MOAI: Module-Optimizing Architecture for Non-Interactive Secure Transformer Inference

Linru Zhang
Nanyang Technological University

Xiangning Wang
Nanyang Technological University

Jun Jie Sim
Ant Group

Zhicong Huang
Ant Group

Jiahao Zhong
Nanyang Technological University

Huaxiong Wang
Nanyang Technological University

Pu Duan
Ant Group

Kwok-Yan Lam
Nanyang Technological University

Abstract

The advent of Large Language Models (LLM) has brought about a new wave productivity, revolutionizing business operations while keeping cost relatively low. The human-like interface of LLM enables it to be easily integrated with business functions, thereby freeing up precious human resources for more complex, valuable tasks. However, due to the intensive computation and memory requirements of LLM inference, it is preferable and cheaper to deploy LLMs with the Cloud Service Providers (CSP) that offer high performance computation resources and low-latency networking. Nevertheless, privacy concerns have been raised about the possibility of data leakage to the CSP. In this work, we seek to address such privacy concerns through the use of Fully Homomorphic Encryption (FHE). FHE enables the CSP to work on data in its encrypted form, thus ensuring that the data stay private and secure. We propose the implementation of LLM inference with FHE. While a series of prior work have demonstrated that it is possible to execute LLM inference in a private manner, it remains a challenge to design a solution that is practical.

Our contributions are as follows: We provide the first end-to-end open-source implementation of a non-interactive transformer inference with FHE. We report an amortized time of 9.6 minutes of one input with 128 tokens when evaluating the BERT model on CPU. Our packing methods for encrypted matrices remove the need to repack ciphertext between encrypted matrix multiplication and activation layers. Additionally, we introduce interleaved batching to eliminate the internal rotations during ciphertext matrix multiplications. Our approach also avoids HE rotations in evaluations of the softmax and layerNorm, leading to a speedup of $4.22\times$ and $122\times$ than existing works respectively. Our implementation supports arbitrary token lengths, in contrast with existing solutions that requires a full token embedding. Our implementation can be found here.[1]

1 Introduction

The transformer architecture proposed in the seminal paper [1] has imbued Artificial Intelligence (AI) with the exceptional capability to interact with humans via text. This has given rise to a class of machine learning models known as Large Language Models (LLM), aptly named due to the sheer size of data required to train them, together with its propensity for language. The ability to understand

[1]https://github.com/dtc2025ag/MOAI

human textual inputs and generate a corresponding output that another human can understand and interact with is credited to the attention mechanism that allows the model to capture the important points in an input.

In light of the recent widespread adoption of LLM, there is an increasing need for privacy and security when applied to business functions. In May 2023, Samsung banned the use of ChatGPT upon discovery that proprietary internal source code was leaked by its employees [2]. Later that year, in September 2023, Google's Bard AI was found to leak conversations with users in Google searches [3]. A common issue in both privacy leaks is that LLMs inadvertently stores a user's data, be it to update future iterations or to enhance the user's experience.

One way to ensure privacy when interacting with such proprietary AI models is to use privacy enhancing technologies (PET) during model inference. PET offers the capability to protect the user's data such that throughout the model inference process, the server never sees or learns anything about the user's data. In this work, we focus on Fully Homomorphic Encryption (FHE), once regarded as the holy grail of cryptography. FHE is special type of encryption that allows computations on encrypted data, such that upon decryption, the result is as though the computations were performed on the data in the clear.

1.1 Related Work

Early works [4, 5, 6] targeting interactive privacy-preserving transformer inference, usually on BERT, relied on a combination of FHE and Multi-Party Computation (MPC) that splits the user's data into shares and distributes a share to the server. The linear layers, namely compromising of matrix multiplication is executed with FHE while the non-linear activation layers such as softmax, layer normalization, and GELU are executed via MPC interactively. Iron [4] proposed a method for matrix multiplication that when given two encrypted matrices, returns shares of the matrix products and MPC approximations for activation functions. A subsequent work, BOLT [5] proposed two encrypted matrix multiplication protocols for different packing methods and alternative and more accurate approximations for activation functions compared to Iron. A conversion protocol from an FHE ciphertext to an MPC share is applied after each linear layer due to the different underlying rings in FHE and MPC. This also helps to eliminate the error induced in FHE computations without the need for the expensive bootstrapping operation. Bumblebee [6], building upon Iron and BOLT, proposed an FHE-based matrix multiplication that works over the MPC underlying ring, eliminating any conversion protocols for larger models like LLaMa-7b. A main drawback using MPC protocols is that the inference process is an interactive one, often requiring high WAN speeds to achieve good latency. Very recently, the first non-interactive solution NEXUS, based purely on FHE, was proposed by [7] (NDSS '25). They proposed a way to compress several encrypted matrices that are frequently used to reduce online computation cost, together with several HE friendly approximations for activation functions. However, NEXUS does not provide an end-to-end inference solution. Its reported performance reflects the aggregated costs of independently evaluated modules, but the methodology for integrating these components into a cohesive inference pipeline remains unspecified.

1.2 Our Contributions

In this paper, we present MOAI: **M**odule-**O**ptimizing **A**rchitecture for non-interactive secure transformer **I**nference. MOAI aims to address the following main challenges for building an end-to-end non-interactive privacy-preserving LLM inference:

(1) A coherent evaluation flow along with corresponding packing strategies to ensure consistency between adjacent blocks.
(2) Optimized evaluation algorithms for each component, including matrix multiplications and nonlinear functions, to enhance overall efficiency.

Our contributions are as follows:

End-to-end open-source implementation. We provide the first end-to-end open-source implementation of a non-interactive transformer inference with FHE.

Complimentary packing method for encrypted matrices. We leverage two encrypted matrix packing methods - column packing and diagonal packing to remove the need to repack a ciphertext between the linear and nonlinear layers. Interleaved batching is used to reduce half of rotations in encrypted matrix multiplication.

Rotation free activation layers. We reduce the number of rotations in the evaluation of the softmax and layerNorm, leading to a speedup of $4.22\times$ and $122\times$ than existing works respectively. This is achieved via novel FHE algorithms that we propose, utilizing the encrypted matrix packing method.

Arbitrary token lengths. Existing FHE solutions assumes a fixed input length equal to the model's maximum supported tokens (e.g., 128 tokens per input in BERT), and the inference result will be incorrect if the input has fewer tokens. For inputs with fewer tokens, it seems as a trivial solution to pad the inputs with the 0 token, but will result in incorrect results as the value of padding slots will be changed and corrupt subsequent computations. Our implementation supports arbitrary token lengths, less than the maximum.

We validate MOAI on the BERT-base-uncased model and report an amortized time per input (up to 128 tokens) of 9.6 minutes on CPU.

2 Preliminaries

We use $[n]$ to represent the set $\{0, 1, ..., n-1\}$. We use bold lowercase letters to denote a row vector $\mathbf{v} \in \mathbb{R}^n$ where $\mathbf{v} = [v_0, v_1, ..., v_{n-1}]$. We also use $[v_i]_{i\in[n]}$ to represent the vector $\mathbf{v}$ where v_i is the i-th entry of the row vector $\mathbf{v}$. Column vectors are represented by $\mathbf{v}^\intercal$. We define the scalar product for $c \in \mathbb{R}$ and $\mathbf{v} \in \mathbb{R}^n$, let $\alpha\mathbf{v} = [\alpha v_i]_{i\in[n]}$. We also define addition and Hadamard product (entry-wise multiplication) between two vectors with dimension n as follows: $\mathbf{v} + \mathbf{u} := [v_i + u_i]_{i\in[n]}, \mathbf{v} \otimes \mathbf{u} := [v_i u_i]_{i\in[n]}$. We use $\mathrm{Rot}_k(\mathbf{v})$, for $k > 0$, to denote the vector $[v_k, ..., v_{n-1}, v_0, ...v_{k-1}]$, a left rotation of $\mathbf{v}$. For $k < 0$, $\mathrm{Rot}_k(\mathbf{v})$ denotes a right rotation.

We use uppercase letters to denote a matrix $A \in \mathbb{R}^{m\times d}$ where $A = (a_{ij})_{i\in[m],j\in[d]}$. We use $\mathbf{a_j}^\intercal$ to represent the j-th column of A, where $\mathbf{a_j}^\intercal = [a_{0j}, a_{1j}, ..., a_{m-1,j}]^\intercal$ and we can write $A = (\mathbf{a_0}^\intercal \parallel \mathbf{a_1}^\intercal \parallel \cdots \parallel \mathbf{a_{d-1}}^\intercal)$. Further, if A is a square matrix with size $m \times m$, define $\mathrm{Diag}_i(A)$ as A's i-th (upper) diagonal for $i \in [m]$: $\mathrm{Diag}_i(A) := [a_{0,i}, a_{1,i+1}, ..., a_{m-i-1,m-1}, a_{m-i,0}, ..., a_{m-1,i-1}]$.

We define the mean and variance of a vector as functions from $\mathbb{R}^n \to \mathbb{R}$. We denote the softmax function and layer normalization as functions from $\mathbb{R}^n \to \{0,1\}^n$, specifically:

$$\sigma(\mathbf{x}) = \left[\frac{\exp(x_i)}{\sum_{r\in[n]}\exp(x_r)}\right]_{i\in[n]} \qquad \mathrm{LayerNorm}(\mathbf{x}) := \left[\gamma \cdot \frac{x_i - \mathrm{mean}(\mathbf{x})}{\sqrt{\mathrm{var}(\mathbf{x})}} + \beta\right].$$

where γ, β are model parameters. The functions σ and $\mathrm{LayerNorm}$ can be directly extended to matrices, by applying the functions to each *row* of the input matrix.

2.1 The BERT model

Embedding. Let the input has m tokens. First it is embedded to m vectorized token embeddings, each of which has dimension d. We denote the it as matrix $X_e \in \mathbb{R}^{m\times d}$. d is also defined as the *hidden size* of the model. Let $P \in \mathbb{R}^{m\times d}$ be the positional embeddings for the input tokens. Then, the embedding input is obtained by computing $X := X_e + P$.

Multi-Head Attention. Let $X \in \mathbb{R}^{m\times d}$ be the input, $W_Q, W_K, W_V \in \mathbb{R}^{d\times d'}$ be weights matrices, and $\mathbf{b}_Q, \mathbf{b}_K, \mathbf{b}_V$ be bias vectors. An attention block in head $h \in [H]$ is defined as $\mathrm{Attention}(Q, K, V) = \sigma\left(\frac{QK^\intercal}{\sqrt{d'}}\right) V \in \mathbb{R}^{d\times d'}$, where: $Q := XW_Q + \mathbf{1}^\intercal\mathbf{b}_Q,\ K := XW_K + \mathbf{1}^\intercal\mathbf{b}_K,\ V := XW_V + \mathbf{1}^\intercal\mathbf{b}_V$. The multi-head attention is the execution of H parallel attention blocks and we set the h-th result to be $X^{(h)}$ where $d' := d/H$. Then the H matrices are concatenated $X^{\mathrm{Concat}} := (X^{(0)} \parallel X^{(1)} \parallel ... \parallel X^{(H-1)}) \in \mathbb{R}^{m\times d}$. The attention block ends with a layer normalization procedure with a residual connection: $X := \mathrm{LayerNorm}(X + X^{\mathrm{Concat}})$.

Feed Forward Network. The feed forward network contains four components. A fully connected layer followed by the Gaussian Error Linear Units (GELU) [8] activation function, another fully connected layer and $\mathrm{LayerNorm}$. The GELU function is defined as $\mathrm{GELU}(x) := \frac{x}{2}\left(1 + \frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}dt\right)$. Since there is no closed-form expression for GELU, [8] suggested an approximation by tanh:

$$\mathrm{GELU}(x) \approx 0.5x\left(1 + \tanh\left[\sqrt{2/\pi}\left(x + 0.044715x^3\right)\right]\right). \tag{1}$$

Let $W_{\mathsf{fc1}}, W_{\mathsf{fc2}}$ be weights matrices, and $\mathbf{b}_{\mathsf{fc1}}, \mathbf{b}_{\mathsf{fc2}}$ be bias vectors. The feed forward network can be described as follows: $X := \mathrm{LayerNorm}(X + \mathrm{GELU}(XW_{\mathsf{fc1}} + \mathbf{1}^\intercal\mathbf{b}_{\mathsf{fc1}})W_{\mathsf{fc2}} + \mathbf{1}^\intercal\mathbf{b}_{\mathsf{fc1}}) \in \mathbb{R}^{m\times d}$. The output of feed forward block is set to be the input of the attention block in the next transformer layer. The BERT-base-uncased model [9] is using $H = 12$ and it has 12 transformer layers, that is, it contains 12 attention blocks and 12 feed forward blocks.

