Computers, Materials & Continua

Tech Science Press

https://doi.org/10.32604/cmc.2026.078473

ARTICLE

Privacy-Preserving Transformer Inference with Optimized Homomorphic Encryption and Secure Collaborative Computing

Tao Bai[1], Yang Tang[2], Kuan Shao[3], Zhenyong Zhang[3,*] and Yuanteng Liu[4]

[1]Guizhou Provincial Meteorological Data Center, Guiyang, China
[2]Technical Department of the People's Procuratorate of Guizhou Province, Guiyang, China
[3]College of Computer Science and Technology, Guizhou University, Guiyang, China
[4]Colorful Guizhou Digital Technology Co., Ltd., Guiyang, China

*Corresponding Author: Zhenyong Zhang. Email: zhangzy@gzu.edu.cn

Received: 31 December 2025; Accepted: 10 March 2026; Published: 08 May 2026

ABSTRACT: In recent years, the rapid development of artificial intelligence has greatly promoted the application of Machine Learning as a Service (MLaaS). Users can upload their requirements through front-end applications, and the server provides model inference services after receiving the user input. However, MLaaS may lead to serious privacy breaches. Large language model services are typical representatives of MLaaS, and the Transformer is a typical structure in large language models. Therefore, this paper proposes a privacy-protected Transformer inference scheme based on the CKKS fully homomorphic encryption scheme to optimize computational and communication efficiency. Firstly, this paper implements efficient matrix multiplication based on ring multiplication and optimizes the matrix partition parameters to adapt to different types (including ciphertext-plaintext and ciphertext-ciphertext) and different matrix dimensions. Secondly, this paper optimizes and designs secure Softmax, LayerNorm, and Gelu protocols based on parameter fuzzing and collaborative computing to perform efficient, secure atomic computations over ciphertexts. Finally, experiments on text classification were conducted on the IMDB and AGNEWS datasets. The results show that, under our experimental settings (including an AMD Ryzen 7 5700G CPU with 32 GB RAM and 8-thread parallel computing using the Lattigo library), the scheme proposed in this paper completes the inference process within 3 s, with communication costs below 1 GB, and the computing accuracy is comparable to that of plaintext computing.

KEYWORDS: Machine learning as a service; privacy preservation; Transformer; collaborative computing

1 Introduction

In recent years, Artificial Intelligence (AI) has witnessed rapid development. Large models such as DeepSeek [1] and ChatGPT [2] have been widely adopted, and Machine Learning as a Service (MLaaS) is increasingly becoming ubiquitous, achieving significant efficacy in fields such as medicine and economics. However, these applications also introduce concerns regarding the leakage of users' private information. For instance, in 2017, due to a failure to implement effective encryption, Spira Toys leaked over 2.2 million voice messages collected from parents and children via smart toy devices, resulting in severe privacy infringements [3].

As a general-purpose neural network architecture, the Transformer leverages the attention mechanism [4] to enhance both model performance and interpretability. By focusing on capturing critical information in input sequence data, the attention mechanism enables the model to understand and process

the data more effectively. Models based on the Transformer architecture, such as ChatGPT and the BERT series [5], have since emerged. The Transformer and its variants have been proven to possess robust natural language processing capabilities, spanning tasks such as knowledge reasoning [6] and image recognition [7]. However, with repeated model iterations, the total number of training parameters has increased significantly. Data indicates that GPT-3's parameter count has reached a staggering 175 billion [8], necessitating substantial computational power and rendering model deployment extremely challenging. Consequently, large models are generally deployed by service providers on cloud servers. To use these services, users typically need to upload their data to the cloud for inference. This results in MLaaS providers holding increasing amounts of user information, often including personal data or commercial secrets. Once leaked, this could cause severe losses to users. Italy [9], for example, once announced a ban on the use of ChatGPT. Therefore, protecting users' private information remains an open problem that must be addressed to realize AI applications based on the Transformer architecture.

Existing research has proposed several solutions; however, these schemes typically incur high communication and computational costs or degrade accuracy due to the substitution of computational processes. For instance, Scheme [10] employs Secure Multi-Party Computation (MPC) [11] to design a secure Transformer inference system. Although this enhances privacy protection, it incurs high communication costs and relies on the non-collusion of multiple parties. Furthermore, to improve computational efficiency, Scheme [12] optimized non-linear activation functions that are unfriendly to MPC. While this reduces computational costs, it requires retraining the model, thereby increasing computational overhead and reducing model accuracy. Consequently, achieving efficient, high-precision, privacy-preserving Transformer inference remains a challenge. Specifically, the challenges are twofold: first, improving the efficiency of the massive matrix operations involved in the inference process while reducing communication costs; and second, ensuring the security of the operation process while maintaining the computational accuracy of non-linear activation functions.

To this end, this paper proposes a privacy-preserving Transformer inference scheme based on Fully Homomorphic Encryption (FHE). We design a secure and efficient matrix multiplication method and propose secure computation protocols for Softmax, LayerNorm, and Gelu. The specific contributions are as follows:

- We design a privacy-preserving Transformer inference framework based on CKKS FHE, optimized for computational and communication efficiency, which enables Transformer inference services while protecting user privacy.
- Based on multiplication over rings, we implement efficient matrix multiplication in the ciphertext domain. We optimize matrix partitioning parameters to accommodate different types (including ciphertext-plaintext and ciphertext-ciphertext matrix multiplication) and varying dimensions, thereby achieving optimal matrix partitioning.
- Leveraging parameter obfuscation and collaborative computing, we optimally design secure protocols for Softmax, LayerNorm, and Gelu to perform these computations in the ciphertext domain.
- We conduct text classification experiments on the IMDB and AGNEWS datasets. The results demonstrate that the proposed scheme completes the inference process within 3 s, maintains communication overhead below 1 GB, and achieves computational accuracy comparable to that of plaintext calculation.

2 Related Work

Privacy-preserving inference efforts based on Transformers primarily rely on Secure Multi-Party Computation (MPC) frameworks. MPC enables multiple parties to jointly compute a function, ensuring that each party obtains the correct result upon completion without disclosing their private information.

Major works on MPC-based privacy-preserving Transformer inference include the following: Chen et al. [13] developed a matrix multiplication protocol based on customized homomorphic encryption and an optimization scheme for third-order non-linear functions by representing matrix operations as polynomial multiplications. Pang et al. [14] constructed a privacy inference framework that supports efficient matrix operations via dynamic compression strategies and higher-precision nonlinear function optimization, although its performance relies on intensive ciphertext rotation and model retraining. Luo et al. [15] constructed MPC protocols for complex non-linear functions such as Gelu, LayerNorm, and Softmax using numerical methods. Li et al. [16] combined MPC with knowledge distillation to approximate non-linear functions with linear ones. Chen et al. [17] proposed reconstructing the nonlinear parts of the Transformer using approximate substitution strategies to significantly accelerate secure inference. Akimoto et al. [18] effectively improved the computational efficiency of a Transformer variant by replacing the traditional Sigmoid function with the ReLU activation function. Zeng et al. [10] evaluated various attention mechanisms using differentiable algorithms to compose a more efficient one, thereby improving the accuracy and efficiency of privacy-preserving inference by replacing the native attention in Transformers.

However, due to the large parameter size of Transformer models, which involve extensive matrix multiplications (ciphertext-plaintext and ciphertext-ciphertext) and nonlinear function computations (Softmax, Gelu, and LayerNorm), existing methods generally face the dual challenges of high computational complexity and substantial communication overhead. To perform secure operations within the Transformer architecture, it is necessary to design more efficient security protocols. Furthermore, directly approximating nonlinear functions in the Transformer architecture introduces errors, thereby affecting computational accuracy.

To address the aforementioned issues, this paper proposes integrating MPC concepts with Homomorphic Encryption (HE) to achieve privacy protection. HE, proposed by Rivest et al. [19], ensures that results from computations on encrypted data are consistent with those from direct computations on plaintext data, effectively guaranteeing data confidentiality. Early research primarily focused on partially homomorphic encryption schemes, such as additive [20] and multiplicative [19] homomorphisms. Gentry [21] proposed the first Fully Homomorphic Encryption (FHE) framework and utilized bootstrapping to control noise growth, achieving homomorphic decryption for circuits of arbitrary depth for the first time. However, traditional FHE schemes are computationally intensive, and efficiency bottlenecks in parallel processing remain unresolved [22]. Among FHE optimization algorithms, the CKKS (Cheon-Kim-Kim-Song) scheme [23] proposed by Cheon et al. and its variant [24] RNS-CKKS (Residue-Number-System CKKS) have been widely studied for their efficient handling of floating-point numbers. Considering the integration with the Transformer architecture, this paper adopts a variant of the CKKS scheme.

In recent years, numerous works combining FHE with machine learning have emerged. Gilad-Bachrach et al. [25] were the first to combine FHE with traditional Deep Neural Networks (DNN). Subsequent studies [26,27] aimed to improve the communication and computational performance of models within FHE. Xu et al. [28] proposed dubbed BLB by decomposing layers into fine-grained operators and fusing adjacent linear operators, reducing the need for HE/MPC conversions. Matrix multiplication is the primary computational process in Transformers; however, prior schemes have not accounted for the impact of matrix partitioning parameters on computational costs. This paper presents analytical models for ciphertext-plaintext and ciphertext-ciphertext matrix multiplications, respectively, to identify optimal matrix partitioning parameters for rapid matrix multiplication. Most previous schemes employ MPC for nonlinear computations, resulting in significant communication costs. To address this, for non-linear computations in Transformers, this paper employs parameter obfuscation to design secure and low-cost non-linear computation protocols based on FHE, specifically for Softmax, LayerNorm, and Gelu.

