THOR: Secure Transformer Inference with Homomorphic Encryption

Jungho Moon
Hanyang University
Desilo Inc.
Seoul, South Korea
jungho.moon@desilo.ai

Dongwoo Yoo
Yonsei University
Seoul, South Korea
aydw0507@yonsei.ac.kr

Xiaoqian Jiang
University of Texas, Health Science Center at Houston
Houston, TX, USA
Xiaoqian.Jiang@uth.tmc.edu

Miran Kim
Hanyang University
Seoul, South Korea
miran@hanyang.ac.kr

Abstract

As large language models are increasingly deployed in cloud environments, privacy concerns have become a significant issue. To address this challenge, we present THOR, a non-interactive framework for secure transformer inference using homomorphic encryption. We first propose efficient matrix multiplication algorithms based on diagonal-major encoding and compact ciphertext packing. We extend these basic algorithms to support plaintext-ciphertext matrix multiplication (PC-MM) using parallel submatrix computation and ciphertext-ciphertext multiplication (CC-MM) with a baby-step giant-step strategy. We also design efficient evaluation strategies for non-linear functions such as softmax, LayerNorm, GELU, and Tanh, by integrating advanced approximation techniques with adaptive iterative methods. Our matrix multiplication algorithms outperform state-of-the-art methods, achieving up to 5.3× speedup in PC-MM for $\mathbb{R}^{768\times768} \times \mathbb{R}^{768\times128}$ over BOLT (Pang et al., IEEE S&P 2024) and 9.7× in CC-MM for $12 \times (\mathbb{R}^{64\times128} \times \mathbb{R}^{128\times128})$ over Powerformer (Park et al., Preprint). THOR enables secure inference on the BERT-base model with 128 tokens in 10 minutes on a single GPU, while maintaining comparable accuracy on GLUE tasks.

1 Introduction

The rapid advancement of *transformer models* has revolutionized the way machines understand and generate sequence data, particularly in natural language processing (NLP). Transformers, built upon the self-attention mechanism [30], have demonstrated superior performance in capturing long-range dependencies compared to earlier architectures such as recurrent neural networks (RNNs) and long short-term memory (LSTM) networks. The introduction of BERT (Bidirectional Encoder Representations from Transformers) exemplified the power of transformer models, leading to significant advancements in various NLP tasks, including question answering, named entity recognition, and translation.

Despite their remarkable capabilities, deep learning models impose substantial computational demands, making their deployment in cloud environments a natural and practical choice. However, this deployment model raises serious concerns about data privacy—particularly when models are applied to sensitive user inputs. Traditional model deployments on cloud computing platforms, such as OpenAI's GPT-4o and Meta's LLaMA, do not adequately address these privacy concerns, highlighting the need for new approaches to ensure data confidentiality during inference.

To mitigate privacy concerns, numerous studies have explored the design of efficient cryptographic protocols to enable transformer inference in a privacy-preserving manner. Several approaches [1, 12, 23] leverage oblivious transfer or secret-sharing-based multi-party computation (MPC) techniques to protect private data throughout all layers, including client input. Hybrid protocols such as Iron [15] and BOLT [25] combine MPC with homomorphic encryption (HE) to optimize matrix multiplication operations via ciphertext packing techniques. Despite these advancements, the interactive nature of MPC leads to substantial communication overheads. For instance, BOLT incurs approximately 25.74 GB of communication for end-to-end inference on the BERT-base model.

Recently, non-interactive methods based on HE have emerged as promising alternatives, enabling computation directly on encrypted data. These approaches allow for inference to be outsourced without compromising the data privacy. Two recent works [26, 29] are HE-based systems that perfectly protect data privacy during transformer inference. However, both focus on the BERT-tiny model, which limits the scalability and efficiency of their matrix multiplication algorithms when applied to larger transformer architectures. In particular, their row-wise matrix encoding and multiplication algorithms are not optimized for large-scale multi-head attention, which necessitates efficient parallel matrix computation at scale. Furthermore, their non-linear approximation methods are restricted to a relatively small domain, limiting their effectiveness for larger models.

On the other hand, NEXUS [31] presents the first demonstration of secure transformer inference on the BERT-base model. However, it has several limitations: (a) Despite its high throughput, it requires a substantial number of operations even for a single prediction, leading to high latency—2.7 hours per query; (b) It does not provide an end-to-end matrix multiplication method for secure inference. While NEXUS adopts multiple packing strategies (component-wise, row-wise, column-wise, and diagonal-wise), the transformations between these formats during secure inference are not explicitly explained.

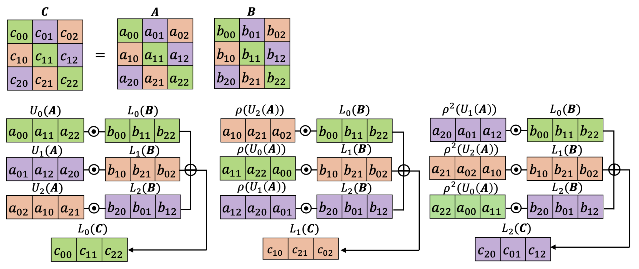

C
A
B
c_{00} c_{01} c_{02}
c_{10} c_{11} c_{12}
c_{20} c_{21} c_{22}
=
a_{00} a_{01} a_{02}
a_{10} a_{11} a_{12}
a_{20} a_{21} a_{22}
b_{00} b_{01} b_{02}
b_{10} b_{11} b_{12}
b_{20} b_{21} b_{22}
$U_0(A)$
$L_0(B)$
$U_1(A)$
$L_1(B)$
$U_2(A)$
$L_2(B)$
$L_0(C)$
$\rho(U_2(A))$
$\rho(U_0(A))$
$\rho(U_1(A))$
$L_1(C)$
$\rho^2(U_1(A))$
$\rho^2(U_2(A))$
$\rho^2(U_0(A))$
$L_2(C)$

Figure 1: Our matrix multiplication algorithm with dimension $n = 3$. Here, $U_i(A)$ and $L_j(B)$ denote the i-th upper and j-th lower diagonals of A and B, respectively. The operators $\oplus$ and $\odot$ represent entrywise addition and multiplication. We use ρ^k to denote the left rotation of a vector by k positions.

1.1 Our contributions

In this study, we present THOR, a secure Transformer inference framework that leverages HomomOrphic encRyption to ensure data privacy. Our contributions are three-fold.

- **Fast homomorphic matrix multiplication.** We present homomorphic matrix multiplication algorithms aimed at reducing the computational overhead of encrypted operations. In a nutshell, we formulate matrix multiplication as vectorized arithmetic operations along matrix diagonals, enabling efficient implementation using input ciphertext rotations and homomorphic multiplications. Specifically, each diagonal vector of the matrix product AB can be computed by summing the entrywise products of the rotated upper diagonals of A and the lower diagonals of B (Fig. 1). To further enhance performance, we utilize all plaintext slots by compactly packing multiple diagonal vectors from different matrices into a single ciphertext, enabling parallel computations over the packed ciphertexts.
  Based on this insight, we present a plaintext-ciphertext matrix multiplication (PC-MM) algorithm that performs parallel multiplications between submatrices and aggregates intermediate results to generate the final output. This blockwise strategy reduces the number of required rotations on the input ciphertexts from the original matrix size to the submatrix size, significantly improving efficiency. We also propose a ciphertext-ciphertext matrix multiplication (CC-MM) based on a *baby-step giant-step* (BSGS) strategy, which defers input ciphertext rotations and instead applies them to partially accumulated results at once. As a result, the number of key-switching operations for computing $\mathbb{R}^{m\times n} \times \mathbb{R}^{n\times n}$ is reduced by a factor of $2s/n$, where s denotes the number of plaintext slots.
  While existing approaches such as BOLT and Powerformer are optimized for either PC-MM or CC-MM, our method consistently outperforms both across all matrix operations required for multi-head attention. Specifically, our approach achieves 5.3× and 11.2× speedups in PC-MM for $\mathbb{R}^{768\times 768} \times \mathbb{R}^{768\times 128}$ compared to BOLT and Powerformer, respectively. For CC-MM involving $12 \times (\mathbb{R}^{64\times 128} \times \mathbb{R}^{128\times 128})$, our method yields 36.0× and 9.7× speedups over BOLT and Powerformer, respectively.
- **Accurate and efficient non-linear homomorphic computations.** We propose an efficient *two-phase method* for evaluating the softmax function over large intervals: we approximate the exponential function over a relatively small interval and gradually extend the approximate range through repeated *square-and-normalize* operations. Specifically, we adopt Goldschmidt's iterative method to approximate the inverse while adaptively adjusting the relaxation coefficient for faster convergence. Compared with the state-of-the-art approach [10], our method achieves a 2.64× speedup when computing 256 softmax operations with input dimension of 128. We also apply the adaptive iterative method to efficiently compute the inverse square root for the LayerNorm. Another contribution is the evaluation of the GELU (Gaussian Error Linear Unit) and Tanh functions by approximating them as compositions of two low-degree polynomials.
- **End-to-end transformer inference.** We design an end-to-end HE-based framework for secure transformer inference. The experimental results show that THOR provides secure inference on 128 tokens with the BERT-base model (12 attention layers and 768 hidden dimensions) with a latency of 10 minutes on a single GPU, while maintaining a comparable inference accuracy.

1.2 Related Work

1.2.1 Homomorphic Matrix Multiplication. Several works have investigated how to perform linear transformations on encrypted vectors [13, 14, 18], which can be naturally extended to the matrix-matrix setting. Jiang et al. [16] introduced a general method for encrypted matrix multiplication by expressing it as HE-friendly operations over packed ciphertexts, which was optimized by Powerformer [26]. However, its performance remains suboptimal, particularly in the plaintext-ciphertext setting, as it inherits limitations from the underlying algorithm of [16], which was originally designed for CC-MM. Another recent work by Chen et al. [6] optimized Halevi-Shoup's matrix-vector algorithm [13]. However, this approach has several limitations: (a) the matrix dimensions must be co-prime and (b) the matrix is represented as a one-dimensional vector using the Chinese Remainder Theorem. When the matrix is too large to fit into a single ciphertext, it should be processed in a blockwise manner, making it less suitable for large-scale matrix multiplication.

Recent works such as [3, 27] introduced a new matrix multiplication framework that reduces computation to plaintext linear algebra by leveraging multi-secret variants of RLWE to compactly represent multiple ciphertexts. However, the dimensions of the operations involved in this reduction do not match the actual dimensions of the matrix, which can increase the computational complexity. This limitation is especially evident when the matrix dimension is smaller than the ring degree of the underlying HE scheme, as is the common case in secure transformer inference scenarios.

On the other hand, homomorphic matrix multiplication algorithms have been explored for hybrid HE-MPC implementation frameworks, such as Iron [15] and BOLT [25]. While BOLT achieves state-of-the-art performance for PC-MM, it still incurs high computational complexity in CC-MM. This inefficiency stems from its design focus on minimizing communication overhead in MPC by reducing the multiplicative depth of matrix multiplication to accommodate the constraints of HE parameters in an MPC setting. However, it leads to degraded computational performance.

1.2.2 Softmax Computation. One way of implementing the softmax function on encrypted data is to replace it with HE-friendly alternatives such as sigmoid function [17], Gumble softmax function [20], and a quadratic polynomial [2]. More recently, ReLU-based functions [26, 32] and Gaussian kernels [28] have been used to avoid division, but their effectiveness has not been validated on larger BERT models. Another line of work focuses on accurately approximating the softmax function. To avoid overflow and enhance the numerical stability of the exponential function, it is common practice to subtract the row maximum before exponentiation. However, in the encrypted domain, computing the exact maximum requires a substantial number of comparison operations. NEXUS [31] instead uses a fixed constant, but if this offset is too large, the exponentials vanish within the precision limits, causing the outputs to collapse to zero. Thus, the fixed-constant approach introduces numerical instability and limits applicability.

