PrivTF: Efficient and Privacy-Preserving Transformer Inference with Optimized Protocols for Secure NLP

Ziwei Peng, Kaiping Xue, *Senior Member, IEEE*, Bin Zhu, *Graduate Student Member, IEEE*, Yaxuan Huang, Jian Li, *Senior Member, IEEE*, Meng Hao, Tianwei Zhang, *Member, IEEE*

***Abstract*—As many natural language processing services employ language models like Transformer in the cloud, privacy concerns arise for both users and model owners. Secure inference is proposed in the literature to perform the computing service securely. While prior schemes have studied convolutional neural networks and analogous models, they do not easily apply to Transformer because it requires more efficient protocols of different layers with large-sized weight matrices and complex non-linear functions on high-dimensional vectors. Therefore, we propose a privacy-preserving scheme PrivTF for secure transformer inference. PrivTF consists of protocols for Transformer-unique layers (encoder/decoder layer and their attention sub-layer) with our specialized design. Specifically, we present protocols of base sub-layers to address the heavy performance overhead: a new protocol of embedding layer, protocols of matrix multiplication and softmax that make use of mixed bitwidths, and protocols of GeLU functions and normalization. Analysis and experiments demonstrate that our protocols outperform previous secure inference works and maintain accuracy on practical inference tasks.**

***Index Terms*—Privacy preservation, Transformer, Secure inference, Secure Computation.**

I. Introduction

Transformer [1] is a popular neural network architecture in natural language processing (NLP) and derives a family of models. Unlike convolutional neural networks (CNNs) or recurrent neural networks (RNNs), Transformer models use the attention mechanism to process parallely across sequences. They are also larger and computationally powerful, with a multitude of parameters. Due to their special characteristics, various Transformer models [2]–[4] outperform previous neural networks and become a widely used option for cloud-based language services.

In many cloud applications of Transformer models, the issue of privacy concerns arises for both the service provider and the client. This is the case of a typical cloud computing scenario, where the serving model and input data are held seperately by two parties. On the one hand, the client wants to keep the data and the computation private since it implies sensitive identity information or intellectual property. Otherwise, plaintext computation may reveal the sensitive data [5]. For example, The scheme in [5] once recovered the input by attacks on the embedded vectors of many pre-trained language models. On the other hand, proprietary models of companies are private and not shareable with others. Model owners typically take much time to get a trained model and their models are confidential as commercially profitable services. Owners are also concerned that models reveal private information learned from the training dataset. Thus privacy preservation for language models in cloud services is needed.

Research on privacy-preserving machine learning proposed secure inference to conduct private computation in cloud client-server service. Secure inference ensures that engaging in the secure two-party computation (2PC) the server cannot learn the private input and the client cannot learn the model. The main techniques of secure inference are cryptographic primitives: homomorphic encryption (HE) [6]–[9] and secure computation techniques such as secret sharing, oblivious transfer (OT) [10]–[16], garbled circuits (GC) [17]–[19].

Transformer has complex layers such as the embedding layer and non-linear functions, which poses new challenges in secure inference. Firstly, the embedding layer on the plaintext is in essence a lookup over a vocabulary table, which cannot be evaluated easily on ciphertexts. The vocabulary table, as a large matrix, is costly to manipulate with ciphertext operations. Since the vocabulary is an integral and private component of the model, the lookup operation is not merely a simple retrieval but rather a process that must yield a preserved or secret-shared output to ensure privacy preservation. A straightforward solution is to obliviously do multiplication over the matrix via homomorphic encryption [20], [21] or secret sharings [18]. Considering the embedding vocabulary is tens of thousands long, traversing the entire matrix for each word entry by cryptographic computation costs large amount of unnecessary overhead. Secondly, the non-linear functions, including softmax of the attention sub-layer, GeLU of the feed-forward sub-layer, and layer normalization, also present challenges of achieving computational efficiency and accuracy. Iron [20] follows the approach of using lookup-table (LUT) proposed by SIRNN [14] to design non-linear function protocols, while BOLT [22] and BumbleBee [21] uses integer approximation or high-degree polynomial approximation. The lookup-table-based protocols are particularly advantageous than approximation techniques as it is able to universally

Z. Peng, K. Xue, B. Zhu, Y. Huang, J. Li are with the School of Cyber Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, China.

T. Zhang is with the School of Computer Science and Engineering, Nanyang Technological University, 639798, Singapore.

M. Hao is with the School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, Sichuan 610054, China, and also with the School of Computer Science and Engineering, Nanyang Technological University, 639798, Singapore.

support complex math functions with great accuracy and flexibility. However, the schemes of Iron still incur significant overhead and low computational efficiency. With these challenges in mind, we aim to study further on efficient secure inference of Transformer.

In this work, we propose a framework to achieve computationally efficient two-party secure inference of Transformer, PrivTF. Generally, PrivTF consists of several specialized and efficient protocols for the complicated Transformer layers. Firstly, we provide a novel protocol PrivEmbedding of the embedding layer. This method offers a new insight for securely computing the embedding layer. Unlike previous approaches that rely on $O(w)$ homomorphic multiplications ([20], [21]), where w denotes the vocabulary size, our protocol avoids such costly operations. Instead, it leverages the pre-sharing of a matrix that is masked using pseudo-random functions (PRFs). The target vector is then retrieved through secure two-party computation techniques. The pre-shared masked matrix can be reused multiple times, significantly reducing the latency of online services. Once the real-time embedding process begins, our scheme requires only local XOR computation plus a two-party computation of PRF circuits. This approach consumes less running time than the direct method of multiplying with the overall matrix.

Secondly, for the attention sub-layer we devise protocols for large matrix multiplication and softmax using the idea of mix-bitwidth. We identify that in the attention mechanism the intermediate results of exponentiation and softmax normalization are known to scale to small ranges. Therefore, a small bitwidth suffices to avoid numeric overflow, while also alleviating the computation and communication overhead. So we give efficient protocols MatMul, Softmax to make use of variable bitwidths and improve the performance.

Thirdly, we develop protocols of GeLU and layer normalization in feed-forward sub-layers and encoder/decoder layers. For GeLU, we exploit its sigmoid expression and propose an efficient protocol, GeLU. By leveraging the symmetry of the sigmoid function, we reduce the number of secure computation operators compared with Iron [20]. Our method is not only designed to compute GeLU accurately and efficiently, but also provides a more efficient approach to computing the sigmoid function, which is applicable to other neural networks. For layer normalization, we adopt an alternative pre-norm pattern to devise our layer protocols. By adopting the pre-norm pattern we fuse the normalization protocol PreLN and reduce much computation, thus improving performance. We further construct protocols of Transformer's unique layers (attention sub-layer and encoder/decoder layer). The contributions of our work are summarized as follows:

- We design a new protocol for the embedding layer, utilizing pre-shared masked matrices and secure two-party computation. This method has cryptographic operation complexity independent of the vocabulary size n, reducing online computation latency by two times.
- We propose a privacy-preserving inference framework, called PrivTF, for Transformer models. PrivTF features specialized protocols, including efficient PrivEmbedding, PrivAttention with mixed-bitwidth optimization, and PrivFeedForward with lookup-table-based GeLU. These protocols are computationally efficient and achieve high accuracy with less than one unit of error, without relying on approximation.
- We implement PrivTF on practical models and classification tasks. PrivTF achieves higher efficiency and accuracy of non-linear functions than existing schemes. Analysis and experiments results verify our scheme's security, accuracy, efficiency and practicality.

The remainder of the paper is organized as follows. Section II introduces related work. Section III introduces necessary preliminaries of Transformer and cryptographic techniques. Section IV presents our problem establishment. Section V first gives an overview of Transformer secure inference and then describes the proposed scheme PrivTF. Then we present security analysis in Section VI and experimental evaluation in SectionVII. Finally, we conclude our work in Section VIII.

II. Related Work

A. Secure Neural Network Inference

This section reviews the literature of secure neural network inference, including Transformer models. Early works on secure inference can be classified according to their core cryptographic techniques. One technical approach employs homomorphic encryption (HE). CryptoNets [6] proposes HE-based protocols for convolutional neural networks (CNNs) in client-server settings, implementing linear layers through homomorphic operations and approximating activations with polynomials. Subsequent work by Gazelle [7] enhances efficiency using packed additive HE for CNN linear layers, while Falcon [8] further improves secure convolution through Fourier transform techniques. Alternatively, other approaches leverage secure multi-party computation (MPC) primitives such as secret sharing, oblivious transfer (OT), and garbled circuits (GC). SecureML [10] and Quotient [12] introduce multi-server frameworks with non-colluding security assumptions. MiniONN [11] proposes an offline-online approach in the client-server setting to accelerate online weight multiplication. XONN [17] uses GC to securely evaluate neural networks. General-purpose frameworks like ABY [18] and ABY^3 [19] combine secret sharing with GC for secure multi-server machine learning. However, these early systems incur substantial computational overhead, restricting their deployments to small-scale models (e.g., 5-layer networks on MNIST).

Works such as [13], [15], [16] focused on coordinating diverse cryptographic primitives across network layers and developing specialized protocols for secure deep neural network inference . Rathee *et al.* [13] presented a hybrid scheme CrypTFlow2, a hybrid framework featuring optimized HE-based protocols and OT-based protocols that enable secure inference for deep CNNs like ResNet50. Huang *et al.* [15] further advanced this direction by proposing Cheetah, which converts HE-based protocols into the OT domain through novel polynomial encoding. Cheetah also avoids the SIMD-style design with costly homomorphic rotations, achieving faster secure inference. Other works proposed general frameworks that integrate secure MPC primitives as application interfaces, including tensor operations and neural network compilers (e.g., [23]–[25] in the multi-server setting, [9], [26], [27] in the client-server setting). Parallel efforts [28]–[30] targeted outsourced inference in distributed edge networks. They proposed new protocols for convolution and activation functions, where cloud or edge parties compute on secret sharings using pre-generated multiplication triplets.

In addition to convolutional neural networks, a few works focused on secure inference of language models. For LSTM language models, SecureNLP [31] uses multiplicative secret sharing to evaluate activation functions such as sigmoid and tanh within a multi-party security model. For recurrent neural networks, SIRNN [14] proposes the idea of secure computation on variable bit-lengths and lookup-tables. It gives efficient protocols of precise math functionalities based on OT and secret sharing. We leverage the math functionalities and further propose specific protocols for Transformer.

B. Secure Transformer inference

Transformer models are more complicated, leading to many recent efforts on secure protocols for Transformer. In the client-server setting, some works [20]–[22] followed the approach of mixed frameworks, i.e., using homomorphic encryption for linear computation and other MPC primitives for non-linear functions. MPCformer [32] and PrivFormer [33] adopt a different three-server model and use secret sharing-based multi-party computation. TABLE I compares the system models. Recent advances adopt a two-party pre-processing paradigm. Works such as Orca [34] leverage function secret sharing [35] to design efficient protocols for complex neural network operations. The security model assumes a dealer for pre-processing [36], [37], so their approach of utilizing pre-computed key generation demonstrates improvements in online phase efficiency, offering a potential direction for practical secure inference systems. Zeng *et al.* [38] focused on distributed secure inference of Transformer with multiple parties. Zhang *et al.* [39] used fully homomorphic encryption with approximation to achieve non-interactive secure computation of Transformer models.

Designing protocols of secure Transformer inference presents challenges in two areas: linear computation (matrix multiplication in the embedding and attention layers) and non-linear functions. We review the existing works in relation to these two aspects. Specifically, for the domain of linear computation, many studies explored HE-based matrix multiplication protocols. Iron [20] observes that matrix-matrix multiplication, rather than matrix-vector multiplication, is the dominant operation in transformer inference, and Cheetah's encoding of homomorphic encrypted vectors [15] is not applicable. Iron replaces this encoding by a new matrix encoding method, saving up to one order of magnitude in terms of computation and communication overhead. BOLT [22] and BumbleBee [21] further optimize HE-based protocols of matrix multiplication, which is surveyed by He *et al.* [40]. However, they introduce additional costly homomorphic rotation and require more advanced hardware, such as 64 vCPUs and 256GB memory.