3 Preliminaries

3.1 Cheon-Kim-Kim-Song

Cheon-Kim-Kim-Song (CKKS) is a level FHE that supports a deep multiplication in the encrypted form. The input message and the encrypted result of CKKS are elements of the polynomial ring:

$$\mathbb{R}_Q = \mathbb{Z}_Q[X]/(X^N + 1), \tag{1}$$

where $Q = \prod_{i=0}^{L} q_i$ and any two q_is are different. Once the ciphertext level becomes too low, it is necessary to invoke the Bootstrapping operation to refresh it to a higher level, thereby enabling further computations.

CKKS supports Single Instruction Multiple Data (SIMD) [29], which allows for the encryption of a vector $z \in \mathbb{R}^{N/2}$ and the batch processing of these encrypted elements. This enables the scheme to process encrypted data in parallel, thereby accelerating operation speeds. To encrypt z in SIMD format, it first uses the encoding algorithm Ecd to encode z into a polynomial in $\mathbb{R}_Q$, and then uses the encryption algorithm Enc to encrypt the polynomial. In the following, we introduce the encryption process of CKKS:

- $Ecd\&Dcd$. Encodes a vector $z \in \mathbb{R}^{N/2}$ into the plaintext ring, or decodes a plaintext element $m \in \mathbb{R}_Q$ into a vector.

$$m = Ecd(z) \in \mathbb{R}_Q, z = Dcd(m) \in \mathbb{R}^{N/2}. \tag{2}$$

- $Enc\&Dec$. Encrypts the plaintext $m \in \mathbb{R}_Q$ into a ciphertext, or decrypts the ciphertext $\underline{m} \in \mathbb{R}_Q^2$ into a plaintext.

$$\underline{m} = Enc(m) \in \mathbb{R}_Q^2, m = Dec(\underline{m}) \in \mathbb{R}_Q. \tag{3}$$

- $Add1$. Adding the ciphertext $\underline{m_1}$ and the plaintext ring element m_2 yields a new ciphertext, the decryption of which is equivalent to the sum of the plaintext ring elements m_1 and m_2.

$$Dec(Add1(\underline{m_1}, m_2)) = m_1 + m_2 \in \mathbb{R}_Q. \tag{4}$$

- $Add2$. Adding the ciphertext $\underline{m_1}$ and the ciphertext $\underline{m_2}$ yields a new ciphertext, the decryption of which is equivalent to the sum of the plaintext ring elements m_1 and m_2.

$$Dec(Add2(\underline{m_1}, \underline{m_2})) = m_1 + m_2 \in \mathbb{R}_Q \tag{5}$$

- $Mul1$. Multiplying the ciphertext $\underline{m_1}$ by the plaintext ring element m_2 yields a new ciphertext, the decryption of which is equivalent to the multiplication of the plaintext ring elements m_1 and m_2.

$$Dec(Mul1(\underline{m_1}, m_2)) = m_1 \cdot m_2 \in \mathbb{R}_Q \tag{6}$$

- $Mul2$. Multiplying the ciphertext $\underline{m_1}$ by the ciphertext $\underline{m_2}$ yields a new ciphertext, the decryption of which is equivalent to the multiplication of the plaintext ring elements m_1 and m_2.

$$Dec(Mul2(\underline{m_1}, \underline{m_2})) = m_1 \cdot m_2 \in \mathbb{R}_Q \tag{7}$$

- Rot. Rotating the ciphertext $\underline{m}$ corresponding to the vector z by i positions yields a new ciphertext, which is equivalent to the ciphertext corresponding to the new vector z' obtained by rotating the vector z by i positions.

$$Rot(\underline{m}, i) \Leftrightarrow Enc(Ecd(z)) \in \mathbb{R}_Q. \tag{8}$$

3.2 Transformer Framework

The Transformer is a representative neural network architecture introduced by Google engineers for machine translation. It is widely used because its architecture, with a self-attention mechanism, improves inference accuracy on long-sequence data. For instance, in translation tasks, compared to Recurrent Neural Networks (RNNs), the Transformer requires less time to achieve the same accuracy on identical sequence data. The Transformer is composed of a stack of multiple encoders and decoders; each encoder primarily consists of two components: the Self-Attention Mechanism and the Feed-Forward Neural Network, as illustrated in the Transformer framework diagram in Fig. 1. The decoder's architecture is fundamentally similar to that of the encoder and will not be elaborated further here.

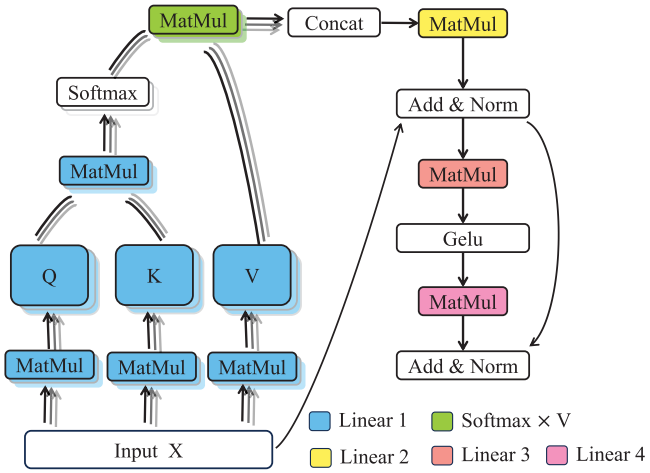

Figure 1: The framework of Transformer.

As the core component of the Transformer, it allows the model to simultaneously attend to information from all other positions in the sequence while processing each specific position, thereby dynamically computing the degree of association between positions. Through this mechanism, the model can better capture contextual semantic information within sequence elements, overcoming the limitations imposed by the vanishing gradient problem in Recurrent Neural Networks (RNNs) when processing long sequences. Furthermore, Multi-Head Attention, which is directly extended from the self-attention mechanism, enables the Transformer to process data in parallel; each attention head can learn information from different subspaces, significantly enhancing computational efficiency. The specific process is as follows: the input sequence undergoes word-embedding and positional-encoding transformations to yield the matrix X. Subsequently, X is multiplied by the corresponding weight matrices W_Q, W_K, and W_V to obtain the corresponding Query (Q), Key (K), and Value (V) matrices. Then, the attention scores are calculated via the Softmax function:

$$Attention(Q, K, V) = Softmax(\frac{QK^T}{\sqrt{d_k}})V, \tag{9}$$

where d_k represents the key vector dimension. The Multi-Head Attention mechanism is obtained by computing multiple attention functions in parallel:

$$\begin{aligned} &MultiHead(Q, K, V) = [head_1, head_2, \cdots, head_n], \\ &head_i = Attention(QW_Q^i, KW_K^i, VW_V^i). \end{aligned} \tag{10}$$

Here, W_O is a linear projection matrix. For i ranging from 1 to H, upon completion of the computations, the multiple resulting matrices are concatenated to obtain a new matrix, referred to as the Multi-Head Attention (MHA) matrix, which contains information from different attention heads.

$$MHA = Contact(head_1, head_2, \cdots, head_H)W_O. \tag{11}$$

Residual Connection and Layer Normalization (Add & Norm): The matrix resulting from the processing of sequence data by the attention layer is first input into the residual connection and layer normalization layer. Specifically, this layer consists of two components: the residual connection and layer normalization:

$$LayerNorm(X + sublayer(X)). \tag{12}$$

As shown in the Transformer framework diagram in Fig. 1, a sublayer can be either an Attention layer or a Feed-Forward layer. Where:

$$LayerNorm(x) = \frac{x - E(x)}{\sqrt{Var(x) + \varepsilon}} * \gamma + \beta. \tag{13}$$

In Eq. (13), $E(x)$ represents the mean, and $Var(x)$ represents the variance. The notation ε is introduced to prevent division by zero, while γ and β are two learnable parameters. The residual connection helps mitigate network degradation during training, reducing training errors and improving accuracy. Meanwhile, the objective of normalization is to enhance training stability while simultaneously accelerating training speed.

Feed-Forward Neural Network (FFNN) Layer: This is a fully connected neural network that incorporates a non-linear activation function. Specifically, a linear transformation is first performed from a low dimension to a high dimension, followed by the introduction of non-linearity via the Gelu activation function, and finally, another linear transformation maps the data back to the low dimension. This structure endows the model with non-linearity. Since the preceding matrix multiplications and encryption operations involved only linear transformations, the model's expressive capacity was significantly limited. However, the non-linear Gelu activation in this layer applies a nonlinearity to the input, thereby enhancing the model's expressive power and enabling it to address complex nonlinear problems. The formula is as follows:

$$FFN(x) = [Gelu(xW_1 + b_1)]W_2 + b_2. \tag{14}$$

Here, W_1 and W_2 are two weight matrices, and b_1 and b_2 are the corresponding biases, representing the parameters for the first and second linear transformations, respectively. As indicated by the model architecture, the output of the first linear transformation serves as the input to the non-linear activation function. The Gelu formula is defined as follows:

$$Gelu(x) = xP(X \leq x). \tag{15}$$

After the input undergoes Gelu non-linear activation, it proceeds to the computations of the subsequent layers.