The most closely related approach to ours is that of Cho et al. [10], which proposed an iterative method for the softmax approximation based on a normalize-and-square strategy. Compared to their normalization using inverse square roots, our approach adopts a square-and-normalize strategy based on an inverse approximation and incorporates the adaptive iterative method from [24] to achieve more efficient and stable normalization. This formulation enables faster convergence over a wider input range with fewer iterations, thereby significantly reducing computational overhead and improving efficiency.

2 Preliminaries

We denote the binary logarithm by $\log(\cdot)$. We denote by $[n]$ the set $\{0, 1, \ldots, n-1\}$ where $n \in \mathbb{N}$. $a \pmod{n}$ or $[a]_n$ denotes the unique integer r such that $r \in [n]$ and $r = a \pmod{n}$. The entry of the i-th row and j-th column of an $m \times n$ matrix A is denoted by $A[i, j]$ or A_{ij}. A_i and A^j denote the i-th row and j-th column vectors. $\langle u, v\rangle$ denotes the inner product. If two matrices A and B have the same number of rows, $(A|B)$ denotes a matrix formed by horizontal concatenation.

2.1 Homomorphic Encryption

The CKKS scheme [9] supports approximate arithmetic on encrypted complex numbers and enables parallel computation in a single instruction multiple data (SIMD) manner. It provides the following basic arithmetic and advanced operations.

- Vectorized arithmetic operations: Entrywise addition and multiplication (denoted by + and $\odot$) can be performed between two ciphertexts (`Add` and `Mult`), or between a ciphertext and a plaintext (`PAdd` and `PMult`).
- Rotation: Given a ciphertext ct of $\mathbf{m} \in \mathbb{C}^s$, $\texttt{Rot}(\texttt{ct}; r)$ returns a ciphertext of the left rotation $\rho^r(\mathbf{m})$ by r amounts.
- Key-switching: A ciphertext is transformed into a ciphertext of the same message, but under a different secret key. For example, homomorphic multiplication involves a key-switching operation—commonly referred to as *relinearization*—which converts the result back to a normal ciphertext under the original secret after performing a tensor product of the input ciphertexts.
- Bootstrapping: The bootstrapping operation refreshes a ciphertext by increasing its coefficient modulus, thereby enabling additional computations without a significant loss of precision.

2.2 Transformers

The transformer is based on an encoder-decoder architecture, with both parts having a similar structure. Therefore, we mainly focus on the encoder. The encoder is composed of a stack of L layers, each comprising *multi-head attention* followed by two normalizations, with feed-forward layers in between. Given the number of tokens n and the model dimension d, multi-head attention operates as follows: (a) the input sequence $X \in \mathbb{R}^{n\times d}$ is linearly projected to H heads so that the query matrix $Q_h = XW_{Q,h}$, the key matrix $K_h = XW_{K,h}$, and the value matrix $V_h = XW_{V,h}$ with the parameter matrices $W_{Q,h}, W_{K,h} \in \mathbb{R}^{d\times d_k}$, and $W_{V,h} \in \mathbb{R}^{d\times d_v}$ for $h \in [H]$; (b) the scaled dot-product attention is computed in each head:

$$\texttt{ATTENTION}(Q_h, K_h, V_h) = \texttt{Softmax}\left(Q_h {K_h}^\intercal / \sqrt{d_k}\right) \times V_h;$$

and (c) the results of multiple heads are concatenated and linearly transformed by $W^O \in \mathbb{R}^{Hd_v\times d}$. Following the parameters from [30], we use $d_k = d_v = d/H$. A feed-forward layer consists of two linear transformations with a GELU activation in between. The first token from the L encoder layers is fed into a pooler, which is a dense layer followed by a `Tanh` activation. The output is then fed into a classifier, which is another dense layer.

3 Basic Matrix Multiplication

In this section, we introduce HE-friendly matrix multiplication algorithms, which serve as the foundation of the linear layers in secure transformer inference.

3.1 Definitions & Lemmas

Definition 3.1. For a matrix $A \in \mathbb{R}^{a\times b}$ and $0 \le k < \min\{a, b\}$, the k-th upper diagonal vector $\mathsf{U}_k(A) \in \mathbb{R}^{\max\{a,b\}}$ is defined by $\mathsf{U}_k(A)[t] = A[t \pmod{a}, k + t \pmod{b}]$. For a matrix $B \in \mathbb{R}^{b\times c}$ and $0 \le \ell < \min\{b, c\}$, the ℓ-th lower diagonal vector $\mathsf{L}_\ell(B) \in \mathbb{R}^{\max\{b,c\}}$ is defined by $\mathsf{L}_\ell(B)[t] = B[\ell + t \pmod{b}, t \pmod{c}]$.

We start with useful lemmas that characterize the upper and lower diagonal vectors, which are later used for homomorphic matrix multiplications.

Lemma 3.2. *For any $a, b \in \mathbb{R}^n$ and $r \in \mathbb{Z}$, we have $\langle a, b\rangle = \langle \rho^r(a), \rho^r(b)\rangle$. That is, $\rho^r(\cdot) : \mathbb{R}^n \to \mathbb{R}^n$ is an isometry on the real inner product space $\mathbb{R}^n$.*

Lemma 3.3. *For any $A \in \mathbb{R}^{a\times b}$ and $k \in \mathbb{Z} \cap [0, \min\{a, b\})$, we have $\mathsf{U}_k(A) = \mathsf{L}_k(A^\intercal)$.*

Lemma 3.4. *Suppose that n is divisible by m. Then we have*

$$\mathsf{L}_\ell(A) = \begin{cases} \rho^{-m+\ell}(\mathsf{U}_{m-\ell}(A)) & \text{if } A \in \mathbb{R}^{m\times n}, \\ \rho^{-n+\ell}(\mathsf{U}_{m-\ell}(A)) & \text{if } A \in \mathbb{R}^{n\times m}. \end{cases}$$

Lemma 3.5. *For a square matrix $A \in \mathbb{R}^{n\times n}$ and any $t \in [n]$, we have $(\mathsf{U}_k(A)[t])_{k\in[n]} = \rho^t(A_t) \in \mathbb{R}^n$.*

3.2 Matrix Multiplication

In this section, we propose matrix multiplication algorithms for the case when the diagonal vectors of two input matrices are provided. In a nutshell, it follows from Lemma 3.2 that, given two square matrices of A and B of size n, we have

$$\left\langle \rho^t(A_{r+t}), \rho^t(B^t) \right\rangle = \left\langle A_{r+t}, B^t \right\rangle = AB[r+t, t] = \mathsf{L}_r(AB)[t] \quad (1)$$

for $r, t \in [n]$. Note that the ℓ-th entries of $\rho^t(A_{r+t})$ and $\rho^t(B^t)$ correspond to the t-th entries of $\rho^r(\mathsf{U}_{\ell-r}(A))$ and $\mathsf{L}_\ell(B)$, respectively. Consequently, the t-th entry of $\mathsf{L}_r(AB)$ corresponds to the t-th entry of the vector obtained by summing the entrywise products of $\rho^r(\mathsf{U}_{\ell-r}(A))$ and $\mathsf{L}_\ell(B)$ for $\ell \in [n]$. In a SIMD fashion, this can be formulated as the following propositions.

Proposition 3.6. *For $n \geq m$, suppose that n is divisible by m. Given two matrices $A \in \mathbb{R}^{n\times m}$ and $B \in \mathbb{R}^{m\times n}$, for $r \in [n]$, the r-th lower diagonal vector of the matrix AB can be computed as follows:*

$$\mathsf{L}_r(AB) = \sum_{\ell\in[m]} \rho^r(\mathsf{U}_{\ell-r}(A)) \odot \mathsf{L}_\ell(B), \quad (2)$$

where the index of $\mathsf{U}_k(A)$ is taken modulo m.

Proof. See Appendix A.1. □

Proposition 3.7. *For $n \geq m$, suppose that n is divisible by m. Given two matrices $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{n\times n}$, for $r \in [m]$, the r-th lower diagonal vector of the matrix AB can be computed as follows:*

$$\mathsf{L}_r(AB) = \sum_{\ell\in[n]} \rho^r(\mathsf{U}'_{\ell-r}(A)) \odot \mathsf{L}_\ell(B), \quad (3)$$

where the upper diagonal vector $\mathsf{U}'_k(A) \in \mathbb{R}^n$ is defined by

$$\mathsf{U}'_k(A) = \begin{cases} \mathsf{U}_k(A) & \text{if } k < m, \\ \rho^{m\cdot\lfloor k/m \rfloor}(\mathsf{U}_{(k \bmod m)}(A)) & \text{if } m \leq k < n. \end{cases} \quad (4)$$

Proof. See Appendix A.2. □

We now propose matrix multiplication algorithms for lower-diagonal inputs.

Corollary 3.8. *For $n \geq m$, suppose that n is divisible by m. Given two matrices $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{n\times n}$, for $r \in [m]$, the r-th lower diagonal vector of the matrix AB can be computed as follows:*

$$\mathsf{L}_r(AB) = \sum_{\ell\in[n]} \rho^\ell(\mathsf{L}_{r-\ell}(A)) \odot \mathsf{L}_\ell(B), \quad (5)$$

where the index of $\mathsf{U}_k(A)$ is taken modulo m.

Proof. See Appendix A.3. □

Corollary 3.9. *For $n \geq m$, suppose that n is divisible by m. Given two matrices $A \in \mathbb{R}^{n\times m}$ and $B \in \mathbb{R}^{m\times n}$, for $r \in [n]$, the r-th lower diagonal vector of the matrix AB can be computed as follows:*

$$\mathsf{L}_r(AB) = \sum_{\ell\in[m]} \rho^\ell(\mathsf{L}'_{r-\ell}(A)) \odot \mathsf{L}_\ell(B), \quad (6)$$

where the lower diagonal vector $\mathsf{L}'_k(A) \in \mathbb{R}^n$ is defined by

$$\mathsf{L}'_k(A) = \begin{cases} \mathsf{L}_k(A) & \text{if } k < m, \\ \rho^{m\cdot\lfloor k/m \rfloor}(\mathsf{L}_{(k \bmod m)}(A)) & \text{if } m \leq k < n. \end{cases}$$

Proof. See Appendix A.4. □

In the context of HE, the aforementioned algorithms are HE-friendly, as they can be efficiently implemented using input rotations and homomorphic multiplications. Specifically, assume that the number of slots is equal to n (that is, each diagonal vector is fully packed in encrypted form). If a matrix A is provided in cleartext, Eq. (2) can be evaluated as follows: $\sum_{\ell\in[m]} \texttt{PMult}(\mathsf{ct.B}_\ell, \mathsf{pt.A}_{\ell,r})$, where $\mathsf{pt.A}_{\ell,r}$ and $\mathsf{ct.B}_\ell$ denote the plaintext of $\rho^r(\mathsf{U}_{\ell-r}(A))$ and a ciphertext of $\mathsf{L}_\ell(B)$, respectively. The resulting ciphertext corresponds to an encryption of $\mathsf{L}_r(AB)$. Alternatively, when both inputs are provided in encrypted form, it can be computed as follows: $\sum_{\ell\in[m]} \texttt{Mult}(\texttt{Rot}(\mathsf{ct.A}_{\ell-r}; r), \mathsf{ct.B}_\ell)$, where $\mathsf{ct.A}_\ell$ represents an encryption of $\mathsf{U}_\ell(A)$. As such, our matrix multiplication algorithms preserve the diagonal structure of the matrix product within the encrypted domain.