For non-linear functions, the methods include using approximation or scientific computation. Approximation methods introduce different expressions such as splines (piecewise polynomials) and series expansion. Scientific computation follows provably precise algorithms of iteration evaluation or table-lookups. THE-X [41] proposes approximations in training to integrate with fully homomorphic encryption, but converting the architecture to achieve MPC requires extra training and lacks flexibility of deployment. MPCformer [32] and PrivFormer [33] propose approximations with MPC-friendly operators to speed up the inference efficiency. MPCformer [32] uses polynomial approximations for softmax and GeLU. PrivFormer [33] uses Chebychev polynomials for exponentiation and Newton iteration for normalization. Iron [20] follows SIRNN and uses lookup-table-based exponentiation to provide protocols for the softmax and GeLU functions. BOLT [22] uses a quantization-based algorithm for softmax and a spline approximation for GeLU, while BumbleBee [21] proposes two spline approximations for exponentiation-based softmax and GeLU.

TABLE I and TABLE II summarize the related works and explain the differences of our approach from existing studies. As discussed in Section I, few prior works focused on the embedding layer. Schemes using homomorphic encryption are inefficient due to the high computational complexity associated with large-sized vocabularies. Instead, our scheme provides a new embedding layer protocol using the insight of pre-sharing. Additionally, for the attention layer involving intensive computation, we leverage secret sharings with variable bitwidths of to design optimized protocols, improving the computation efficiency than the schemes using uniform bitwidths. For non-linear functions of the attention and feed-forward layer, approximation-based methods have accuracy deviations and limited usability [42], thus we use lookup-table-based (LUT-based) computation to design approximation-free protocols for softmax and sigmoid-based GeLU.

Table I. Scheme Overview of Related Works.

	System Model	Embedding Layer	Attention Layer
THE-X [41]	Client-Server	None	HE-based
Iron [20]	Client-Server	Homomorphic Multiplication	Mix techniques (HE, SS)
MPCFormer [32]	Three-party	None	Secret Sharing-based
Privformer [33]	Three-party	None	Optimizing mask attention
BOLT [22]	Client-Server	None	Mix techniques (HE, SS)
BumbleBee [21]	Client-Server	Homomorphic Multiplication	Mix techniques (HE, SS)
Ours	Client-Server	New Method of Pre-sharing	Mix techniques with variable bitwidths

Table II. Non-linear Function Schemes of Related Works.

	Non-linear functions	Proposed Protocols
THE-X [41]	Approximation	Approximation in Training
Iron [20]	LUT-based Computation	Tanh-based GeLU
MPCFormer [32]	Approximation	Polynomial approximations
Privformer [33]	Approximation	Polynomial for softmax, ReLU for GeLU
BOLT [22]	Approximation	Quantization for softmax, spline for GeLU
BumbleBee [21]	Approximation	Splines for softmax and GeLU
Ours	LUT-based Computation	LUT-based softmax, sigmoid-based GeLU

III. Preliminaries

A. Transformer Architecture

Transformer [1] follows the commonly used encoder-decoder structure and each encoder/decoder layer consists of two sub-layers: multi-head attention and feed-forward. Besides, there are embedding layer and layer normalization.

Embedding layer with a table $\mathbf{W}$ converts the input tokens to vectors of shape d_m. Apart from the semantic embedding, certain models has a positional embedding $\mathbf{B}$, which can be viewed as an addition. For the i-th token in a sequence with table index x, its embedding is as follows:

$$\mathsf{Embedding}(x) = \mathbf{W}[x] + \mathbf{B}_i.$$

Multi-head attention operates on three input matrices: $\mathbf{Q}$ and $\mathbf{K}$ of dimension d_k, and values $\mathbf{V}$ of dimension d_v (typically d_k and d_v equal to d_m). These inputs first undergo linear transformations: $\mathbf{Q} \leftarrow \mathbf{W}_Q\mathbf{Q}, \mathbf{K} \leftarrow \mathbf{W}_K\mathbf{K}, \mathbf{V} \leftarrow \mathbf{W}_V\mathbf{V}$, where $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V$ are model parameters. The projected inputs are then divided into h heads, with each head computing an independent scaled dot-product attention function:

$$\mathsf{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathsf{softmax}(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}})\mathbf{V}.$$

Finally outputs of each head are concatenated together as multi-head attention output.

The attention layer is followed by a feed-forward network with two dimension transformations and a GeLU activation:

$$\mathsf{Feed\text{-}Forward}(\mathbf{X}) = \mathsf{GeLU}(\mathbf{X}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2,$$

where $\mathbf{W}_1$ expands to d_f, and $\mathbf{W}_2$ projects to d_m.

The layer normalization (LN) in Transformer is to normalize all the dimensions in the same hidden layer. With input and output dimension d_m, the LN operation can be formulated as:

$$\mathsf{LN}(\mathbf{x}) = \frac{\mathbf{x} - \mu}{\sqrt{\sigma^2}} \odot \gamma + \beta.$$

The mean and variance are calculated over real-time input. γ and β are linear transform parameters.

B. Secret sharing

In arithmetic secret sharing, we split an l-bit value x into additive shares in the ring $\mathbb{Z}_{2^l}$. In our schemes, the shares $\langle x\rangle^{\mathcal{S}}$ and $\langle x\rangle^{\mathcal{C}}$ are distributed to the server and the client, with reconstruction via modulo addition: $x = \langle x\rangle^{\mathcal{S}} + \langle x\rangle^{\mathcal{C}} \bmod 2^l$. The bitwidth l is variable when performing non-linear computations on numbers within a limited range.

We follow prior works [14], [15], [20] to use fixed-point arithmetic in secret sharing. With a predetermined precision f, a real number $\tilde{x}$ is encoded as a fixed-point integer $x := \lfloor \tilde{x} \cdot 2^f \rfloor \bmod 2^l$, which is then shared as $\langle x\rangle$. After each multiplication, truncation of the low f-bit is needed to keep the arithmetic. Our implementation adopts the interactive truncation protocol from SIRNN [14], though we omit this step in protocol descriptions for clarity.

In addition to arithmetic sharing, we use boolean sharing $[\![x]\!]$ in the embedding layer and the attention sub-layer. A value x is shared as $[\![x]\!]^{\mathcal{S}}, [\![x]\!]^{\mathcal{C}}$, such that $[\![x]\!]^{\mathcal{S}} \oplus [\![x]\!]^{\mathcal{C}} = x$. This supports *xor* ($\oplus$), *and* ($\wedge$) operations. The functionality $\mathsf{B2A}$ can transform boolean sharing to arithmetic sharing [18].

C. Oblivious Transfer

Oblivious Transfer [43] is also an important 2PC primitive. In 1-out-of-2 OT, a sender inputs two messages (m_0, m_1) and a receiver inputs a choice bit $c \in \{0, 1\}$. At the end of the protocol, the receiver obtains m_c, while the sender learns nothing. Recent research on cryptography proposed new OT and OT extension protocols [44]. We use the correlated-OT variant in our framework as a black box: $\binom{1}{2}$-COT^l. It takes $x \in \mathbb{Z}_2$ from the receiver, takes $\mathrm{u} \in \mathbb{Z}_{2^l}$ from the sender and generates a correlation $\mathrm{w} := x \cdot \mathrm{u} + \mathrm{v} \in \mathbb{Z}_{2^l}$. The receiver learns w while the sender learns $\mathrm{v} \in \mathbb{Z}_{2^l}$.

D. Two-party Functionalities

The following protocols (proposed by [14], [18], [45]) serve as underlying operations in PrivTF's construction. We use them as sub-protocols that are replaced with calls to secure 2PC functionalities. The inputs may be a single number or a vector. For vectors, the functionalities perform an element-wise operation. In this paper, we use boldface $\mathbf{x}$ to denote vectors and capital $\mathbf{X}$ matrices. Table III summarizes additional key notations in this paper.

- Comparison: The functionality $\mathsf{CMP}(\langle x\rangle, 0)$ outputs a boolean indicator $b = \mathbf{I}\{x < 0\}$ shared in $\mathbb{Z}_2$.
- Multiplexer: Given $\langle x\rangle, [\![b]\!]$ as input, the functionality $\mathsf{MUX}(\langle x\rangle, [\![b]\!])$ outputs $\langle y\rangle$ such that $y = x$ if $b = 1$ and 0 otherwise. The multiplexer along with comparison can be used to build maximum function $\mathsf{Max}(\langle \mathbf{x}\rangle)$ where the output $\langle y\rangle$ is the share of the maximal element of $\mathbf{x}$.
- Exponentiation: Given $\langle u\rangle \in \mathbb{Z}_{2^l}, u < 0$, the functionality $\mathsf{Exp}^{l,l'}(\langle u\rangle)$ outputs $\langle v\rangle \in \mathbb{Z}_{2^{l'}}$ such that $v = e^u$ in l'-bit fixed-point arithmetic.
- Product: Given $\langle u\rangle \in \mathbb{Z}_{2^s}$, $\langle v\rangle \in \mathbb{Z}_{2^l}$, $\mathsf{Product}^{s,l}$ $(\langle u\rangle, \langle v\rangle)$ returns $\langle w\rangle \in \mathbb{Z}_{2^l}$ such that $w = uv$ (for vectors, it is a hadamard product $\mathbf{u} \odot \mathbf{v}$).
- Division: For a public denominator de, $\mathsf{Div}^{l,de}(\langle u\rangle)$ divides a secret-shared input $\langle u\rangle \in Z_{2^l}$ by de, producing $\langle v\rangle \in Z_{2^l}$; For a private denominator de, $\mathsf{Div}^{l,l'}(\langle u\rangle, \langle de\rangle)$ computes fixed-point arithmetic division, producing $\langle v\rangle \in Z_{2^{l'}}$ where $v = u/de$.
- Inverse Squre Root: Given $\langle u\rangle \in \mathbb{Z}_{2^l}$, $\mathsf{ISqrt}(\langle u\rangle)$ outputs $\langle w\rangle \in \mathbb{Z}_{2^l}$ such that w represents the fixed-point arithmetic result of u's inverse square root.
- Secure AES: The functionality sAES securely computes AES, where one party inputs a key k and the other inputs x. The outputs are boolean sharings $[\![s]\!]^{\mathcal{S}}, [\![s]\!]^{\mathcal{C}}$ such that $[\![s]\!]^{\mathcal{S}} \oplus [\![s]\!]^{\mathcal{C}} = \mathrm{AES}_k(x)$.

IV. Problem Statement

We consider a two-party setting: Server $\mathcal{S}$ and Client $\mathcal{C}$ jointly execute secure inference of a Transformer model. Fig. 1 depicts the system model. The server possesses a pre-trained language model while the client holds data samples to conduct language inference. They have checked a common vocabulary by negotiation and Client needs to input sequences of word indexes. For each stage of the online execution (embedding and several encoder/decoder layers, depending on the model

Table III. Key Notations.

Notations	Definitions
n	sequence length
d_m	model embedding dimension
d_q, d_k, d_v	the dimensions of the query, key, value
h, d_h	the number of heads, the dimension of each head
d_f	the dimension of the feed-forward sub-layer
$\langle\cdot\rangle, [\![\cdot]\!]$	arithmetic sharing and boolean sharing
l, s	bit-lengths
f	precision parameter for the fixed arithmetic
$\mathbf{x}, \mathbf{X}$	vector and matrix
$\mathbf{x}_i, \mathbf{x}_{ij}$	vector element and matrix element
$\odot$	hadamard product
$\mathcal{S}, \mathcal{C}$	the server and the client
$\langle\cdot\rangle^{\mathcal{S}}, \langle\cdot\rangle^{\mathcal{C}}, [\![\cdot]\!]^{\mathcal{S}}, [\![\cdot]\!]^{\mathcal{C}}$	private shares of the server and the client

architecture), secure inference protocols operate the input and output in privacy-preserving forms. After the protocol ends, the inference result is revealed to Client the client.

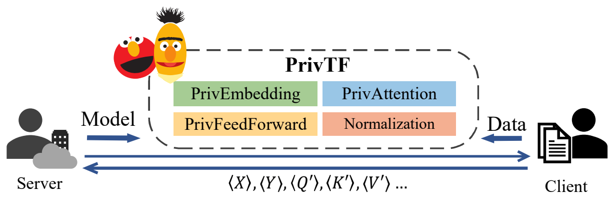

PrivTF
PrivEmbedding
PrivAttention
PrivFeedForward
Normalization
Model
Data
Server
Client
$\langle X\rangle, \langle Y\rangle, \langle Q'\rangle, \langle K'\rangle, \langle V'\rangle$ ...

Fig. 1. PrivTF System Model.