4 Privacy-Preserving Transformer

This paper proposes a privacy-preserving Transformer inference framework based on Fully Homomorphic Encryption (FHE), as illustrated in Fig. 2. Since the Transformer involves extensive matrix

multiplications, directly applying FHE to the Transformer model results in a drastic decline in inference performance [12]. The primary challenges faced are as follows:

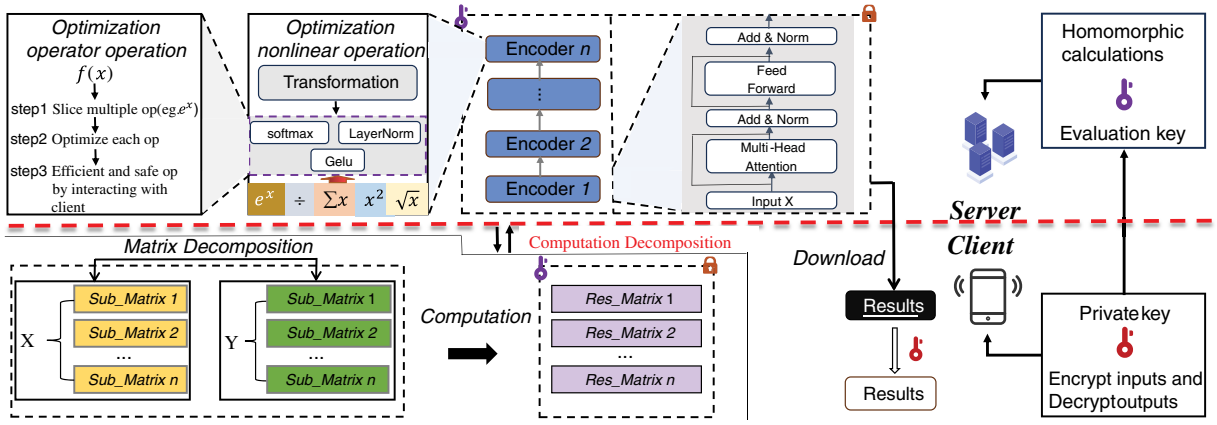

Figure 2: An overview of the privacy-preserving Transformer.

Challenge 1: Matrix multiplication is the primary computational process in Transformers and typically involves large dimensions, resulting in an excessive volume of data. The limited size of plaintext slots makes it impossible to accommodate all input data and weight matrices. Consequently, it is necessary to decompose a single large matrix multiplication into multiple smaller ones. Different partitioning methods yield varying communication costs and computational efficiencies. Balancing communication costs and computational efficiency during matrix partitioning remains a challenge.

Challenge 2: Due to the linear computation constraints of FHE, complex non-linear computations cannot be performed directly. Previous works primarily addressed this via polynomial approximation or Garbled Circuits (GC); however, the former results in a loss of model inference accuracy, while the latter incurs substantial communication costs. Consequently, achieving low-cost non-linear computation while guaranteeing both computational accuracy and security remains a significant challenge.

Next, we provide an overview of the privacy-preserving Transformer inference framework proposed in this paper.

4.1 Privacy-Preserving Transformer Inference Framework

As shown in Fig. 2, it illustrates the inference framework of the privacy-preserving inference scheme. This framework comprises two entities: the client and the server.

- **Client:** Holds the data requiring inference. The user uploads the data to the Transformer model server for secure inference, then retrieves the inference results.
- **Server:** The server possesses a pre-trained Transformer model. It receives the client's uploaded data, performs secure Transformer inference, and returns the results to the client.

The client (user) holds private data requiring inference, encrypts the message, and uploads it to the server. The server possesses a pre-trained Transformer model. Upon receiving data from the client, the server invokes the pre-trained model to perform privacy-preserving inference on encrypted data. This paper divides the inference process into two parts: matrix multiplication and non-linear function processing. For the matrix multiplication part, we employ computational decomposition techniques to reduce communication costs and enhance computational efficiency; relevant details will be presented in Section 5. For the nonlinear function part, we design secure operators based on parameter obfuscation via addition

and multiplication, and construct an efficient nonlinear collaborative inference process using these secure operators; details will be presented in Section 6. Finally, the server returns the obtained results to the client, who decrypts them using the private key. The final decrypted results are consistent with those obtained using plaintext inference. It is worth noting that the entire inference process in this paper is conducted within an encrypted environment.

4.2 Security Threat

The threat model adopted in this paper is the semi-honest threat model. This assumption, widely employed in previous schemes [10–12,14–19], is well-suited for practical scenarios and specifically encompasses two potential hypotheses: First, the model holder (server) may continuously harvest data uploaded by the user (client), which may contain the user's private information. Second, the client may attempt to extract model parameters through repeated inference. As these parameters have high commercial value to the model holder, it is essential to ensure they remain protected from theft throughout the inference process. This paper guarantees the security of both user data and model weight parameters throughout the entire inference process via the semantic security of Fully Homomorphic Encryption (FHE). Specifically, the security of this paper relies on the hardness of the Ring Learning With Errors (RLWE) problem, a classical hard problem in post-quantum cryptography widely recognized as unbreakable.

4.3 Design Goal

The objectives of the Transformer-based user data privacy protection proposed in this paper are as follows:

- **Privacy:** Throughout the entire inference process, the security of the data uploaded by the user (client) and the model weight parameters must be guaranteed. Specifically, both must be protected against leakage throughout the inference process to safeguard data privacy.
- **Integrity:** No one shall be permitted to tamper with the inference results.
- **Real-Time Performance:** The server must guarantee a response to the user (client) data uploaded within a specific timeframe—specifically, the inference results—because, in certain scenarios, users (clients) cannot tolerate prolonged delays in receiving the inference response.

5 Optimal Partitioning Method for Matrix Multiplication

The Transformer architecture involves extensive matrix multiplications; consequently, achieving rapid matrix multiplication within the ciphertext domain has garnered widespread attention from the research community. The matrix-vector multiplication method proposed by Juvekar et al. [30] requires numerous ciphertext rotations, which incurs substantial computational cost. To address this, Huang et al. [31] proposed embedding data directly in the ring domain, thereby enabling vector-matrix multiplication via underlying ring multiplication. Chen et al. [13] further enhanced this scheme by directly embedding matrices into the ring domain to achieve fast homomorphic matrix-matrix multiplication. This approach requires embedding matrices X and Y into the ring, respectively, as follows:

$$\begin{aligned} &\hat{x} = \pi_{in}^{i}(v), \quad where \quad \hat{x}[i \cdot n \cdot k + (n-1) - k] = X[i,k], \\ &\hat{y} = \pi_{in}^{w}(W), \quad where \quad \hat{y}[j \cdot n + k] = Y[k,j]. \end{aligned} \tag{16}$$

Multiplying the two ring elements $\hat{x}$ and $\hat{y}$ yields a new ring element $\hat{z}$, i.e.:

$$\hat{z}[i \cdot n \cdot k + (j+1) \cdot n - 1] = \sum_{k=0}^{n_i - 1} X[i,k] \cdot Y[k,j]. \tag{17}$$

Assume two matrices $X \in \mathbb{R}^{A \times B}$ and $Y \in \mathbb{R}^{B \times C}$ are multiplied to yield the matrix $Z = XY \in \mathbb{R}^{A \times C}$. It is necessary to select parameters a, b, and c to partition the matrices. This is required because the matrices are too large for the degree of the ring polynomial to accommodate all elements simultaneously, i.e., $abc \leq N < ABC$. However, different partitioning parameters a, b, and c result in varying computational and communication costs. To address this, this paper analyzes in detail the impact of partitioning parameters on computational and communication costs for both plaintext-ciphertext and ciphertext-ciphertext matrix multiplications, and constructs a model for the optimal partitioning parameter problem.

5.1 Ciphertext-Plaintext Matrix Multiplication

First, matrix X needs to be embedded into $\frac{A}{a} \cdot \frac{B}{b}$ ciphertexts, and the resulting matrix Z is stored in $\frac{A}{a} \cdot \frac{C}{c}$ ciphertexts. The required number of multiplications is $\frac{A}{a} \cdot \frac{B}{b} \cdot \frac{C}{c}$, and the number of ciphertexts for communication is $\frac{A}{a}(\frac{B}{b} + \frac{C}{c}) = \frac{ABc+ACb}{abc}$. We aim to minimize the number of multiplications by setting $abc = N$, which yields a minimum multiplication count of $\frac{ABC}{N}$ and a ciphertext count of $\frac{ABc+ACb}{N}$. Assuming the communication cost of a single input ciphertext is x, and the communication cost of a single ciphertext after multiplication is y, the overall communication cost is formulated as:

$$\frac{A}{a}\left(\frac{B}{b}x + \frac{C}{c}y\right) = \frac{ABxc + ACyb}{N}. \tag{18}$$

Here, A, B, C, a, b, c, x, and y are all constants. Similarly, assuming that the encryption and decryption times for a single ciphertext are x' and y', respectively, the overall computation time is formulated as:

$$\frac{A}{a}\left(\frac{B}{b}x' + \frac{C}{c}y'\right) = \frac{ABx'c + ACy'b}{N}. \tag{19}$$

By integrating the above equations, we formulate a constrained optimization problem that considers both computation time and communication costs:

$$\begin{aligned} &\min_{a,c} \quad ac + \beta b \\ &s.t. \quad \alpha = \frac{AB}{N}[rx + (1-r)x'] \\ &\beta = \frac{AC}{N}[ry + (1-r)y'] \\ &abc = N. \end{aligned} \tag{20}$$

Here, r is a constant representing the degree of emphasis on computational efficiency relative to communication costs. We clarify below how the optimization framework extends beyond the idealized assumption $abc = N$, and how padding, slot underutilization, and non-square matrices are handled in practice. The constraint $abc = N$ is used in the analysis to capture the ideal case of full slot utilization, in which all CKKS slots are used. This formulation simplifies the analytical objective function and provides clear intuition for the trade-offs among partitioning parameters. In practical deployments, matrix dimensions may not perfectly align with CKKS slot counts, resulting in padding or partial slot usage. In these cases,

the constraint is relaxed to: $abc \leq N$, where unused slots are padded with zeros. We have: (i) Padding with zeros does not affect correctness, since zero slots do not contribute to homomorphic products; (ii) Slot underutilization only affects constant factors in efficiency, not the asymptotic behavior captured by the objective function. In the revised implementation, we explicitly account for under-utilization by introducing a utilization factor $abc \leq N$, and the effective cost is scaled accordingly when comparing candidate partitions.