After 12 transformer layers, the model applies a linear layer with bias and $\tanh(\cdot)$ to the first token ([CLS]) as the aggregated sequence representation for classification tasks. Let this vector be $\mathbf{x}_{[\mathrm{CLS}]} \in \mathbb{R}^d$. The classification layer first computes $\tanh(\mathbf{x}_{[\mathrm{CLS}]}W + \mathbf{1}^\intercal\mathbf{b}) \in \mathbb{R}^d$, denoted as $\mathbf{x}$. Finally let c is the number of classes, the output is $\mathbf{x}W_{\mathsf{cls}} + \mathbf{1}^\intercal\mathbf{b}_{\mathsf{cls}} \in \mathbb{R}^c$.

2.2 Fully Homomorphic Encryption - CKKS Scheme

Fully Homomorphic Encryption (FHE) is a privacy enhancing technology that enables computations on encrypted data without decryption. The Cheon-Kim-Kim-Song scheme (CKKS) scheme [10] is an FHE scheme that supports approximate arithmetic operations over complex vectors. We give brief description of the CKKS scheme and we defer the reader to Appendix F and [10] for more details on the encryption scheme.

Let N be the degree of a polynomial ring, whose value is a power of 2. Q be a product of distinct primes $Q = \prod_0^L q_i$, where L is called the *level* of the ciphertext. A fresh ciphertext starts from level L and drops to $L-1$ after a homomorphic multiplication. When remaining level is 0, decryption fails. The bootstrapping process [11] can refresh the ciphertext and restore the remaining level, enabling continued evaluations. The CKKS scheme supports Single Instruction Multiple Data SIMD-style computation by encoding $N/2$ messages into a single ciphertext, enabling parallel homomorphic operations across all slots. Let $\mathbf{v}, \mathbf{v}'$ be two vectors in $\mathbb{C}^{N/2}$ and $\mathrm{ct} = \mathrm{Enc}(\mathbf{v})$ and $\mathrm{ct}' = \mathrm{Enc}(\mathbf{v}')$. The operations supported by the CKKS scheme are as follows

- $\mathrm{Add}(\mathrm{ct}, \mathrm{ct}')$. Return a ciphertext that encrypts the vector sum $\mathbf{v} + \mathbf{v}'$.
- $\mathrm{Mult}(\mathrm{ct}, \mathrm{ct}')$. Return a ciphertext that encrypts the Hadamard product $\mathbf{v} \otimes \mathbf{v}'$.
- $\mathrm{pMult}(\mathrm{ct}, \mathbf{v}')$. Return a ciphertext that encrypts the Hadamard product $\mathbf{v} \otimes \mathbf{v}'$.
- $\mathrm{sMult}(s, \mathrm{ct})$. Return a ciphertext that encrypts the product $(sv_0, sv_1, ..., sv_{N/2})$.
- $\mathrm{Rot}_k^{\mathrm{CKKS}}(\mathrm{ct})$. Return a ciphertext that encrypts $\mathrm{Rot}_k(\mathbf{v}) = [v_k, ..., v_{N/2-1}, v_0, ...v_{k-1}]$, i.e., the left rotation of $\mathbf{v}$.

3 Matrix Packing and Matrix Multiplications

The SIMD capability of an FHE ciphertext allows multiple data to be packed into a single ciphertext, which offers an array-like paradigm to encode matrices for encryption. We list three cases where encrypted matrix multiplications is required in the attention mechanism.

(I) Ciphertext-Plaintext matrix multiplication when calculating Q, K, V.

(II) Ciphertext-Ciphertext matrix multiplication when calculating $Q \cdot K^\intercal$.

(III) Ciphertext-Ciphertext matrix multiplication when calculating output of softmax with V.

There are mainly three ways to encode an encrypted matrix in FHE [12], namely row, column, diagonal packing. The main difficulty in designing an encrypted matrix multiplication protocol is that the encrypted matrix must be able to switch between various forms, in this case, its transpose ($K^\intercal$), or post-function evaluation (output of softmax) in an efficient manner. For instance, if a matrix is in row packing form, then the transpose would require the extraction of each matrix element from its ciphertext slot and sending it into another ciphertext.

To this end, we propose to use both the column packing and diagonal packing in [12], to realize seamless transition for encrypted matrix multiplication.

Definition 3.1 (Column Packing). *Given a matrix $X \in \mathbb{R}^{m \times d}$, the column packing of X is a set of d CKKS ciphertexts, denoted as $\mathrm{Enc}_{col}(X) := \left\{ \mathrm{Enc}(\mathbf{x}_0^\intercal), \mathrm{Enc}(\mathbf{x}_1^\intercal), ..., \mathrm{Enc}(\mathbf{x}_{d-1}^\intercal) \right\}$, where $\mathbf{x_j}^\intercal$ is the j-th column of X. We say that the ciphertext of X is in **column packing**.*

Definition 3.2 (Diagonal Packing). *Given a **square** matrix $X \in \mathbb{R}^{m \times m}$, the upper diagonal packing of X is a set of m CKKS ciphertexts, denoted as $\mathrm{Enc}_{diag}(X) := \left\{ \mathrm{Enc}(\mathrm{Diag}_0(X)), \mathrm{Enc}(\mathrm{Diag}_1(X)), ..., \mathrm{Enc}(\mathrm{Diag}_{m-1}(X)) \right\}$, where $\mathrm{Diag}_j(X)$ is the j-th (upper) diagonal of X. We use $\mathrm{Diag}_j(X)$ as a **row** vector. And we say that the ciphertext of X is in **diagonal packing**.*

Encrypted matrix-vector multiplication for both packing methods were proposed in [12]. We give a brief description of the algorithms for encrypted matrix-vector multiplication for both packing methods. Let C be a square matrix of dimensions $m \times m$ and v be a vector of length m.

For column packing, denote C with its m columns as $\{\mathrm{Col}_0(C), \mathrm{Col}_1(C), ..., \mathrm{Col}_{m-1}(C)\}$. [12] first proposed an auxiliary algorithm `Replicate` that takes the vector $\mathbf{v}$ and return m-many vectors $\mathbf{v_i}$, each encrypting v_i, that is $\{\mathbf{v_i} = [v_i, v_i, \ldots, v_i]\}$. The matrix-vector product is computed as $Cv = \sum \mathrm{Col}_i(C) \odot \mathbf{v_i}$. We can extend the matrix-vector product to matrix-matrix product by iterating over all columns of the right hand side matrix.

For diagonal packing, denote C with its m diagonals as $\{\mathrm{Diag}_0(C), \mathrm{Diag}_1(C), ..., \mathrm{Diag}_{m-1}(C)\}$. The matrix-vector product is computed as $Cv = \sum \mathrm{Diag}_i(C) \odot \mathrm{Rot}(\mathbf{v}, i)$. A subsequent improvement in [13] reduced the number of rotations needed to $O(\sqrt{m})$ by using the baby-step giant-step method (see Appendix B.1 for details).

We describe how we realize encrypted matrix multiplication for three cases described earlier with both column and diagonal packing.

Case I: Calculating Q, K, $V \in \mathbb{R}^{m \times d'}$ with $\mathrm{Enc}(X)W_i + \mathbf{1}^\intercal \mathbf{b}_i$ for $i \in \{Q, K, V\}$. We encrypt X in column packing while the weights and biases are not encrypted. We note that since the weights matrices are not encrypted, the matrix product can be interpreted as concatenations of linear combinations of X.

$$XW = \big[\sum_{i \in [d']} w_{i,0} \mathbf{x}_0^\intercal \parallel \sum_{i \in [d']} w_{i,1} \mathbf{x}_1^\intercal \parallel \cdots \parallel \sum_{i \in [d']} w_{i,d'-1} \mathbf{x}_{d'-1}^\intercal \big].$$

The result is a matrix encrypted in column packing. The detailed algorithm is given as Algorithm 2 in Appendix B.2.

Case II: Calculating $QK^\intercal$. We remind the reader here that Q and K are encrypted in column packing as per the result of Case I. We then apply the following lemma that allows us to compute the matrix product $QK^\intercal$ where both Q and K are encrypted in column packing, without transposing K.

Lemma 3.1. *Suppose $Q, K \in \mathbb{R}^{m \times d'}$, so $QK^\intercal$ is square matrix with size $m \times m$. Then for $j \in [m]$: $\mathrm{Diag}_j(QK^\intercal) = \sum_{i=0}^{d'-1} \mathbf{q}_i \otimes \mathrm{Rot}_j(\mathbf{k}_i)$, where $\mathbf{q}_i$ and $\mathbf{k}_i$ are the i-column of Q and K respectively.*

We note here that the result is a matrix encrypted in diagonal form. The detailed algorithm is given as Algorithm 3 in Appendix B.2.

Case III: Calculating $\sigma(QK^\intercal/\sqrt{d'}) \cdot \mathrm{Enc}(V)$. The encrypted matrix V is encrypted in column packing, as per Case I. For now, we briefly mention that applying the softmax function on a diagonally packed matrix will return a diagonally packed matrix. Thus, $\sigma(QK^\intercal/\sqrt{d'})$ is encrypted in diagonal packing. For this matrix product, we can apply the matrix-vector product for diagonally packed matrices described earlier [12, 13] over all encrypted columns of V. The detailed algorithm is given as Algorithm 4 in Appendix B.2.

4 Non-linear Function Evaluation

In this section we propose optimized evaluations of non-linear functions used in BERT, namely softmax, layer normalization of GELU. Our novelty is that we do not require any rotations in the evaluations. This greatly improves the speed of our algorithms. Table 1 gives a comparison between ours and the state-of-the-art work [7] (NDSS'25) and a concurrent work [14].

Table 1: Comparison with NEXUS [7], THOR [14] and MOAI softmax and layer normalization.

Function	NEXUS #Rot$^{\text{CKKS}}$	THOR #Rot$^{\text{CKKS}}$	Our #Rot$^{\text{CKKS}}$
$\text{Softmax}(X \in \mathbb{R}^{m \times m})$	$\lceil \log_2(m) \rceil \frac{m}{\frac{N}{4}/2^{\lceil \log_2(m) \rceil}}$	$\lceil \log_2(m) \rceil \frac{m}{\frac{N}{2}/2^{\lceil \log_2(m) \rceil}} \cdot 2$	0
$\text{LayerNorm}(X \in \mathbb{R}^{m \times d})$	$\lceil \log_2(d) \rceil \frac{d}{\frac{N}{4}/2^{\lceil \log_2(d) \rceil}}$	$\lceil \log_2(d) \rceil \frac{d}{\frac{N}{2}/2^{\lceil \log_2(d) \rceil}} \cdot 2$	0

Softmax. The softmax function appears in $\sigma(QK^\top/\sqrt{d'})V$, but we eliminate the $\sqrt{d'}$ term by folding it into W_K, saving one CKKS level. The input of softmax is in diagonal packing, and we want the output of the softmax function to be in diagonal packing, as the input to the next module. We introduce a novel FHE softmax algorithm (Algorithm 1, Figure 1) that preserves this packing format and eliminates the need for rotations, based on the key insight in Lemma 4.

Lemma 4.1. *Let $C \in \mathbb{R}^{m \times m}$ be a square matrix. The summation of columns equals to the summation of diagonals:* $\sum_i \mathbf{c}_i^\intercal = \sum_i \text{Diag}_i(C)$.

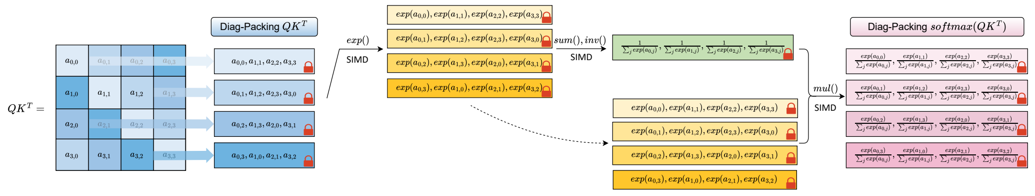


Figure 1: Our no-rotation softmax evaluation whose input and output are both in diagonal packing.