A. Threat Model

We follow the same honest-but-curious threat model as most previous works [13], [15]. This means that Server and Client perform the protocol as promised and at most one of them is corrupted by an honest-but-curious adversary. Assume that the adversary $\mathcal{A}$ that is computationally bounded in polynomial time follows the protocol but tries to infer information about honest parties. In the simulation-based security model [46], for a protocol Π, there are two interaction views coordinated by the environment $\mathcal{Z}$: a real interaction in the presence of adversary $\mathcal{A}$ and an ideal interaction with the trusted functionality $\mathcal{F}$. In the real interaction the adversary has a realistic view: $\mathsf{view}_{\mathcal{A}}^{\Pi}$, whereas in the ideal interaction he has a simulated view generated by a simulator $\mathsf{Sim}_{\mathcal{F}}$. The real view includes all intermediate information and protocol outputs owned by the corrupted party. The security holds if for any adversary there exists a polynomial-time simulator Sim such that the adversary cannot distinguish the two views: $\mathsf{view}_{\mathcal{A}}^{\Pi} \approx \mathsf{Sim}_{\mathcal{F}}$.

PrivTF does not hide the computation parameters as prior works [7], [15] do, including the model architecture (dimension of weight matrices and numbers of layers) and sequence length. Besides, PrivTF does not aim to protect against model attacks such as model extraction attack [47] and membership inference attack [48] as prior works do. Defending against these attacks is orthogonal to secure inference. One can defend by complementary techniques like differential privacy [49], query auditing [50] and access control.

B. Design Goals

Our work has specific design goals. The first is privacy preservation. Client only learns the inference result and nothing else. In the interaction of two parties, the model owner has only sharings of the intermediate result and learns nothing about the client's input. Both model and user privacy are protected.

Secondly, the protocol should ensure correctness and output accurate results. For building blocks we evaluate the precision by certain error analysis standards and for model accuracy we compare against plaintext output to ensure the correctness.

Thirdly, computation and communication efficiency should be realized as much as possible. The protocols are expected to scale to vectors of high dimensions. To achieve efficiency we try to design tailored protocols for Transformer models.

V. PrivTF Scheme

A. PrivTF Overview

Generally PrivTF works as a series of secure layer protocols with mixed cryptographic primitives. All secure layers are organized on secret sharing of a general ring. Each layer receives a shared input, invokes protocols based on two-party secure computation techniques and produces a desired secure output sharing to the next layer.

The Transformer architecture comprises an embedding layer followed by encoder/decoder layers. We begin by presenting our embedding layer scheme in Section V-B. For the encoder/decoder construction, we study their attention sub-layer and feed-forward sub-layer. The attention sub-layer is composed of matrix multiplication and softmax functions, and the feed-forward sub-layer is composed of matrix multiplication and GeLU functions. Our approach leverages existing uniform bitwidth matrix multiplication schemes while introducing optimizations for non-uniform bitwidth cases. Consequently, we develop four secure computation components to support these sub-layers: matrix multiplication of uniform bitwidths and non-uniform bitwidths, softmax and GeLU. In Section V-C, we detail our protocols (matrix multiplication and softmax) for the attention sub-layer. Then, for the feed-forward sub-layer, we reuse the matrix multiplication protocols from the attention sub-layer and introduce a newly optimized GeLU protocol. Both sub-layers in the encoder and decoder layer employ layer normalization in layer connections, for which we provide a optimization method. Building upon these components, we finally construct the encoder and decoder layer protocols (Section V-D).

B. Embedding Layer

Embedding converts tokens to vectors based on a large vocabulary matrix $\mathbf{W}$ of size $(w \times d_m)$. The vocabulary size is w and the embedding dimension is d_m. We propose the idea of applying PRF as boolean masks to the matrix and give an lightweight protocol for the embedding layer. The core intuition is to send Client a transformed matrix $\mathbf{W}' \in \mathbb{Z}_{2^l}^{w \times d_m}$ that is computationally indistinguishable from random, analogous to $\mathbf{W}$'s boolean secret shares. This is achieved by

Algorithm 1: PrivEmbedding

```
Input: Client's word indices $\mathbf{x}$ of length $n$,
Server's semantic embedding matrix $\mathbf{W} \in \mathbb{Z}_{2^l}^{w \times d_m}$ and an
extra embedding matrix $\mathbf{B} \in \mathbb{Z}_{2^l}^{n \times d_m}$.
Output: $\langle \mathbf{V} \rangle^{\mathcal{S}}, \langle \mathbf{V} \rangle^{\mathcal{C}}$, $\mathbf{V} \in \mathbb{Z}_{2^l}^{n \times d_m}$.
Pre-sharing:
Server masks $\mathbf{W}'[i] = \mathbf{W}[i] \oplus \mathrm{AES}_k(i)$ and sends to Client.
Online:
1. for $\mathbf{x}_i$ in $\mathbf{x}$ do
    1.1. Client encodes $\mathbf{x}_i$ as $\mathbf{u}$ where $\mathbf{u}[\mathbf{x}_i] = 1$.
    1.2. Client shares $\mathbf{u}$ as boolean sharings $[\![\mathbf{u}]\!]^{\mathcal{S}}, [\![\mathbf{u}]\!]^{\mathcal{C}}$.
    1.3. Both parties compute locally
      $[\![\mathbf{W}'[x_i]]\!] = \bigoplus_{j=0}^{w-1} [\![\mathbf{u}_j]\!] \wedge \mathbf{W}'[j]$.
    1.4. Server inputs key $k$ and Client inputs index $\mathbf{x}_i$,
      sAES returns sharings of the mask s.t.
      $[\![s]\!]^{\mathcal{S}} \oplus [\![s]\!]^{\mathcal{C}} = \mathrm{AES}_k(i)$.
    1.5. Server and Client invokes B2A to convert
      $[\![\mathbf{W}'[x_i]]\!] \oplus [\![s]\!]$ to arithmetic sharing $\langle \mathbf{v} \rangle$.
end
Concatenate all $\langle \mathbf{v} \rangle$ together to get the embedded input $\langle \mathbf{V} \rangle$.
2. Extra embedding: $\langle \mathbf{V} \rangle^{\mathcal{S}} \leftarrow \langle \mathbf{V} \rangle^{\mathcal{S}} + \mathbf{B}$, $\langle \mathbf{V} \rangle^{\mathcal{C}} \leftarrow \langle \mathbf{V} \rangle^{\mathcal{C}}$.
```

masking $\mathbf{W}$ with a PRF (the PRF will be instantiated via AES):

$$\mathbf{W}'[i] = \mathbf{W}[i] \oplus \mathrm{PRF}_k(i), i = \{0, 1, ..., w-1\}.$$

Only the server knows the PRF key. When the inference starts, Client encodes the index of interest x as a one hot vector $\mathbf{u}$ of length w, where only the x-th bit equals to one. Client shares it as boolean indicators $[\![\mathbf{u}]\!]^{\mathcal{S}}, [\![\mathbf{u}]\!]^{\mathcal{C}}$. Then both parties use the indicators to compute boolean AND with $\mathbf{W}'$ and XOR row-wise. Because the matrices of both parties are identical, they yield a boolean sharing of the masked entry:

$$[\![\mathbf{W}'[x]]\!] = \bigoplus_{j=0}^{w-1} [\![\mathbf{u}_j]\!] \wedge \mathbf{W}'[j].$$

Next, the secure computation of PRF is needed to unmask the extracted entry: $\mathbf{W}[x] = \mathbf{W}'[x] \oplus \mathrm{PRF}_k(x)$. We instantiate the secure computation of PRF using sAES [18], [45]. Finally, a sharing conversion B2A [18] switches the entry to arithmetic sharing. The advantage of this design is that pre-sharing is performed only once, and computation over the matrix remains local.

Algorithm 1 shows the details of PrivEmbedding. In the pre-sharing phase, Server masks the embedding matrix by a PRF key and sends it to Client. When the inference starts, step 1 is for token embedding, where there are five operations. In step 1.1, Client encodes its input sequence as a series of one-hot vector $\mathbf{u}$. $\mathbf{u}$ is composed of as many as w bits, where only one bit at index x_i is 1 and others are zero, representing the i-th word. In step 1.2, Client splits $\mathbf{u}$ into two boolean shares and sends to Server. In step 1.3, both parties perform local computation over the masked embedding matrix. In step 1.4 and 1.5, they execute sAES and B2A to unmask the result and obtain arithmetic shares of $\langle \mathbf{v} \rangle$. Finally, step 2 involves additional embedding for outputs.

C. Attention Sub-Layer

The attention sub-layer involves matrix multiplication and softmax function. The following section describes two

Algorithm 2: MatMulCrossTerm

```
Input: $\langle \mathbf{X} \rangle^{\mathcal{S}} \in \mathbb{Z}_{2^s}^{n_0 \times n_1}, \langle \mathbf{Y} \rangle^{\mathcal{C}} \in \mathbb{Z}_{2^l}^{n_1 \times n_2}$.
Output: $\{\langle \mathbf{Z} \rangle^{\mathcal{S}}, \langle \mathbf{Z} \rangle^{\mathcal{C}}\} \in \mathbb{Z}_{2^l}^{n_0 \times n_2}$, $\mathbf{Z} = \langle \mathbf{X} \rangle^{\mathcal{S}} \langle \mathbf{Y} \rangle^{\mathcal{C}}$.
foreach $\langle \mathbf{x}_{ij} \rangle^{\mathcal{S}}$ in $\langle \mathbf{X} \rangle^{\mathcal{S}}$ do
    Multiplying $\langle \mathbf{x}_{ij} \rangle^{\mathcal{S}}$ with the j-th row of $\langle \mathbf{Y} \rangle$ ($\langle \mathbf{y}_j \rangle^{\mathcal{C}}$):
    for the b-th bit ($x_b$) of $\langle \mathbf{x}_{ij} \rangle^{\mathcal{S}}$ do //Operate bit by bit.
        Invoke $\binom{1}{2}$-COT$^{l-b}$ for $n_2$ times, where Server is
          the sender with input $x_b$ and Client is the receiver
          with $n_2$ inputs $\{\langle \mathbf{y}_{jk} \rangle^{\mathcal{C}}\}_{k=\{0,1,\ldots,n_2-1\}}$ and learns
          shares of $x_b \times \langle \mathbf{y}_{jk} \rangle^{\mathcal{C}}$ of length $l-b$.
        The shares of $\langle \mathbf{x}_{ij} \rangle^{\mathcal{S}} \langle \mathbf{y}_{jk} \rangle^{\mathcal{C}}$ are computed via the
          local summation: $\sum_{b=0}^{s-1} 2^b \times \langle x_b \times \langle \mathbf{y}_{jk} \rangle^{\mathcal{C}} \rangle$.
    end
end
Compute $\langle \mathbf{z}_{ik} \rangle = \sum_{j=0}^{n_1-1} \langle \langle \mathbf{x}_{ij} \rangle^{\mathcal{S}} \langle \mathbf{y}_{jk} \rangle^{\mathcal{C}} \rangle$ to obtain $\langle \mathbf{Z} \rangle$.
```

protocols for these two building blocks and then a construction of the full attention sub-layer.

1) Matrix Multiplication: We briefly introduce the protocols of matrix multiplication proposed by previous works [20]–[22]. The protocols are based on homomorphic encryption and work in the case of uniform bitwidths. We refer to the protocols as MatMul^l operates on a fixed bitwidth l: $\langle \mathbf{Z} \rangle^{\mathcal{S}}, \langle \mathbf{Z} \rangle^{\mathcal{C}} \leftarrow \mathsf{MatMul}^l(\mathbf{X}, \mathbf{Y})$, where the private matrices are $\mathbf{X} \in \mathbb{Z}_{2^l}^{n_0 \times n_1}, \mathbf{Y} \in \mathbb{Z}_{2^l}^{n_0 \times n_1}, \mathbf{Z} = \mathbf{XY}, \langle \mathbf{Z} \rangle \in \mathbb{Z}_{2^l}^{n_0 \times n_2}$.

