$, and 8. The parameter setting is the same as Section 9.4. The results are shown in Table 3. As the batch size increases, the computation time and communication bandwidth of the encoder and decoder increase linearly, and the execution time per input shows little change. Since this experiment was performed on CPUs and no parallelization was not introduced, no improvement in the per-sample execution time is observed. However, the amortized computation time per sample can be improved by using a parallel implementation using GPUs or other devices. Improvement of computation time by parallelization is also a promising direction for future research.

F. Summary of Protocols

Table 4 is a summary of the inputs and outputs of every entity and invoked subprotocols for each protocol.

TABLE 4. Summary of Protocols

Protocol (Type)	Input					Output	Invoked Protocol
	Public	Query User	Model Owner	Comp Servers (from prev. protocol)	Comp Servers (from query user/model owner inputs)	Comp Servers	
Π_{ReLUAtt} (Transformer primitive)		Q, K, V	Ω		$\langle Q \rangle_L, \langle K \rangle_L, \langle V \rangle_L, \langle \Omega \rangle_L$	$\langle \mathrm{ReLUAtt}(Q, K, V) \rangle_L$	linear op., $\Pi_{\mathrm{MutMul}}, \Pi_{\mathrm{ReLU}}$
$\Pi_{\mathrm{MReLUAtt}_{QK}}$ (Transformer primitive)		Q, K, V	Ω		$\langle Q \rangle_L, \langle K \rangle_L, \langle V \rangle_L, \langle \Omega \rangle_L$	$\langle \mathrm{MReLUAtt}(Q, K, V) \rangle_L$	linear op., $\Pi_{\mathrm{MutMul}}, \Pi_{\mathrm{ReLU}}$
$\Pi_{\mathrm{MReLUAtt}_{VK}}$ (Transformer primitive)		Q, K, V	Ω		$\langle Q \rangle_L, \langle K \rangle_L, \langle V \rangle_L, \langle \Omega \rangle_L$	$\langle \mathrm{MReLUAtt}(Q, K, V) \rangle_L$	linear op., $\Pi_{\mathrm{MutMul}}, \Pi_{\mathrm{ReLU}}$
$\Pi_{\mathrm{SqrtInverse}}$ (General primitive)	$1, 2^{\mathrm{FP}/2} \cdot 2^{\mathrm{FP}}$				$\langle 1 \rangle_L, \langle 2^{\frac{\mathrm{FP}}{2}} \cdot 2^{\mathrm{FP}} \rangle_L$	$\langle \frac{1}{\sqrt{b}} 2^{\mathrm{FP}} \rangle_L$	$\Pi_{\mathrm{DReLU}}, \Pi_{\mathrm{Select}}, \Pi_{\mathrm{Mult}}$
Π_{MHA} (Transformer layer)		Q, K, V	Ω W_i^Q, W_i^K, W_i^V		$\langle Q \rangle_L, \langle K \rangle_L, \langle V \rangle_L, \langle \Omega \rangle_L$ $\langle W_i^Q \rangle_L, \langle W_i^K \rangle_L, \langle W_i^V \rangle_L$	$\langle \mathrm{MHA}(Q, K, V) \rangle_L$	linear op., $\Pi_{\mathrm{ReLUAtt}}, \Pi_{\mathrm{MReLUAtt}}$
Π_{LN} (Transformer layer)	ϵ $1, 2^{\mathrm{FP}/2} \cdot 2^{\mathrm{FP}}$		β, γ	$\langle X \rangle_L$	$\langle \beta \rangle_L, \langle \gamma \rangle_L, \langle \epsilon \rangle_L$ $\langle 1 \rangle_L, \langle 2^{\frac{\mathrm{FP}}{2}} \cdot 2^{\mathrm{FP}} \rangle_L$	$\langle \mathrm{LN}_{\beta,\gamma}(X) \rangle_L$	linear op., $\Pi_{\mathrm{Mult}}, \Pi_{\mathrm{SqrtInv}}$
Π_{FFN} (Transformer layer)			$W^{F1}, B^{F1}, W^{F2}, B^{F2}$	$\langle X \rangle_L$	$\langle W_i^{F1} \rangle_L, \langle B_i^{F1} \rangle_L, \langle W_i^{F2} \rangle_L, \langle B_i^{F2} \rangle_L$	$\langle \mathrm{FFN}(X) \rangle_L$	linear op., $\Pi_{\mathrm{MatMul}}, \Pi_{\mathrm{ReLU}}$
Π_{Encoder} (Transormer)	$1, 2^{\mathrm{FP}/2} \cdot 2^{\mathrm{FP}}$	Q, K, V	$\Omega, \beta_i, \gamma_i$ W_i^Q, W_i^K, W_i^V $W_i^{F1}, B_i^{F1}, W_i^{F2}, B_i^{F2}$		$\langle Q \rangle_L, \langle K \rangle_L, \langle V \rangle_L, \langle \beta_i \rangle_L, \langle \gamma_i \rangle_L$ $\langle W_i^Q \rangle_L, \langle W_i^K \rangle_L, \langle W_i^V \rangle_L$ $\langle W_i^{F1} \rangle_L, \langle B_i^{F1} \rangle_L, \langle W_i^{F2} \rangle_L, \langle B_i^{F2} \rangle_L$ $\langle 1 \rangle_L, \langle 2^{\frac{\mathrm{FP}}{2}} \cdot 2^{\mathrm{FP}} \rangle_L$	$\langle \mathrm{Encoder}(Q, K, V) \rangle_L$	$\Pi_{\mathrm{MHA}}, \Pi_{\mathrm{LN}}, \Pi_{\mathrm{FFN}}$
Π_{Decoder} (Transformer)	$1, 2^{\mathrm{FP}/2} \cdot 2^{\mathrm{FP}}$	Q, K, V	$\Omega, \beta_i, \gamma_i$ W_i^Q, W_i^K, W_i^V $W_i^{F1}, B_i^{F1}, W_i^{F2}, B_i^{F2}$	$\langle \mathrm{Encoder}(Q, K, V) \rangle_L$	$\langle Q \rangle_L, \langle K \rangle_L, \langle V \rangle_L, \langle \beta_i \rangle_L, \langle \gamma_i \rangle_L$ $\langle W_i^Q \rangle_L, \langle W_i^K \rangle_L, \langle W_i^V \rangle_L$ $\langle W_i^{F1} \rangle_L, \langle B_i^{F1} \rangle_L, \langle W_i^{F2} \rangle_L, \langle B_i^{F2} \rangle_L$ $\langle 1 \rangle_L, \langle 2^{\frac{\mathrm{FP}}{2}} \cdot 2^{\mathrm{FP}} \rangle_L$	$\langle \mathrm{Decoder}(Q, K, V) \rangle_L$	$\Pi_{\mathrm{MHA}}, \Pi_{\mathrm{MMHA}}$, $\Pi_{\mathrm{LN}}, \Pi_{\mathrm{FFN}}$