Algorithm 1 HE Softmax evaluation: $\text{Enc}_{diag}(QK^\intercal) \to \text{Enc}_{diag}(\sigma(QK^\intercal))$

```
Input: $\text{Enc}_{diag}(QK^\intercal)$: Diagonal packing of encrypt matrix $QK^\intercal \in \mathbb{R}^{m \times m}$.
Output: $\text{Enc}_{diag}(\sigma(QK^\intercal))$: Diagonal packing of encrypt matrix $\sigma(QK^\intercal) \in \mathbb{R}^{m \times m}$.
1: Write $\text{Enc}_{diag}(QK^\intercal)$ as $\{\text{Enc}(\text{Diag}_0(QK^\intercal)), \text{Enc}(\text{Diag}_1(QK^\intercal)), ..., \text{Enc}(\text{Diag}_{m-1}(QK^\intercal))\}$.
2: Evaluate $\text{Enc}(\exp(\text{Diag}_i(QK^\intercal)))$ by SIMD polynomial evaluation $(1 + x/2^r)^{2^r}$.
3: Add ciphertexts to obtain $\text{ct}_{sum} = \text{Enc}(\sum_i \exp(\text{Diag}_i(QK^\intercal)))$.
4: Evaluate FHE inverse algorithm on $\text{ct}_{sum}$ using the Goldschmidt division algorithm [15]. Let
   the result be $\text{ct}_{sum^{-1}} := \text{Enc}(1/\sum_i \exp(\text{Diag}_i(QK^\intercal)))$.
5: For $i \in [m]$, return $\text{Mult}(\text{ct}_{sum^{-1}}, \text{Enc}(\exp(\text{Diag}_i(QK^\intercal))))$ as the $i$-th component.
```

Layer normalization. Note that the input of LayerNorm is in column packing. We first give the SIMD style mean and variance algorithm, given input as $\text{Enc}_{col}(X)$ where $X \in \mathbb{R}^{m \times d}$. For $\text{Enc}(mean(X))$, compute the summation of all encrypted columns of X and then multiply $1/d$: $\text{sMult}(1/d, \sum_i \text{Enc}(\mathbf{x}_i^\intercal))$. For $\text{Enc}(var(X))$. First compute $\text{ct}_{x^2} := \text{Mult}(\text{Enc}(\mathbf{x}_i) - \text{Enc}(mean(X)), \text{Enc}(\mathbf{x}_i) - \text{Enc}(mean(X))), i \in [d]$. Then compute $\text{sMult}(1/d, \sum_{i \in [d]} \text{ct}_{x^2})$. We can merge two scalar multiplication of $1/d$ to save one FHE level. Goldschmidt division algorithm [15, 16] is applied to obtain the inverse square root $(1/\sqrt{x})$ of variance. Please refer to Algorithm 5 for the detailed explanation in Appendix C.

GELU. For the encrypted GELU algorithm using Equation 1, it suffices to approximate $\tanh(\cdot)$ function by polynomial. The main challenge is the balance between the size of effective input interval and the polynomial degree. By [17], a degree-$\Omega(R)$ polynomial is required when using minimax approximation with fixed maximum error on the domain interval $[-R, R]$. In our work, we approximate $\tanh(\cdot)$ in Equation 1 using a degree-23 polynomial $P_{23}(x)$ on interval $[-20, 10]$ which is enough in our experiment. If the model needs larger input interval, we apply the *domain extension polynomial* $B_r(x) := \frac{x}{L^r} - \frac{4}{27B^2L^{3r}}x^3$ from [17]. Let $P(x)$ be the minimax polynomial of $\tanh(x)$ on $[-B, B]$. Then $P \circ B_0(x)$ can take input from a larger interval $[-LB, LB]$ for proper L. (Theorem 4 in [17]). This can be further extended to input interval $[-L^sB, L^sB]$. We also point out that using piecewise polynomial in NEXUS [7] to approximate GELU has precision problem. Please refer to Appendix C.2 for a detailed analysis.

5 Interleaved Batching

In column packing (Definition 3.1) and diagonal packing (Definition 3.2), each ciphertext encrypts a vector of dimension m, i.e., the sequence length of input. Meanwhile, the number of slots $N/2$ in the CKKS scheme is usually 2^{15}. So batching multiple vectors into one ciphertext will reduce the amortized cost. The batching method should be (1) compatible with the HE matrix multiplication in Section 3 and FHE evaluations in Section 4; and (2) requiring as few as possible expensive homomorphic operations like rotation. In this section, We propose an interleaved batching method that reduces the number of rotations by half compared to existing approaches ([7, 14]).

Assume $m|(N/2)$. Let $N/(2m)$ vectors $\{\mathbf{x}^{(r)} \in \mathbb{R}^m\}_{r\in[N/(2m)]}$ be:

$$\mathbf{x}^{(0)} = [x_0^{(0)}, x_1^{(0)}, ..., x_{m-1}^{(0)}], ..., \mathbf{x}^{(N/(2m)-1)} = [x_0^{(N/(2m)-1)}, x_1^{(N/(2m)-1)}, ..., x_{m-1}^{(N/(2m)-1)}] \, ,$$

We define the *interleaved batching* by packing the the first entry of each vector from $\{\mathbf{x}^{(r)} \in \mathbb{R}^m\}_{r\in[N/(2m)]}$, then the second entry, and at last the last entry. We use $\tilde{\mathbf{x}} \in \mathbb{R}^{N/2}$ to represent the interleaved batching of $N/(2m)$ vectors in $\mathbb{R}^m$.

$$\tilde{\mathbf{x}} := [x_0^{(0)}, x_0^{(1)}, x_0^{(2)}, ..., x_0^{(N/(2m)-1)}, ...x_{m-1}^{(0)}, x_{m-1}^{(1)}, x_{m-1}^{(2)}, ..., x_{m-1}^{(N/(2m)-1)}] \, . \quad (2)$$

We use Lemma 5.1 to address the its compatibility with rotation and include its proof in Appendix D.2.

Lemma 5.1. *Let $\tilde{\mathbf{x}} \in \mathbb{R}^{N/2}$ be the interleaved batching vector of $\{\mathbf{x}^{(r)} \in \mathbb{R}^m\}_{r\in[N/(2m)]}$ given by Eq. (2). Then $\mathrm{Rot}_{jN/(2m)}(\tilde{x})$ is the interleaved batching vector of $\{\mathrm{Rot}_j(\mathbf{x}^{(r)})_{r\in[N/(2m)]}\}$.*

Compared to the naive batching method in existing works, which require a process called "internal rotation", our interleaved batching reduces half of rotations in matrix multiplication.

Finally, we define interleaved column/diagonal packing by merging interleaved batching with our matrix packing methods directly. Please refer to Appendix D.3.

6 Experimental Results

Implementation. We implement MOAI in C++, utilizing the SEAL library [18] for CKKS homomorphic encryption, FHE-MP-CNN[2] for CKKS bootstrapping, and OpenMP [19] for multi-thread programming. We set the degree of the polynomial ring as $N = 2^{16}$, yielding 2^{15} slots in a ciphertext. We chose a 1743-bit ciphertext modulus of in accordance to the Homomorphic Encryption Standard [20] to achieve 128-bit security. We apply the scale propagation technique [21] to better manage the noise propagation and set the scale to 2^{46} in our implementation. All results are evaluated on a Intel(R) Xeon(R) Platinum 8480+ CPU at 2.0 GHz with 56 cores and 112 threads. Our codes are available here.[3]

Model and Dataset. The BERT model has $H = 12$ attention heads and 12 layers with an embedding dimension of $d = 768$. The input token length is set to be 128, which is the maximum number of tokens allowed in BERT. In evaluations, we use the pre-trained BERT-base-uncased model fine-tuned on SST-2, QNLI, and RTE tasks.[4]

6.1 Micro-benchmark evaluation of Softmax and LayerNorm

We compare our implementation of the softmax and layer normalization functions with NEXUS [7]. The amortized time taken for both functions are given in Table 2, evaluated under identical precision levels and single-thread setting on the same machine. We find that removing FHE rotations in the evaluation of these two functions resulted in a speedup of $4.22\times$ and $122\times$ compared to NEXUS.

[2]https://github.com/snu-ccl/FHE-MP-CNN

[3]https://github.com/dtc2025ag/MOAI

[4]`https://github.com/huggingface/transformers/tree/main/examples/pytorch/text-classification`

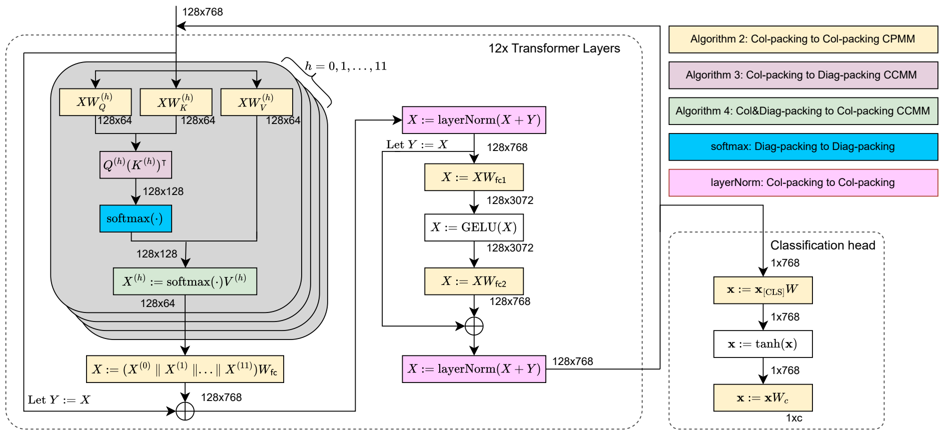
Table 2: Comparison between Softmax and LayerNorm implementations

	NEXUS		MOAI		Speedup
	Packing method	Amortized time(s)	Packing method	Amortized time(s)	
Softmax	Row packing	2.53	Diagonal packing	0.62	4.22×
LayerNorm	Row packing	46.7	Column packing	0.38	122×

6.2 End-to-end Inference Evaluation

The name MOAI also comes from the monolithic human figures on Easter Island. They are huge and stable. They are similar to our scheme: large and stable end-to-end inference, and with very few "rotations". In this section, we show our full diagram in Figure 2 of the BERT-base-uncased model. We illustrate the dimensions and the packing formats of encrypted matrices in each module. The packing formats include column packing (Col-packing, Definition 3.1) and diagonal packing (Diag-packing, Definition 3.2). CPMM stands for ciphertext-plaintext matrix multiplication, while CCMM stands for ciphertext-ciphertext matrix multiplications. As shown in Figure 2, MOAI does not need conversion between the two packing methods.

Figure 2: The full diagram of MOAI. The parameters are from the BERT-base-uncased model, which has 12 transformer layers and 12 heads in each layer. c is the number of classes.



128x768
12x Transformer Layers
$h = 0, 1, \dots, 11$
$XW_Q^{(h)}$
$XW_K^{(h)}$
$XW_V^{(h)}$
128x64
128x64
128x64
$Q^{(h)}(K^{(h)})^\top$
128x128
softmax(·)
128x128
$X^{(h)} := \text{softmax}(\cdot)V^{(h)}$
128x64
$X := (X^{(0)} \parallel X^{(1)} \parallel \dots \parallel X^{(11)})W_{fc}$
Let $Y := X$
128x768
$X := \text{layerNorm}(X + Y)$
Let $Y := X$
128x768
$X := XW_{fc1}$
128x3072
$X := \text{GELU}(X)$
128x3072
$X := XW_{fc2}$
128x768
$X := \text{layerNorm}(X + Y)$
128x768
Algorithm 2: Col-packing to Col-packing CPMM
Algorithm 3: Col-packing to Diag-packing CCMM
Algorithm 4: Col&Diag-packing to Col-packing CCMM
softmax: Diag-packing to Diag-packing
layerNorm: Col-packing to Col-packing
Classification head
1x768
$\mathbf{x} := \mathbf{x}_{[CLS]}W$
1x768
$\mathbf{x} := \tanh(\mathbf{x})$
1x768
$\mathbf{x} := \mathbf{x}W_c$
1xc

Arbitrary Token Length Support. Unlike NEXUS, which assumes a fixed input length equal to the model's maximum supported tokens (e.g., 128 tokens per input in BERT), our implementation supports arbitrary token lengths (up to the maximum) while preserving correctness and efficiency in homomorphic evaluation. We introduce a *tracing vector* mechanism to track which ciphertext slots contain valid tokens versus padding. This vector is maintained throughout both column and diagonal packing. For slots holding real tokens, the corresponding entry in the tracing vector is set to 1; otherwise, it is set to 0. During homomorphic evaluations, padding slots may be changed to non-zero values, which corrupts subsequent computations. To address this, we multiply the trace vector with the ciphertexts, effectively restoring the padding slots to 0. For example, in Softmax evaluation, computing $\exp(x)$ over padding slots returns 1, leading to incorrect summations $\sum_i \exp(\text{diag}_i(QK^\top))$. Our method ensures these slots are zeroed out before summation. A similar issue arises in LayerNorm evaluation, and our method resolves it as well. To the best of our knowledge, our system is the first FHE-based solution to correctly and efficiently support arbitrary token lengths within transformer models, addressing a the limitation in prior work.