However, in the attention sub-layer we identify that the number range is limited and the computation overhead of multiplication can be improved by using non-uniform bitwidths. Therefore, we introduce SIRNN's [14] scheme of matrix multiplication of non-uniform bitwidths. The protocol is denoted as MatMul^{l,s}:

$$\langle \mathbf{Z} \rangle^{\mathcal{S}}, \langle \mathbf{Z} \rangle^{\mathcal{C}} \leftarrow \mathsf{MatMul}^{l,s}(\langle \mathbf{X} \rangle^{\mathcal{S}}, \langle \mathbf{Y} \rangle^{\mathcal{S}}, \langle \mathbf{X} \rangle^{\mathcal{C}}, \langle \mathbf{Y} \rangle^{\mathcal{C}}),$$

where $\mathbf{X} \in \mathbb{Z}_{2^s}^{n_0 \times n_1}, \mathbf{Y} \in \mathbb{Z}_{2^l}^{n_1 \times n_2}$, $s \leq l$, $\mathbf{Z} = \mathbf{XY} \in \mathbb{Z}_{2^l}^{n_0 \times n_2}$.

Multiplying two signed matrices shared in different rings is given by:

$$(\langle \mathbf{X} \rangle^{\mathcal{S}} + \langle \mathbf{X} \rangle^{\mathcal{C}} - 2^{s-1} - 2^s \times \mathsf{CMP}(\langle \mathbf{X} \rangle^{\mathcal{S}} + \langle \mathbf{X} \rangle^{\mathcal{C}} - 2^s, 0))$$
$$\cdot(\langle \mathbf{Y} \rangle^{\mathcal{S}} + \langle \mathbf{Y} \rangle^{\mathcal{C}} - 2^{l-1} - 2^l \times \mathsf{CMP}(\langle \mathbf{Y} \rangle^{\mathcal{S}} + \langle \mathbf{Y} \rangle^{\mathcal{C}} - 2^l, 0)) \bmod 2^l.$$

Note that the Y's CMP item would be removed by modulo 2^l. For remaining items, the computation involves three steps.

1) The first step is to locally compute the item $\langle \mathbf{X} \rangle^{\mathcal{S}} \langle \mathbf{Y} \rangle^{\mathcal{S}}$, $\langle \mathbf{X} \rangle^{\mathcal{C}} \langle \mathbf{Y} \rangle^{\mathcal{C}}$, $2^{s-1}(\langle \mathbf{Y} \rangle^{\mathcal{S}} + \langle \mathbf{Y} \rangle^{\mathcal{C}})$ and $2^{l-1}(\langle \mathbf{X} \rangle^{\mathcal{S}} + \langle \mathbf{X} \rangle^{\mathcal{C}})$.
2) The second step is to compute the cross-term multiplication. Take multiplying $\langle \mathbf{X} \rangle^{\mathcal{S}}$ with $\langle \mathbf{Y} \rangle^{\mathcal{C}}$ as an example. ($\langle \mathbf{X} \rangle^{\mathcal{C}} \langle \mathbf{Y} \rangle^{\mathcal{S}}$ follows a similar process.) This matrix multiplication involves the multiplication of a matrix element $\langle \mathbf{x}_{ij} \rangle$ with the j-th row of $\langle \mathbf{Y} \rangle$ and summation. We realize it by MatMulCrossTerm (Algorithm 2), where we use correlated oblivious transfer on each bit of $\langle \mathbf{x}_{ij} \rangle$. Each multiplication of $\langle \mathbf{x}_{ij} \rangle$ with the vector $\langle \mathbf{y}_j \rangle$ requires s instances of COT([15], [44]) with a communication cost of $n_2(l - s/2 + 1/2)s$ bits.
3) The last step involves performing a secure comparison on $\langle \mathbf{X} \rangle^{\mathcal{S}} + \langle \mathbf{X} \rangle^{\mathcal{C}} - 2^s$, yielding a boolean sharing $[\![\mathbf{B}_X]\!]$. This is used for multiplexer-style multiplications with $(\langle \mathbf{Y} \rangle^{\mathcal{S}} + \langle \mathbf{Y} \rangle^{\mathcal{C}})$, pre-multiplied by 2^s. The multiplexers are realized by MUX on $n_0 n_1 n_2$ elements.

The communication overhead involves two cross-term matrix multiplications requiring $2n_0n_1n_2(l-s/2+1/2)s$ bits from COT, along with multiplexer-style multiplications that require CMP on n_0n_1 elements and MUX on $n_0n_1n_2$ elements.

Improving by vector oblivious linear evaluation: It is possible to leverage vector oblivious linear evaluation (VOLE) [51] to compute the non-uniform matrix multiplication. One of its extensions, called Subfield VOLE, extends n instances of $\mathbf{w} = \mathbf{v} + \Delta\mathbf{u} \in \mathbb{Z}_{2^l}^n$ quickly at low cost. With many instances at hand, two parties can exchange $\mathbf{x}_{ij} - \Delta$ to securely derive a product $\mathbf{x}_{ij}\mathbf{u}$. The advantage of VOLE-based multiplication over the direct use of correlated oblivious transfer is not immediately clear and merits further investigation.

2) Softmax: We also utilize mix-bitwidths in Softmax, which is combined with the protocol for matrix multiplication of non-uniform bitwidths. Softmax is mathematically defined as $\frac{e^{\mathbf{x}_i}}{\sum_{i=0}^{n-1} e^{\mathbf{x}_i}}$, where the input vector $\mathbf{x}$ is of length n.

Specifically, we operate the input sequence matrix row-wise, computing the softmax function for each row in five steps. In step 1 and step 2, we find the maximal value to shift all components of the input vector to negative values: $\mathbf{x}_i - \mathbf{x}_{max}$. This step ensures numerical stability and also meets the requirements of Exp functionality. After the subtraction, step 3 involves Exp to compute the exponentiation and summing them together to obtain the sharings of the denominator. Here the output is smaller than 1, so we set the bit-length $l_e \leftarrow f + 2 + \log n$, where we allocate additional $\log n$ bits for the summation of n numbers in step 4. In step 5, Div computes the quotient, which can represent decimal values using only an $f+2$-bit length representation.

Additionally, Transformer's softmax function employs two types of negative infinity matrices for sequence padding and masking. Since fixed-point arithmetic cannot represent infinity, our protocol implements an alternative approach: (1) padding and masking positions are identified through a parameter (n' in Algorithm 3), and (2) these positions are ignored during computation with zeros directly assigned to the corresponding outputs. Algorithm 3 provides the details of Softmax.

3) PrivAttention: The secure multi-head attention sub-layer protocol PrivAttention is composed of MatMul^l, $\mathsf{MatMul}^{l,s}$, Softmax. The protocol is shown in Algorithm 4. The inputs are secret sharing of queries $\langle\mathbf{Q}\rangle$, keys $\langle\mathbf{K}\rangle$, values $\langle\mathbf{V}\rangle$ of shape $(n \times d_m)$ and weight matrices $\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v$. Step 1 is to perform linear projections using weight matrices. In step 2, Server and Client split the inputs into distinct heads $\langle\mathbf{Q}_i\rangle, \langle\mathbf{K}_i\rangle, \langle\mathbf{V}_i\rangle$ of dimension $d_h = d_m/h$. For each head, they perform the attention mechanism which involves two distinct matrix multiplication protocols and Softmax (step 3.1-3.3). The uniform bitwidth multiplication MatMul^l can be applied to compute the query-key product since both matrices share the same bitwidth l. However, for multiplying the normalized attention matrix (represented with $s = f+2$ bits) with the value matrix (of bitwidth l), we employ the non-uniform bitwidth multiplication protocol. After step 3, the single head attention outputs $\langle\mathbf{Y}_i\rangle$ of head shape $(n \times d_h)$. Finally, Server and Client concatenate all head dimensions together as outputs $\langle\mathbf{Y}\rangle$ of shape $(n \times d_m)$.

Algorithm 3: Softmax

```
Input: Attention matrix $\langle\mathbf{X}\rangle^{\mathcal{S}}, \langle\mathbf{X}\rangle^{\mathcal{C}} \in \mathbb{Z}_{2^l}^{n\times n}$. The length
       before padding is $m$. The fixed-point precision is $f$.
Output: Activated attention matrix$\{\langle\mathbf{X}'\rangle^{\mathcal{S}}, \langle\mathbf{X}'\rangle^{\mathcal{C}}\} \in \mathbb{Z}_{2^{l'}}^{n\times n}$.
Two functional indicators: padding and masking.
foreach row $\langle\mathbf{x}_i\rangle$ in $\langle\mathbf{X}\rangle$ do
    if padding then
        if $i >= m$ then break; //skipping padded rows
        $n' \leftarrow m$.
    if masking then $n' \leftarrow i$;
    Truncate $\langle\mathbf{x}\rangle$ to its first $n'$ elements.
    1. $\langle\mathbf{x}_{max}\rangle \leftarrow$ Max $(\langle\mathbf{x}\rangle)$;
    2. for $\langle\mathbf{x}_j\rangle$ in $\langle\mathbf{x}\rangle$ do $\langle\mathbf{x}_j\rangle \leftarrow \langle\mathbf{x}_j\rangle - \langle\mathbf{x}_{max}\rangle$;
    3. Compute the exponentiation with an appropriate
     bitwidth $l_e \leftarrow (f + 2 + \log n')$:
    $\langle\mathbf{e}\rangle \leftarrow \mathsf{Exp}^{l,l_e}(\langle\mathbf{x}\rangle)$.
    4. $\langle sum\rangle = \Sigma_{j=0}^{n'-1}\langle\mathbf{e}_j\rangle$.
    5. Division: $l' \leftarrow f+2, \langle\mathbf{x}'\rangle \leftarrow \mathsf{Div}^{l_e,l'}(\langle\mathbf{e}\rangle, \langle sum\rangle)$.
end
Server and Client pad then concatenate $\langle\mathbf{x}'\rangle$ to output $\langle\mathbf{X}'\rangle$.
```

Algorithm 4: PrivAttention

```
Input: $\langle\mathbf{Q}\rangle, \langle\mathbf{K}\rangle, \langle\mathbf{V}\rangle \in \mathbb{Z}_{2^l}^{n\times d_m}$, Server's matrices
       $\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v \in \mathbb{Z}_{2^l}^{d_m\times d_m}$.
Output: $\langle\mathbf{Y}\rangle \in \mathbb{Z}_{2^l}^{n\times d_m}$.
1. for $(\mathbf{X}, \mathbf{W}_x)$ in $\{(\mathbf{Q}, \mathbf{W}_q), (\mathbf{K}, \mathbf{W}_k), (\mathbf{V}, \mathbf{W}_v)\}$ do
   $\langle\mathbf{X}\rangle^{\mathcal{S}}, \langle\mathbf{X}\rangle^{\mathcal{C}} \leftarrow \mathsf{MatMul}^l(\mathbf{W}_x, \langle\mathbf{X}\rangle^{\mathcal{C}})$,
   $\langle\mathbf{X}\rangle^{\mathcal{S}} \leftarrow \langle\mathbf{X}\rangle^{\mathcal{S}} + \mathbf{W}_x\langle\mathbf{X}\rangle^{\mathcal{S}}$;
2. Split $\langle\mathbf{Q}\rangle, \langle\mathbf{K}\rangle, \langle\mathbf{V}\rangle$ to $h$ heads (of shape $(n \times d_h)$).
3. for head $i$ do
    3.1 $\langle\mathbf{X}_i\rangle \leftarrow \mathsf{MatMul}^l(\langle\mathbf{Q}_i\rangle^{\mathcal{S}}, \langle\mathbf{K}_i\rangle^{\mathcal{C}})$ +
     $\mathsf{MatMul}^l(\langle\mathbf{Q}_i\rangle^{\mathcal{C}}, \langle\mathbf{K}_i\rangle^{\mathcal{S}}) + \langle\mathbf{Q}_i\rangle^{\mathcal{S}}\langle\mathbf{K}_i\rangle^{\mathcal{S}}$ (or
     $\langle\mathbf{Q}_i\rangle^{\mathcal{C}}\langle\mathbf{K}_i\rangle^{\mathcal{C}}$);
    3.2 $\langle\mathbf{X}'_i\rangle \leftarrow \mathsf{Softmax}(\langle\mathbf{X}_i\rangle)$;
    3.3 $\langle\mathbf{Y}_i\rangle \leftarrow \mathsf{MatMul}^{l,s}(\langle\mathbf{X}'_i\rangle^{\mathcal{S}}, \langle\mathbf{V}_i\rangle^{\mathcal{S}}, \langle\mathbf{X}'_i\rangle^{\mathcal{C}}, \langle\mathbf{V}_i\rangle^{\mathcal{C}})$,
     where $s \leftarrow f+2$, $f$ is the fixed-point precision.
end
4. Server and Client concatenate each head
   $\{\langle\mathbf{Y}_1\rangle, \langle\mathbf{Y}_2\rangle, ..., \langle\mathbf{Y}_h\rangle\}$ to output $\langle\mathbf{Y}\rangle$.
```

D. Encoder and Decoder Layer

In this section, we firstly design a protocol for GeLU function in order to construct the protocol of the feed-forward sub-layer. Then we propose the layer normalization protocol. With all these components and sub-layer protocols, we present protocols for encoder and decoder layers (PrivEncoder, PrivDecoder).