Remark 1. It follows from Lemma 3.5 that the vector formed by the t-th entries of each upper diagonal vector corresponds to a left rotation by t positions of the t-th row of a matrix $A \in \mathbb{R}^{n\times n}$. Therefore, if the target function $f(X)$ is applied row-wise to A, it can be homomorphically evaluated directly on the encrypted upper diagonal vectors. Specifically, let $\alpha = f(A)$ denote the result of applying f to each row of A, and assume that the upper diagonals of A are provided in encrypted form, where $\mathsf{ct.A}_k$ denotes an encryption of the k-th upper diagonal. Then the evaluated result $f(\mathsf{ct.A}_0, \ldots, \mathsf{ct.A}_{n-1})$ corresponds to the upper diagonals of α.

4 Homomorphic Matrix Multiplication

In this section, we describe how to evaluate the proposed matrix multiplication algorithms, incorporating compact ciphertext packing and SIMD techniques.

4.1 Multi-diagonal Batched Encryption

Throughout Section 3.2, we assume that the matrix dimension is equal to the number of plaintext slots. However, in many real-world applications including transformer inference, the slot parameter is chosen to be sufficiently large to support the evaluation of deep circuits. In such cases, if each ciphertext encrypts only one diagonal of a matrix, most of the slots remain unused. To address this inefficiency, we pack diagonals from multiple matrices in a single plaintext, enabling efficient *parallel* matrix computations. Furthermore, we introduce a *compact packing* method that encodes as many packed diagonal vectors as possible into a single ciphertext.

Given matrices $A^{(z)} \in \mathbb{R}^{m\times n}$ with $m \leq n$ for $z \in [H]$, we assume that the slot parameter s is divisible by nH. Suppose that $s = cnH$ for some integer c, and denote $A = (A^{(z)})_{z\in[H]}$. First, for $\ell \in [m]$, the ℓ-th lower diagonal vectors from the matrices $A^{(z)}$ are interlaced to form a batched vector $\bar{\mathsf{L}}_\ell(A) \in \mathbb{R}^{nH}$, defined as $\bar{\mathsf{L}}_\ell(A)[t] = \mathsf{L}_\ell(A^{(\bar{t})})[\lfloor t/H \rfloor]$ where $\bar{t} = t \pmod H$ for $t \in [nH]$. Next, multiple batched vectors are concatenated to be fully-packed into a single ciphertext. For simplicity of presentation, we assume that c divides m and denote $m_c = m/c$. For $j \in [m_c]$, we define the concatenation of the interlaced lower diagonals as

$$\widehat{\mathsf{L}}_j(A) = (\bar{\mathsf{L}}_{cj}(A) | \bar{\mathsf{L}}_{cj+1}(A) | \ldots | \bar{\mathsf{L}}_{c(j+1)-1}(A)) \in \mathbb{R}^s, \quad (7)$$

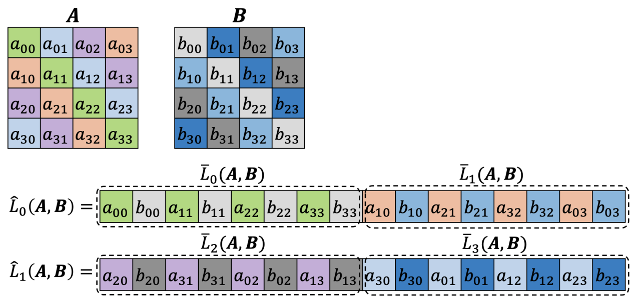

A
B
a_{00} a_{01} a_{02} a_{03}
a_{10} a_{11} a_{12} a_{13}
a_{20} a_{21} a_{22} a_{23}
a_{30} a_{31} a_{32} a_{33}
b_{00} b_{01} b_{02} b_{03}
b_{10} b_{11} b_{12} b_{13}
b_{20} b_{21} b_{22} b_{23}
b_{30} b_{31} b_{32} b_{33}
$\bar{L}_0(A,B)$
$\bar{L}_1(A,B)$
$\hat{L}_0(A,B) =$ a_{00} b_{00} a_{11} b_{11} a_{22} b_{22} a_{33} b_{33} a_{10} b_{10} a_{21} b_{21} a_{32} b_{32} a_{03} b_{03}
$\bar{L}_2(A,B)$
$\bar{L}_3(A,B)$
$\hat{L}_1(A,B) =$ a_{20} b_{20} a_{31} b_{31} a_{02} b_{02} a_{13} b_{13} a_{30} b_{30} a_{01} b_{01} a_{12} b_{12} a_{23} b_{23}

Figure 2: Illustration of the multi-diagonal batched encoding when $m = n = 4, H = 2, s = 16$ and $c = s/nH = 2$.

which we refer to as the *multi-diagonal batched encoding* of A. This encoding strategy maximizes SIMD utilization by parallelizing H operations across multiple matrices and packing c diagonals into a single ciphertext. We refer to c as the *diagonal packing capacity*. This encoding method is illustrated in Fig. 2.

4.2 Plaintext-ciphertext Matrix Multiplication

We observe that all the plaintext-ciphertext matrix multiplications in the linear layers of the BERT model can be expressed as $C = AB$, where $A \in \mathbb{R}^{d\times d}$ is provided in plaintext (e.g., a weight matrix) and $B \in \mathbb{R}^{d\times n}$ is provided in encrypted form (e.g., an encrypted output from the previous layer), given that $n \leq d$ (see Section 6.2 for details). Without loss of generality, we assume that d is divisible by n and denote $d_n = d/n$; otherwise, zeros can be padded.

4.2.1 Extension of Basic Matrix Multiplication. One approach is to extend the basic matrix multiplication algorithms in Section 3.2 to support this type of matrix multiplication. Similar to Proposition 3.7, for $r \in [n]$, the r-th lower diagonal of C in $\mathbb{R}^d$ can be computed as

$$\mathsf{L}_r(C) = \sum_{\ell\in[d]} \rho^r(\mathsf{U}_{\ell-r}(A)) \odot \mathsf{L}'_\ell(B), \tag{8}$$

where $\mathsf{L}'_\ell(B) = \rho^{n\cdot\lfloor \ell/n\rfloor}(\mathsf{L}_{\ell \bmod n}(B))$. To utilize all available slots, c diagonal vectors of dimension d from the matrix B are fully packed into a ciphertext, where c is an integer such that $s = cd$. For simplicity, we assume that c divides n, and define $n_c = n/c$ and $d_c = d/c$. Given a ciphertext $\mathsf{ct.B}_j$ of the multi-diagonal vector $(\mathsf{L}_{cj}(B)|\dots|\mathsf{L}_{c(j+1)-1}(B)) \in \mathbb{R}^{c\cdot d} = \mathbb{R}^s$ for $j \in [n_c]$, we first need encryptions of the extended diagonal $\mathsf{L}'_\ell(B)$ in a multi-diagonal format. Specifically, for each $t \in [1, d_n)$ and $j \in [n_c]$, we generate a ciphertext $\mathsf{ct.B}_{n_c\cdot t+j}$ of $(\mathsf{L}'_{nt+cj}(B)|\dots|\mathsf{L}'_{nt+c(j+1)-1}(B))$. This requires rotations within a subslot, which are not natively supported in CKKS, as it only provides one-dimensional rotations. To address this, we apply an *internal rotation* technique[1], which can be implemented using two homomorphic rotations and one plaintext multiplication. Once the extended diagonals are obtained in packed format, the r-th multi-diagonal encoding of C for $r \in [n_c]$ can be computed as follows:

$$\sum_{j\in[d_c],\ell\in[c]} \mathtt{PMult}(\mathtt{Rot}(\mathsf{ct.B}_j; d\cdot\ell), \mathsf{pt.A}_{j,\ell,r}), \tag{9}$$

where $\mathsf{pt.A}_{j,\ell,r}$ denotes a plaintext derived appropriately from A.

Since the internal rotation operation is performed for each $t \in [1, d_n]$ on $\mathsf{ct.B}_j$ for $j \in [n_c]$, this step requires $2(d_n - 1)\cdot n_c \approx 2d_c$ rotations. Next, in Eq. (9), each ciphertext $\mathsf{ct.B}_j$ for $j \in [d_c]$ is rotated $(c-1)$ times, incurring additional $(c-1)\cdot d_c \approx d$ rotations. In total, the number of rotations is approximately $2d_c + d$.

[1]For example, given a ciphertext of the vector $(u_0, u_1, u_2, u_3, v_0, v_1, v_2, v_3)$, we first apply a masking operation to extract $(0, u_1, u_2, u_3, 0, v_1, v_2, v_3)$ and subtract it from the original ciphertext to obtain $(u_0, 0, 0, 0, v_0, 0, 0, 0)$. Next, we rotate the first ciphertext to the left by one and the second ciphertext to the right by three. By adding these two rotated ciphertexts, we obtain a ciphertext of $(u_1, u_2, u_3, u_0, v_1, v_2, v_3, v_0)$.

Algorithm 1 PLAINTEXT-CIPHERTEXT MATMUL

Input: Plaintexts $\mathsf{pt.A}_{i,j,\ell,r}$ of the internally rotated upper diagonal vectors of $A \in \mathbb{R}^{d\times d}$; ciphertexts $\mathsf{ct.B}_j$ of the multi-lower-diagonal batched encodings of $B = (B_k)$ for $i \in [d/n], j, r \in [n/c], \ell \in [c]$ with $B_k \in \mathbb{R}^{n\times n}$ for $k \in [d/n]$.
Output: Ciphertexts $\mathsf{ct.C}_r$ of the multi-lower-diagonal batched encodings of $C = (C_k)$ for $r \in [n/c]$ with $C_k \in \mathbb{R}^{n\times n}$ for $k \in [d/n]$.

```
1: $d_n := d/n, n_c := n/c$
2: for $j = 0$ to $n_c - 1$ do
3:     $\mathsf{ct.Br}_{j,0} := \mathsf{ct.B}_j$
4:     for $\ell = 1$ to $c - 1$ do
5:         $\mathsf{ct.Br}_{j,\ell} \leftarrow \mathtt{Rot}(\mathsf{ct.B}_j; d\cdot\ell)$
6: for $i = 0$ to $d_n - 1$ do
7:     for $r = 0$ to $n_c - 1$ do
8:         for $j = 0$ to $n_c - 1$, $\ell = 0$ to $c - 1$ do
9:             $\mathsf{ct}_{i,j,\ell,r} \leftarrow \mathtt{PMult}(\mathsf{ct.Br}_{j,\ell}, \mathsf{pt.A}_{i,j,\ell,r})$
10:        $\mathsf{ct}_{i,r} \leftarrow \sum_{j\in[n_c],\ell\in[c]} \mathsf{ct}_{i,j,\ell,r}$
11: for $r = 0$ to $n_c - 1$ do
12:     for $i = 1$ to $d_n - 1$ do
13:         $\mathsf{ct}_{i,r,R} \leftarrow \mathtt{PMult}(\mathsf{ct}_{i,r}, \mathsf{pt}_i)$
14:         $\mathsf{ct}_{i,r,L} \leftarrow \mathsf{ct}_{i,r} - \mathsf{ct}_{i,r,R}$
15:         $\mathsf{ct}'_{i,r} \leftarrow \mathtt{Rot}(\mathsf{ct}_{i,r,R}; d(d_n - i)) + \mathtt{Rot}(\mathsf{ct}_{i,r,L}; -di)$
16:     $\mathsf{ct.C}_r \leftarrow \mathtt{PMult}(\mathsf{ct}_{0,r}, \mathsf{pt}_0) + \mathsf{ct}'_{i,r}$
17: return $\mathsf{ct.C}_r$
```

4.2.2 Diagonal Block Matrix Computation. To further enhance efficiency, we partition the input matrices into $n\times n$ blocks and compute the matrix product via parallel multiplications of these submatrices.