Placement of Bootstrapping. As described earlier, bootstrapping is an expensive process that resets the noise level of the ciphertext. Thus the placement of bootstrapping is critical to the performance of our implementation. We make the following two observations: (1) For a sufficiently accurate softmax result, the Goldschmidt division algorithm, line 4 in Algorithm 1, requires at least 10 iterations. We computed empirically range of the values require and find that based on Figure 5, at least 10 iterations are required. In total, evaluating the softmax function requires at least 20 layers, 8 for computing e^x and 11 for Goldschmidt division, and 1 for the final multiplication; (2) The time it takes for a ciphertext matrix multiplication varies with its noise level. Ciphertext at a higher level require a longer computation time due to its larger modulus. Therefore, we optimized the softmax algorithm by performing bootstrapping on ct_{sum} after the third line of Algorithm 1 and feed the refreshed ciphertext as an input to the Goldschmidt division algorithm. This optimization reduces the overall depth of the Softmax evaluation from 20 layers to 10. As a result, the two matrix multiplications preceding the softmax function (first two rows in Table 3) can be executed at a lower level, allowing the evaluation of the attention block to begin with 14 layers instead of a level larger than 20. Performing a single bootstrapping operation on one ciphertext significantly reduces the time cost of the two matrix multiplications, and further reduces the total time taken. The details of the optimized softmax algorithm are provided in Algorithm 9.

End-to-end performance. Table 3 presents the end-to-end performance (amortized for 256 inputs). The results are obtained from the execution of the BERT model with 12 transformer layers. We emphasize that MOAI's evaluation flow and corresponding packing strategies ensure consistency across all layers, enabling an efficient end-to-end implementation, rather than adding the time taken across individual modules like NEXUS. We report an amortized time of 9.6 minutes per input with up to 128 tokens when evaluating the BERT model on CPU.

Table 3: Breakdown of MOAI encrypted inference, amortized over 256 inputs

Component	Operation in each layer	Depth	Packing	(Data dimensions) $\times$ (Number of operations)	Total time for 12 layers(s)
Attention	Pt-ct MatrixMul	$15 \to 14$	Col $\to$ Col	$(\mathbb{R}^{128\times768} \times \mathbb{R}^{768\times64}) \times 36$	37.4
	Ct-ct MatrixMul	$14 \to 13$	Col$\to$ Diag	$(\mathbb{R}^{128\times64} \times \mathbb{R}^{64\times128}) \times 12$	40.3
	Softmax	$13 \to 3$	Diag $\to$ Diag	$\mathbb{R}^{128\times128} \times 12$	53.3
	Ct-ct MatrixMul	$3 \to 2$	Diag $\to$ Col	$(\mathbb{R}^{128\times128} \times \mathbb{R}^{128\times64}) \times 12$	1.4
Self Output	Pt-ct MatrixMul	$2 \to 1$	Col $\to$ Col	$(\mathbb{R}^{128\times768} \times \mathbb{R}^{768\times768})$	1.7
	Bootstrapping	$1 \to 21$	Col $\to$ Col	$\mathbb{R}^{128\times768}$	95.4
	LayerNorm	$21 \to 1$	Col $\to$ Col	$\mathbb{R}^{128\times768}$	0.6
	Bootstrapping	$1 \to 21 \to 10$	Col $\to$ Col	$\mathbb{R}^{128\times768}$	95.8
Intermediate	Pt-ct MatrixMul	$10 \to 9$	Col $\to$ Col	$(\mathbb{R}^{128\times768} \times \mathbb{R}^{768\times3072})$	44.1
	GELU	$9 \to 2$	Col $\to$ Col	$\mathbb{R}^{128\times3072}$	3.3
Final	Pt-ct MatrixMul	$2 \to 1$	Col $\to$ Col	$(\mathbb{R}^{128\times3072} \times \mathbb{R}^{3072\times768})$	7.1
	Bootstrapping	$1 \to 21$	Col $\to$ Col	$\mathbb{R}^{128\times768}$	98.8
	LayerNorm	$21 \to 1$	Col $\to$ Col	$\mathbb{R}^{128\times768}$	0.6
	Bootstrapping	$1 \to 21 \to 15$	Col $\to$ Col	$\mathbb{R}^{128\times768}$	94.8
Total					**574.6**

Accuracy. We also provide an analysis of the error propagation throughout the encrypted evaluation. We recorded the output at each layer and compared them against the output from a plaintext evaluation, which is plotted in Figure 3. We observe that the error magnitude increases during the first 6 to 7 layers and stabilizes subsequently, indicating that error actually remains bounded. This suggest that an FHE implementation does not result in a drastic loss of accuracy in the evaluated outputs. More details on the error variation within a single layer are provided in subsection E.2. To investigate the effect of the noise on inference results, we sample Gaussian noise with a mean of 0.000223 and a variance of 0.0449 determined from the error of the last layer in our end-to-end evaluation, and add them to the last layer's output to get scores on the GLUE benchmarks. Table 4 shows that the additional error from FHE will only have a negligible impact on the inference results.

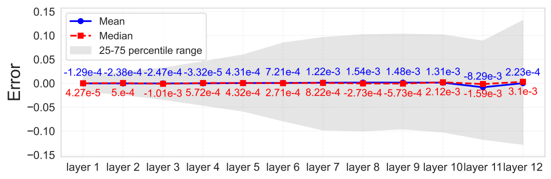


Figure 3: Error analysis of 12 layers' output

Dataset	SST-2	QNLI	RTE
Score without noise	93.9	90.7	65.7
Score with noise	93.8	90.5	65.5
Match count	1818	5437	2959
Mismatch count	4	27	42

Table 4: Scores on the GLUE benchmarks

6.3 Discussion on GPU implementation

A limitation of this work is that all experiments were completed on a CPU, with no GPU results reported. Our implementation uses the CKKS scheme from the SEAL library [18], which was designed for a CPU environment. As a result, our current performance benchmarks do not reflect the potential acceleration that could be achieved through GPU parallelism. Developing a GPU-accelerated version of MOAI remains an important direction for future work to further improve its efficiency.

References

[1] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[2] Siladitya Ray. *Samsung Bans ChatGPT Among Employees After Sensitive Code Leak — forbes.com*. `https://www.forbes.com/sites/siladityaray/2023/05/02/samsung-bans-chatgpt-and-other-chatbots-for-employees-after-sensitive-code-leak/`. [Accessed 13-05-2025].

[3] Chris Stokel-Walker. *Google was accidentally leaking its Bard AI chats into public search results*. `https://www.fastcompany.com/90958811/google-was-accidentally-leaking-its-bard-ai-chats-into-public-search-results`. [Accessed 13-05-2025].

[4] Meng Hao et al. “Iron: Private inference on transformers”. In: *Advances in neural information processing systems* 35 (2022), pp. 15718–15731.

[5] Qi Pang et al. “Bolt: Privacy-preserving, accurate and efficient inference for transformers”. In: *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2024, pp. 4753–4771.

[6] Wen-jie Lu et al. “Bumblebee: Secure two-party inference framework for large transformers”. In: *Cryptology ePrint Archive* (2023).

[7] Jiawen Zhang et al. “Secure transformer inference made non-interactive”. In: *Cryptology ePrint Archive* (2024).

[8] Dan Hendrycks and Kevin Gimpel. “Gaussian error linear units (gelus)”. In: *arXiv preprint arXiv:1606.08415* (2016).

[9] Jacob Devlin et al. “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, volume 1 (long and short papers)*. 2019, pp. 4171–4186.

[10] Jung Hee Cheon et al. “Homomorphic encryption for arithmetic of approximate numbers”. In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2017, pp. 409–437.

[11] Jung Hee Cheon et al. “Bootstrapping for approximate homomorphic encryption”. In: *Advances in Cryptology–EUROCRYPT 2018: 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29-May 3, 2018 Proceedings, Part I 37*. Springer. 2018, pp. 360–384.

[12] Shai Halevi and Victor Shoup. “Algorithms in helib”. In: *Annual Cryptology Conference*. Springer. 2014, pp. 554–571.

[13] Shai Halevi and Victor Shoup. “Bootstrapping for helib”. In: *Journal of Cryptology* 34.1 (2021), p. 7.

[14] Jungho Moon et al. *THOR: Secure Transformer Inference with Homomorphic Encryption*. Cryptology ePrint Archive, Paper 2024/1881. 2024. URL: `https://eprint.iacr.org/2024/1881`.

[15] Robert E Goldschmidt. “Applications of division by convergence”. PhD thesis. Massachusetts Institute of Technology, 1964.

[16] Hongyuan Qu and Guangwu Xu. “Improvements of homomorphic evaluation of inverse square root”. In: *Available at SSRN 4258571* ().

[17] Jung Hee Cheon, Wootae Kim, and Jai Hyun Park.
"Efficient homomorphic evaluation on large intervals".
In: *IEEE Transactions on Information Forensics and Security* 17 (2022), pp. 2553–2568.

[18] *Microsoft SEAL (release 4.1).* `https://github.com/Microsoft/SEAL`.
Microsoft Research, Redmond, WA. Jan. 2023.

[19] Leonardo Dagum and Ramesh Menon.
"OpenMP: an industry standard API for shared-memory programming".
In: *IEEE computational science and engineering* 5.1 (1998), pp. 46–55.

[20] Martin Albrecht et al. "Homomorphic encryption standard".
In: *Protecting privacy through homomorphic encryption* (2021), pp. 31–62.

[21] Jean-Philippe Bossuat et al.
"Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys".
In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2021, pp. 587–617.

[22] Jingwei Chen et al.
Secure Transformer-Based Neural Network Inference for Protein Sequence Classification.
Cryptology ePrint Archive, Paper 2024/1851. 2024.
URL: `https://eprint.iacr.org/2024/1851`.

[23] Donghwan Rho et al. "Encryption-friendly LLM architecture".
In: *arXiv preprint arXiv:2410.02486* (2024).

[24] Dongjin Park, Eunsang Lee, and Joon-Woo Lee.
Powerformer: Efficient Privacy-Preserving Transformer with Batch Rectifier-Power Max Function and Optimized Homomorphic Attention.
Cryptology ePrint Archive, Paper 2024/1429. 2024.
URL: `https://eprint.iacr.org/2024/1429`.

[25] Xiaoqian Jiang et al.
"Secure outsourced matrix computation and application to neural networks". In: *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*. 2018, pp. 1209–1222.

[26] Eugene Y Remez. "Sur la détermination des polynômes d'approximation de degré donnée".
In: *Comm. Soc. Math. Kharkov* 10.196 (1934), pp. 41–63.

[27] Ye Dong et al. "Puma: Secure inference of llama-7b in five minutes".
In: *arXiv preprint arXiv:2307.12533* (2023).

A A detailed comparison between existing works

In this section, we give a more detailed comparison between existing HE-based transformer inference. NEXUS (NDSS'25) [7] is the state-of-the-art HE-based non-interactive transformer inference, providing HE algorithms for different parts transformer. There exist a list of limitations of NEXUS. First of all, it is not an end-to-end inference. Only HE algorithms of different parts in transformer are proposed, and these algorithms cannot be combined into an end-to-end inference. More concretely, their matrices packing methods are not consistent between adjacent blocks, making it impossible to be connected, unless expensive conversion between packing methods are used. (Please refer to Table 5). In addition, their non-polynomial functions are not correct in the scenario of BERT model inference, as we will elaborate in Appendix C. Compared with NEXUS, we address all the above problem and propose the first open-source end-to-end non-interactive transformer inference.