1) GeLU of the feed-forward sub-layer: We compute GeLU using its sigmoid-based expression: $x\mathsf{sigmoid}(1.702x)$ ($\mathsf{sigmoid}(x) = \frac{1}{1+e^{-x}}$). This approach simplifies the computation compared to the original expression used in Iron [20]: $0.5x(1+\mathsf{Tanh}[\sqrt{2/\pi}(x+0.04x^3)])$. This method also enables further cost reductions with our two insights.

The first insight is using mixed bitwidths. The exponentiation output is converged into the range [0,1]. Therefore, we use the bitwidth $f+2$ for the output of sigmoid and save computation as well as communication costs of the following multiplication. The second insight is to leverage the unique property of the sigmoid function, which is defined such that $\mathsf{sigmoid}(x) + \mathsf{sigmoid}(-x) = 1$. SIRNN [14] computes sigmoid by firstly $\frac{1}{1+\exp(-|x|)}$ and then uses another multiplication with $\exp(-|x|)$. But we can leverage the unique symmetry to avoid the multiplication. Let $h = \frac{1}{1+\exp(-1.702|x|)}$. Concretely, $\mathsf{sigmoid}(1.702x)$ is exactly h if x is non-negative and $1-h$ else. Let t be $h - \frac{1}{2}$, then we have:

$$\mathsf{sigmoid}(1.702x) = \begin{cases} \frac{1}{2} + t, x \leq 0 \\ \frac{1}{2} - t, x < 0 \end{cases}.$$

This can be computed by one comparison and one multiplexer: $\frac{1}{2} + t - 2\mathbf{I}(x > 0) \cdot t$. If we take the final product of $\mathsf{sigmoid}(1.702x)$ and x together, GeLU's output is like:

$$\mathsf{GeLU}(x) = \frac{x}{2} + xt - 2\mathbf{I}(x > 0) \cdot xt.$$

Therefore, after computing t and the product xt, we can use a multiplexer MUX and the indicator $b := \mathbf{I}(x > 0)$ from a secure CMP to get the final output.

Algorithm 5 illustrates the protocol (the operations are described for a vector). The numbers $\frac{1}{2}$ and 1 are scaled by the fixed-point precision parameter f. Specifically, there are five steps. Step 1 is comparing the input with zero to determine its sign. The sign is then used to compute the absolute value of $\mathbf{x}_i$, followed by a negation and a multiplication (step 2). The sigmoid-like item is computed via exponentiation and division (step 3), followed by a multiplication with the original input to produce $\mathbf{xt}$ (step 4). Finally, the GeLU output is derived by combining the scaled inputs, $\mathbf{xt}$ (step 5).

Algorithm 5: GeLU

```
Input: $\langle \mathbf{x} \rangle^{\mathcal{S}}, \langle \mathbf{x} \rangle^{\mathcal{C}} \in \mathbb{Z}_{2^l}^{d_m}$. The fixed-point precision is $f$.
Output: $\langle \mathbf{y} \rangle^{\mathcal{S}}, \langle \mathbf{y} \rangle^{\mathcal{C}} \in \mathbb{Z}_{2^l}^{d_m}$.
1. Comparison $[\![\mathbf{b}]\!] \leftarrow \mathsf{CMP}(\langle \mathbf{x} \rangle, 0)$;
2. $\langle \mathbf{x}' \rangle \leftarrow \langle \mathbf{x} \rangle - 2 \times \mathsf{MUX}([\![\mathbf{b}]\!], \langle \mathbf{x} \rangle)$;
   $\langle \mathbf{x}' \rangle \leftarrow \mathrm{int}(1.702 \times 2^f) \times \langle \mathbf{x} \rangle'$ ;
3. The sigmoid-like item $\langle \mathbf{e} \rangle \leftarrow \mathsf{Exp}^{l,f+2}(\langle \mathbf{x}' \rangle)$;
   $\langle \mathbf{h} \rangle \leftarrow \mathsf{Div}^{f+2,f+2}(\langle 2^f \rangle, \langle 2^f \rangle + \langle \mathbf{e} \rangle)$; $\langle \mathbf{t} \rangle \leftarrow \langle \mathbf{h} \rangle - \langle 2^{f-1} \rangle$;
4. Multiplication: $\langle \mathbf{xt} \rangle \leftarrow \mathsf{Product}^{f+2,l}(\langle \mathbf{t} \rangle, \langle \mathbf{x} \rangle)$;
5. $\langle \mathbf{y} \rangle \leftarrow 2^{f-1} \times \langle \mathbf{x} \rangle + \langle \mathbf{xt} \rangle - 2 \times \mathsf{MUX}([\![\mathbf{b}]\!], \langle \mathbf{xt} \rangle)$;
```

The computation overhead compared with Iron [20] is listed in Table IV. Our new insight gives a more efficient method of computing sigmoid and thus achieves a more efficient protocol of GeLU than Iron. We replace two products as multiplexer and save three more products.

Table IV. Approximation Methods for GeLU.

Schemes	Methods	Overhead
Iron [20]	$0.5x(1 + \tanh[a(x + bx^3)])$	1Exp, 1Div, 6Prod, 1CMP
Ours	$x\mathsf{sigmoid}(1.702x)$	1Exp, 1Div, 1Prod, 1CMP, 2MUX

2) PrivFeedForward: We now provide the protocol of the feed-forward sub-layer, PrivFeedForward. The inputs are secret sharings $\langle \mathbf{X} \rangle$ of shape $(n \times d_m)$ and Server's weight matrices $\mathbf{W}_1$ of shape $(d_m \times d_f)$, $\mathbf{W}_2$ of shape $(d_f \times d_m)$. Server and Client execute the first linear transformation MatMul^l to project the sequence to a higher dimension space $(n \times d_f)$. This multiplication is followed by GeLU. Server and Client perform the second MatMul^l to project the sequence back to dimension d_m as $\langle \mathbf{Y} \rangle$. We omit the protocol listing to avoid description redundancy.

3) Layer Normalization: Layer normalization can be viewed as a combination of two steps: linear affine transformation and non-linear normalization. Linear computation can be evaluated by homomorphic encryption. The non-linear part of evaluating mean and variance can be realized by 2PC sub-protocols of Section III-D. More importantly, we find that there is an optimization of fusing the linear part. The typical Transformer pipeline appends layer normalization to each sub-layer. Another pattern [52], [53] is to place layer normalization ahead of sub-layers. It is termed as pre-norm while the former pattern is post-norm. Fig. 2 illustrates the two patterns.

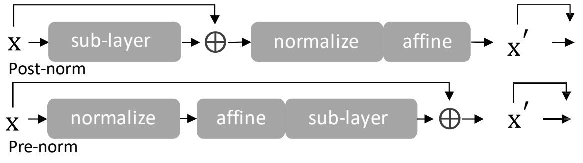


Fig. 2. Two Layer Normalization Patterns.

We prefer pre-norm and adopt it in PrivTF. Pre-norm sends the normalized output directly to the following sub-layer. Since γ and β are fixed and the normalized output is not needed by residual connection, we can fuse the linear part, reducing one multiplication and truncation.

We accordingly propose the secure protocol of normalization PreLN with fusion (Algorithm. 6). The pre-processing phase is computing the parameters, i.e., combining γ, β and weight matrices in the next sub-layer together to get updated parameters. The online phase performs normalization using real-time computed mean and variance. First, the server and client securely compute the mean by summing all the components and applying Div^{l,d_m} to the aggregate value. Variance calculation is similar: both parties invoke Product to compute squared terms, then derive the inverse square root through additive combination, the shifting protocol Div^{l,d_m}, and the ISqrt protocol. Finally, the normalized result is generated by multiplying the mean-centered inputs and the inverse standard deviation through Product.

Algorithm 6: PreLN

```
Input: $\{\langle \mathbf{x} \rangle^{\mathcal{S}}, \gamma, \beta\}$, $\{\langle \mathbf{x} \rangle^{\mathcal{C}}\}$, $\mathbf{x} \in \mathbb{Z}_{2^l}^{d_m}$.
Output: $\langle \mathbf{x}' \rangle^{\mathcal{S}}, \langle \mathbf{x}' \rangle^{\mathcal{C}} \in \mathbb{Z}_{2^l}^{d_m}$.
Model Processing:
The linear coefficients of the next sub-layer are $\mathbf{W}, \mathbf{b}$. Server
  outputs new parameters $\mathbf{W}^* \leftarrow \gamma \mathbf{W}$ and $\mathbf{b}^* \leftarrow \beta \mathbf{W} + \mathbf{b}$.
Online:
1. Server and Client jointly compute the average:
   $\langle sum \rangle = \Sigma_{i=0}^{d_m - 1} \langle \mathbf{x}_i \rangle$ and $\langle \mu \rangle \leftarrow \mathsf{Div}^{l,d_m}(\langle sum \rangle)$.
2. for $\langle \mathbf{x}_i \rangle$ in $\langle \mathbf{x} \rangle$ do $\langle \mathbf{x}_i \rangle \leftarrow \langle \mathbf{x}_i \rangle - \langle \mu \rangle$; //subtraction.
3. Server and Client jointly compute the square root of
   variance: $\langle \mathbf{y} \rangle \leftarrow \mathsf{Product}^{l,l}(\langle \mathbf{x} \rangle, \langle \mathbf{x} \rangle)$;
   $\langle sum' \rangle = \Sigma_{i=0}^{d_m - 1} \langle \mathbf{y}_i \rangle$;
   $\langle \sigma^2 \rangle \leftarrow \mathsf{Div}^{l,d_m}(\langle sum' \rangle)$; $\langle \frac{1}{\sigma} \rangle \leftarrow \mathsf{ISqrt}(\langle \sigma^2 \rangle)$.
4. Server and Client output $\langle \mathbf{x}' \rangle \leftarrow \mathsf{Product}^{l,l}$ $(\langle \mathbf{x} \rangle, \langle \frac{1}{\sigma} \rangle)$.
```

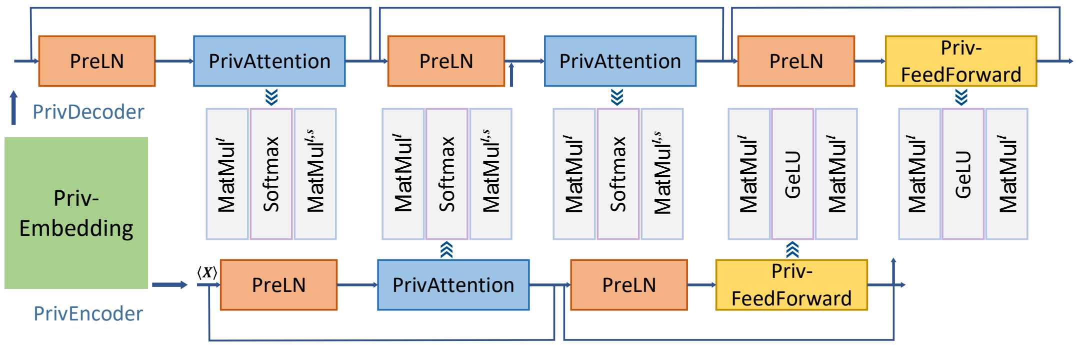


Fig. 3. PrivTF Scheme Execution Flow.