We denote the (i,k)-th submatrix of A by $A_{ik} \in \mathbb{R}^{n\times n}$ and the k-th submatrix of B by $B_k \in \mathbb{R}^{n\times n}$ for $i, k \in [d_n]$. The j-th multi-diagonal batched encoding of $B = (B_k)_{k\in[d_n]}$ is defined as $\widehat{\mathsf{L}}_j(B) = (\bar{\mathsf{L}}_{cj}(B)|\bar{\mathsf{L}}_{cj+1}(B)|\dots|\bar{\mathsf{L}}_{c(j+1)-1}(B)) \in \mathbb{R}^s$, and its encryption is denoted by $\mathsf{ct.B}_j$ for $j \in [n_c]$. Similarly, we define $A^{(i)} = (A_{i+k,k})_{k\in[d_n]}$ as the set of submatrices along the i-th diagonal, and the plaintext of their interlaced diagonal vectors for $A^{(i)}$ is denoted by $\mathsf{pt.A}_{i,j,\ell,r}$ for $j, r \in [n_c]$ and $\ell \in [c]$. The i-th submatrix of the product $C = AB$ is computed via blockwise multiplication as $C_i = \sum_k A_{ik}B_k$ for $i \in [d_n]$. As illustrated in Fig. 3, this blockwise computation can be efficiently implemented as follows:

- **Step 1.** Let $C_{i,k} = A_{i,k}B_k \in \mathbb{R}^{n\times n}$ and $C^{(i)} = (C_{i+k,k})_{k\in[d_n]}$. We perform parallel matrix multiplications between submatrices aligned along the same diagonal. To this end, we first rotate the

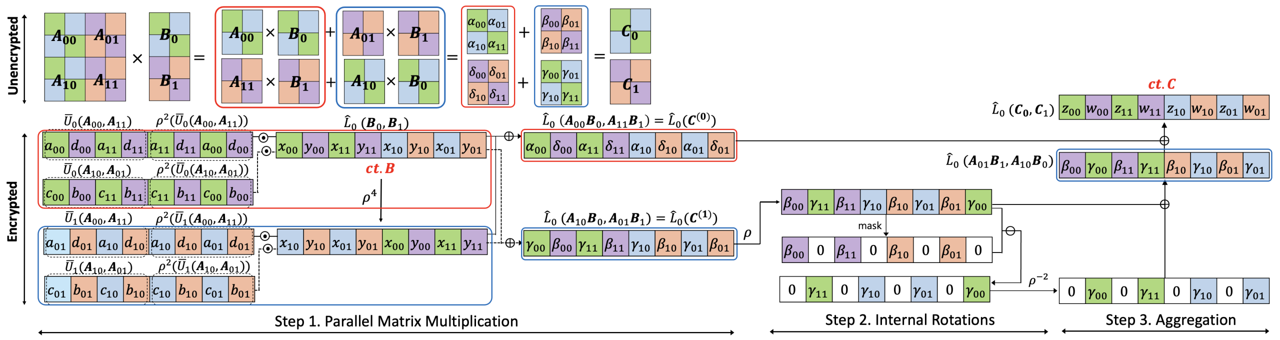

Unencrypted
Encrypted
ct.B
ct.C
Step 1. Parallel Matrix Multiplication
Step 2. Internal Rotations
Step 3. Aggregation
mask

Figure 3: Diagonal block PC-MM with parameters $n = 2$, $d = 4$, $s = 8$, and $c = s/d = 2$. The ciphertexts ct.B and ct.C represent the multi-diagonal batched encodings $\widehat{\mathsf{L}}_0(B_0, B_1) = (\mathsf{L}_0(B_0, B_1)|\mathsf{L}_1(B_0, B_1))$ and $\widehat{\mathsf{L}}_0(C_0, C_1) = (\mathsf{L}_0(C_0, C_1)|\mathsf{L}_1(C_0, C_1))$, respectively.

Table 1: Comparison of the PC-MM methods of AB where $A \in \mathbb{R}^{d\times d}$ and $B \in \mathbb{R}^{d\times n}$ with $n \le d$ and $c = s/d$.

Method	#Pmult[†]	#Rot[‡]
Basic	$\frac{2d}{c} + \frac{dn}{c}$	$d + \frac{2d}{c}$
DiagBlock	$\frac{d}{c} + \frac{dn}{c}$	$n + \frac{2d}{c}$

†: Plaintext-ciphertext multiplication, ‡: Rotation

input ciphertexts of B and evaluate Eq. (2) to obtain the multi-diagonal batched vector $\widehat{\mathsf{L}}_r(C^{(i)})$ for $i \in [d_n]$ and $r \in [n_c]$ (lines 2 and 10 in Algorithm 1).

- **Step 2**. We internally rotate the resulting partial submatrix products within output diagonals of the same order, yielding the result in the order $(C_{0,d_n-i}, \dots, C_{i,0}, \dots, C_{d_n-1,d_n-1-i})$ (lines 11 to 15).
- **Step 3**. Aggregate the intermediate results to produce the final result (line 16).

Parallel submatix multiplication requires approximately $(c-1) \cdot n_c \approx n$ rotations for input ciphertexts. The second internal rotation process involves $2(d_n - 1) \cdot n_c \approx 2d/c$ rotations to compute all aligned results of the submatrix products. While additional computations are needed to rotate slots for interactions between values from different submatrices, the intermediate submatrix products of dimension n can be parallelized in a SIMD manner. As a result, the number of rotations is reduced from $2d/c + d = O(d)$ in the naive method to $n + 2d/c = O(n)$. We summarize the results in Table 1.

4.3 Ciphertext-ciphertext Matrix Multiplication

When two input matrices are provided in encrypted form, the upper-lower matrix multiplication algorithms can be applied in a similar manner to the plaintext-ciphertext case. However, it is non-trivial to generate the required ciphertexts from the input encryptions. To reduce computational complexity, we leverage the lower diagonals of the input matrices with optimized cryptographic techniques. For simplicity, we consider the matrix multiplication between $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{n\times n}$ under the assumption that $m \le n$ and $s = cn$ for some integer c. For simplicity of presentation, we first focus on the case of a single matrix multiplication. For the sake of brevity, we abuse the notation of $\cdot$ to denote plaintext-ciphertext multiplication.

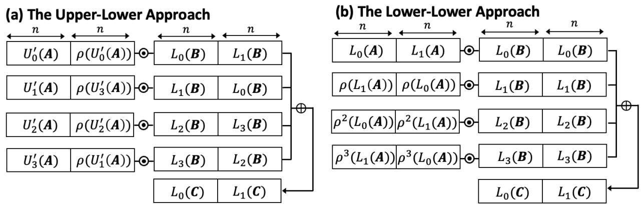


Figure 4: The CC-MM methods of $C = AB$ where $A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{n\times n}$ with $m = 2n, s = 2n$, and $c = s/n = 2$. (a) The upper-lower approach where $\mathsf{U}'_k(A) \in \mathbb{R}^n$ is defined in Eq. (4). (b) The lower-lower basic approach without the BSGS strategy.

4.3.1 *Basic Ciphertext-ciphertext Matrix Multiplication.*

Let $m_c = m/c$ and $n_c = n/c$. Suppose that we are given the encrypted multi-diagonal encodings of $(\mathsf{U}_{cj}(A)|\dots|\mathsf{U}_{c(j+1)-1}(A))$ and $(\mathsf{U}_{c\ell}(B)|\dots|\mathsf{U}_{c(\ell+1)-1}(B))$ for $j \in [m_c]$ and $\ell \in [n_c]$. Our goal is to securely compute the multi-diagonal encoding $(\mathsf{L}_{cj}(C)|\dots|\mathsf{L}_{c(j+1)-1}(C))$ for $j \in [m_c]$. According to Proposition 3.7, this can be efficiently achieved by evaluating the following computation in a packed manner: $\mathsf{L}_{cj+r}(C) = \sum_{\ell\in[n]} \rho^{cj+r}(\mathsf{U}'_{\ell-cj-r}(A)) \odot \mathsf{L}_\ell(B)$ for $r \in [c]$. As illustrated in Fig. 4(a), the r-th subslots of length n across the ciphertexts for A correspond to the diagonals of the matrix A rotated by the r positions. In other words, each ciphertext should contain the same diagonal of A, but rotated differently across the subslots. The resulting ciphertexts are then multiplied by the appropriately rotated ciphertexts of B. In conclusion, the upper-lower matrix multiplication method requires approximately $(m+1)n$ rotations, mn plaintext multiplications, and mn/c ciphertext multiplications. A detailed complexity analysis is provided in Appendix B.1.

Now suppose that we are given ciphertexts $\mathsf{ct.A}_j$ representing the multi-lower-diagonal vector $\widehat{\mathsf{L}}_j(A) = (\mathsf{L}_{cj}(A)|\dots|\mathsf{L}_{c(j+1)-1}(A))$ for $j \in [m_c]$. Each ciphertext $\mathsf{ct.B}_\ell$ is assumed to contain c copies of the ℓ-th lower diagonal vector of B for $\ell \in [n]$; that is, $\mathsf{ct.B}_\ell$ represents $(\mathsf{L}_\ell(B)|\dots|\mathsf{L}_\ell(B))$. If the diagonal vectors of B are initially provided in multi-diagonal encoded form, they are transformed into this replicated format using the “rotate-and-sum” algorithm[2]

[2]Assume that a single diagonal vector of size n is packed into a ciphertext. The input ciphertext is first rotated by the n positions and added to the original. This process is

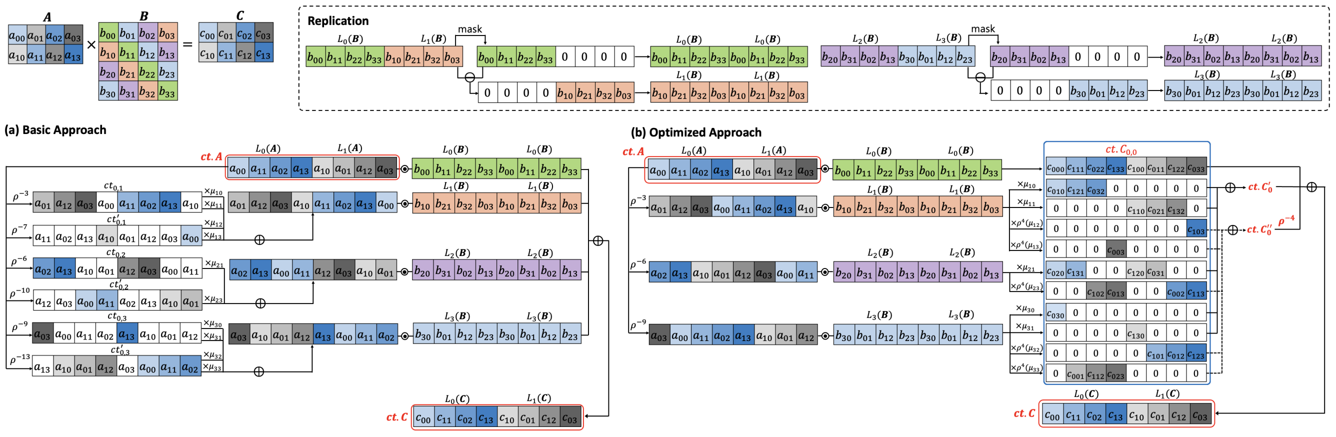


Figure 5: The CC-MM methods with $m = 2$, $n = 4$, $s = 8$, and $c = s/n = 2$. The ciphertext ct.A represents the multi-lower-diagonal encoding $\widehat{\mathsf{L}}_0(A) = (\mathsf{L}_0(A)|\mathsf{L}_1(A))$. (a) The basic approach: The internally rotated ciphertexts of A are generated and multiplied with the replicated diagonals of B. (b) The optimized approach: We perform homomorphic multiplications between the simply rotated diagonals of A and the replicated diagonals of B, followed by masking operations. The resulting ciphertexts hold intermediate products $c_{ijk} = a_{ij}b_{jk}$. To align the elements of the same color in the blue box, we apply a final rotation ρ^{-4} to the partially accumulated result ct.C_0''. As a result, we obtain the ciphertext ct.C representing $\widehat{\mathsf{L}}_0(C) = (\mathsf{L}_0(C)|\mathsf{L}_1(C))$.