5.2 Ciphertext-Ciphertext Matrix Multiplication

First, matrices X and Y need to be embedded into $\frac{A}{a} \cdot \frac{B}{b}$ and $\frac{B}{b} \cdot \frac{C}{c}$ ciphertexts, respectively, and the resulting matrix Z is stored in $\frac{A}{a} \cdot \frac{C}{c}$ ciphertexts. The required number of multiplications is $\frac{A}{a} \cdot \frac{B}{b} \cdot \frac{C}{c}$. The number of ciphertexts for communication is $\frac{A}{a} \cdot \frac{B}{b} + \frac{B}{b} \cdot \frac{C}{c} + \frac{A}{a} \cdot \frac{C}{c}$. Similarly, this paper minimizes the number of multiplications by setting $abc = N$, yielding a minimum multiplication count of ABC/N and a ciphertext count of $\frac{ABc+ACb+BCb}{N}$. Assuming the communication cost of a single input ciphertext is x and the communication cost of a single ciphertext after multiplication is y, the overall communication cost is formulated as:

$$\frac{A}{a} \cdot \frac{B}{b} x + \frac{B}{b} \cdot \frac{C}{c} x + \frac{A}{a} \cdot \frac{C}{c} y = \frac{ABxc + ACyb + BCxa}{N} \tag{21}$$

Here, A, B, C, a, b, c, x, y, and N are all constants. Similarly, assuming that the encryption and decryption times for a single ciphertext are x' and y', respectively, the overall computation time is formulated as:

$$\frac{A}{a} \cdot \frac{B}{b} x' + \frac{B}{b} \cdot \frac{C}{c} x' + \frac{A}{a} \cdot \frac{C}{c} y' = \frac{ABx'c + ACy'b + BCx'a}{N} \tag{22}$$

By integrating the above equations, we can formulate an optimization problem that simultaneously considers both computation time and communication costs:

$$\begin{aligned} &\min_{a,b,c} \quad \alpha c + \beta a + \gamma b \\ &s.t. \quad \alpha = \frac{AB}{N}[rx + (1-r)x'], \\ &\beta = \frac{BC}{N}[rx + (1-r)x'], \\ &\gamma = \frac{AC}{N}[ry + (1-r)y'], \\ &abc = N. \end{aligned} \tag{23}$$

6 A Secure Protocol for the Nonlinear Activation Functions

Here, we provide secure protocols for the nonlinear activation functions. Our approach does not rely on complex or task-specific preprocessing.

6.1 Secure Softmax

The softmax is described by:

$$softmax(X)_i = \frac{e^{x_i}}{\sum_{j=0}^{n-1} e^{x_j}}. \tag{24}$$

It can be observed that the required computations involve exponentiation, division, and summation. Among these, exponentiation and division cannot be directly performed in CKKS. Previous schemes [15,17]

used approximate computations to simplify calculations at the cost of computational accuracy; however, in most scenarios, computational accuracy is also important to users. To this end, by leveraging client interaction, this paper proposes a secure Softmax function protocol without loss of precision.

6.1.1 Exponential Calculation

This paper implements this using parameter obfuscation and collaborative computing; the obfuscated parameter is removed during computation, ensuring computational accuracy is unaffected. First, the Server adds a random number $-r$ to the computed ciphertext result ct, and the Client decrypts it to obtain the result $x - r$. Subsequently, the Client and Server independently compute $e^{x-r} = e^x e^{-r}$ and e^r, respectively. Finally, the Client encrypts e^{x-r} to generate its ciphertext and sends it to the Server. The Server then performs homomorphic multiplication to recover the ciphertext of e^x.

6.1.2 Summation Calculation

Previous summation computations required extensive homomorphic rotation operations [14], which incurred substantial computational costs and noise accumulation, thereby degrading computational accuracy. This paper transforms the summation operation into the form of matrix multiplication, i.e.:

$$X \cdot W = \begin{bmatrix} e^{x_{0,0}} & \cdots & e^{x_{0,n-1}} \\ \vdots & \cdots & \vdots \\ e^{x_{m-1,0}} & \cdots & e^{x_{m-1,n-1}} \end{bmatrix} \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \cdots & \vdots \\ 1 & \cdots & 1 \end{bmatrix} = \begin{bmatrix} \sum_{j=0}^{n-1} e^{x_{0,j}} & \cdots & \sum_{j=0}^{n-1} e^{x_{0,j}} \\ \vdots & \cdots & \vdots \\ \sum_{j=0}^{n-1} e^{x_{m-1,j}} & \cdots & \sum_{j=0}^{n-1} e^{x_{m-1,j}} \end{bmatrix} \tag{25}$$

This method eliminates the need for homomorphic rotation operations; for specific calculation methods, please refer to Section 4.

6.1.3 Division Calculation

Previous schemes [32] employed Goldschmidt division for approximate division, which incurs high computational costs and introduces errors. To address these issues, this paper proposes a solution based on parameter obfuscation and collaborative computing. First, the Server multiplies the computed ciphertext result ct by a random number r, and the Client decrypts this to obtain the result xr. Next, the Client performs plaintext computation to obtain $\frac{1}{xr}$. Finally, the Client encrypts $\frac{1}{xr}$ to generate its ciphertext and sends it to the Server. The Server then performs homomorphic multiplication to recover the ciphertext of $\frac{1}{x}$. The secure Softmax protocol is as follows:

- First, the Client decrypts ct to obtain $x_i - r_i$, performs the exponential computation to yield $e^{x_i - r_i} = e^{x_i} e^{-r_i}$, and then encrypts and uploads $e^{x_i} e^{-r_i}$.
- Next, the Server computes e^{r_i} and uses it to recover the ciphertext of e^{x_i}. We observe that the ciphertext recovery and summation computation can be performed simultaneously; specifically, by transforming W into: $\begin{bmatrix} e^{r_{0,0}} & \cdots & e^{r_{m-1,0}} \\ \vdots & \cdots & \vdots \\ e^{r_{0,n-1}} & \cdots & e^{r_{m-1,n-1}} \end{bmatrix}$. In this way, both summation and ciphertext recovery can be performed in a single matrix multiplication.
- Subsequently, the Server applies a multiplicative obfuscation parameter l_i to the obtained result, i.e., $l_i \sum_{j=0}^{n-1} e^{x_{i,j}}$. Similarly, the application of the obfuscation parameter can be performed simultaneously with the summation computation, specifically by transforming W into: $\begin{bmatrix} l_0 e^{r_{0,0}} & \cdots & l_{m-1} e^{r_{m-1,0}} \\ \vdots & \cdots & \vdots \\ l_0 e^{r_{0,n-1}} & \cdots & l_{m-1} e^{r_{m-1,n-1}} \end{bmatrix}$.

- Subsequently, the Server sends the calculation result to the Client. The Client decrypts it to obtain $l_i \sum_{j=0}^{n-1} e^{x_{i,j}}$ and computes the reciprocal to yield $\frac{1}{l_i \sum_{j=0}^{n-1} e^{x_{i,j}}}$. The Client then encrypts $e^{x_i} e^{-r_i}$ and $\frac{1}{l_i \sum_{j=0}^{n-1} e^{x_{i,j}}}$ to generate ciphertexts ct_0 and ct_1, respectively, and sends them to the Server.
- Finally, the Server multiplies ciphertexts ct_0 and ct_1 and multiplies the resulting product by $l_i e^{r_i}$ to obtain the result $Softmax(X)$.

6.2 Secure LayerNorm Protocol

LayerNorm is a method for normalizing all features of a single sample within a neural network. In a neural network layer with multiple neurons, LayerNorm computes the mean and variance of each sample's feature vector, then normalizes each element of the vector. Its formula is expressed as follows:

$$LayerNorm(x)_i = r\frac{x_i - u}{\sqrt{\sum_{j=0}^{n-1}(x_j - u)^2}} + \beta = r\frac{z_i}{\sqrt{\sum_{j=0}^{n-1} z_j^2}} + \beta. \tag{26}$$

Here, $u = \frac{1}{n}\sum_{i=0}^{n-1} x_i$. It can be observed that the required computations involve summation, squaring, square root, and division. However, square root and division operations cannot be performed directly in CKKS. Scheme [14] performs these computations via MPC, which increases communication costs, whereas Scheme [15] uses approximate computations, sacrificing accuracy to avoid these operations. In contrast, this paper achieves lower communication overhead while maintaining computational accuracy through client interaction.