4) Layer Protocols: We provide the protocols for the encoder/decoder layers and the full execution flow in Fig. 3. After the embedding layer, the sequences pass through a stack of multiple encoder or decoder layers. The encoder layer consists of one attention sub-layer and one feed-forward sub-layer, while the decoder includes three sub-layers. In PrivEncoder, the input is $\langle \mathbf{X} \rangle$ of shape $(n \times d_m)$. We apply the optimization of pre-layer normalization here. After normalization we pass $\mathbf{X}$ through the secure attention sub-layer, where the inputs $\mathbf{Q}$, $\mathbf{K}$, $\mathbf{V}$ are equal to $\mathbf{X}$. In PrivAttention, this sub-layer applies attention mechanism (involving MatMul^l, MatMul^{l,s} and Softmax). After a residual connection followed by another pre-norm, the next sub-layer is the feedforward sub-layer. Our PrivFeedForward includes two matrix multiplications of uniform bitwidths and one activation function protocol GeLU. PrivFeedForward outputs $\langle \mathbf{Y} \rangle$, which also serves as the encoder output, maintaining the same shape $(n \times d_m)$ as the encoder input. PrivDecoder is similar to the encoder but includes one additional secure attention sub-layer. The embedding layer and encoder/decoder layer together constitute a secure Transformer model.

VI. Security Analysis

This section analyzes the security of PrivTF. Intuitively, since the input data are encrypted or random sharings, Client's privacy is protected. The weight parameters of the model owner (Server) are also protected because Client only obtains the masked intermediate result. Next we prove in a formal way that PrivTF protocols are secure in the presence of an honest-but-curious adversary. That is, for any adversary $\mathcal{A}$ [46] there exists a simulator Sim that can construct a simulated world. In this simulated world, the view is computationally indistinguishable from the adversary's view in the real world. For the sub-protocols used in PrivTF, we describe their security using the hybrid model to provide the proof. In this model, sub-protocols can be seen as invocations of an ideal functionality. A protocol invoking the ideal functionality $\mathcal{F}$ is said to be in the $\mathcal{F}$-hybrid model.

Theorem 6.1 The protocol PrivEmbedding is secure in honest-but-curious model, assuming the existence of pseudo-random function, the security of sAES and B2A.

Proof. Given the security of the PRF (instantiated as AES), the mask of the embedding matrix is a random value unknown to Client. Consequently, from the Client's perspective, the distribution of the masked matrix $\mathbf{W}'$ is uniformly random, and the view of Client can be simulated. In the online phase, PrivEmbedding involves local computations and invocations of two 2PC sub-protocols. Therefore, it follows the security in $(\mathcal{F}_{\text{sAES}}, \mathcal{F}_{\text{B2A}})$-hybrid model.

Theorem 6.2 The protocol MatMul^{l,s} is secure in honest-but-curious model, assuming the security of proposed sub-protocol COT.

Proof. The protocol is mainly built from COT other than local computation. The security of MatMul^{l,s} follows in $(\mathcal{F}_{\text{COT}})$-hybrid model.

Theorem 6.3 The protocols Softmax, GeLU and PreLN are secure in honest-but-curious model, assuming the security of proposed sub-protocols CMP, Max, MUX, Exp, Div, Product, ISqrt.

Proof. In Softmax, the view of the corrupted party is view $= \{\langle \mathbf{e} \rangle, \langle sum \rangle\}$. Among the view, $\{\mathbf{e}\}$, as the secure outputs of Exp, is indistinguishable. The intermediate result $\langle sum \rangle$ is also secure because it is computed from the secret-shared exponentiation result and keeps uniformly random on the ring. For either Server or Client, they cannot learn the original denominator since it is randomized by the other party. In the last step, Div outputs the sharings of the quotient $\langle \mathbf{x}' \rangle$, which is also secure. The view as well as the output $\langle \mathbf{x}' \rangle$ can be simulated and indistinguished by any probabilistic polynomial-time algorithm. Thus, Softmax is secure in $(\mathcal{F}_{\text{Max}}, \mathcal{F}_{\text{Exp}}, \mathcal{F}_{\text{Div}})$-hybrid model.

The cases of GeLU and PreLN are similar. In GeLU, the view of the adversary is view $= \{[\![\mathbf{b}]\!], \langle \mathbf{x}' \rangle, \langle \mathbf{e} \rangle, \langle \mathbf{h} \rangle, \langle \mathbf{t} \rangle, \langle \mathbf{xt} \rangle, \langle \mathbf{y} \rangle\}$. Among the view, $[\![\mathbf{b}]\!]$, $\langle \mathbf{x}' \rangle$, $\langle \mathbf{e} \rangle$, $\langle \mathbf{h} \rangle$, $\langle \mathbf{xt} \rangle$, $\langle \mathbf{y} \rangle$ are outputs of CMP, MUX, Exp, Div, Product. Therefore, GeLU is secure $(\mathcal{F}_{\text{CMP}}, \mathcal{F}_{\text{Exp}}, \mathcal{F}_{\text{MUX}}, \mathcal{F}_{\text{Div}}, \mathcal{F}_{\text{Product}})$-hybrid model. In PreLN, the view is view $= \{\langle sum \rangle, \langle sum' \rangle, \langle \mu \rangle, \langle \sigma^2 \rangle, \langle \frac{1}{\sigma} \rangle, \langle \mathbf{x}' \rangle\}$. $\langle \mu \rangle$, $\langle \sigma^2 \rangle$, $\langle \frac{1}{\sigma} \rangle$, $\langle \mathbf{x}' \rangle$ are outputs of Div, Product and ISqrt. The intermediate results $\langle sum \rangle, \langle sum' \rangle$ are uniformly random on $\mathbb{Z}_{2^l}$. Therefore, the view and the final outputs $\langle \mathbf{x}' \rangle$ are simulatable and random. PreLN is secure $(\mathcal{F}_{\text{Product}}, \mathcal{F}_{\text{ISqrt}}, \mathcal{F}_{\text{Div}})$-hybrid model.

Theorem 6.4 PrivAttention, PrivFeedForward, PrivEncoder, PrivDecoder are secure in honest-but-curious model, assuming the the security of MatMul^l, MatMul^{l,s}, Softmax, GeLU and PreLN.

Proof. The layer protocols have PrivAttention, PrivFeedForward as sub-layers and a layer normalization module PreLN. Firstly, PrivAttention is the sequential combination of local computations and invocations of MatMul^l, $\mathsf{MatMul}^{l,s}$, Softmax. Secondly, PrivFeedForward is also the sequential combination of local computations and invocations of MatMul^l, GeLU, PreLN. Simulation follows directly from composing the corresponding simulators. Thus the encoder layer protocol PrivEncoder and the decoder layer protocol PrivDecoder is secure in (MatMul^l, $\mathsf{MatMul}^{l,s}$, Softmax, PreLN)-hybrid model.

VII. Performance Evaluation

A. Implementation

To implement our framework, we build upon the open source library SCI (secure and correct inference library built by SCI library [26], [54]). In the secret domain used by PrivTF, data are represented in fixed-point integers and shared over ring Z_{2^l}. We basically set the ring as $Z_{2^{37}}$ in our implementations ($l = 37$). This setting of ring is consistent with prior work [13], [55]. To present the weights to 37-bit fixed-point integers, we use a scaling factor of 2^{12}, meaning that the fixed-point precision f is 12-bit.

All our experiments are conducted on a computer with 3.10 GHz Core 10Gen CPU and 16 GB RAM. Our programs are implemented in C++ and compiled by gcc 9.4.0. The communication is simulated in LAN setting using gigabit ethernet network interfaces NetIO. The communication cost includes both parties' sending messages but excludes the setup phase.

For our experiments, we select two representative BERT models of different sizes: BERT-tiny [56] and BERT-base [2], following common practice in related work. We additionally implement a vision Transformer model (ViT) [3] with the optimization of pre-norm. These three models employ encoder-only configurations, so we also include Transformer-base (TF-base) [1] with six decoder layers for measurement. Table V details each model's parameter configurations, including the number of layers, embedding dimensions, attention sub-layer specifications (multiple 64-dimensional heads), and feed-forward layer sizes. All models process sequences of up to 128 tokens.

Table V. Model Parameter.

Model Parameter	Layers	d_m	$h\&d_h$	d_f	n
Bert-tiny [56]	2	128	2&64	512	$\leq$128
Bert-base [2]	12	768	12&64	3072	$\leq$128
TF-base [1]	6	512	8&64	2048	$\leq$128
ViT [3]	12	768	12&64	3072	$\leq$128

B. Performance of Embedding

The performance of PrivEmbedding is shown in Table VI. In the comparison the embedding matrix has a vocabulary of size 30522 or 50257, each embedding vector is d_m-dimensional and the input is one token. d_m is the model dimension in Table V.

Our protocol initiates the computation with a pre-shared masked matrix. Although this initial step involves communication, it is a one-time effort, as the pre-sharing can be amortized to subsequent queries. Since we only do local traversing, PrivEmbedding has communication overhead independent of the vocabulary size. For vocabulary with the same dimension 768, our PrivEmbedding takes the same 1.1s for computation time and 40MB for communication. The computation time and communication requirements scale with the embedding dimension, as the secure computation of AES involves more blocks. For vocabulary of size 30522, our PrivEmbedding takes 0.1s for 128-dimensional vectors and 1.1s for 768-dimensional vectors. Our protocol requires one round for pre-sharing and three rounds for secure computation of AES in the online phase, while existing approaches based on homomorphic encryption require two rounds.

Table VI. Performance of Embedding Layer Protocols.

Vocabulary	Scheme	Time	Comm.
(30522×128)	Cheetah [15]	1.5s	62MB
	Iron [20]	0.8s	45MB
	BumbleBee [21]	0.5s	22MB
	Ours (pre-sharing)	0.1s (35ms)	3.1MB (17MB)
(30522×768)	Cheetah [15]	8.1s	357MB
	Iron [20]	2.4s	45MB
	BumbleBee [21]	1.8s	22MB
	Ours (pre-sharing)	1.1s (0.4s)	19MB (103MB)
(50257×768)	Cheetah [15]	12s	877MB
	Iron [20]	4.9s	77MB
	BumbleBee [21]	2.9s	24MB
	Ours (pre-sharing)	1.1s (0.8s)	19MB (171MB)

For the embedding layer with a vocabulary size of (30522×128), our PrivEmbedding achieves a computation time of 0.1 seconds and communicates at 3.1MB, which is significantly lower than the 1.5 seconds and 62MB required by Cheetah [15], the 0.8 seconds and 45MB required by Iron [20], and the 0.5 seconds and 22MB of BumbleBee [21]. While our approach involves a one-time pre-sharing overhead of 17MB, this initial cost is amortized across multiple queries, resulting in negligible per-query communication overhead for batch processing scenarios.

When scaling up to a higher-dimensional embedding layer with a vocabulary size of (30522×768), our PrivEmbedding achieves a computation time of 1.1 seconds and communication cost of 19MB. Compared to prior work, we achieve 54% faster computation than Iron (2.4 seconds) and 38% faster than BumbleBee (1.8 seconds). In communication efficiency, we demonstrate more than ten times and two times reductions compared to Cheetah and Iron, respectively. While maintaining communication overhead comparable to BumbleBee, our protocol requires lower computation time. This performance advantage persists with an expanded vocabulary size of (50257×768). Our PrivEmbedding maintains the performance with computation time of 1.1 seconds and a communication overhead of 19MB. Iron's performance degrades to 4.9 seconds and 77MB. BumbleBee requires 24MB communication and 2.9 seconds.

C. Performance of the Attention Sub-Layer

1) Matrix Multiplication: We evaluate our matrix multiplication protocol and compare with other protocols. The input are $\mathbf{X} \in \mathbb{Z}_{2^l}^{128\times128}\mathbf{Y} \in \mathbb{Z}_{2^l}^{128\times64}$. Our $\mathsf{MatMul}^{l,s}$ allows multiplication over two matrices of different bitwidths. An alternative of computing over uniform bitwidths is to align the input to the same bitwidth and conduct multiplication protocols (the underlying technique can be HE or OT). However, it takes extra cost to expand the bitwidth and operating on the uniform bitwidth also brings unnecessary overhead of time and communication. Our non-uniform design brings 1.5-2.1× improvement of running time than the methods of uniform bitwidths. In terms of communication, our protocol has 5.6× more costs than the HE-based method (which is reasonable as HE support compact packing) and 1.35× less costs than the OT method. Our protocol require 19 communication rounds, compared with 23 rounds of the approach of uniform bitwidths.

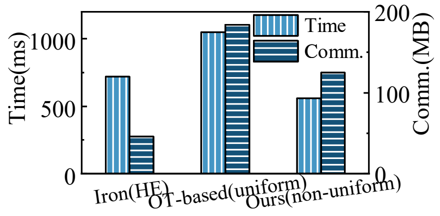


Fig. 4. Performance Comparison of Matrix Multiplication Protocols.