A concurrent pre-print, THOR [14] also claims an end-to-end non-interactive transformer inference. They only provide a very high level description of their solution in paper, and their implementations are not open-source yet. They use several kinds of diagonal packing for non-square matrices, and they need to perform conversion between different packing formats, which is very expensive in time and space. For example, many rotations are need to convert encrypted matrix K to encrypted matrix $K^{\intercal}$, which is required before the ciphertext-ciphertext matrix multiplication $QK^{\intercal}$. Further, they need rotations in their HE algorithms of softmax and layerNorm, as shown in Table 1. Therefore, we claim that our solution is faster than THOR. Below in Table 5, we will show that our number of rotations is much smaller than existing works.

In NEXUS, THOR and our work, the functions in the transformers are not modified. On the contrary, another line of research works are trying to replace the expensive functions in HE, i.e., softmax and layerNorm, by simpler functions. However, changing these functions will require re-training the model [7]. Otherwise, the model can only contains one or two layers. [22] replaces softmax and layerNorm by quadratic function, and it only has one transformer layer. [23] has two transformer layers. They replace softmax with gaussian kernal to remove the need of evaluating $1/x$. [24] replaces softmax and replaces GELU with a simpler ReLU.

To show our efficiency, we compare the number of the expensive HE rotations in the attention blocks, between different existing works. The comparison involves MPC based solution [5], the state-of-the-art HE based solution NEXUS [7], and a concurrent pre-print THOR [14]. The table also includes PowerFormer [24], in which the state-of-the-art ciphertext-ciphertext matrix multiplication [25] is applied. However, as explained in THOR [14], their matrix packing methods are not optimal in transformer inference, because packing format conversions are required. We follow the comparison table in [14] and add our result, as listed in Table 5. The "N/A" means that NEXUS [7] did not provide this algorithm.

Table 5: Comparison of HE matrix multiplications in transformer architecture. The concrete parameters are based on BERT-base-uncased model: $m = 128$, $d = 768$, $d' = 64$, $H = 12$. N is the CKKS ring dimension. $s = 2^{15}$ is from BOLT, while $c = 16$ is from THOR. The equations of the first 4 schemes are from THOR [14].

Operation	Scheme	#KeySwitch in HE rotations	
		Equation	Number
$\{XW_*^{(h)}\}_{h\in[H]}$	BOLT	$\frac{63md}{s} + (\frac{s}{64m} - 1)\frac{md'}{s} \cdot 3H$	297
$X \in \mathbb{R}^{m\times d}$	NEXUS	$6dd'H/(N/(4m))$	27648
$W_* \in \mathbb{R}^{d\times d'}$	PowerFormer	$\left(\sqrt{3m+d'} + (\sqrt{3m+d'} + \log(\frac{m}{d'}))\frac{3H}{2}\right)\frac{d}{m}$	2612
$* = Q, K, V$	THOR	$\frac{m}{2} + 6(\frac{d}{2d'} - 1)\frac{d'}{c} + \frac{3d'}{c}$	180
	MOAI (ours)	0	**0**
$\{Q^{(h)}(K^{(h)})^\intercal\}_{h\in[H]}$	BOLT	$(2m - 2 + m\log(m/2)) \cdot H$	12264
$Q^{(h)} \in \mathbb{R}^{m\times d'}$	NEXUS	$m^2H/(N/(4m))$	1536
$K^{(h)} \in \mathbb{R}^{m\times d'}$	PowerFormer	$(5m/2 + 4\sqrt{m}) \cdot H/2$	2196
	THOR	$8\frac{m}{c} + \frac{m}{8}(\log c + 3) + 2m + \frac{3m}{2c}$	444
	MOAI (ours)	$\frac{md'}{N/(2m)} \cdot H$	**384**
$\{\sigma(Q^{(h)}(K^{(h)})^\intercal)V^{(h)}\}_{h\in[H]}$	BOLT	$(2m-1)(\log m) \cdot H$	21420
$\sigma(Q^{(h)}(K^{(h)})^\intercal) \in \mathbb{R}^{m\times m}$	NEXUS	N/A	N/A
$V^{(h)} \in \mathbb{R}^{m\times d'}$	PowerFormer	$(4m + 3\sqrt{14m} + 8\sqrt{m} + 2\sqrt{2m}) \cdot \frac{H}{2}$	4614
	THOR	$m(\log c + 3)/4 + m(1 + \frac{m+2}{4})/c$	492
	MOAI (ours)	$d'(\sqrt{m} + m)/(N/(2m)) \cdot H$	**420**
$(X^{(0)} \parallel ... \parallel X^{(H-1)})W_{fc}$	BOLT	$31md/s + (\frac{s}{32m} - 1)\frac{md'}{s} \cdot H$	177
$X^{(h)} \in \mathbb{R}^{m\times d'}$	NEXUS	$2d^2H/((N/(4m)))$	110592
$W_{fc} \in \mathbb{R}^{d\times d}$	PowerFormer	$4\sqrt{m} \cdot (d/m)^2$	1656
	THOR	$d'/2 + 2(d'/m - 1)m/c$	118
	MOAI (ours)	0	**0**

B Missing materials in Section 3

B.1 The baby-step-giant-step method

In [13] they optimized the algorithm by baby-step-giant-step method, and the number of HE rotations is reduced to $O(\sqrt{m})$. The idea is to write $m = g \times b$. First for $i \in [b]$, we compute $\mathrm{Rot}_i(\mathbf{v})$ for b times. To compute rotation $\mathrm{Rot}_k(\mathbf{v})$ for $k >= b$, we can write $k = \alpha b + r$ with $r < b$, and

$$\mathrm{rot}_k(\mathbf{v}) := \mathrm{rot}_{\alpha b}\big(\mathrm{rot}_r(\mathbf{v})\big)\ .$$

Combining the above, and with the diagonals of C, we will have

$$C\mathbf{v}^\intercal := \sum_{\alpha\in[g]} \mathrm{rot}_{\alpha b}\big(\sum_{r\in[b]} \mathrm{rot}_{m-\alpha b}\big(\mathrm{Diag}_{\alpha b+r}(C)\big) \otimes \mathrm{rot}_r(\mathbf{v})\big)\ . \tag{3}$$

B.2 Pseudo codes in Section 3

Algorithm 2 describes the detailed implementation of CPMM when X is encrypted in column packing.

Algorithm 2 Ciphertext-plaintext matrix multiplication $\mathrm{Mult}_{ct,pt}(\mathrm{Enc}_{col}(X), W, B)$.

Input: $\mathrm{Enc}_{col}(X)$: Column packing of encrypted $X \in \mathbb{R}^{m \times d}$; W: Weight matrix with size $d \times d'$. P: Bias matrix with size $m \times d'$.
Output: $\mathrm{Enc}_{col}(XW + B)$: Column packing of encrypted $XW + B \in \mathbb{R}^{m \times d'}$.

```
1: Let $Ans := \emptyset$.
2: Parse $\mathrm{Enc}_{col}(X)$ as $\{\mathrm{Enc}(\mathbf{x}_0^\intercal), \mathrm{Enc}(\mathbf{x}_1^\intercal), ..., \mathrm{Enc}(\mathbf{x}_{d-1}^\intercal)\}$.
3: for $j \in [d']$ do
4:     Let $ct_j := (0,0) \in R_Q^2$ be a naive ciphertext.
5:     for $i \in [d]$ do
6:         $ct_j := \mathrm{Add}\left(ct_j, \mathrm{sMult}(w_{ij}, \mathrm{Enc}(\mathbf{x}_i^\intercal))\right)$.
7:     end for
8:     Let $\mathbf{p}_j^\intercal$ be the $j$-th column of $P$.
9:     $Ans := Ans \cup \mathrm{Add}(ct_j, \mathbf{p}_j^\intercal)$.
10: end for
11: Return $Ans$.
```

Algorithm 3 describes the detailed implementation for encrypted inputs. For the sake of simplicity, we assume $m = N/2$ in Algorithm 3 and 4 now. Then the rotation of a vector with dimension m can be directly evaluated using one homomorphic rotation. When $m < N/2$ and $m|(N/2)$, [12] uses sparse packing, i.e., using m slots with index $0, N/(2m), 2 \cdot N/(2m), ..., (m-1) \cdot N/(2m)$. We will elaborate the our method of full packing in Section 5.

Algorithm 3 Ciphertext-ciphertext matrix multiplication: $\mathrm{Mult}_{col,col}(\mathrm{Enc}_{col}(Q), \mathrm{Enc}_{col}(K))$.

Input: $\mathrm{Enc}_{col}(Q), \mathrm{Enc}_{col}(K)$: Column packing of encrypted matrices $Q, K \in \mathbb{R}^{m \times d'}$.
Output: $\mathrm{Enc}_{diag}(QK^\intercal)$: Diagonal packing of encrypted matrix $QK^\intercal \in \mathbb{R}^{m \times m}$.

```
1: Let $Ans := \emptyset$.
2: Parse $\mathrm{Enc}_{col}(Q)$ as $\{\mathrm{Enc}(\mathbf{q}_0^\intercal), \mathrm{Enc}(\mathbf{q}_1^\intercal), ..., \mathrm{Enc}(\mathbf{q}_{d'-1}^\intercal)\}$.
3: Parse $\mathrm{Enc}_{col}(K)$ as $\{\mathrm{Enc}(\mathbf{k}_0^\intercal), \mathrm{Enc}(\mathbf{k}_1^\intercal), ..., \mathrm{Enc}(\mathbf{k}_{d'-1}^\intercal)\}$.
4: for $j \in [m]$ do
5:     Let $ct_j := (0,0) \in R_Q^2$ be a naive ciphertext.
6:     for $i \in [d']$ do
7:         if $j > 0$ then
8:             $temp := \mathrm{Rot}_j^{\mathrm{CKKS}}(\mathrm{Enc}(\mathbf{k}_i^\intercal))$.
9:         else
10:            $temp := \mathrm{Enc}(\mathbf{k}_i^\intercal)$
11:        end if
12:        $ct_j := \mathrm{Add}\left(ct_j, \mathrm{Mult}(\mathrm{Enc}(\mathbf{q}_i^\intercal), temp)\right)$.
13:    end for
14:    $Ans := Ans \cup ct_j$.
15: end for
16: Return $Ans$.
```

C Missing materials in Section 4

C.1 Layer normalization details

In this section we give the detailed implementation of $\mathrm{LayerNorm}(X)$. The overview of the flow is shown in Figure 4. Recall that for matrix input, layerNorm is a "row-to-row"" function applied to each row of the input. Given one row $\mathbf{x} = [x_i]_i \in [d]$ from X, the i-th term of $\mathrm{LayerNorm}(\mathbf{x})$ is

Algorithm 4 Ciphertext-ciphertext matrix multiplication: $\mathrm{Mult}_{diag,col}(\mathrm{Enc}_{diag}(C), \mathrm{Enc}_{col}(V))$.

```
Input: $\mathrm{Enc}_{diag}(C)$: Diagonal packing of encrypted matrices $C \in \mathbb{R}^{m\times m}$. $\mathrm{Enc}_{col}(V)$: Column packing of encrypted matrix $V \in \mathbb{R}^{m\times d'}$.
Output: $\mathrm{Enc}_{col}(CV)$: Column packing of encrypted matrix $CV \in \mathbb{R}^{m\times d'}$.
1: Let $Ans := \emptyset$.
2: Parse $\mathrm{Enc}_{diag}(C)$ as $\{\mathrm{Enc}(\mathrm{Diag}_0(C)), \mathrm{Enc}(\mathrm{Diag}_1(C)), ..., \mathrm{Enc}(\mathrm{Diag}_{m-1}(C))\}$. Let $c_i := \mathrm{Enc}(\mathrm{Diag}_i(C))$, for $i \in [m]$.
3: Parse $\mathrm{Enc}_{col}(V)$ as $\{\mathrm{Enc}(\mathbf{v}_0^\intercal), \mathrm{Enc}(\mathbf{v}_1^\intercal), ..., \mathrm{Enc}(\mathbf{v}_{d'-1}^\intercal)\}$.
4: Let $b := \lceil\sqrt{m}\rceil$, $g := \lceil m/b \rceil$.
5: for $j \in [d']$ do
6:     Baby step: For $i \in [b]$, let $\beta_i := \mathrm{Rot}_i^{\mathrm{CKKS}}(\mathrm{Enc}(\mathbf{v}_j^\intercal))$.
7:     Giant step: Compute the ciphertext of $C\mathbf{v}_j^\intercal$:
           $$ct_j := \sum_{\alpha\in[g]} \mathrm{Rot}_{\alpha b}^{\mathrm{CKKS}}\left(\sum_{r\in[b]} \mathrm{Mult}(\mathrm{Rot}_{m-\alpha b}^{\mathrm{CKKS}}(c_{\alpha b+r}), \beta_r)\right).$$
8:     $Ans := Ans \cup ct_j$.
9: end for
10: Return $Ans$.
```