6.2.1 Summation Calculation

The summation computation here can be implemented directly using collaborative computing. The Server adds a random number $-r_i$ to the computed ciphertext result ct, and the Client decrypts it to obtain the result $x_i - r_i$. The Client and Server independently perform plaintext computations to obtain $\sum_{i=0}^{n-1}(x_i - r_i) = nu - \sum_{i-0}^{n-1} r_i$ and $\sum_{i=0}^{n-1} r_i$, respectively. The Server recovers the value by adding $\sum_{i=0}^{n-1} r_i$ during the computation, thereby realizing secure collaborative summation.

6.2.2 Square Calculation

This squaring computation requires only the homomorphic multiplication of the recovered ciphertext of x_i.

6.2.3 Square Root Calculation

For the square root computation, the data x can be multiplied by a random number r, i.e., $\sqrt{x} = \frac{\sqrt{x \cdot r}}{\sqrt{r}}$. Specifically, the Server applies a multiplicative obfuscation parameter r to the data x's ciphertext. Subsequently, the Client and Server independently compute $\sqrt{x \cdot r}$ and $\frac{1}{\sqrt{r}}$, respectively. Finally, the Server recovers the value by multiplying by $\frac{1}{\sqrt{r}}$, thereby realizing a secure square root computation.

6.2.4 Division Calculation

This division computation is identical to the division computation in Section 5.1. In summary, the secure LayerNorm protocol is as follows:

- First, the Client decrypts ct to obtain $x_i - r_i$. The Client and Server then independently perform summation and division by n to obtain $u - \frac{1}{n}\sum_{i=0}^{n-1} r_i$ and $\frac{1}{n}\sum_{i=0}^{n-1} r_i$, respectively. The Client proceeds

with local computations to yield $x_i - u + \frac{1}{n}\sum_{i=0}^{n-1} r_i = z_i + \frac{1}{n}\sum_{i=0}^{n-1} r_i$. Finally, the Client encrypts $z_i + \frac{1}{n}\sum_{i=0}^{n-1} r_i$ and sends the resulting ciphertext.

- Next, the Server homomorphically adds $-\frac{1}{n}\sum_{i=0}^{n-1} r_i$ to the ciphertext of $z_i + \frac{1}{n}\sum_{i=0}^{n-1} r_i$ to obtain the ciphertext of z_i. Thus, the Server obtains the ciphertext of z_i^2 via homomorphic multiplication. The Server then adds an additive obfuscation parameter $-r_i$ via homomorphic addition to obtain the ciphertext of $z_i^2 - r_i$ and sends it to the Client.
- The Client decrypts the data to obtain $z_i^2 - r_i$ and performs a local summation operation to yield $\sum_{i=0}^{n-1} z_i^2 - \sum_{i=0}^{n-1} r_i$. The Client then encrypts $\sum_{i=0}^{n-1} z_i^2 - \sum_{i=0}^{n-1} r_i$ to obtain its ciphertext and sends it to the Server.
- The Server recovers the ciphertext of $\sum_{i=0}^{n-1} z_i^2$ by adding $\sum_{i=0}^{n-1} r_i$ via homomorphic addition. Subsequently, it applies the multiplicative obfuscation parameter l through homomorphic multiplication to obtain the ciphertext of $l\sum_{i=0}^{n-1} z_i^2$ and sends it to the Client.
- The Client decrypts the data to obtain $l\sum_{i=0}^{n-1} z_i^2$ and performs a square root computation to yield $\sqrt{l}\sqrt{\sum_{i=0}^{n-1} z_i^2}$. The Client then proceeds to compute the reciprocal to obtain $\frac{1}{\sqrt{l}} \cdot \frac{1}{\sqrt{\sum_{i=0}^{n-1} z_i^2}}$, encrypts the result, and sends it to the Server.
- The Server multiplies the ciphertext of $\frac{1}{\sqrt{l}} \cdot \frac{1}{\sqrt{\sum_{i=0}^{n-1} z_i^2}}$ and subsequently multiplies by $\sqrt{l}$ to realize the LayerNorm computation.

6.3 Secure Gelu Protocol

The Gelu (Gaussian Error Linear Unit) [33] function enhances the model's non-linear representational capacity and accelerates model convergence. The computational form of this activation function is expressed as:

$$Gelu(x) = xP(X \le x) = x\int_{x}^{-\infty} \frac{e^{\frac{-(X-u)^2}{2\sigma^2}}}{\sqrt{2\pi}\sigma} dX. \tag{27}$$

Alternatively, an approximation can be employed, with the formula expressed as:

$$Gelu(x) \approx 0.5x\left(1 + tanh\left(\sqrt{\frac{2}{\pi}}(x + 0.044715x^3)\right)\right) = 0.5x\left(1 + tanh(y)\right) = 0.5x\left(1 + \frac{e^{2y}-1}{e^{2y}+1}\right). \tag{28}$$

Here, $y = \sqrt{\frac{2}{\pi}}(x + 0.044715x^3)$, and the required computations involve polynomial evaluation, exponentiation, and division. Previous schemes [14] used Oblivious Transfer for computation, thereby guaranteeing computational accuracy but increasing communication costs. By leveraging homomorphic encryption and client-server interaction, this paper ensures computational accuracy while reducing communication overhead.

6.3.1 Polynomial Calculation

Polynomial computation can be directly implemented via homomorphic multiplication.

6.3.2 Exponential Calculation

The exponentiation calculation method is identical to that in Section 5.1.

6.3.3 Division Calculation

The division calculation method is identical to that in Section 5.1.

In summary, the secure Gelu protocol is as follows:

- First, the Client decrypts the data to obtain $x - r_1$, encodes it into the Slot domain, encrypts it, and sends it to the Server.
- Next, the Server recovers the ciphertext of x via homomorphic addition and obtains the ciphertext of x^3 via homomorphic multiplication. The Server then obtains the ciphertext of $x^3 - r_2$ via homomorphic encryption and parameter obfuscation, and sends it to the Client.
- The Client and Server independently perform computations to obtain $\sqrt{\frac{2}{\pi}}(x + 0.044715x^3) - \sqrt{\frac{2}{\pi}}(r_1 + 0.044715r_2) = y - R$ and $\sqrt{\frac{2}{\pi}}(r_1 + 0.044715r_2) = R$, respectively. Subsequently, the Client and Server proceed to independently compute the exponentials, yielding $e^{2y-2R} = e^{2y}e^{-2R}$ and e^{2R}, respectively. Finally, the Client encrypts $e^{2y}e^{-2R}$ and sends it to the Server.
- The Server recovers the ciphertext of e^{2y} via homomorphic multiplication and obtains the ciphertexts of $e^{2y} + 1$ and $e^{2y} - 1$ via homomorphic operations. Subsequently, the Server applies a multiplicative obfuscation parameter via homomorphic multiplication to obtain $l(e^{2y} + 1)$ and sends it to the Client.
- The Client decrypts the data to obtain $l(e^{2y} + 1)$, computes $\frac{1}{l} \cdot \frac{1}{e^{2y}+1}$, encrypts the result, and sends it to the Server.
- Finally, the Server obtains the ciphertext of $\frac{1}{l} \cdot \frac{e^{2y}-1}{e^{2y}+1}$ via homomorphic multiplication and multiplies it by l to yield $tanh(y)$, subsequently obtaining the ciphertext of $1 + tanh(y)$ via homomorphic addition. The Server then proceeds to multiply the ciphertext of $1 + tanh(y)$ by the ciphertext of x and the constant 0.5 to obtain the ciphertext of $Gelu(x)$.

6.4 Security Analysis

6.4.1 Information Leakage during Interactions

All repeated interactions in our system arise from non-linear protocols (Gelu, Softmax, LayerNorm), which combine CKKS-based FHE with collaborative computation. We analyze leakage across rounds as follows:

- All messages exchanged during interaction rounds are CKKS ciphertexts encrypted under the client's public key. Under the IND-CPA security of CKKS, each ciphertext is computationally indistinguishable from random to the server, even across multiple rounds. Consequently, repeated exposure to ciphertexts does not accumulate information leakage about the underlying plaintext values.
- The number of interaction rounds, message sizes, and execution order are fixed and input-independent for a given model configuration. Therefore, no control-flow, branching, or early-termination behavior leaks information about intermediate activations.
- In collaborative steps where the client performs plaintext-domain operations (e.g., normalization or polynomial evaluation), the client observes only values that are either: (i) Derived from its own secret key and inputs, or (ii) Explicit protocol outputs that are already implied by the final inference result. No additional side information is revealed to the server during these steps.
- Each interaction round is secure under the CKKS encryption scheme and the semi-honest model. By the standard sequential composition theorem, the overall protocol remains secure, and the cumulative leakage is limited to the union of per-round leakage, which in our case consists solely of public parameters and input-independent metadata.

6.4.2 Malicious Setting

Our system is analyzed under the semi-honest (honest-but-curious) model, in which all parties are assumed to follow the protocol specification correctly, while possibly attempting to infer additional information from observed transcripts. Under this model, the client performs plaintext exponentiation, division, and square-root operations correctly and as specified.

Even if a client were to deviate from the specified computation, such behavior would: (i) Not reveal additional information about the server's model parameters, since all values returned to the server remain encrypted under the client's public key; (ii) Affect correctness, not privacy: an incorrect client computation can only bias its own inference result, not extract extra information about the model.

6.4.3 Security of Random Injections

Exposing masked sums of exponentials warrants careful analysis. We clarify below why this does not leak relative magnitude or sparsity information under our threat model.