2) Softmax function: We analyze the precision of Softmax using the units in last place (ULP) error metric, which is widely adopted by standard math libraries. Specifically, the ULP error measures the number of representable floating-point values between the exact result and its finite-bit approximation. For comparison, we also evaluate the ULP errors of other approximation methods. Table VII summarizes the methods used in prior works and their error analysis results. Our protocols maintain high computational accuracy, as the underlying math functionalities are precise [14]. Our $\mathsf{Softmax}$ implementation achieves an average ULP error of 0.039 and a maximum ULP error of 1, which is comparable to BOLT [22]. MPCformer [32] replaces the exponentiation with rough approximation so the average ULP error is high. PrivFormer [33] has better precisions using the special polynomial and Newton iteration but the error is still above ours. For further details on ULP, we refer readers to [14].

For performance analysis, Table VIII presents a comparative study between our optimized protocol and the Iron scheme. The input is a vector of size 128. $\mathsf{Softmax}$ with our specific design of variable bitwidths reduces 20% of running time and communication than Iron [20]. Considering softmax appears many times in Transformer inference, the optimization has practical significance. Table VIII further highlights the differences in communication complexity. Our scheme

Table VII. Approximation Methods for Softmax.

Schemes	Methods	ULP Error AVG	ULP Error MAX
MPCformer-1	$\frac{ReLU(\mathbf{x})}{\sum ReLU(\mathbf{x})}$	10	21
MPCformer-2	$\frac{(\mathbf{x}+c)^2}{\sum(\mathbf{x}+c)^2}$	42	129
PrivFormer	$\exp(x) = (1 + x + 0.5x^2 +$ $0.1665x^3 + 0.0438x^4)$	0.703	7
BOLT	$(0.385(p + 1.353)^2 + 0.344) >> z$	0.059	2
BumbleBee	$\exp(x) = (1 + \frac{x}{2^k})^{2^k}$	10	20
Ours	Look-ups for $\exp(x)$ Goldschmidht method for x^{-1}	0.039	1

demonstrates a lower communication overhead, with an average data transmission of 1.87MB, compared to 2.70MB in Iron. We also require only 185 rounds, reducing 100 rounds.

Table VIII. Performance of Non-linear Function Protocols.

Scheme	Softmax Time	Softmax Rounds	Softmax Comm.	GeLU Time	GeLU Rounds	GeLU Comm.
Iron	141ms	281	2.70MB	623ms	255	67MB
Ours	107ms	185	1.87MB	248ms	154	29MB

D. Performance of the Encoder/Decoder Layer

Firstly, we evaluate the performance of our GeLU protocol. Table VIII shows that our scheme demonstrates an efficiency improvement compared to Iron. When processing a vector of size 3072, our approach finishes in approximately 248 milliseconds, which is a substantial improvement over Iron's 623 milliseconds. Our protocol requires less interaction rounds and communication. Our protocol reduces operators of secure computation and saves 10 communication rounds than Iron. The total communication cost is also 1.84× lower, as our GeLU requires 29MB while Iron requires 67 MB. Table VIII shows that our scheme demonstrates lower communication rounds. We reduce 100 rounds, improving by 1.5 times.

Then we evaluate our layer protocols $\mathsf{PrivEncoder}$, $\mathsf{PrivDecoder}$. $\mathsf{PrivEncoder}$ and $\mathsf{PrivDecoder}$ are composed of sub-layer protocols and non-linear functions. Table IX shows the performance of encoder/decoder protocols of four models. For ViT which naturally supports a pre-layer normalization, we compare our scheme with pre-norm pattern against the counterpart with post-norm pattern. Results demonstrate that pre-norm improves efficiency. PreLN with pre-norm pattern saves one multiplicative depth due to the linear transformation parameters absorbed into the next linear layer. Doing this we reduce one execution of multiplication and truncation, which is more efficient than the normal post-norm. In an encoder and decoder our optimization gains improvement of two percent of the execution time and reduces six percent of communication.

Performance of Different Sequence Lengths: To evaluate the impact of sequence length, we measure the execution time of the encoder layer on Bert-base under varying lengths (shown in Fig 5). For short sequences (32 tokens), the

Table IX. Performance of Layer Protocols.

Protocol	Model&Pattern		Time	Comm.
PrivEncoder	Bert-tiny	PostLN	88s	9.72GB
	Bert-base	PostLN	244s	20.2GB
	ViT	PostLN	245s	20.9GB
	ViT	PreLN	240s	19.6GB
PrivDecoder	TF-base	PostLN	162s	8.6GB
	TF-base	PreLN	154s	8GB

Table X. End-to-end Performance.

Model	Scheme	Time	Comm.
Bert-tiny	Iron [20]	204s	9.7GB
	Ours	177s	9.5GB
Bert-base	Iron [20]	3654s	316GB
	Ours	2938s	242GB
TF-base	Iron [20]	976s	52GB
	Ours	748s	45GB

protocol takes about three seconds, while longer sequences incur higher overheads. Fig. 5 provides a detailed breakdown of the overhead, which can be categorized into protocols of linear and non-linear computation. The linear component encompasses matrix multiplication operations, along with necessary truncation which are essential for numeric accuracy. The non-linear component consists of operations such as softmax, GeLU and normalization. We collect the rest and denote it as a miscellaneous part including local vector operations and others. Matrix multiplication is observed to be more time-consuming for longer sequences due to the expansion of matrix dimensions and the increased size of ciphertexts. Truncation also contributes significantly to the overhead, and we believe addressing it will be a key challenge for future work. For non-linear functions, overhead increases gradually, but its proportion of total time decreases to approximately 20%, as processing vectors amortizes costs.

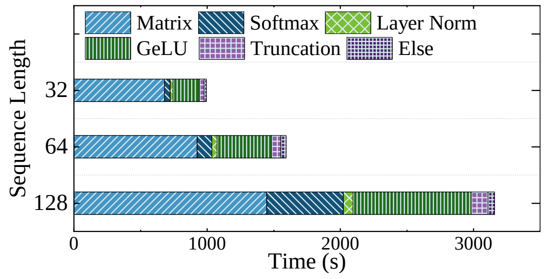


Fig. 5. Performance Breakdown of Different Sequence Lengths.

E. End-to-end Inference

To evaluate end-to-end performance, we implement our scheme on three models and compare with prior work Iron [20] in Table X. Iron is set in two-party model without the assistance of third parties or pre-generated randomness, which is consistent with PrivTF. Our improvements stem from specialized designs for Transformer layers, including large vocabulary matrix embedding and optimized protocols for attention and layer normalization computations. Results demonstrate that PrivTF achieves approximately 1.3 times reductions in runtime and communication costs compared to Iron [20].

Performance on Datasets: We implement BERT securely with our protocols for four NLP tasks over the datasets of the Microsoft Research Paraphrase Corpus (MRPC), the Multi-Genre Natural Language Inference Corpus (MNLI), the Stanford Question Answering Dataset (QNLI) and the Stanford Sentiment Treebank (SST-2) from GLUE benchmarks [57]. In Fig. 6a, we compare the dataset accuracy with the plaintext baseline. The accuracy of PrivTF on fixed-point computation is maintained well. Specifically, the accuracy has 0.3% loss in average over four datasets and is even slightly higher on MNLI. In Fig. 6b, we also record the output value of several samples to compare with plaintext. The classification probability is in the range from zero to one. The output of PrivTF is numerically close to the plaintext computation result, at a deviation less than 0.1. The above experiment results illustrate that our protocols are numerically precise.

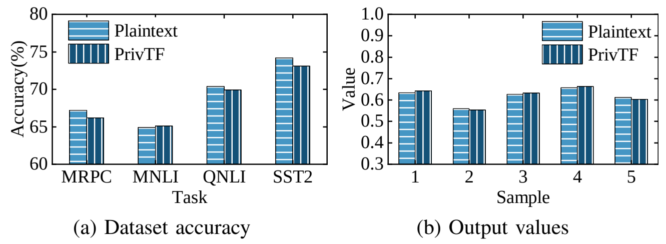


Fig. 6. Task Performance Compared with Plaintext.

VIII. Conclusion

We presented PrivTF, a privacy-preserving scheme for Transformer two-party inference. Unlike convolutional neural networks, Transformers have unique complexities that have not been well addressed in secure computation. To tackle the heavy performance overhead, PrivTF introduces specialized protocols for key Transformer layers: PrivEmbedding for the embedding layer, $\mathsf{MatMul}^{l,s}$ and Softmax with mixed bitwidths for the attention sub-layer, as well as GeLU and PreLN that reduce secure computation operators. These protocols collectively form the complete PrivTF scheme. Our analysis and experiments on practical models confirm that PrivTF is both correct and efficient, achieving significant performance improvements.

Acknowledgment

This work is supported in part by the National Natural Science Foundation of China (NSFC) under Grants No. 61972371 and No. 62372425, Anhui Provincial Key Research and Development Plan under Grant No. 2022a05020050, Youth Innovation Promotion Association of Chinese Academy of Sciences (CAS) under Grant No. Y202093.

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, pp. 6000–6010, Curran Associates Inc., 2017.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pp. 4171–4186, ACL, 2019.

[3] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *Proceedings of the 9th International Conference on Learning Representations (ICLR)*, 2021.

[4] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, vol. 33, pp. 1877–1901, 2020.

[5] C. Song and A. Raghunathan, "Information leakage in embedding models," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pp. 377–390, ACM, 2020.

[6] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning (PMLR)*, pp. 201–210, PMLR, 2016.

[7] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *Proceedings of the 27th USENIX Security Symposium (USENIX Security)*, pp. 1651–1669, USENIX Association, 2018.

[8] S. Li, K. Xue, B. Zhu, C. Ding, X. Gao, D. Wei, and T. Wan, "FALCON: A Fourier transform based approach for fast and secure convolutional neural network predictions," in *Prceedings of the 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8702–8711, IEEE/CVF, 2020.

[9] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "DELPHI: A cryptographic inference service for neural networks," in *Proceedings of the 29th USENIX Security Symposium (USENIX Security)*, pp. 2505–2522, USENIX Association, 2020.

[10] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *Proceedings of the 2017 IEEE symposium on security and privacy (S&P)*, pp. 19–38, IEEE, 2017.

[11] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via MiniONN transformations," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pp. 619–631, ACM, 2017.

[12] N. Agrawal, A. Shahin Shamsabadi, M. J. Kusner, and A. Gascón, "QUOTIENT: Two-party secure neural network training and prediction," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pp. 1231–1247, ACM, 2019.

[13] D. Rathee, M. Rathee, N. Kumar, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "CrypTFlow2: Practical 2-party secure inference," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pp. 325–342, ACM, 2020.

[14] D. Rathee, M. Rathee, R. K. Kiran Goli, D. Gupta, R. Sharma, N. Chandran, and A. Rastogi, "SIRNN: A math library for secure RNN inference," in *Proceedings of the 2021 IEEE Symposium on Security and Privacy (S&P)*, pp. 1003–1020, IEEE, 2021.

[15] Z. Huang, W.-j. Lu, C. Hong, and D. Jiansheng, "Cheetah: Lean and fast secure two-party deep neural network inference," in *Proceedings of the 31st USENIX Security Symposium (USENIX Security)*, pp. 809–826, USENIX Association, 2022.

[16] J. Feng, Y. Wu, H. Sun, S. Zhang, and D. Liu, "Panther: Practical secure two-party neural network inference," *IEEE Transactions on Information Forensics and Security*, vol. 20, pp. 1149–1162, 2025.

[17] M. S. Riazi, M. Samragh, H. Chen, K. Laine, K. Lauter, and F. Koushanfar, "XONN: XNOR-based oblivious deep neural network inference," in *Proceedings of the 27th USENIX Security Symposium (USENIX Security)*, pp. 1651–1669, USENIX Association, 2018.

[18] D. Demmler, T. Schneider, and M. Zohner, "ABY-A framework for efficient mixed-protocol secure two-party computation," in *Proceedings of the 2015 Network and Distributed System Security Symposium (NDSS)*, The Internet Society, 2015.

[19] P. Mohassel and P. Rindal, "ABY^3: A mixed protocol framework for machine learning," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pp. 35–52, ACM, 2018.

[20] M. Hao, H. Li, H. Chen, P. Xing, G. Xu, and T. Zhang, "Iron: Private inference on transformers," in *Advances in Neural Information Processing Systems*, vol. 35, pp. 15718–15731, Curran Associates, Inc., 2022.