(Algorithm 4 in Appendix B.2). According to Corollary 3.8, the multi-diagonal encoding $\widehat{\mathsf{L}}_j(C) = (\mathsf{L}_{cj}(C)|\dots|\mathsf{L}_{c(j+1)-1}(C))$ can be securely computed by evaluating the following equations in parallel:

$$\mathsf{L}_{cj+r}(C) = \sum_{\ell\in[n]} \rho^{\ell}(\mathsf{L}_{cj+r-\ell}(A)) \odot \mathsf{L}_{\ell}(B) \quad (10)$$

for $r \in [c]$. Let ct.$\mathsf{A}_{j,\ell}$ the ciphertext representing the concatenation of the rotated diagonals from $\rho^{\ell}(\mathsf{L}_{cj-\ell}(A))$ to $\rho^{\ell}(\mathsf{L}_{c(j+1)-1-\ell}(A))$. As illustrated in Fig. 4(b), Eq. (10) can be securely evaluated as

$$\sum_{\ell\in[n]} \mathtt{Mult}(\mathsf{ct.A}_{j,\ell}, \mathsf{ct.B}_{\ell}), \quad (11)$$

resulting in a ciphertext of $\widehat{\mathsf{L}}_j(C)$ for $j \in [m_c]$.

Finally, it remains to generate the ciphertext ct.$\mathsf{A}_{j,\ell}$ from the input ct.A_j. In contrast to the upper-lower matrix multiplication case, this ciphertext consists of distinct lower diagonal vectors that are rotated by the same amount. As a result, it can be efficiently constructed using internal rotations, as described in Section 4.2.1. Specifically, we generate two rotated ciphertexts $\mathsf{ct}_{j,\ell} = \mathtt{Rot}(\mathsf{ct.A}_j; -n \cdot [\ell]_c + \ell)$ and $\mathsf{ct}'_{j,\ell} = \mathtt{Rot}(\mathsf{ct.A}_j; -n \cdot [\ell]_c + \ell - n)$ for $1 \le \ell < n$. Then $\mathsf{ct}_{j,\ell}$ is applied by two masks $\mu_{\ell,0}$ and $\mu_{\ell,1}$ to extract the first $(n-\ell)$ entries of $\rho^{\ell}(\mathsf{L}_{cj+r}(A))$ for each $c - [\ell]_c \le r < c$ and $0 \le r < c - [\ell]_c$, respectively. Similarly, $\mathsf{ct}'_{j,\ell}$ is masked to extract the last ℓ entries from each $\rho^{\ell}(\mathsf{L}_{cj+r}(A))$ using the masks $\mu_{\ell,2}$ and $\mu_{\ell,3}$. Next, these ciphertexts are appropriately added to generate the desired ciphertext ct.$\mathsf{A}_{j,\ell}$ as follows:

$$\mathsf{ct.A}_{j,\ell} = \mathsf{ct}_{j-1,\ell}\cdot\mu_{\ell,0} + \mathsf{ct}_{j,\ell}\cdot\mu_{\ell,1} + \mathsf{ct}'_{j-1,\ell}\cdot\mu_{\ell,2} + \mathsf{ct}'_{j,\ell}\cdot\mu_{\ell,3}. \quad (12)$$

then repeated for $\log(s/n)$ iterations, with the rotation amount doubling each time and the results being aggregated at each step.

A simplified illustration of this process is provided in Fig. 5(a). Consequently, the lower-lower matrix multiplication method requires $2mn/c$ rotations, $4mn/c$ plaintext multiplications, and mn/c ciphertext multiplication. We refer to Appendix B.1 for a detailed complexity analysis.

4.3.2 *Optimized Lower-lower Matrix Multiplication.*

We reformulate the evaluation of Eq. (11) by adapting the BSGS strategy, which reduces the number of required rotations by roughly a factor of two. The main observation is that the ciphertext $\mathsf{ct}'_{j,\ell}$ can be obtained by rotating $\mathsf{ct}_{j,\ell}$ by the same n position across all j and ℓ. Rather than applying rotations to each input ciphertext of A, we delay these rotations. We instead perform ciphertext-multiplication between $\mathsf{ct}_{j,\ell}$ and the input ciphertexts of B, extract the valid entries, aggregate the resulting intermediate ciphertexts corresponding to $\mathsf{ct}'_{j,\ell}$, and apply only a single rotation to the partially summed result.

From Eq. (12), the ciphertext ct.$\mathsf{A}_{j,\ell}$ can be computed as

$$\begin{aligned}\mathsf{ct.A}_{j,\ell} &= \mathsf{ct}_{j-1,\ell}\cdot\mu_{\ell,0} + \mathsf{ct}_{j,\ell}\cdot\mu_{\ell,1} \\ &\quad + \mathtt{Rot}(\mathsf{ct}_{j-1,\ell}; -n)\cdot\mu_{\ell,2} + \mathtt{Rot}(\mathsf{ct}_{j,\ell}; -n)\cdot\mu_{\ell,3} \\ &= \mathsf{ct}_{j-1,\ell}\cdot\mu_{\ell,0} + \mathsf{ct}_{j,\ell}\cdot\mu_{\ell,1} \\ &\quad + \mathtt{Rot}(\mathsf{ct}_{j-1,\ell}\cdot\rho^{n}(\mu_{\ell,2}) + \mathsf{ct}_{j,\ell}\cdot\rho^{n}(\mu_{\ell,3}); -n).\end{aligned}$$

The first equality follows from the fact that $\mathsf{ct}'_{j,\ell} = \mathtt{Rot}(\mathsf{ct}_{j,\ell}; -n)$. We substitute this into Eq. (11) as follows:

$$\sum_{\ell\in[n]} \mathtt{Mult}(\mathsf{ct}_{j-1,\ell}\cdot\mu_{\ell,0} + \mathsf{ct}_{j,\ell}\cdot\mu_{\ell,1}, \mathsf{ct.B}_{\ell})$$
$$+ \sum_{\ell\in[n]} \mathtt{Mult}(\mathtt{Rot}(\mathsf{ct}_{j-1,\ell}\cdot\rho^{n}(\mu_{\ell,2}) + \mathsf{ct}_{j,\ell}\cdot\rho^{n}(\mu_{\ell,3})); -n), \mathsf{ct.B}_{\ell}).$$

Let $\mathrm{ct.C}_{j,\ell} = \mathtt{Mult}(\mathrm{ct}_{j,\ell}, \mathrm{ct.B}_\ell)$. The first term can be expressed as follows:

$$\begin{aligned}&\sum_{\ell\in[n]} \mathtt{Mult}(\mathrm{ct}_{j-1,\ell}, \mathrm{ct.B}_\ell)\cdot\mu_{\ell,0} + \mathtt{Mult}(\mathrm{ct}_{j,\ell}, \mathrm{ct.B}_\ell)\cdot\mu_{\ell,1}\\ &= \sum_{\ell\in[n]} \mathrm{ct.C}_{j-1,\ell}\cdot\mu_{\ell,0} + \mathrm{ct.C}_{j,\ell}\cdot\mu_{\ell,1}.\end{aligned}$$

Following $\mathrm{ct.B}_\ell = \mathtt{Rot}(\mathrm{ct.B}_\ell; -n)$, the second term can be expressed as follows:

$$\begin{aligned}&\mathtt{Rot}(\sum_{\ell\in[n]} \mathtt{Mult}(\mathrm{ct}_{j-1,\ell}\cdot\rho^n(\mu_{\ell,2}) + \mathrm{ct}_{j,\ell}\cdot\rho^n(\mu_{\ell,3})), \mathrm{ct.B}_\ell); -n)\\ &= \mathtt{Rot}(\sum_{\ell\in[n]} \mathrm{ct.C}_{j-1,\ell}\cdot\rho^n(\mu_{\ell,2}) + \mathrm{ct.C}_{j,\ell}\cdot\rho^n(\mu_{\ell,3}); -n).\end{aligned}$$

This implies that, after summing up the multiplication results, it suffices to apply a single rotation by $-n$ positions. Fig. 5(b) illustrates the idea of the BSGS strategy with comparison of the basic approach.

Parallel Matrix Multiplication. The CC-MM algorithm can be parallelized in a SIMD manner without additional cost. We aim to compute the matrix products $C^{(z)} = A^{(z)}B^{(z)}$ for $z \in [H]$, where $A^{(z)} \in \mathbb{R}^{m\times n}$ and $B^{(z)} \in \mathbb{R}^{n\times n}$ under $s = cnH$. Given ciphertexts $\mathrm{ct.A}_j$ of the batched vector $\widehat{\mathsf{L}}_j(A) = (\bar{\mathsf{L}}_{cj}(A)|\ldots|\bar{\mathsf{L}}_{c(j+1)-1}(A))$ for $j \in [m_c]$ and ciphertexts $\mathrm{ct.B}_\ell$ of the batched vector $\widehat{\mathsf{L}}_\ell(B) = (\bar{\mathsf{L}}_{c\ell}(B)|\ldots|\bar{\mathsf{L}}_{c(\ell+1)-1}(B))$ for $\ell \in [n_c]$, their products are executed in one shot by adjusting the rotation amounts and updating the masks accordingly within a single matrix multiplication algorithm. The detailed procedure is provided in Algorithm 2. The outputs correspond to multi-diagonal batched encodings $\widehat{\mathsf{L}}_j(C)$ where $C = (C^{(z)})_{z\in[H]}$. We remark that the replication procedure is also performed in a SIMD manner (Algorithm 4 in Appendix B.2, denoted as Replication), yielding encryptions of the replicated batched diagonal vectors.

The cost of the optimized parallel CC-MM is as follows. We first apply a single rotation to compute $\mathrm{ct}_{j,\ell}$ for each j and ℓ (line 7). Next, we perform $m_c n$ multiplications between $\mathrm{ct}_{j,\ell}$ and $\mathrm{ct.B}_\ell$ (line 8). After that, we carry out $4m_c(n-1)$ plaintext multiplications of the resulting ciphertexts with the masks $\mu_{\ell,0}, \mu_{\ell,1}, \rho^{nH}(\mu_{\ell,2})$, and $\rho^{nH}(\mu_{\ell,3})$ (lines 11 to 13). This number can be further reduced by $3m_c(n-1)$ using $\mathrm{ct.C}_{j,\ell}\cdot\rho^{nH}(\mu_{\ell,3}) = \mathrm{ct.C}_{j,\ell} - \mathrm{ct.C}_{j,\ell}\cdot\mu_{\ell,0} - \mathrm{ct.C}_{j,\ell}\cdot\mu_{\ell,1} - \mathrm{ct.C}_{j,\ell}\cdot\rho^{nH}(\mu_{\ell,2})$ (line 14). Finally, the output ciphertexts are appropriately summed with rotations by nH positions (lines 16 to 18). As shown in Fig. 5, the basic approach requires $2\cdot 3 = 6$ rotations for the encryption of A, whereas the optimized method reduces this to only 3 rotations for the input ciphertext and one additional rotation for the intermediate ciphertext, when $m = 2$ and $n = 4$. In general, the main advantage of the BSGS strategy is that it reduces the number of rotations by half—from $2mn/c$ to mn/c. In Table 2, we summarize the computational complexity of the optimized CC-MM method.

4.4 Optimization

We apply a lazy rescaling technique during plaintext-ciphertext multiplications by adjusting a scaling factor only once within the resulting ciphertext (e.g., lines 5 and 10 in Algorithm 1). We also rely on a lazy relinearization technique during ciphertext multiplications, enabling summation over extended ciphertexts along with only a single relinearization (e.g., lines 16 and 17 of Algorithm 2).