In the relevant non-linear protocols (e.g., Softmax), the client receives values of the form:

$$\tilde{S} = S + r, \tag{29}$$

where $S = \sum_i e^{x_i}$ is the true sum of exponentials and r is a fresh, randomly sampled mask drawn from a sufficiently large domain. The client never observes S itself, nor any unmasked e^{x_i}.

Because the mask r is additive, independent, and information-theoretically unknown to the client: (i) The distribution of $\tilde{S}$ is statistically independent of the magnitude of S; (ii) Given a single masked observation, the client cannot distinguish whether S is large or small relative to other queries; (iii) Across multiple queries, fresh randomness is used, preventing cross-query comparison or normalization. As a result, relative magnitude information is computationally hidden, even if the client chooses inputs adaptively.

Sparsity information (e.g., whether only a few terms dominate the sum) would require access to either: Individual exponentials e^{x_i}, or ratios between partial sums. In our protocol: (i) The client observes only a single aggregated value per interaction round; (ii) No partial sums or per-token contributions are ever revealed; (iii) The masking prevents inference based on absolute value or scale. Therefore, the client cannot infer whether S is composed of many small terms or a few dominant ones.

6.4.4 Security of Repeated Interactive Queries

The repeated queries do not leak model weights. The reasons are as follows:

- Model weights are never revealed in plaintext: All server-side computations involving model weights are performed either on plaintext weights combined with encrypted activations, or entirely within the encrypted domain. At no point does the client receive plaintext or partially decrypted model parameters.
- Ciphertext indistinguishability across queries: All messages returned to the client are CKKS ciphertexts encrypted under the client's public key. Under the IND-CPA security of CKKS, ciphertexts corresponding to different masked linear combinations of model weights are computationally indistinguishable from random ciphertexts, even under multiple adaptive queries.
- No linear equations exposed to the client: The client never observes multiple plaintext outputs corresponding to linearly independent combinations of the same intermediate values. Thus, adaptive querying does not allow the client to construct solvable systems of equations to recover model weights.

- Interaction structure is input-independent: The number of interaction rounds, message sizes, and execution order are fixed and independent of the client's inputs. This prevents adaptive behavior from influencing protocol flow or extracting side information through control-flow leakage.

7 Experimental Evaluation

7.1 Experimental Setting

7.1.1 Dataset

The evaluation datasets are provided as follows.

- **IMDB:** This is a sentiment analysis dataset utilized in the fields of Natural Language Processing and Machine Learning. It contains 50,000 highly polarized reviews, with 25,000 allocated to the training set and 25,000 to the test set. The labels are categorized as positive and negative.
- **AGNEWS:** This dataset encompasses news from four major categories: World, Sports, Business, and Science/Technology. It aggregates the title and description fields of articles from the AG corpus corresponding to these four categories. The dataset contains 30,000 training samples and 1900 test samples per category.

7.1.2 Training Parameters

We train the Transformer in plaintext on the IMDB and AGNEWS datasets using the PyTorch [34] library, an open-source machine learning framework. Plaintext training is performed across varying numbers of encoders and sequence lengths. The specific model training framework is detailed in Table 1, Transformer Model Framework Information.

Table 1: The parameters of the Transformer under different datasets.

Dataset	Length of Sequence	Number of Encoders	Dimension of the Encoder
IMDB	128	2	128
	128	3	128
	128	4	128
AGNEWS	64	2	128
	64	3	128
	64	4	128

7.1.3 Ciphertext and Other Settings

In this paper, we conduct ciphertext inference tests using the Lattigo library [35], adopting the CKKS homomorphic encryption scheme. All experiments are executed on a computer equipped with an AMD R7 5700G CPU and 32GB of RAM, employing 8-thread parallel computing. We select two sets of ciphertext parameters to accommodate different computational processes, as detailed in Table 2.

The two CKKS parameter sets listed in Table 2 are selected to match the multiplicative-depth requirements of different computational components in Transformer inference, while maintaining sufficient numerical precision and minimizing computational overhead. Transformer inference consists of two distinct classes of operations: (i) Linear operations (matrix multiplications, additions), which require low multiplicative depth but dominate runtime; (ii) Non-linear operations (Gelu, Softmax, LayerNorm), which involve multiple ciphertext–ciphertext multiplications and therefore require greater noise budget. Using two

specialized parameter sets allows us to decouple efficiency and depth requirements: (i) Parms1 minimizes latency and communication cost for the dominant linear computations; (ii) Parms2 ensures correctness and precision for noise-sensitive non-linear functions without resorting to approximation or bootstrapping.

Table 2: Ciphertext parameters used in this paper.

Ciphertext Parameters	$logN$	$\{logq_i\}$	Scale
Parms1	12	68, 40	2^{40}
Parms2	12	58, 40, 40, 40, 40	2^{40}

7.2 Test of Basic Operations

7.2.1 Test of the Matrix Multiplication

To analyze the impact of matrix dimensions on computational efficiency and communication costs, this paper conducts matrix-multiplication tests with varying matrix sizes. Given that matrix multiplication requires only a single multiplication operation and consumes a multiplicative depth of 1, Parms1 is selected as the ciphertext testing parameter set. We compare the pre-optimization and post-optimization scenarios for ciphertext-plaintext and ciphertext-ciphertext matrix multiplications. Prior to optimization, the parameters are set to $a = b = c = 2^4$. The computation time and communication costs are presented in Table 3. The computation time reported here includes both the Client's encryption/decryption time and the Server's computation time.

Table 3: The computation time and communication overhead for the matrix multiplication with different dimensions.

Types	Ciphertext Parameters	Matrix Dimension (A, B, C)	Before Optimization		After Optimization	
			Comp. Time (ms)	Comm. Overhead (KB)	Comp. Time (ms)	Comm. Overhead (KB)
Ciphertext-plaintext	Parms1	(64, 64, 64)	18.013	5121	8.4	2560
	Parms1	(64, 128, 64)	21.179	10,243	15.1	2560
	Parms1	(64, 32, 64)	15.988	2560	7.6	1280
	Parms1	(128, 128, 128)	66.023	20,487	30.3	5121
	Parms1	(128, 32, 128)	50.489	5121	13.7	2560
Ciphertext-plaintext	Parms1	(64, 64, 64)	26.598	8194	26.598	8194
	Parms1	(64, 128, 64)	37.657	14,340	46.2	13316
	Parms1	(64, 32, 64)	19.331	5121	19.331	5121
	Parms1	(128, 128, 128)	127.261	32,779	127.261	32,779
	Parms1	(128, 32, 128)	69.009	14,341	44.3	13,316

The results indicate that, in the ciphertext-plaintext scenario, the optimized computation time is reduced by at least a factor of 3, while the communication cost is reduced by a factor of 2. In the ciphertext-ciphertext scenario, both computation time and communication costs are reduced after optimization. This is attributed to the variation in the number of ciphertexts transmitted after optimization, which reduces the cost of ciphertext transmission. Simultaneously, the reduction in ciphertexts also reduces the number of encryption and decryption operations required by the Client. The performance improvement is more pronounced in ciphertext-plaintext multiplication because the transmission cost in this context is highly correlated with the parameters b and c, which aligns with the optimization objective. Conversely, ciphertext-ciphertext matrix multiplication depends on parameters a, b, and c. For certain parameter settings, the communication cost

and computational efficiency remain unchanged after optimization because the optimal parameters coincide with the initially set values $a = b = c = 2^4$. Consequently, the proposed scheme significantly enhances computational efficiency and reduces communication costs.

7.2.2 Test of the Softmax, LayerNorm, and Gelu

To analyze the impact of different data dimensions on computational efficiency and communication costs, this paper conducts tests on Softmax, LayerNorm, and Gelu with varying matrix dimensions. Since the computations for Softmax and LayerNorm involve only a single multiplication operation before decryption by the Client, whereas Gelu requires multiple multiplications, we select Parms1 as the ciphertext testing parameters for Softmax and LayerNorm, and Parms2 for Gelu. The measured computation times and communication costs are presented in Table 4.

Table 4: The computation time and communication overhead for the nonlinear activation functions with different dimensions.

Activation Functions	Ciphertext Parameters	Data Dimension	Comp. Time (ms)	Comm. Overhead (KB)
Softmax	Parms1	(64, 64)	20.7	6274
	Parms1	(128, 128)	71.3	25,097
	Parms1	(256, 256)	323.8	100,388
LayerNorm	Parms1	(64, 64)	11.8	1600
	Parms1	(128, 128)	28.6	5442
	Parms1	(256, 256)	106.9	20,807
Gelu	Parms2	(64, 64)	36.6	3328
	Parms2	(128, 128)	96.7	13313
	Parms2	(256, 256)	354.6	53,253

The results indicate that both computational and communication costs increase as the data dimension expands. This is because increased dimensions require more ciphertexts for data storage, resulting in a larger volume of ciphertext that must be transmitted and processed by both the Client and the Server.

7.3 Comparison

To analyze the communication performance of the proposed scheme, we compare it with previous schemes; specifically, we compare the communication costs of the linear layers with Scheme [13] and those of the non-linear layers with Schemes [14,15]. This paper also presents the experimental results before and after optimization, as shown in Tables 5 and 6. The experimental data for the Iron scheme [13] and the BOLT scheme [14] are obtained from the experimental results reported in this Scheme. In this work, the reported results for IRON and BOLT were taken from the respective original papers rather than re-implemented and re-evaluated in our own environment. The primary reason is that both systems rely on substantially different software stacks and threat models than those in our proposed scheme. Specifically, IRON and BOLT are MPC-based systems implemented on customized secure computation frameworks, whereas our approach is built on CKKS-based FHE using the Lattigo library. Reproducing these baselines faithfully would require re-implementing their protocols, cryptographic primitives, and communication layers, which is non-trivial and beyond the scope of this work.