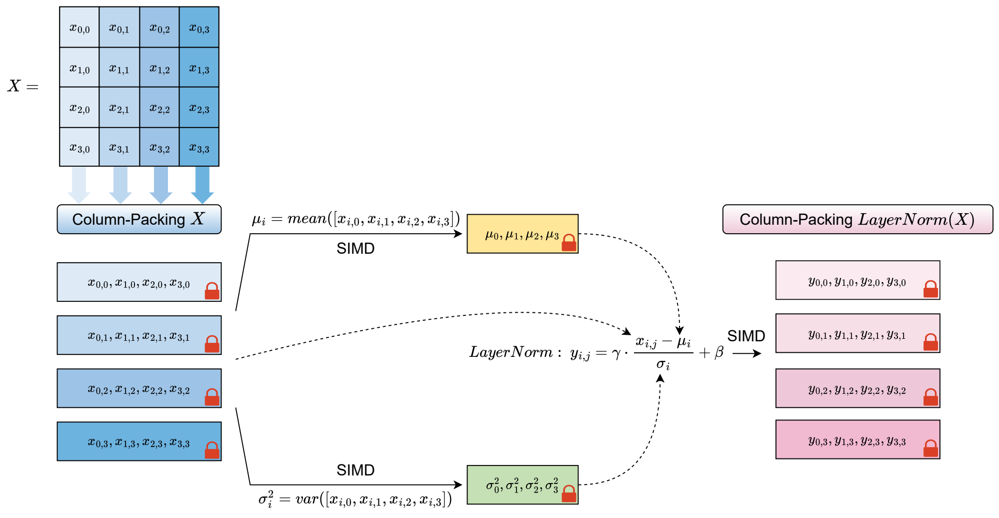

$X =$
$x_{0,0}$ $x_{0,1}$ $x_{0,2}$ $x_{0,3}$
$x_{1,0}$ $x_{1,1}$ $x_{1,2}$ $x_{1,3}$
$x_{2,0}$ $x_{2,1}$ $x_{2,2}$ $x_{2,3}$
$x_{3,0}$ $x_{3,1}$ $x_{3,2}$ $x_{3,3}$
Column-Packing X
$x_{0,0}, x_{1,0}, x_{2,0}, x_{3,0}$
$x_{0,1}, x_{1,1}, x_{2,1}, x_{3,1}$
$x_{0,2}, x_{1,2}, x_{2,2}, x_{3,2}$
$x_{0,3}, x_{1,3}, x_{2,3}, x_{3,3}$
$\mu_i = mean([x_{i,0}, x_{i,1}, x_{i,2}, x_{i,3}])$
SIMD
$\mu_0, \mu_1, \mu_2, \mu_3$
$LayerNorm: \; y_{i,j} = \gamma \cdot \frac{x_{i,j} - \mu_i}{\sigma_i} + \beta$
SIMD
SIMD
$\sigma_i^2 = var([x_{i,0}, x_{i,1}, x_{i,2}, x_{i,3}])$
$\sigma_0^2, \sigma_1^2, \sigma_2^2, \sigma_3^2$
Column-Packing $LayerNorm(X)$
$y_{0,0}, y_{1,0}, y_{2,0}, y_{3,0}$
$y_{0,1}, y_{1,1}, y_{2,1}, y_{3,1}$
$y_{0,2}, y_{1,2}, y_{2,2}, y_{3,2}$
$y_{0,3}, y_{1,3}, y_{2,3}, y_{3,3}$

Figure 4: The flow of HE evaluation of layer normalization.

$\gamma\frac{x_i - \mathrm{mean}(\mathbf{x})}{\sqrt{\mathrm{var}(\mathbf{x})}} + \beta$. After computing mean() and var() we can obtain two different forms of layerNorm where $S := \sum_i x_i$:

$$\frac{\gamma}{\sqrt{d}} \frac{dx_i - S}{\sqrt{\sum_i (dx_i - S)^2/d^2}} + \beta\,. \tag{4}$$

$$\frac{\gamma}{d} \frac{dx_i - S}{\sqrt{\sum_i (dx_i - S)^2/d^3}} + \beta\,. \tag{5}$$

We give the detailed implementation in Algorithm 5. The parameter $t \in \{1, 2\}$ means to choose Equation 4 or Equation 5. It is for the higher precision of the inverse square root $1/\sqrt{x}$. After attention block, we choose $t = 1$ and compute layerNorm according to Equation 4. After feed forward block, we choose $t = 2$ and compute layerNorm according to Equation 5.

Algorithm 5 HE evaluation of layer normalization.

```
Input: $\mathrm{Enc}_{col}(X)$: Column packing of encrypted $X \in \mathbb{R}^{m\times d}$; model parameter $\gamma, \beta \in \mathbb{R}$; algorithm type $t \in \{1,2\}$.
Output: $\mathrm{Enc}_{col}(\mathrm{LayerNorm}(X))$: Column packing of encrypted $\mathrm{LayerNorm}(X) \in \mathbb{R}^{m\times d}$.
1: Let $Ans := \emptyset$.
2: Parse $\mathrm{Enc}_{col}(X)$ as $\left\{ \mathrm{Enc}(\mathbf{x}_0^\intercal), \mathrm{Enc}(\mathbf{x}_1^\intercal), ..., \mathrm{Enc}(\mathbf{x}_{d-1}^\intercal) \right\}$.
3: Compute $\mathrm{Enc}(d \cdot \mathrm{mean}(X))$, where $\mathrm{mean}(X) \in \mathbb{R}^m$ is a vector containing the mean of each row in $X$:
       $$\mathrm{Enc}(d \cdot \mathrm{mean}(X)) := \sum_{i\in[d]} \mathrm{Enc}(\mathbf{x}_i^\intercal)\ ,$$
   i.e., the $i$-th element in $\mathrm{mean}(X)$ is the mean of the $i$-th *row* in X.
4: Compute $\mathrm{Enc}(d^3 \cdot \mathrm{var}(X))$, where $\mathrm{var}(X) \in \mathbb{R}^m$ is a vector containing the variance of each row in $X$:
5: for $j \in [d]$ do
6:     Compute $tmp := \mathrm{sMult}(d, \mathrm{Enc}(\mathbf{x}_i^\intercal)) - \mathrm{Enc}(d \cdot \mathrm{mean}(X))$.
7:     Let $tmp_j := \mathrm{Mult}(tmp, tmp)$.
8: end forLet $\mathrm{Enc}(d^3 \cdot \mathrm{var}(X)) = \sum_j tmp_j$.
9: if $t = 1$ then
10:    Compute $\mathrm{ct} := \mathrm{sMult}(\frac{1}{d^2}, \mathrm{Enc}(d^3 \cdot \mathrm{var}(X)))$.
11:    Use the Goldschmidt division algorithm [15, 16] to obtain the inverse square root $(1/\sqrt{x})$ of ct, denoted as $\mathrm{ct}_{sqroot}$.
12:    Compute $\mathrm{ct}_1 := \mathrm{sMult}(\frac{\gamma}{\sqrt{d}}, \mathrm{ct}_{sqroot})$.
13: end if
14: if $t = 2$ then
15:    Compute $\mathrm{ct} := \mathrm{sMult}(\frac{1}{d^3}, \mathrm{Enc}(d^3 \cdot \mathrm{var}(X)))$.
16:    Use the Goldschmidt division algorithm [15, 16] to obtain the inverse square root $(1/\sqrt{x})$ of ct, denoted as $\mathrm{ct}_{sqroot}$.
17:    Compute $\mathrm{ct}_2 := \mathrm{sMult}(\frac{\gamma}{d}, \mathrm{ct}_{sqroot})$.
18: end if
19: for $i \in [d]$ do
20:    $tmp_i := \mathrm{Mult}\left(\mathrm{ct}_t, (\mathrm{sMult}(d, \mathrm{Enc}(\mathbf{x}_i^\intercal)) - \mathrm{Enc}(d \cdot \mathrm{mean}(X)))\right) + \beta$.
21:    $Ans := Ans \cup tmp_i$.
22: end for
23: Return $Ans$.
```

Remark. We point out the problem of the HE layerNorm implementation of NEXUS [7]. Comparing with our Equation 4 and 5, they propose:

$$\sqrt{d}\gamma \cdot \frac{dx_i - S}{\sqrt{\sum_i (dx_i - S)^2}} + \beta\ .$$

Note that the input to the inverse square root evaluation is $\sum_i (dx_i - S)^2$, which is about $\Theta(d^3)$. In the BERT-base-uncased model $d = 768$, which makes $\sum_i (dx_i - S)^2$ extremely large and is out of the supported data range. Therefore their algorithm is unable to compute layerNorm in BERT-base-uncased model.

C.2 GELU approximation

We approximate $\tanh(\cdot)$ in Equation 1 using a degree-23 polynomial $P_{23}(x)$ on interval $[-20, 10]$ which is enough in our experiment. The approximation method is the famous Remez algorithm [26]. It is noteworthy to point out that all polynomial-approximation method only support a specific interval. As suggested by [17], a degree-$\Omega(R)$ polynomial is required when using Remez or other minimax approximation with fixed maximum error on the domain interval $[-R, R]$. If the model needs larger input interval, we apply the *domain extension polynomial* $B_r(x) := \frac{x}{L^r} - \frac{4}{27B^2L^{3r}}x^3$ from [17]. In their algorithm, let $P(x)$ be the approximation polynomial of $\tanh(x)$ on $[-B, B]$. Then $P \circ B_0(x)$

can take input from a larger interval $[-LB, LB]$ for proper L. (Theorem 4 in [17]). This can be further extended to input interval $[-L^s B, L^s B]$, by using $P \circ B_{s-1} \circ ... \circ B_0(x)$.

Very recently, [23] proposed another method to control the data range. They pre-fix the desired data range of the output of each component in the transformer. Then they design punish function accordingly and perform pre-training to the model weights.

We also point out that using the piecewise polynomial in NEXUS [7] to approximate GELU has several problems. NEXUS uses a different HE evaluation of GELU which is based on the line of MPC-based solution [6, 27]. More concretely, NEXUS calculate GELU ($\mathbb{R} \to \mathbb{R}$) by the following piecewise polynomial:

$$\mathrm{GELU}(x) := \begin{cases} 0 & x \le -4 \\ P_3(x) & -4 < x \le -1.95 \\ P_6(x) & -1.95 < x \le 3 \\ x & x > 3 \end{cases} \tag{6}$$

Here $P_3(x)$ and $P_6(x)$ are polynomials with degree 3 and 6.

First, NEXUS reports the supported input range is $[-8, 8]$. We find that this range is not enough for BERT-base-uncased model, and in our experiment the range is $[-20, 10]$. Next, their piecewise polynomial relies heavily on determining which interval does the encrypted x fall in. This is done by HE evaluation of the an *approximated* sign function. For encrypted number x, NEXUS uses 4 encrypt bits $b_1(x), b_2(x), b_3(x), b_4(x)$ so that $b_i(x) = 1$ if and only if x belongs to the i-th segment of Equation 6. Ideally, $\mathrm{GELU}(x)$ from Equation 6 can be evaluated by the following for $x \in [-8, 8]$:

$$0 \cdot b_1(x) + P_3(x) b_2(x) + P_6(x) b_3(x) + 3 b_4(x) \ .$$

But in fact, each b_i are approximated bits and it is a small value ϵ instead of being 0 exactly. This will cause problems. For example, when $x = -8$, in general $P_6(-8)$ is huge, and thus $P_6(-8) b_3(-8) \approx \epsilon P_6(-8)$ is far from 0. What is more, the actually data range we observed is $[-20, 10]$, making $P_6(-20)$ extremely huge. Theoretically one can choose extreme parameters of the CKKS evaluation of sign function so that the error ϵ is small, but this will increase the time cost dramatically.