Algorithm 2 Ciphertext-Ciphertext MatMul

```
Input: Ciphertexts $\mathrm{ct.A}_j$ of the multi-lower-diagonal batched encoding of $(A^{(z)})_{z\in[H]}$ for $j \in [m/c]$ where $A^{(z)} \in \mathbb{R}^{m\times n}$ and $c = s/nH$; ciphertexts $\mathrm{ct.B}_\ell$ of the replicated batched lower diagonal vectors or the multi-lower-diagonal batched vectors of $(B^{(z)})_{z\in[H]}$ where $B^{(z)} \in \mathbb{R}^{n\times n}$
Output: Ciphertexts $\mathrm{ct.C}_j$ of the multi-lower-diagonal batched encoding of $(C^{(z)})_{z\in[H]}$ where $C^{(z)} = A^{(z)}B^{(z)}$
1: $m_c := m/c$
2: if $B^{(z)}$ are not replicated then
3:     $\mathrm{ct.B}_\ell \leftarrow$ Replication($\mathrm{ct.B}_\ell$)
4: for $j = 0$ to $m_c - 1$ do
5:     $\mathrm{ct.C}_{j,0} \leftarrow \mathtt{Mult}(\mathrm{ct.A}_j, \mathrm{ct.B}_0)$
6:     for $\ell = 1$ to $n - 1$ do
7:         $\mathrm{ct_{rot}} \leftarrow \mathtt{Rot}(\mathrm{ct.A}_j; (-n\cdot[\ell]_c + \ell)\cdot H)$
8:         $\mathrm{ct.C}_{j,\ell} \leftarrow \mathtt{Mult}(\mathrm{ct_{rot}}, \mathrm{ct.B}_\ell)$
9: for $\ell = 1$ to $n - 1$ do
10:    for $j = 0$ to $m_c - 1$ do
11:        $\mathrm{ct}_{j,\ell,0} \leftarrow \mathtt{PMult}(\mathrm{ct.C}_{j,\ell}, \mu_{\ell,0})$
12:        $\mathrm{ct}_{j,\ell,1} \leftarrow \mathtt{PMult}(\mathrm{ct.C}_{j,\ell}, \mu_{\ell,1})$
13:        $\mathrm{ct}_{j,\ell,2} \leftarrow \mathtt{PMult}(\mathrm{ct.C}_{j,\ell}, \rho^{nH}(\mu_{\ell,2}))$
14:        $\mathrm{ct}_{j,\ell,3} \leftarrow \mathrm{ct.C}_{j,\ell} - \mathrm{ct}_{j,\ell,0} - \mathrm{ct}_{j,\ell,1} - \mathrm{ct}_{j,\ell,2}$
15: for $j = 0$ to $m_c - 1$ do
16:    $\mathrm{ct.C}'_j \leftarrow \sum_{1\le\ell<n}(\mathrm{ct}_{j-1,\ell,0} + \mathrm{ct}_{j,\ell,1})$
17:    $\mathrm{ct.C}''_j \leftarrow \sum_{1\le\ell<n}(\mathrm{ct}_{j-1,\ell,2} + \mathrm{ct}_{j,\ell,3})$
18:    $\mathrm{ct.C}_j \leftarrow \mathrm{ct.C}_{j,0} + \mathrm{ct.C}'_j + \mathtt{Rot}(\mathrm{ct.C}''_j; -n\cdot H)$
19: return $\mathrm{ct.C}_j$
```

Table 2: Comparison of CC-MM methods for computing $A^{(z)}B^{(z)}$ for $z \in [H]$, where $A^{(z)} \in \mathbb{R}^{m\times n}$ and $B^{(z)} \in \mathbb{R}^{n\times n}$ with $n \ge m$ and $c = s/nH$.

Method	Optimization		#PMult	#Mult		#Rot
	BSGS	Cplx		#TProd†	#Relin‡	
U-L	-	✗	mn	$\frac{mn}{c}$	$\frac{m}{c}$	$(m+1)n$
L-L	✗	✗	$\frac{4mn}{c}$	$\frac{mn}{c}$	$\frac{m}{c}$	$\frac{2mn}{c}$
	✓	✗	$\frac{3mn}{c}$	$\frac{mn}{c}$	$\frac{2m}{c}$	$\frac{mn}{c}$
	✓	✓	$\frac{3mn}{2c}$	$\frac{mn}{2c}$	$\frac{2m}{c}$	$\frac{mn}{2c} + \frac{m}{c}$

†: Tensor product, ‡: Relinearization

Another optimization technique is *complexification*, which allows two ciphertexts of real-valued vectors to be packed into a single ciphertext. For simplicity of presentation, we focus on Proposition 3.6 given that m is even. We use $\mathrm{Re}(x)$ and $\mathrm{Conj}(x)$ to denote the real part and the complex conjugate of $x \in \mathbb{C}$, respectively. Then Eq. (2) can be reformulated as follows:

$$\mathsf{L}_r(C) = \mathrm{Re}(\sum_{\ell\in[m/2]} \rho^r(\mathsf{U}_{[\ell-r]_m}(\bar{A})) \odot \mathsf{L}_\ell(\bar{B})),$$

Table 3: Comparison with Jiang et al. [16] for computing $A^{(z)}B^{(z)}$ for $z \in [H]$, where $A^{(z)}, B^{(z)} \in \mathbb{R}^{n\times n}$ and $s = n^2$. The concrete numbers are based on $n = 2^7$ and $s = 2^{14}$.

Equation		H				
		1	2	4	8	16
[16]	$H(5\sqrt{n}+4n)$	572	1144	2272	4576	9152
Our work	$H(n/2+3)+n/4(\log(n/H)+3)$	387	422	520	760	1264

where $\mathsf{U}_k(\bar{A}) = \mathsf{U}_k(A) - \sqrt{-1}\mathsf{U}_{k+m/2}(A)$ and $\mathsf{L}_\ell(\bar{B}) = \mathsf{L}_\ell(B) + \sqrt{-1}\mathsf{L}_{\ell+m/2}(B)$. Since $\mathsf{Re}(2x) = x + \mathsf{Conj}(x)$, we have $\mathsf{L}_r(C) = X + \mathsf{Conj}(X)$ where $X = \sum_{\ell\in[m/2]} \rho^r(\mathsf{U}_{[\ell-r]_m}(\bar{A})/2)) \odot \mathsf{L}_\ell(\bar{B})$. Thus, the number of plaintext multiplications is reduced by half. A similar approach can be applied to all the proposed matrix multiplication algorithms (see Table 2).

4.5 Complexity Comparison

4.5.1 Comparison with [16]. We provide a complexity comparison of our CC-MM method with the state-of-the-art homomorphic matrix multiplication method of Jiang et al. [16]. Suppose that we aim to compute the products of $n \times n$ matrices $A^{(z)}$ and $B^{(z)}$ for $z \in [H]$. For a fair comparison, we set the number of slots to $s = n^2$.

- **Jiang et al. [16]:** Each matrix is encrypted into a single ciphertext using row-major order encoding, resulting in a total of $2H$ ciphertexts. According to Table 2 in [16], the homomorphic multiplication of two square matrices requires $5\sqrt{n}+4n$ key-switching operations. Consequently, the total cost of performing H matrix multiplications is $H(5\sqrt{n}+4n)$ key-switching operations.
- **Ours:** Let $c = s/nH = n/H$ denote the diagonal packing capacity. Specifically, c diagonals from each matrix $A^{(z)}$ are packed into one ciphertext using the multi-diagonal batched form. In other words, each ciphertext encodes $cnH = n^2$ entries from $A^{(z)}$ matrices, resulting in a total of $n^2H/n^2 = H$ ciphertexts. If the $B^{(z)}$ matrices are also provided in the multi-diagonal batched form, a replication step is required to convert them into replicate form (Algorithm 4). As summarized in Table 2, the total cost of key-switching operations is $H(3+n/2) + n/4(\log c + 3) = H(3+n/2) + n/4(\log(n/H)+3)$.

Table 3 compares the number of key-switching operations for different values of H. As H increases, our method achieves a greater reduction in key-switching overhead. For example, when $H = 16$, the number of key-switching operations is reduced by 86%, while maintaining the same number of input ciphertexts as in [16].

4.5.2 Comparison with Prior Secure Transformer Inference Works. In Table 4, we present a detailed comparison of the computational complexity of matrix multiplication algorithms when applied to the multi-head attention mechanism in the BERT model, along with existing methods such as BOLT [25], NEXUS [31], and Powerformer [26] under the assumption that $nH = 2d_kH = 2d$.

- **Case 1:** $\{XW_{i,h}\}_{0\le i<3, h\in[H]}$ of an encrypted matrix $X \in \mathbb{R}^{n\times d}$ and a plaintext matrix $W_{i,h} \in \mathbb{R}^{d\times d_k}$.
- **Case 2:** $\{Q_hK_h^\top\}_{h\in[H]}$ of encrypted $Q_h, K_h \in \mathbb{R}^{n\times n/2}$.
- **Case 3:** $\{\alpha_hV_h\}_{h\in[H]}$ of encrypted $\alpha_h \in \mathbb{R}^{n\times n}, V_h \in \mathbb{R}^{n\times n/2}$.

Table 4: Comparison with prior secure transformer inference works.

	Method	#KeySwitch	
		Equation	Number
Case1	BOLT	$\frac{63nd}{s} + (\frac{s}{64n} - 1)\cdot\frac{nd_k}{s}\cdot 3H$	297
	NEXUS	$2nd$	196608
	Powerformer	$(\sqrt{3n+d_k} + (\sqrt{3n+d_k} + \log(\frac{n}{d_k}))\cdot\frac{3H}{2})\cdot\frac{d}{n}$	2616
	THOR	$\frac{n}{2} + 3(H-2)\cdot\frac{d_k}{c} + 3\cdot\frac{d_k}{c}$	**192**
Case2	BOLT	$2(n-1)(\log n + 1)H$	24468
	NEXUS	$(\frac{n}{2}+1)nH$	99840
	Powerformer	$(\frac{5n}{2} + 6\sqrt{n} + 1)\cdot\frac{H}{2}$	2358
	THOR	$\frac{n}{2c}(\frac{n}{2}+6) + \frac{n}{8}(\log c + 3) + (\frac{n}{2} + \frac{n}{2c})$	**460**
Case3	BOLT	$2n\log n\cdot H$	21420
	Powerformer	$(4n + 3\sqrt{14n} + 8\sqrt{n} + 2\sqrt{2n})\cdot\frac{H}{2}$	4620
	THOR	$\frac{n}{4c}(n+6) + \frac{n}{4}(\log c + 3)$	**492**
Case4	BOLT	$\frac{31nd}{s} + (\frac{s}{32n} - 1)\cdot\frac{nd_k}{s}\cdot H$	177
	NEXUS	$2d^2$	1179648
	Powerformer	$\sqrt{n}\cdot(1+\frac{H}{2})H$	966
	THOR	$\frac{d_k}{2} + 2\cdot(\frac{H}{2}-1)\cdot\frac{n}{c}$	**122**

- **Case 4:** $\mathsf{Concat}(\mathrm{head}_0, \ldots, \mathrm{head}_{H-1})W^O$ of an encrypted matrix $\mathrm{head}_h \in \mathbb{R}^{n\times n/2}$ and a plaintext matrix $W^O \in \mathbb{R}^{d\times d}$.

To ensure a fair comparison, we assume that all methods use the same HE parameters, with the number of slots $s = 2^{15}$. The concrete numbers are based on the BERT-base parameters: $n = 128$, $d = 768$, $d_k = 64$, and $H = 12$. Given that each diagonal vector of the intermediate matrix in the transformer has a size of $n = 128$, we select the largest power-of-two integer c satisfying $c \le s/(nH) = 2^{15}/(2^7\cdot 12)$ in order to fully utilize the available slots. As a result, $c = 16$ multiple diagonal vectors from each matrix are packed together into a single ciphertext. We apply the CC-MM algorithms based on Corollaries 3.8 and 3.9 to Case 3 and Case 2, respectively, both of which incorporate the parallel replication step.