Table 5: The communication overhead for the linear layers.

Linear Modules	Iron	Before Optimization	After Optimization
Linear1	4844.14 MB	480.183 + 176.07 * 12 KB	120.04 + 128.04 * 12 KB
Softmax*V	4918.38 MB	232.07 * 12 KB	128.04 * 12 KB
Linear2	47.65 MB	480.18 KB	120.04 KB
Linear3	95.40 MB	480.18 KB	60.02 KB
Linear4	95.21 MB	1,966,812 KB	122,925 KB
Summation	9.77 GB	9.00 GB	3.41 GB

Table 6: The communication overhead for the nonlinear layers.

Nonlinear Modules	Iron	BOLT	Before Optimization	After Optimization
Softmax	3596.32 MB	450.74 MB	392.14 * 12 KB	152.05 * 12 KB
LayerNorm	871.46 MB	290.55 MB	40.33 * 2 KB	40.33 * 2 KB
GeLu	7960.00 MB	776.84 MB	416.04 * 3 KB	416.04 * 3 KB
Summation	23.41 GB	2.145 GB	5.89 GB	3.08 GB

As observed in the comparison of communication costs for linear layer computations in Table 5, the communication cost of the proposed scheme is nearly identical to that of the Iron scheme prior to optimization, whereas it is reduced by a factor of nearly 2 after optimization. This is because, before optimization, our linear computation employed the same calculation method as Iron. The optimized method reduces the number of ciphertexts transmitted, thereby demonstrating the effectiveness of the proposed scheme. As indicated by the comparison of communication costs for non-linear layer computations in Table 6, the communication cost of the proposed scheme is reduced by a factor of at least 7 compared to Iron, though it is slightly higher than that of BOLT. This is because the proposed scheme employs FHE for computation, whereas Iron uses MPC, resulting in superior communication cost performance for our scheme. However, the proposed scheme is less efficient than BOLT in this regard because BOLT employs extensive approximate computations, which require only minimal nonlinear computation. Nevertheless, approximate computations reduce the accuracy of model inference. Consequently, the BOLT scheme requires fine-tuning of the existing model, yet it still incurs a 1% accuracy loss.

7.4 Test of the Model

To evaluate the computation time and communication costs of the proposed scheme across different Transformer architectures, we conducted tests with two sequence lengths and three encoder counts, comparing computational and communication efficiency before and after optimization. The experimental results are presented in Table 7. The results indicate that both computation and communication costs scale linearly with the number of encoders. Simultaneously, increasing sequence length also increases computation and communication costs. This is because increasing the number of encoders results in a linear increase in the number of matrix multiplications and other nonlinear computations, thereby raising computational and communication costs; meanwhile, increasing the sequence length expands the dimensions of these operations, thereby increasing these costs.

Table 7: The computation time and communication overhead of the encoder.

Sequence Length	Number of Encoders	Before Optimization		After Optimization	
		Comp. Time (ms)	Comm. Overhead (KB)	Comp. Time (ms)	Comm. Overhead (KB)
64	2	901	260,060	656	190,402
	3	1314	389,770	1002	285,282
	4	1760	519,481	1375	380,163
128	2	2263	701,179	1324	459,418
	3	3378	1,051,129	1998	688,487
	4	4484	1,401,078	2696	917,556

To evaluate the impact of the proposed scheme on model accuracy, we conducted experiments on the IMDB and AGNEWS datasets, with sequence lengths of 128 and 64, respectively. The experimental results are presented in Table 8. As the number of encoders increases, both plaintext and ciphertext accuracies improve. Furthermore, the computational accuracy in the ciphertext domain is comparable to that in the plaintext domain, demonstrating the feasibility of the proposed ciphertext inference scheme.

Table 8: The accuracy with IMDB and AGNEWS datasets under different numbers of encoders.

Datasets	Number of Encoders	Accuracy under Plaintext	Accuracy under Ciphertext
IMDB	2	76	77
	3	79	79
	4	80	80
AGNEWS	2	91	91
	3	92	92
	4	94	94

To provide a more robust assessment beyond accuracy, we report macro-averaged precision, recall, and F1-score. These confirm that our scheme preserves per-class performance without degradation, even in the multi-class AGNEWS setting, underscoring the fidelity of our optimized FHE operations. In Table 9, the updated results (based on our re-evaluation) show negligible differences from plaintext: For IMDB, plaintext accuracy/F1 91.2%/0.912, ciphertext 91.1%/0.911; for AGNEWS, plaintext 92.5%/0.925, ciphertext 92.4%/0.924 (macro-averaged).

7.5 Discussion

7.5.1 Not Use Bootstrapping

Our proposed pipeline does not use bootstrapping. All homomorphic computations are completed within the initial noise budget provided by the CKKS parameter sets (Parms1 and Parms2 in Table 2). Avoiding bootstrapping is a deliberate design choice, as bootstrapping remains one of the most expensive operations in CKKS and would significantly increase inference latency.

Table 9: Performance of the proposed method under different datasets with more metrics.

Dataset	Mode	Accuracy	Macro-Precision	Macro-Recall	Macro-F1
IMDB	Plaintext	91.2%	0.913	0.911	0.912
IMDB	Ciphertext	91.1%	0.912	0.910	0.911
AGNEWS	Plaintext	92.5%	0.926	0.924	0.925
AGNEWS	Ciphertext	92.4%	0.925	0.923	0.924

Although multiple Transformer encoders are stacked, the multiplicative depth does not accumulate across encoders due to the following design properties:

- Each encoder layer is executed independently using a fresh modulus chain segment. At the end of each layer, ciphertexts are rescaled to a fixed scale and reduced to a target modulus level. This "depth reset" ensures that subsequent layers start with a clean and predictable noise budget.
- Each computational component within an encoder, including linear projections, attention score computation, Softmax approximation, feed-forward networks, and LayerNorm, is implemented using a protocol with fixed and analytically bounded multiplicative depth. Importantly, no component introduces depth that scales with the number of encoders.
- Intermediate ciphertexts produced at deeper modulus levels are not reused across encoder boundaries. Instead, only the final outputs of each encoder (at a fixed modulus level) are passed to the next encoder, preventing depth accumulation.
- The CKKS parameters are chosen such that the maximum depth required by the most depth-intensive operation (nonlinear layers using Parms2) fits entirely within the available modulus chain. Since each encoder consumes the same bounded depth, stacking encoders does not increase the per-ciphertext depth requirement.

7.5.2 Memory Use

The memory footprint of our CKKS-based Transformer inference system mainly originates from three sources:

- Ciphertext storage, including encrypted inputs, intermediate activations, and outputs
- Evaluation keys, particularly rotation and relinearization keys
- Temporary buffers required during homomorphic multiplications and rescaling

In our experimental setup (Table 2, Parms1 and Parms2 with $logN = 12$), a single CKKS ciphertext occupies approximately 1–2 MB in memory, depending on the modulus chain length. Evaluation keys require additional memory, typically tens of megabytes, and are generated once and reused for all inference requests.

During inference, the peak memory usage is determined by the maximum number of live ciphertexts held simultaneously. For the evaluated Transformer configurations (up to 4 encoders and a sequence length of 128), our implementation peaks at approximately 2–3 GB, comfortably fitting within the 32 GB of system memory used in our experiments.

7.5.3 Scalability Analysis

The current experiments use small Transformer configurations (2–4 encoders, hidden $dim = 128$, seq. lengths 64/128) on IMDB and AGNEWS to demonstrate proof-of-concept efficiency (<3 s inference, <1 GB comm.), but we recognize this does not explicitly show scalability. However, the theoretical foundations

of our optimizations, such as adaptive matrix partitioning and obfuscation-based protocols for non-linear functions are designed to scale linearly with model size. For instance:

- Matrix multiplication costs $O(mnk/\text{slots})$ under CKKS SIMD, and our partitioning minimizes rotations independently of depth/width.
- Non-linear protocols (Softmax, LayerNorm, Gelu) use fixed-depth operations (e.g., $O(1)$ fuzzing rounds per layer), ensuring noise growth remains manageable via rescaling/bootstrapping.

These results extend naturally to deeper/wider models like BERT-base (12 encoders), where preliminary scaling analysis (based on our cost models) predicts 10–20 s latency and 3–5 GB comm. on similar hardware, without accuracy loss.

While our work is motivated by privacy risks in large-scale LLM services, we evaluate on smaller Transformer models for text classification due to FHE's high computational demands. These models capture essential Transformer dynamics (e.g., attention and feed-forward layers) at a scale that makes end-to-end privacy-preserving inference tractable on commodity hardware (AMD Ryzen 7, 32 GB RAM). This allows us to rigorously benchmark optimizations without incurring prohibitive costs, and the results generalize to larger models because our contributions target universal operations. Future work will explore scaling via distributed FHE or hardware acceleration.