D Missing materials in Section 5

D.1 Naive batching and interleaved batching

Suppose we are given $N/(2m)$ vectors $\{\mathbf{x}^{(r)} \in \mathbb{R}^m\}_{r \in [N/(2m)]}$:

$$\mathbf{x}^{(0)} = [x_0^{(0)}, x_1^{(0)}, ..., x_{m-1}^{(0)}], ..., \mathbf{x}^{(N/(2m)-1)} = [x_0^{(N/(2m)-1)}, x_1^{(N/(2m)-1)}, ..., x_{m-1}^{(N/(2m)-1)}] \ ,$$

the naive idea of batching is to arrange them one by one (e.g., in [7]):

$$\bar{\mathbf{x}} := [\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, ..., \mathbf{x}^{(N/(2m)-1)}] = [x_0^{(0)}, ..., x_0^{(1)}, ..., x_0^{(N/(2m)-1)}, ..., x_{m-1}^{(N/(2m)-1)}] \in \mathbb{R}^{N/2} \ .$$

We call $\bar{\mathbf{x}}$ the naive batching of $N/(2m)$ vectors $\{\mathbf{x}^{(r)} \in \mathbb{R}^m\}_{r \in [N/(2m)]}$. For example, consider batching $[x_0, x_1, x_2]$ and $[y_0, y_1, y_2]$. The naive batching generates $[x_0, x_1, x_2, y_0, y_1, y_2]$, while the interleaved batching generates $[x_0, y_0, x_1, y_1, x_2, y_2]$. This naive batching method is compatible with the ciphertext-plaintext matrix multiplication (Algorithm 2), but it is not compatible with the ciphertext-ciphertext matrix multiplications (Algorithm 3 and Algorithm 4). More concretely, this naive batching method is not compatible with rotation. When "rotate" the batched vector (e.g., Line 8 of Algorithm 3), the desired output should be a ciphertext encrypting the following "internal rotations":

$$[\underset{j}{\mathrm{Rot}}(\mathbf{k}^{(0)}), \underset{j}{\mathrm{Rot}}(\mathbf{k}^{(1)}), ..., \underset{j}{\mathrm{Rot}}(\mathbf{k}^{(N/(2m)-1)})] \ .$$

In other words, it should be rotating each $\mathbf{k}^{(i)}$ "internally", which is different from directly rotating the batched vector by $\mathrm{Rot}_j(\bar{\mathbf{k}})$. [14] proposes a method to obtain internal rotation, using 2 normal CKKS rotations and 2 plaintext-ciphertext multiplications. As a toy example, consider batching $[x_0, x_1, x_2]$ and $[y_0, y_1, y_2]$. To "internally" left rotate by 1, we should use the naive batching $\bar{\mathbf{x}} := [x_0, x_1, x_2, y_0, y_1, y_2]$ to generate $[x_1, x_2, x_0, y_1, y_2, y_0]$. This can be done by $\mathrm{Rot}_1(\bar{\mathbf{x}}) \otimes [1, 1, 0, 1, 1, 0] = [x_1, x_2, 0, y_1, y_2, 0]$. Then $\mathrm{Rot}_{-2}(\bar{\mathbf{x}}) \otimes [0, 0, 1, 0, 0, 1] = [0, 0, x_0, 0, 0, y_0]$. By summing them together we obtain $[x_1, x_2, x_0, y_1, y_2, y_0]$.

D.2 Proof of Lemma 5.1

Proof. On the one hand, by Eq. (2), we have

$$\begin{aligned}\underset{jN/(2m)}{\mathrm{Rot}}(\tilde{\mathbf{x}}) := [&x_j^{(0)}, x_j^{(1)}, x_j^{(2)}, ..., x_j^{(N/(2m)-1)},\\ &x_{j+1}^{(0)}, x_{j+1}^{(1)}, x_{j+1}^{(2)}, ..., x_{j+1}^{(N/(2m)-1)},\\ &...\\ &x_{j-1}^{(0)}, x_{j-1}^{(1)}, x_{j-1}^{(2)}, ..., x_{j-1}^{(N/(2m)-1)}]\ .\end{aligned}$$

On the other hand, $\forall r \in [N/(2m)]$, we replace $\mathbf{x}^{(r)}$ in Eq. (2) by $\mathrm{Rot}_j(\mathbf{x}^{(r)}) = [x_j^{(r)}, x_{j+1}^{(r)}, ..., x_{j-1}^{(r)}]$. Then we also obtain

$$\begin{aligned}\underset{jN/(2m)}{\mathrm{Rot}}(\tilde{\mathbf{x}}) := [&x_j^{(0)}, x_j^{(1)}, x_j^{(2)}, ..., x_j^{(N/(2m)-1)},\\ &x_{j+1}^{(0)}, x_{j+1}^{(1)}, x_{j+1}^{(2)}, ..., x_{j+1}^{(N/(2m)-1)},\\ &...\\ &x_{j-1}^{(0)}, x_{j-1}^{(1)}, x_{j-1}^{(2)}, ..., x_{j-1}^{(N/(2m)-1)}]\ .\end{aligned}$$

This finishes the proof. □

D.3 Interleaved batching for matrices

Similarly, suppose we are given $N/(2m)$ matrices $\{X^{(r)} \in \mathbb{R}^{m\times d}\}_{r\in[N/(2m)]}$. Let $\tilde{X} \in \mathbb{R}^{\frac{N}{2}\times d}$ be the *interleaved batching* matrix of $\{X^{(r)} \in \mathbb{R}^{m\times d}\}_{r\in[N/(2m)]}$. The i-th column of $\tilde{X}$ is $\tilde{\mathbf{x}}_i^{\intercal} \in \mathbb{R}^{N/2}$, which is the interleaved batching vector of **all** the i-th column of each $X^{(r)}$, for $r \in [N/(2m)]$, according to Eq. (2) [5]:

$$\begin{aligned}\tilde{\mathbf{x}}_i^{\intercal} := [&x_{i,0}^{(0)T}, x_{i,0}^{(1)T}, x_{i,0}^{(2)T}, ..., x_{i,0}^{(N/(2m)-1)T},\\ &x_{i,1}^{(0)T}, x_{i,1}^{(1)T}, x_{i,1}^{(2)T}, ..., x_{i,1}^{(N/(2m)-1)T},\\ &...\\ &x_{i,m-1}^{(0)T}, x_{i,m-1}^{(1)T}, x_{i,m-1}^{(2)T}, ..., x_{i,m-1}^{(N/(2m)-1)T}]\ .\end{aligned} \tag{7}$$

Now we are can formally define the column packing and the diagonal packing for the interleaved batching.

Definition D.1 (Column Packing for Interleaved Batching)**.** *Suppose we are given $N/(2m)$ matrices $\{X^{(r)} \in \mathbb{R}^{m\times d}\}_{r\in[N/(2m)]}$, the interleaved batching matrix of them is $\tilde{X} \in \mathbb{R}^{\frac{N}{2}\times d}$, whose i-th column is given by Eq. (7), for $i \in [d]$. Then the column packing for $\tilde{X}$ is a set of d CKKS ciphertexts, denoted as:*

$$\mathrm{Enc}_{col}(\tilde{X}) := \left\{ \mathrm{Enc}(\tilde{\mathbf{x}}_0^{\intercal}), \mathrm{Enc}(\tilde{\mathbf{x}}_1^{\intercal}), ..., \mathrm{Enc}(\tilde{\mathbf{x}}_{d-1}^{\intercal}) \right\}\ ,$$

*where $\tilde{\mathbf{x}}_i^{\intercal} \in \mathbb{R}^{N/2}$ is the i-th column of $\tilde{X}$. We say that $\mathrm{Enc}_{col}(\tilde{X})$ is the **interleaved column packing** of encrypted $\{X^{(r)} \in \mathbb{R}^{m\times d}\}_{r\in[N/(2m)]}$.*

Definition D.2 (Diagonal Packing for Interleaved Batching)**.** *Suppose we are given $N/(2m)$ square matrices $\{X^{(r)} \in \mathbb{R}^{m\times m}\}_{r\in[N/(2m)]}$. Define:*

$$\widetilde{diag}(X) := \{\widetilde{diag}(X)_i \in \mathbb{R}^{N/2}\}_{i\in[m]}\ ,$$

where $\widetilde{diag}(X)_i$ is the interleaved batching vector given by Eq. (2) using all the i-th (upper) diagonals of $X^{(r)}$, for each $r \in [N/(2m)]$. The (upper) diagonal packing of is a set of m CKKS ciphertexts, denoted as:

$$\mathrm{Enc}_{diag}(\widetilde{diag}(X)) := \left\{ \mathrm{Enc}(\widetilde{diag}(X)_0), \mathrm{Enc}(\widetilde{diag}(X)_1), ..., \mathrm{Enc}(\widetilde{diag}(X)_{m-1}) \right\}\ .$$

*We use $\widetilde{diag}(X)_i$ as a **row** vector. We say that $\mathrm{Enc}_{diag}(\widetilde{diag}(X))$ is the **interleaved diagonal packing** of encrypted $\{X^{(r)} \in \mathbb{R}^{m\times m}\}_{r\in[N/(2m)]}$.*

[5] Here, we use $x_{i,j}^{(r)T}$ to represent the j-th entry of vector $\mathbf{x}_i^{(r)T}$.

We conclude that the column/diagonal packing for interleaved batching is compatible with all the three homomorphic matrix multiplication algorithms (Algorithm 2, 3 and 4) using Lemma 5.1. Note that all the input and output ciphertexts of these algorithms will be encrypting the vectors in the form of the interleaved batching. Algorithm 6, 7 and 8 are the modified algorithms for homomorphic matrix multiplications using the interleaved batching.

For the HE evaluations of softmax (Algorithm 1), layer normalization (5) and GELU, we assert that the algorithm with interleaved batching input remains the same. This is because in these algorithms, the rows of the input matrix are independently evaluated. In the HE evaluations of softmax and layer normalization, if we add a permutation of rows of the input matrix, there will be the same permutation of rows of the output matrix. Note that interleaved batching is effectively a permutation of rows of the matrix. Therefore, Algorithm 1 and 5 remains the same when the input is in interleaved batching.

Algorithm 6 $\mathrm{Mult}_{ct,pt}(\mathrm{Enc}_{col}(\tilde{X}), W, P)$ using interleaved batching.

```
Input: $\mathrm{Enc}_{col}(\tilde{X})$: Interleaved column packing of encrypted $\{X^{(r)} \in \mathbb{R}^{m\times d}\}_{r\in[N/(2m)]}$; $W$: Weight matrix with size $d \times d'$. $P$: Bias matrix with size $m \times d'$.
Output: $\mathrm{Enc}_{col}(\tilde{X}W + \tilde{P})$: Interleaved column packing of encrypted $\{X^{(r)}W + P \in \mathbb{R}^{m\times d'}\}_{r\in[N/(2m)]}$.
1: Let $Ans := \emptyset$.
2: Parse $\mathrm{Enc}_{col}(\tilde{X})$ as $\{\mathrm{Enc}(\tilde{\mathbf{x}}_0^\intercal), \mathrm{Enc}(\tilde{\mathbf{x}}_1^\intercal), ..., \mathrm{Enc}(\tilde{\mathbf{x}}_{d-1}^\intercal)\}$.
3: Let $\tilde{P}$ be the interleaved batching matrix of $N/(2m)$ repeated matrices $P$.
4: for $j \in [d']$ do
5:     Let $ct_j := (0,0) \in R_Q^2$ be a naive ciphertext.
6:     for $i \in [d]$ do
7:         $ct_j := \mathrm{Add}\left(ct_j, \mathrm{Mult}(w_{ij}, \mathrm{Enc}(\tilde{\mathbf{x}}_i^\intercal))\right)$.
8:     end for
9:     Let $\tilde{\mathbf{p}}_j^\intercal \in \mathbb{R}^{N/2}$ be $j$-th column of $\tilde{P}$, i.e., the interleaved batching vector of $N/(2m)$ repeated $j$-th column of $P$.
10:    $Ans := Ans \cup \mathrm{Add}(ct_j, \tilde{\mathbf{p}}_j^\intercal)$.
11: end for
12: Return $Ans$.
```