BOLT exhibits relatively fewer key-switching operations in PC-MM compared to CC-MM. In contrast, PowerFormer requires a similar number of key-switching operations for both PC-MM and CC-MM. NEXUS does not provide a specific matrix multiplication algorithm for Case 3 and is therefore excluded from this case. Additionally, it requires coefficient or component-wise encoding for Case 1. However, the output of one encoder layer does not naturally align with this format, leading to substantial computational overhead to convert the output accordingly. Overall, THOR consistently outperforms existing state-of-the-art HE-based approaches across all matrix operations required for multi-head attention. For example, in Case 3, our CC-MM algorithm reduces the number of key-switchings by 89% to 98% over PowerFormer and BOLT.

5 Secure Non-linear Computations

5.1 Softmax

For $\mathbf{x} = (x_i) \in \mathbb{R}^n$, the softmax function is defined as $\mathtt{Softmax}(\mathbf{x}) = (\exp(x_i)/\sum_{j\in[n]}\exp(x_j))_{i\in[n]}$. The exponential function can be directly approximated over a given interval. However, for larger intervals, direct polynomial approximation becomes less effective due to the rapid growth of the exponential function. To address this issue, the least squares method is applied to approximate the scaled exponential function $e^{x/\delta}$ with a low-degree polynomial and then raise the result to the power of δ. After approximating the exponential function, we aggregate the results and compute their inverse using Goldschmidt's division algorithm.

A large input range introduces additional computational challenges beyond numerical stability, as it necessitates a substantial number of fractional bits to maintain high precision. To address this issue and enable accurate yet efficient softmax evaluation, we propose a *two-phase method* consisting of the following steps. First, we shift the input vector to improve the numerical stability during exponentiation and to enhance the tractability of the approximation. Specifically, for an input vector with elements within the range $[a, b]$, each element is adjusted by subtracting the constant $(a + b)/2$. As a result, the shifted elements are within the interval $[-M/2, M/2]$, where $M = b - a$. Next, we introduce three scaling factors δ_0, δ_1, and δ_2. The parameter δ_0 defines the range for the exponential approximation; δ_1 manages the power of the initial scaled exponentiation; and δ_2 determines the number of repeated square-and-normalize operations.

- **Phase 1:** The input vector is scaled by a factor of $1/(\delta_1 \cdot \delta_2)$. Then each element of the scaled input vector lies within the interval $[-M/(2\delta_1\delta_2), M/(2\delta_1\delta_2)] \approx [-\delta_0/2, \delta_0/2]$, allowing stable exponential computations in subsequent steps. Then, we compute the δ_1-th power of these values. Finally, the resulting exponentials are normalized to form the initial probability distribution by dividing each value by the sum of all exponentials.
- **Phase 2:** To enhance the numerical stability and accuracy, the probabilities are refined through an iterative process of square-and-normalize for $\log \delta_2$ iterations.

Algorithm 3 provides an explicit description of our softmax approximation. Here, AExp and AInv denote the approximated exponential and inverse functions, respectively. We approximate the inverse function using Goldschmidt's algorithm, which iteratively refines the estimate by the update $f(x) = x(2 - x)$. To accelerate the convergence of this process, we incorporate the Adaptive Successive Over-Relaxation (aSOR) method [24], which dynamically adjusts the relaxation factor k_i at each iteration. The refined update rule is given by $x_{i+1} = k_i x_i (2 - k_i x_i)$, where $k_i = 2/(1 + \epsilon_i)$ is selected to maximize the convergence rate and ϵ_i represents the minimum value in the current iteration (see Appendix C for details).

The initial output $\mathbf{y}^{(0)} = (y_i^{(0)})$ of the first phase is an approximation of $\mathtt{Softmax}(\mathbf{x}/\delta_2)$. Since $\mathtt{Softmax}(2\mathbf{x})_i = y_i^2 / \sum_{j\in[n]} y_j^2$ for $\mathbf{y} = (y_i) = \mathtt{Softmax}(\mathbf{x})$, the j-th update $\mathbf{y}^{(j)} = (y_i^{(j)})$ in the second phase is approximate to $\mathtt{Softmax}(2^j \mathbf{x}/\delta_2)$. After $\log \delta_2$ iterations, this process yields an approximate result for $\mathtt{Softmax}(\mathbf{x})$.

5.2 Other Non-linear Functions

The attention value is normalized across the layer using LayerNorm $\mathtt{LayerNorm}(\mathbf{x})_j = \gamma \cdot (x_j - \mu)/\sqrt{\sigma^2 + \epsilon} + \beta$ for $\mathbf{x} = (x_j) \in \mathbb{R}^d$, where μ is the mean, σ^2 is the variance, and ϵ is a small constant for numerical stability. The inverse square root is approximated using the Goldschmidt algorithm combined with the aSOR technique.

Algorithm 3 ASoftmax($\mathbf{x}$)

```
Input: $\mathbf{x} = (x_i) \in [-M/2, M/2]^n$, $\delta_1, \delta_2 \in \mathbb{Z}^+$
Output: An approximated softmax of $\mathbf{x}$
1: $z_i \leftarrow (\mathsf{AExp}(x_i/(\delta_1\delta_2)))^{\delta_1}$,  $i = 0..n-1$
2: $S^{(0)} \leftarrow \sum_{i\in[n]} z_i$
3: $y_i^{(0)} \leftarrow z_i \cdot \mathsf{AInv}(S^{(0)})$,  $i = 0..n-1$
4: for $j = 1$ to $\log(\delta_2)$ do
5:     $z_i \leftarrow (y_i^{(j-1)})^2$,  $i = 0..n-1$
6:     $S^{(j)} \leftarrow \sum_{i\in[n]} z_i$
7:     $y_i^{(j)} \leftarrow z_i \cdot \mathsf{AInv}(S^{(j)})$,  $i = 0..n-1$
8: return $\mathbf{y} = (y_i^{(\log(\delta_2))})$
```

Prior works [12, 23, 31] approximate GELU using piecewise polynomials requiring three sign operations. In contrast, we adopt a two-step approach to approximate GELU with a composition of low-degree polynomials: $f_1(x)$ from least-squares fitting and $f_2(x)$ for residual refinement, resulting in the composite $f_2(f_1(x))$ as a close approximation of GELU. The NEXUS method needs 56 multiplications and depth 14, while ours uses degree-31 and degree-27 polynomials evaluated with the Paterson–Stockmeyer algorithm in 39 multiplications and depth 11. The same two-step approximation strategy is applied to the Tanh function with two degree-15 polynomials. We refer to Appendix C for the detailed parameters used in the approximations.

6 Secure Transformer Inference

In this section, we present the threat model and detailed architecture of THOR for secure transformer inference.

6.1 System Setup & Threat Model

Our system consists of two parties: the service provider, which holds a fine-tuned transformer model, and the client, which requests inference services from this model. The threat model of this paper is similar to that of previous private transformer inference works [26, 31]. We assume that the server is honest-but-curious. Our security goal is to ensure that the server does not learn any information about the client's private input. This can be guaranteed by the semantic security of the underlying HE scheme, which relies on the hardness of the Ring LWE problem. We also remark that the CKKS scheme is secure against the key-retrieval attack [22] if the plaintext results of the decryption are revealed only to the secret-key owner. Our protocol is secure against this attack, since the decrypted results are not shared with any external party.

6.2 THOR

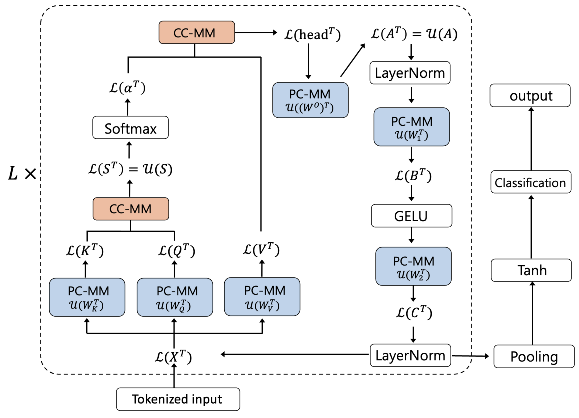
The client first computes the embeddings for a tokenized sentence using the embedding tables. The resulting output is then encrypted and fed into the transformer model for secure inference.

6.2.1 Our Strategy. We consider the computation of $Q = XW$, where W can be provided in cleartext. According to the original matrix multiplication, the encrypted diagonals of Q are obtained by combining rotated ciphertexts of X with plaintexts of W. For example, in the case of $n \times n$ square matrix multiplication (Eq. (2)), it requires n^2 rotations to securely compute $\rho^r(\mathsf{U}_{\ell-r}(X))$ from encryptions of $\mathsf{U}_k(X)$. Alternatively, the encrypted diagonals of $Q^\top = W^\top X^\top$ can be computed using ciphertexts of $X^\top$ with rotated plaintexts of $W^\top$, eliminating the need for ciphertext rotations. As a result, the transposed approach offers improved computational efficiency. Throughout the paper, we adopt this encoding format for transformer inference.

We also note that the row-wise computation over diagonal representations, as discussed in Remark 1, naturally extends to a multi-diagonal batched format, enabling efficient evaluation of non-linear function evaluations such as softmax and LayerNorm.

6.2.2 Workflow of THOR. In the following, we describe the workflow of THOR with a detailed breakdown of each step in the transformer inference process.

- **Attention layer.** Given the weight matrices $W_{Q,h} \in \mathbb{R}^{d \times d_k}$ for $h \in [H]$, each query matrix is calculated as $Q_h = XW_{Q,h}$, and these matrices are horizontally stacked to form the query matrix $Q \in \mathbb{R}^{n \times d}$. In other words, we have $Q^\top = W_Q^\top X^\top$, where $W_Q^\top \in \mathbb{R}^{d \times d}$ is a matrix formed by vertically stacking $W_{Q,h}^\top$ for $h \in [H]$. This formulation enables us to employ the diagonal block matrix multiplication method, yielding the ciphertexts $\mathsf{ct.Q}_j$ of the multi-lower-diagonal batched encoding of $(Q_h^\top)_{h \in [H]}$ for $j \in [d_k/c]$. We repeat this procedure to generate the ciphertexts $\mathsf{ct.K}_j$ and $\mathsf{ct.V}_j$ of the key matrices $(K_h^\top = W_{K,h}^\top X^\top)_{h \in [H]}$ and value matrices $(V_h^\top = W_{V,h}^\top X^\top)_{h \in [H]}$, respectively.
- **Attention score.** Using the property from Lemma 3.4 that $\mathsf{L}_\ell(A) = \rho^{-n+\ell}(\mathsf{U}_{m-\ell}(A)) = \rho^{-n+\ell}(\mathsf{L}_{m-\ell}(A^\top))$ for $A \in \mathbb{R}^{n \times d_k}$, we compute the lower diagonals of K_h's from ciphertexts $\mathsf{ct.K}_j$, denoted the resulting ciphertext as $\mathsf{ct.K}'_j$ for $j \in [d_k/c]$ (see Appendix B.3 for details). The ciphertexts $\mathsf{ct.Q}_j$ are transformed to contain c copies of the lower diagonal vectors of $Q_h^\top$'s, and we denote the resulting ciphertexts as $\mathsf{ct.Q}'_\ell$ for $\ell \in [d_k]$ (see Appendix B.2 for details). Finally, to compute $S_h^\top = K_h Q_h^\top / \sqrt{d_k} \in \mathbb{R}^{n \times n}$ for $h \in [H]$, we apply the CC-MM algorithm to $\mathsf{ct.K}'_j$ and $\mathsf{ct.Q}'_\ell$, yielding the ciphertexts $\mathsf{ct.S}_r$ for $r \in [n/c]$. These ciphertexts represent the multi-lower-diagonal batched encoding of $S_h^\top$'s, which is equivalent to its upper diagonals of S_h's.
- **Soft weights.** Let $\alpha_h = \mathtt{Softmax}(S_h) \in \mathbb{R}^{n \times n}$ be the row-wise application of the softmax operation of S_h. Specifically, Algorithm 3 is applied to the ciphertexts $\mathsf{ct.S}_r$, yielding approximate results of the multi-upper-diagonal batched encoding of α_h's, or equivalently, its lower diagonals of $\alpha_h^\top$'s. We denote the output ciphertexts as $\mathsf{ct.}\alpha_r$ for $r \in [n/c]$.
- **Attention head.** Let $\mathrm{head}_h = \alpha_h V_h \in \mathbb{R}^{n \times d_k}$ be the h-th attention head. First, we transform the input ciphertexts $\mathsf{ct.}\alpha_r$ to contain c copies of the lower diagonals of $\alpha_h^\top$'s, resulting in the output ciphertexts $\mathsf{ct.}\alpha'_\ell$ for $\ell \in [n]$. We then perform the CC-MM algorithm on the ciphertexts $\mathsf{ct.V}_j$ and $\mathsf{ct.}\alpha'_\ell$. The resulting ciphertexts, denoted as $\mathsf{ct.H}_j$ for $j \in [d_k/c]$, correspond to the multi-lower-diagonal batched encoding of $\mathrm{Head}_h^\top$'s.
- **Multi-head attention.** The multi-head attention is computed as $A = \mathrm{Concat}(\mathrm{head}_0, \ldots, \mathrm{head}_{H-1}) W^O \in \mathbb{R}^{n \times d}$ with $W^O \in \mathbb{R}^{d \times d}$. We apply diagonal block PC-MM to the ciphertexts $\mathsf{ct.H}_j$ to compute the transposed multi-head attention $A^\top$.