[21] W. jie Lu, Z. Huang, Z. Gu, J. Li, J. Liu, C. Hong, K. Ren, T. Wei, and W. Chen, "BumbleBee: Secure two-party inference framework for large transformers," in *Proceedings of the 2025 Network and Distributed System Security Symposium (NDSS)*, The Internet Society, 2025.

[22] Q. Pang, J. Zhu, H. Möllering, W. Zheng, and T. Schneider, "BOLT: Privacy-preserving, accurate and efficient inference for transformers," in *Proceedings of the 2024 IEEE symposium on security and privacy (S&P)*, pp. 4753–4771, IEEE, 2024.

[23] B. Knott, S. Venkataraman, A. Hannun, S. Sengupta, M. Ibrahim, and L. van der Maaten, "CrypTen: Secure multi-party computation meets machine learning," in *Advances in Neural Information Processing Systems*, vol. 34, pp. 4961–4973, Curran Associates, Inc., 2021.

[24] S. Tan, B. Knott, Y. Tian, and D. J. Wu, "CryptGPU: Fast privacy-preserving machine learning on the gpu," in *Proceedings of the 2021 IEEE Symposium on Security and Privacy (S&P)*, pp. 1021–1038, IEEE, 2021.

[25] M. Keller, "MP-SPDZ: A versatile framework for multi-party computation," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pp. 1575–1590, ACM, 2020.

[26] N. Chandran, D. Gupta, A. Rastogi, R. Sharma, and S. Tripathi, "EzPC: Programmable and efficient secure two-party computation for machine learning," in *Proceedings of the 2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 496–511, 2019.

[27] L. K. L. Ng and S. S. M. Chow, "GForce: Gpu-friendly oblivious and rapid neural network inference," in *Proceedings of the 30th USENIX Security Symposium (USENIX Security)*, pp. 2147–2164, USENIX Association, 2021.

[28] X. Liu, Y. Zheng, X. Yuan, and X. Yi, "Securely outsourcing neural network inference to the cloud with lightweight techniques," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 1, pp. 620–636, 2023.

[29] J. Wang, D. He, A. Castiglione, B. B. Gupta, M. Karuppiah, and L. Wu, "PCNN$_{\text{CEC}}$: Efficient and privacy-preserving convolutional neural network inference based on cloud-edge-client collaboration," *IEEE Transactions on Network Science and Engineering*, vol. 10, no. 5, pp. 2906–2923, 2023.

[30] R. Zhao, Y. Xie, D. He, K.-K. R. Choo, and Z. L. Jiang, "Practical privacy-preserving convolutional neural network inference framework with edge computing for health monitoring," *IEEE Transactions on Network Science and Engineering*, vol. 11, no. 6, pp. 5995–6006, 2024.

[31] Q. Feng, D. He, Z. Liu, H. Wang, and K.-K. R. Choo, "SecureNLP: A system for multi-party privacy-preserving natural language processing," *IEEE Transactions on Information Forensics and Security*, vol. 15, no. 1, pp. 3709–3721, 2020.

[32] D. Li, H. Wang, R. Shao, H. Guo, E. P. Xing, and H. Zhang, "MPCFORMER: Fast, performant and private transformer inference with MPC," in *Proceedings of the 11th International Conference on Learning Representations (ICLR)*, 2023.

[33] Y. Akimoto, K. Fukuchi, Y. Akimoto, and J. Sakuma, "Privformer: Privacy-preserving transformer with MPC," in *Proceedings of 8th European Symposium on Security and Privacy (EuroS&P)*, pp. 392–410, IEEE, 2023.

[34] N. Jawalkar, K. Gupta, A. Basu, N. Chandran, D. Gupta, and R. Sharma, "Orca: Fss-based secure training and inference with gpus," in *Proceedings of the 2024 IEEE Symposium on Security and Privacy (S&P)*, pp. 597–616, IEEE, 2024.

[35] E. Boyle, N. Gilboa, and Y. Ishai, "Function secret sharing: Improvements and extensions," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pp. 1292–1303, ACM, 2016.

[36] M. Hao, H. Li, H. Chen, P. Xing, and T. Zhang, "FastSecNet: An efficient cryptographic framework for private neural network inference," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 2569–2582, 2023.

[37] H. Chen, H. Li, Y. Wang, M. Hao, G. Xu, and T. Zhang, "PriVDT: An efficient two-party cryptographic framework for vertical decision trees," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 1006–1021, 2023.

[38] C. Zeng, D. He, Q. Feng, X. Yang, and Q. Luo, "SecureGPT: A framework for multi-party privacy-preserving transformer inference in gpt," *IEEE Transactions on Information Forensics and Security*, vol. 19, pp. 9480–9493, 2024.

[39] J. Zhang, X. Yang, L. He, K. Chen, W. jie Lu, Y. Wang, X. Hou, J. Liu, K. Ren, and X. Yang, "Secure transformer inference made non-interactive," in *Proceedings of the 2025 Network and Distributed System Security Symposium (NDSS)*, The Internet Society, 2025.

[40] J. He, K. Yang, G. Tang, Z. Huang, L. Lin, C. Wei, Y. Yan, and W. Wang, "Rhombus: Fast homomorphic matrix-vector multiplication for secure two-party inference," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pp. 2490–2504, ACM, 2024.

[41] T. Chen, H. Bao, S. Huang, L. Dong, B. Jiao, D. Jiang, H. Zhou, J. Li, and F. Wei, "THE-X: Privacy-preserving transformer inference with homomorphic encryption," in *Findings of the Association for Computational Linguistics (ACL)*, pp. 3510–3520, ACL, 2022.

[42] Q. Zhang, C. Xin, and H. Wu, "SecureTrain: An approximation-free and computationally efficient framework for privacy-preserved neural network training," *IEEE Transactions on Network Science and Engineering*, vol. 9, no. 1, pp. 187–202, 2022.

[43] V. Kolesnikov and R. Kumaresan, "Improved OT extension for transferring short secrets," in *Proceedings of the 33rd Annual Cryptology Conference (CRYPTO)*, pp. 54–70, Springer Berlin Heidelberg, 2013.

[44] K. Yang, C. Weng, X. Lan, J. Zhang, and X. Wang, "Ferret: Fast extension for correlated OT with small communication," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pp. 1607–1626, ACM, 2020.

[45] G. Dessouky, F. Koushanfar, A.-R. Sadeghi, T. Schneider, S. Zeitouni, and M. Zohner, "Pushing the communication barrier in secure computation using lookup tables," in *Proceedings the 2017 Network and Distributed System Security Symposium (NDSS)*, The Internet Society, 2017.

[46] Y. Lindell, *How to Simulate It – A Tutorial on the Simulation Proof Technique*, pp. 277–346. Springer International Publishing, 2017.

[47] N. Carlini, F. Tramer, E. Wallace, M. Jagielski, A. Herbert-Voss, K. Lee, A. Roberts, T. Brown, D. Song, U. Erlingsson, *et al.*, "Extracting training data from large language models," in *Proceedings of the 30th USENIX Security Symposium (USENIX Security)*, pp. 2633–2650, USENIX Association, 2021.

[48] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *Proceeings of the 2017 IEEE symposium on security and privacy (S&P)*, pp. 3–18, IEEE, 2017.

[49] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pp. 308–318, ACM, 2016.

[50] M. Juuti, S. Szyller, S. Marchal, and N. Asokan, "PRADA: Protecting against dnn model stealing attacks," in *Proceedings of the 2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 512–527, IEEE, 2019.

[51] E. Boyle, G. Couteau, N. Gilboa, and Y. Ishai, "Compressing vector OLE," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pp. 896–912, ACM, 2018.

[52] Q. Wang, B. Li, T. Xiao, Z. Jingbo, C. Li, D. F. Wong, and L. S. Chao, "Learning deep transformer models for machine translation," in *Proceedings of the 2019 Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 1810–1822, ACL, 2019.

[53] R. Xiong, Y. Yang, D. He, K. Zheng, S. Zheng, C. Xing, H. Zhang, Y. Lan, L. Wang, and T. Liu, "On layer normalization in the transformer architecture," in *Proceedings of the 37th International Conference on Machine Learning (PMLR)*, vol. 119, pp. 10524–10533, PMLR, 2020.

[54] "Secure and correct inference (SCI) library." https://github.com/mpc-msri/EzPC/tree/master/SCI. accessed on Jun. 2025.

[55] "Opencheetah." https://github.com/Alibaba-Gemini-Lab/OpenCheetah. accessed on Jun. 2025.

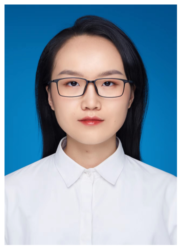
[56] I. Turc, M.-W. Chang, K. Lee, and K. Toutanova, "Well-read students learn better: On the importance of pre-training compact models." https://arxiv.org/abs/1908.08962, 2019. arXiv preprint.

[57] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman, "GLUE: A multi-task benchmark and analysis platform for natural language understanding," in *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, ICLR, 2019.

Ziwei Peng received his B.S. degree in Information Security from School of the Gifted Young, University of Science and Technology of China (USTC) in July, 2022. He is currently working toward the Ph.D degree from the School of Cyber Science and Technology, USTC. His research interests include applied cryptography and privacy computing.

Kaiping Xue (M'09-SM'15) received the bachelor's degree from the Department of Information Security, University of Science and Technology of China (USTC), in 2003 and received the Ph.D degree from the Department of Electronic Engineering and Information Science (EEIS), USTC, in 2007. From May 2012 to May 2013, he was a postdoctoral researcher with the Department of Electrical and Computer Engineering, University of Florida. Currently, he is a Professor in the School of Cyber Science and Technology, USTC. He is also the director of Network and Information Center, USTC. His research interests include next-generation Internet architecture design, transmission optimization and network security. His work won best paper awards in IEEE MSN 2017 and IEEE HotICN 2019, the Best Paper Honorable Mention in ACM CCS 2022, the Best Paper Runner-Up Award in IEEE MASS 2018, and the best track paper in MSN 2020. He serves on the Editorial Board of several journals, including the IEEE Transactions on Information Forensics and Security (TIFS), the IEEE Transactions on Dependable and Secure Computing (TDSC), the IEEE Transactions on Wireless Communications (TWC), and the IEEE Transactions on Network and Service Management (TNSM). He has served as a Guest Editor for many reputed journals/magazines, and he has also served as a track/symposium chair for some reputed conferences. He is an IET Fellow and an IEEE Senior Member.

Bin Zhu (Graduate Student Member, IEEE) received his bachelor's degree in Information Security from the School of Cyber Science and Technology, University of Science and Technology of China (USTC) in 2019. He is currently working toward the Ph.D degree from the School of Cyber Science and Technology, USTC. His research interests include Network security and applied cryptography.



Yaxuan Huang received her becholar's degree from the Department of Information Security, University of Science and Technology of China (USTC) in 2021. She is currently working toward the Ph.D degree from the School of Cyber Science and Technology, USTC. Her research interests include applied cryptography and privacy computing.

Jian Li (Senior Member, IEEE) received his bachelor's degree from the Department of Electronics and Information Engineering, Anhui University, in 2015, and received doctor's degree from the Department of Electronic Engineering and Information Science (EEIS), University of Science and Technology of China (USTC), in 2020. From Nov. 2019 to Nov. 2020, he was a visiting scholar with the Department of Electronic and Computer Engineering, University of Florida. From Dec. 2020 to Dec. 2022, he was a Post-Doctoral researcher with the School of Cyber Science and Technology, USTC. He is currently an associate researcher with the School of Cyber Science and Technology, USTC. He also serves as an Editor of China Communications. His research interests include wireless networks, next-generation Internet, quantum networks, and network security.

Meng Hao received the B.S. degree in information security in 2018, from the University of Electronic Science and Technology of China (UESTC), where he is currently working toward the Ph.D. degree in cyber security at UESTC. His research interests include applied cryptography and privacy-preserving deep learning.

Tianwei Zhang (Member, IEEE) received the bachelor's degree from Peking University in 2011 and the Ph.D. degree from Princeton University in 2017. He is currently an Assistant Professor with the School of Computer Science and Engineering, Nanyang Technological University. his research focuses on computer system security. He is particularly interested in security threats and defenses in machine learning systems, autonomous systems, computer architecture and distributed systems.