Algorithm 7 $\mathrm{Mult}_{col,col}(\mathrm{Enc}_{col}(\tilde{Q}), \mathrm{Enc}_{col}(\tilde{K}))$ using interleaved batching.

```
Input: $\mathrm{Enc}_{col}(\tilde{Q}), \mathrm{Enc}_{col}(\tilde{K})$: Interleaved column packing of encrypted $\{Q^{(r)} \in \mathbb{R}^{m\times d'}\}_{r\in[N/(2m)]}$ and encrypted $\{K^{(r)} \in \mathbb{R}^{m\times d'}\}_{r\in[N/(2m)]}$, respectively.
Output: $\mathrm{Enc}_{diag}(\widetilde{diag}(QK^\intercal))$: Interleaved diagonal packing of encrypted $\{Q^{(r)}(K^{(r)})^\intercal \in \mathbb{R}^{m\times m}\}_{r\in[N/(2m)]}$.
1: Let $Ans := \emptyset$.
2: Parse $\mathrm{Enc}_{col}(\tilde{Q})$ as $\{\mathrm{Enc}(\tilde{\mathbf{q}}_0^\intercal), \mathrm{Enc}(\tilde{\mathbf{q}}_1^\intercal), ..., \mathrm{Enc}(\tilde{\mathbf{q}}_{d'-1}^\intercal)\}$.
3: Parse $\mathrm{Enc}_{col}(\tilde{K})$ as $\{\mathrm{Enc}(\tilde{\mathbf{k}}_0^\intercal), \mathrm{Enc}(\tilde{\mathbf{k}}_1^\intercal), ..., \mathrm{Enc}(\tilde{\mathbf{k}}_{d'-1}^\intercal)\}$.
4: for $j \in [m]$ do
5:     Let $ct_j := (0,0) \in R_Q^2$ be a naive ciphertext.
6:     for $i \in [d']$ do
7:         if $j > 0$ then
8:             $temp := \mathrm{Rot}_{jN/(2m)}(\mathrm{Enc}(\tilde{\mathbf{k}}_i^\intercal))$.
9:         else
10:            $temp := \mathrm{Enc}(\tilde{\mathbf{k}}_i^\intercal)$
11:        end if
12:        $ct_j := \mathrm{Add}\left(ct_j, \mathrm{Mult}(\mathrm{Enc}(\tilde{\mathbf{q}}_i^\intercal), temp)\right)$.
13:    end for
14:    $Ans := Ans \cup ct_j$.
15: end for
16: Return $Ans$.
```

Algorithm 8 $\text{Mult}_{diag,col}(\text{Enc}_{diag}(\widetilde{diag}(C)), \text{Enc}_{col}(\tilde{V}))$ using interleaved batching.

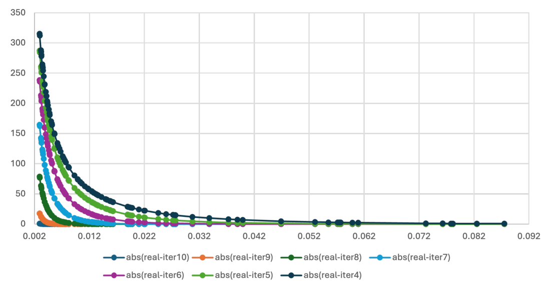
```
Input: $\text{Enc}_{diag}(\widetilde{diag}(C))$: Interleaved diagonal packing of encrypted $\{C^{(r)} \in \mathbb{R}^{m \times m}\}_{r \in [N/(2m)]}$.
    $\text{Enc}_{col}(\tilde{V})$: Interleaved column packing of encrypted $\{V^{(r)} \in \mathbb{R}^{m \times d'}\}_{r \in [N/(2m)]}$.
Output: $\text{Enc}_{col}(\widetilde{CV})$: Interleaved column packing of encrypted $\{C^{(r)}V^{(r)} \in \mathbb{R}^{m \times d'}\}_{r \in [N/(2m)]}$.
 1: Let $Ans := \emptyset$.
 2: Parse $\text{Enc}_{diag}(\widetilde{diag}(C))$ as $\{\text{Enc}(\widetilde{diag}(C)_0), \text{Enc}(\widetilde{diag}(C)_1), ..., \text{Enc}(\widetilde{diag}(C)_{m-1})\}$. Let
    $c_i := \text{Enc}(\widetilde{diag}(C)_i)$, for $i \in [m]$.
 3: Parse $\text{Enc}_{col}(\tilde{V})$ as $\{\text{Enc}(\tilde{\mathbf{v}}_0^\intercal), \text{Enc}(\tilde{\mathbf{v}}_1^\intercal), ..., \text{Enc}(\tilde{\mathbf{v}}_{d'-1}^\intercal)\}$.
 4: Let $b := \lceil \sqrt{m} \rceil$, $g := \lceil m/b \rceil$.
 5: for $j \in [d']$ do
 6:     Baby step: For $i \in [b]$, let $\beta_i := \text{Rot}\, iN/(2m)(\text{Enc}(\tilde{\mathbf{v}}_j^\intercal))$.
 7:     Giant step: Compute the encrypted interleaved batching vector of $\{C^r(\mathbf{v}_j^{(r)})^\intercal\}_{r \in [N/(2m)]}$:
            $$ct_j := \sum_{\alpha \in [g]} \underset{\alpha b N/(2m)}{\text{Rot}} \left( \sum_{r \in [b]} \text{Mult}(\underset{(m-\alpha b)N/(2m)}{\text{Rot}}(c_{\alpha b + r}), \beta_r) \right).$$
 8:     $Ans := Ans \cup ct_j$.
 9: end for
10: Return $Ans$.
```

E Missing materials in Section 6

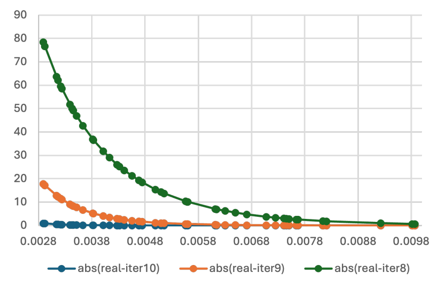
E.1 Optimized Softmax evaluation algorithm

In this section we elaborate the optimized softmax algorithm. Figure 5 shows the error of the Goldschmidt division algorithm. Curves in different colors represent different iterations in the Goldschmidt division algorithm. The y-axis is the absolute error while the x-axis is the real data. The absolute error grows rapidly if the input data is near 0. For stable solution and higher precision, we have to ensure that input data range is not close to 0, and we need at least 10 iterations in the Goldschmidt division algorithm.

Next, at a cost of adding a bootstrapping on only one ct_{sum} after the third line of Algorithm 1, it can help to reduce the ciphertext modulus in previous matrix multiplications and can significantly reduce the time cost. The detailed optimized HE evaluation of softmax is shown in Algorithm 9.



(a) Absolute error of Goldschmidt division algorithm in Softmax



(b) Details of the bottom left corner of the figure

Figure 5: Absolute error of Goldschmidt division algorithm in Softmax

E.2 Error analysis in one transformer layer

In this section, we present the error variation with a single layer. We divide one layer into 4 parts: (1) Row 1 to Row 4 in Table 3. We call it attention layer; (2) Row 5 to Row 8 in Table 3. We call it

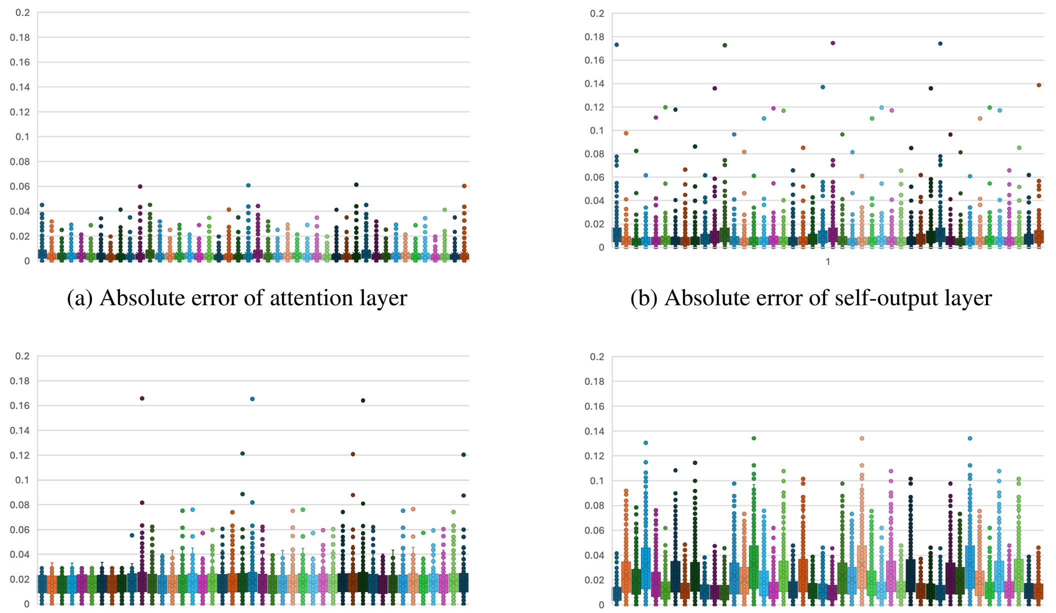
Algorithm 9 HE Softmax evaluation: $\mathrm{Enc}_{diag}(QK^\intercal) \to \mathrm{Enc}_{diag}(\sigma(QK^\intercal))$

```
Input: $\mathrm{Enc}_{diag}(QK^\intercal)$: Diagonal packing of encrypt matrix $QK^\intercal \in \mathbb{R}^{m \times m}$.
Output: $\mathrm{Enc}_{diag}(\sigma(QK^\intercal))$: Diagonal packing of encrypt matrix $\sigma(QK^\intercal) \in \mathbb{R}^{m \times m}$.
1: Write $\mathrm{Enc}_{diag}(QK^\intercal)$ as $\{\mathrm{Enc}(\mathrm{Diag}_0(QK^\intercal)), \mathrm{Enc}(\mathrm{Diag}_1(QK^\intercal)), ..., \mathrm{Enc}(\mathrm{Diag}_{m-1}(QK^\intercal))\}$.

2: Evaluate $\mathrm{Enc}(\exp(\mathrm{Diag}_i(QK^\intercal)))$ by SIMD polynomial evaluation $(1 + x/2^r)^{2^r}$.
3: Add ciphertexts to obtain $\mathrm{ct}_{sum} = \mathrm{Enc}(\sum_i \exp(\mathrm{Diag}_i(QK^\intercal)))$.
4: Bootstrap $\mathrm{ct}_{sum}$ to obtain refreshed ciphertext $\mathrm{ct}'_{sum} = \mathrm{Enc}(\sum_i \exp(\mathrm{Diag}_i(QK^\intercal)))$
5: Evaluate HE inverse algorithm on $\mathrm{ct}'_{sum}$ using the Goldschmidt division algorithm [15]. Let the
   result be $\mathrm{ct}'_{sum^{-1}} := \mathrm{Enc}(1/\sum_i \exp(\mathrm{Diag}_i(QK^\intercal)))$.
6: For $i \in [m]$, return $\mathrm{Mult}(\mathrm{ct}'_{sum^{-1}}, \mathrm{Enc}(\exp(\mathrm{Diag}_i(QK^\intercal))))$ as the $i$-th component.
```

self-output layer; (3) Row 9 to Row 10 in Table 3. We call it intermediate layer; (4) Row 11 to Row 14 in Table 3. We call it final output layer.

Figure 6 illustrated the absolute errors of evaluating the first layer with 44 input tokens. It shows that the error grows slowly along the evaluation and the medians of them fall into $[0.01, 0.03]$ in the final output of the layer, which is consistent with our error analysis of evaluating 12 layers.



(a) Absolute error of attention layer

(b) Absolute error of self-output layer

(c) Absolute error of intermediate layer

(d) Absolute error of final output layer

Figure 6: Box and whisker figure of absolute error in one layer

F Set up operations of the CKKS scheme

Finally, we give more details of the CKKS scheme [10, 11]. The set up, encryption and decryption of the CKKS scheme can be summerized as follows.

- $Setup(1^\lambda)$. The input is a security parameter λ. The output is a parameter set $parms$ of the scheme, including the ciphertext modulus Q, and polynomial degree N.

- $KeyGen(parms)$. The output includes secret key sk, public key pk and other auxiliary data called evaluation keys.

- $\mathrm{Enc}_{\mathsf{pk}}(\mathbf{v})$. Given the vector $\mathbf{v} \in \mathbb{C}^{N/2}$, the output ciphertext is ct $:= (a, b) \in R_Q^2$. For convenience, we will use $\mathrm{Enc}(\mathbf{v})$ when the public key is clear from context. We will use *naive ciphertext* to refer to $(0, 0) \in R_Q^2$ which is a ciphertext of **0**.
- $\mathrm{Dec}_{\mathsf{sk}}(\mathrm{ct})$. It outputs the vector $\mathbf{v} \in \mathbb{C}^{N/2}$. sk is omitted when it is clear from context.