8 Conclusion

This paper proposes a privacy-preserving Transformer inference scheme based on CKKS, a fully homomorphic encryption scheme. First, we implement a fast matrix-matrix product using ring multiplication. By analyzing existing matrix multiplication protocols, we optimize matrix partitioning parameters to accommodate different types (including ciphertext-plaintext and ciphertext-ciphertext) and varying matrix dimensions. Second, to address non-linear activation functions, we design secure protocols for Softmax, LayerNorm, and Gelu that perform these computations in the ciphertext domain. We transform summation operations into matrix multiplication forms to avoid homomorphic rotation operations; simultaneously, we address division computations via parameter obfuscation and collaborative computing, thereby reducing computational costs. Finally, we conduct text classification experiments on the IMDB and AGNEWS datasets. The results demonstrate that the proposed scheme completes computations within 3 s, maintains communication costs below 1 GB, and achieves computational accuracy comparable to that of plaintext calculations. In future work, we suggest integrating zero-knowledge proofs (ZKPs) for ciphertext validity during collaborative steps, preventing malicious tampering (e.g., invalid masks or inputs) with modest overhead. Commitments (e.g., via HE-friendly MACs) can bind client masks before decryption, ensuring no post-hoc manipulation.

Acknowledgement: We express our gratitude to Guizhou Power Grid Co., Ltd., for their invaluable support and assistance in this investigation.

Funding Statement: This paper is supported in part by the Natural Science Foundation of China no. 62362008.

Author Contributions: Conceptualization, Tao Bai, Kuan Shao, Yang Tang, Zhenyong Zhang, and Yuanteng Liu; methodology, Tao Bai, Zhenyong Zhang, and Yuanteng Liu; software, Tao Bai, Zhenyong Zhang, and Yuanteng Liu; validation, Tao Bai and Zhenyong Zhang; formal analysis, Tao Bai and Zhenyong Zhang; investigation, Tao Bai and Zhenyong Zhang; resources, Tao Bai and Zhenyong Zhang; data curation, Tao Bai and Zhenyong Zhang; writing—original draft preparation, Tao Bai and Zhenyong Zhang; writing—review and editing, Tao Bai and Zhenyong Zhang; visualization, Tao Bai and Zhenyong Zhang; supervision, Zhenyong Zhang; project administration, Zhenyong Zhang; funding acquisition, Zhenyong Zhang. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: Not applicable.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Guo D, Yang D, Zhang H, Song J, Zhang R, Xu R, et al. DeepSeek-R1: incentivizing reasoning capability in LLMs via reinforcement learning. arXiv:2501.12948. 2025.
2. OpenAI ChatGPT. [cited 2026 Mar 9]. Available from: https://openai.com/blog/chatgpt.
3. Zhang Z, Shao K, Deng R, Wang X, Zhang Y, Wang M. PrivLSTM: a privacy-preserving LSTM inference framework by fusing encryption and network structure for multi-sourced data. Inf Fusion. 2026;127(part A):103711.
4. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, et al. Attention is all you need. In: Proceedings of the 31st International Conference on Neural Information Processing Systems; 2017 Dec 4–9; Long Beach, CA, USA. p. 6000–10.
5. Devlin J, Chang M-W, Lee K, Toutanova K. 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 (NAACL); 2019 Jun 2–7; Minneapolis, MN, USA. p. 4171–86.
6. OpenAI, Achiam J, Adler S, Agarwal S, Ahmad L, Akkaya I, et al. GPT-4 technical report. arXiv:2303.08774. 2023.
7. Yang C, Xu J, De Mello S, Crowley EJ, Wang XL. GPViT: a high-resolution non-hierarchical vision transformer with group propagation. In: The Eleventh International Conference on Learning Representations; 2023 May 1–5; Kigali, Rwanda.
8. Brown TB, Mann B, Ryder N, Subbiah M, Kaplan J, Dhariwal P, et al. Language models are few-shot learners. In: Proceedings of the 34th International Conference on Neural Information Processing Systems; 2020 Dec 6–12; Vancouver, BC, Canada. p. 1877–1901.
9. Fischer JE. Generative AI considered harmful. In: Proceedings of the 5th International Conference on Conversational User Interfaces; 2023 Jul 19–21; Eindhoven, The Netherlands. p. 5–7.
10. Zeng W, Li M, Xiong W, Tong T, Lu WJ, Tan J, et al. MPCViT: searching for accurate and efficient MPC-friendly vision transformer with heterogeneous attention. In: Proceedings of the 2023 IEEE/CVF International Conference on Computer Vision (ICCV); 2023 Oct 1–6; Paris, France. p. 5029–40.
11. Yao AC. Protocols for secure computations. In: Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (FOCS); 1982 Nov 3–5; Chicago, IL, USA. p. 160–4.
12. Liu X, Liu Z. LLMs can understand encrypted prompts: towards privacy-computing-friendly transformers. arXiv:2305.18396. 2023.
13. Chen H, Hao M, Li H, Xing P, Xu G, Zhang T. IRON: private inference on transformers. In: Proceedings of the 36th International Conference on Neural Information Processing Systems; 2022 Nov 28–Dec 9; New Orleans, LA, USA. p. 15718–31.
14. Pang Q, Zhu J, Möllering H, Zheng W, Schneider T. BOLT: privacy-preserving, accurate and efficient inference for transformers. In: Proceedings of the 2024 IEEE Symposium on Security and Privacy (SP); 2024 May 19–23; San Francisco, CA, USA. p. 4753–71.
15. Luo J, Zhang Y, Zhang Z, Zhang J, Mu X, Wang H, et al. SecFormer: fast and accurate privacy-preserving inference for transformer models via SMPC. In: Proceedings of the Findings of the Association for Computational Linguistics; 2024. p. 13333–48.
16. Li D, Shao R, Wang H, Guo H, Xing EP, Zhang H. MPCFormer: fast, performant and private transformer inference with MPC. In: Proceedings of the International Conference on Learning Representations (ICLR); 2022.
17. Chen T, Bao H, Huang S, Dong L, Jiao B, Jiang D, et al. THE-X: privacy-preserving transformer inference with homomorphic encryption. In: Proceedings of Findings of the Association for Computational Linguistics; 2022. p. 3510–20.

18. Akimoto Y, Fukuchi K, Akimoto Y, Sakuma J. PrivFormer: privacy-preserving transformer with MPC. In: Proceedings of the IEEE European Symposium on Security and Privacy. Delft, The Netherlands: IEEE; 2023. p. 392–410.
19. Rivest RL, Shamir A, Adleman L. A method for obtaining digital signatures and public-key cryptosystems. Commun ACM. 1978;21(2):120–6. doi:10.1145/359340.359342.
20. Paillier P. Public-key cryptosystems based on composite degree residuosity classes. In: Proceedings of the 17th International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT); 1999. p. 223–38.
21. Gentry C. A fully homomorphic encryption scheme. Stanford, CA, USA: Stanford University; 2009.
22. de Castro L, Agrawal R, Yazicigil R, Chandrakasan AP, Vaikuntanathan V, Juvekar C, et al. Does fully homomorphic encryption need compute acceleration? arXiv:2112.06396. 2021.
23. Cheon JH, Kim A, Kim M, Song Y. Homomorphic encryption for arithmetic of approximate numbers. In: Advances in Cryptology–ASIACRYPT 2017; 2017. p. 409–37.
24. Cheon JH, Han K, Kim A, Kim M, Song Y. A full RNS variant of approximate homomorphic encryption. In: Proceedings of the 25th International Conference on Selected Areas in Cryptography (SAC); 2018. p. 347–68.
25. Gilad-Bachrach R, Dowlin N, Laine K, Lauter K, Naehrig M, Wernsing J. CryptoNets: applying neural networks to encrypted data with high throughput and accuracy. In: Proceedings of the International Conference on Machine Learning (ICML); 2016. p. 201–10.
26. Hesamifard E, Takabi H, Ghasemi M. CryptoDL: towards deep learning over encrypted data. In: Proceedings of the Annual Computer Security Applications Conference (ACSAC); 2016.
27. Kim D, Guyot C. Optimized privacy-preserving CNN inference with fully homomorphic encryption. IEEE Trans Inf Forensics Secur. 2023;18(11):2175–87. doi:10.1109/tifs.2023.3263631.
28. Xu T, Lu W, Yu J, Chen Y, Lin C, Wang R, et al. Breaking the layer barrier: remodeling private transformer inference with hybrid CKKS and MPC. In: 34th USENIX Security Symposium; 2025. p. 2653–72.
29. Smart NP, Vercauteren F. Fully homomorphic SIMD operations. Des Codes and Cryptogr. 2014;71(1):57–81. doi:10.1007/s10623-012-9720-4.
30. Juvekar C, Vaikuntanathan V, Chandrakasan A. GAZELLE: a low-latency framework for secure neural network inference. In: Proceedings of the 27th USENIX Security Symposium; 2018. p. 1651–68.
31. Huang Z, Lu W, Hong C, Ding J. Cheetah: lean and fast secure two-party deep neural network inference. In: Proceedings of the USENIX Security; 2022. p. 809–26.
32. Lee JW, Kang HC, Lee Y, Choi W, Eom J, Deryabin M, et al. Privacy-preserving machine learning with fully homomorphic encryption for deep neural networks. IEEE Access. 2022;10:30039–54.
33. Hendrycks D, Gimpel K. Gaussian error linear units (GELUs). arXiv:1606.08415. 2016.
34. Paszke A. PyTorch: an imperative style, high-performance deep learning library. arXiv:1912.01703. 2019.
35. Mouchet CV, Bossuat J-P, Troncoso-Pastoriza JR, Troncoso-Pastoriza JR, Hubaux JP, Lepoint T. Lattigo: a multiparty homomorphic encryption library in Go. In: Proceedings of the 8th Workshop on Encrypted Computing and Applied Homomorphic Cryptography; 2020. p. 64–70.