CC-MM
$\mathcal{L}(\mathrm{head}^T)$
$\mathcal{L}(A^T) = \mathcal{U}(A)$
$\mathcal{L}(\alpha^T)$
PC-MM $\mathcal{U}((W^O)^T)$
LayerNorm
output
Softmax
PC-MM $\mathcal{U}(W_1^T)$
$L \times$
$\mathcal{L}(S^T) = \mathcal{U}(S)$
$\mathcal{L}(B^T)$
Classification
CC-MM
GELU
$\mathcal{L}(K^T)$
$\mathcal{L}(Q^T)$
$\mathcal{L}(V^T)$
PC-MM $\mathcal{U}(W_2^T)$
Tanh
PC-MM $\mathcal{U}(W_K^T)$
PC-MM $\mathcal{U}(W_Q^T)$
PC-MM $\mathcal{U}(W_V^T)$
$\mathcal{L}(C^T)$
$\mathcal{L}(X^T)$
LayerNorm
Pooling
Tokenized input

Figure 6: Overview of secure transformer inference in BERT.

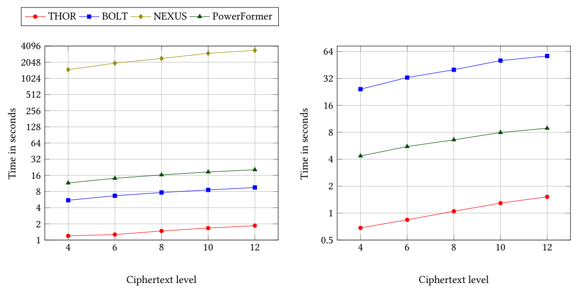
- **LayerNorm1.** We begin by computing the row-wise mean of $X + A$, given the encryptions of their upper diagonals. Similar operations can be applied to compute the mean of squares, and the variance can be obtained by subtracting the square of the means from the mean of squares. The subsequent normalization step, involving an inverse square root approximation, is performed entrywise. In the end, we obtain the ciphertexts $\mathsf{ct.A}_r$ for $r \in [n/c]$, which correspond to the upper diagonals of the normalized input.
- **FC1.** Let $B = AW_1$ with $W_1 \in \mathbb{R}^{d \times fd}$. The weight matrix W_1 is partitioned into f submatrices $W_{1,i} \in \mathbb{R}^{d \times d}$. Let $B_i = AW_{1,i} \in \mathbb{R}^{n \times d}$ for $i = 1, \ldots, f$. We apply diagonal block PC-MM to each $\mathsf{ct.A}_r$ to compute the multi-lower-diagonal batched encoding of $B_i^\top$, denoted as $\mathsf{ct.B}_{i,r}$ for $r \in [n/c]$.
- **GELU.** The approximation of GELU is performed entrywise on encrypted data, and the output is denoted as $\mathsf{ct.G}_{i,r}$.
- **FC2.** Let $C_i = G_i W_{2,i}$ where $G_i = \mathtt{GELU}(B_i)$, $W_{2,i} \in \mathbb{R}^{d \times d}$. The output of the fully connected layer is then $C = \sum_{i \in [f]} C_i \in \mathbb{R}^{n \times d}$. We first apply diagonal block PC-MM to $\mathsf{ct.G}_{i,r}$ to compute the multi-lower-diagonals of $C_i^\top$. Finally, we sum these ciphertexts to obtain the encryption of the lower diagonals of $C^\top$.
- **LayerNorm2.** Similar operations can be applied to obtain the normalized ciphertexts $\mathsf{ct.X}_r$ for $r \in [n/c]$, which represent the multi-lower-diagonal batched encodings of $X^\top$.
- **Pooler & classification.** The final dense layers are computed by applying the PC-MM algorithm to the normalized output.

Fig. 6 provides an overview of secure transformer inference.

7 Evaluation

7.1 Experimental Setup

Libraries and configurations. THOR is evaluated using the Liberate.FHE library [11], which implements the RNS variant of the CKKS scheme [8]. The ciphertext dimension was set as $N = 2^{16}$ to support bootstrapping [7], providing plaintext slots of $s = 2^{15}$. The ciphertext modulus q was chosen to ensure a 128-bit security level [5]. Under these HE parameters, 13 multiplicative levels are



(a) $\mathbb{R}^{768\times768} \times \mathbb{R}^{768\times128}$ (b) $12 \times (\mathbb{R}^{64\times128} \times \mathbb{R}^{128\times128})$

Figure 7: Runtime results of homomorphic matrix multiplication algorithms across different levels: (a) $\mathbb{R}^{768\times768} \times \mathbb{R}^{768\times128}$ PC-MM and (b) $12 \times (\mathbb{R}^{64\times128} \times \mathbb{R}^{128\times128})$ CC-MM.

Table 5: Softmax comparison with Cho et al. [10].

Method	#Iter.	HE operations			Runtime (sec)		Precision
		#Mult	#Rot	#Btps	Total	Amortized	(bits)
[10]	7	53	49	8	9.779	0.0382	-16.956
Our work	3	38	21	3	3.704	0.0145	-14.189

available before bootstrapping is needed. All experiments were conducted on a single NVIDIA A100 GPU. Our source code is available at https://github.com/crypto-starlab/THOR.

BERT model and dataset. We follow the BERT-base model with the number of encoding blocks $L = 12$, the embedding dimension $d = 768$, the number of input tokens $n = 128$. We fix the number of self-attention heads to $H = d/64$ (i.e., $d_k = 64$) and the size of the feed-forward features to $4d$. We use the default fine-tuning hyperparameters from [4], such as a batch size of 16, a learning rate of 5×10^{-5}, and 5 epochs. We evaluated our approach on the GLUE benchmark using three datasets: MRPC, RTE, and SST-2.

7.2 Micro-benchmarks

7.2.1 Homomorphic Matrix Multiplication. Fig. 7 illustrates the runtime performance of PC-MM and CC-MM. Specifically, PC-MM is used for the final linear transformation in multi-head attention (case 4 in Table 4), while CC-MM is used for attention head computation (case 3 in Table 4) including the parallel replication procedure. THOR consistently outperforms BOLT and PowerFormer across different ciphertext levels in both PC-MM and CC-MM tasks. In particular, THOR achieves a 5.3× speedup in PC-MM and 36× speedup in CC-MM over BOLT. Compared to PowerFormer, it yields a 11.2× improvement in PC-MM and a 9.7× improvement in CC-MM.

7.2.2 Softmax Comparison. Table 5 compares our softmax approximation algorithm with Cho et al. [10] to calculate 256 softmax operations with dimension 128, in terms of the number of iterations, the required number of homomorphic operations such as ciphertext multiplication, rotation, and bootstrappings (excluding initial exponentiation), the total/amortized evaluation time, and the worst-case accuracy in bits. The input values in the range $[-44.19, 50.25]$ were collected from the actual softmax inputs in the MRPC dataset. Based on the HE parameter $s = 2^{15}$, one can perform $s/n = 256$ softmax operations of dimension $n = 128$ simultaneously.

According to the implementation details in Table 2 of [10], the input range is represented as $[-2^k, 0]$ with $k = 7$, which requires a total of $k = 7$ normalize-and-square steps. In contrast, our algorithm achieves comparable accuracy with $\log \delta_2 + 1 = 3$ iterations of square-and-normalize using the following scaling factors: $\delta_0 = 11.805, \delta_1 = 2$, and $\delta_2 = 4$. The scaling factors δ_1 and δ_2 are empirically selected via greedy search on the training set, with δ_0 subsequently derived from them, aiming to minimize the overall computational cost. Since each normalization process in both algorithms requires a single bootstrapping, the reduced number of iterations in our method significantly lowers the number of bootstrapping operations (3 in ours vs. 8 in [10]). As a result, our method shows a 2.64× speed-up in evaluation. Specifically, it achieves an amortized cost per softmax operation of 3.7sec/256 ≈ 14.5ms, compared to 9.8sec/256 ≈ 38.2ms in [10]. We also note that the actual cost of the softmax evaluation itself remains below 3% in both methods, indicating that the main efficiency gain stems from the reduced number of bootstrapping operations.

Table 6: Breakdown of the execution time of THOR.

Operation	Input	Time (sec)
Attention layer	$3 \times (\mathbb{R}^{128\times768} \times \mathbb{R}^{768\times64})$	49.77
Attention score	$12 \times (\mathbb{R}^{128\times64} \times \mathbb{R}^{64\times128})$	16.25
Softmax	$12 \times (\mathbb{R}^{128\times128})$	15.53
Attention head	$12 \times (\mathbb{R}^{128\times128} \times \mathbb{R}^{128\times64})$	13.08
Multi-head attention	$\mathbb{R}^{128\times768} \times \mathbb{R}^{768\times768}$	27.43
LayerNorm1	$\mathbb{R}^{128\times768}$	7.13
FC1	$\mathbb{R}^{128\times768} \times \mathbb{R}^{768\times3072}$	49.80
GELU	$\mathbb{R}^{128\times3072}$	29.42
FC2	$\mathbb{R}^{128\times3072} \times \mathbb{R}^{3072\times768}$	49.19
LayerNorm2	$\mathbb{R}^{128\times768}$	4.10
Pooler & Classification	$\mathbb{R}^{128\times768}$	2.70
Bootstrappings	-	337.86
Total	-	**602.26**

7.3 End-to-end Inference Evaluation

7.3.1 End-to-end Performance. Table 6 presents a detailed breakdown of the execution time for each component within a single encoder, along with the bootstrapping time and the overall end-to-end runtime. Given the high computational cost of bootstrapping, THOR's architecture is designed to perform bootstrapping when the number of intermediate or output ciphertexts is minimal. Despite this optimization, bootstrapping operations still account for more than 56.10% of the total runtime. In contrast, the linear layers within the attention mechanism and the fully connected layers together contribute 34.13%. The total inference time of THOR for one input with 128 tokens is 10.04 minutes in a single GPU setting.

On the other hand, NEXUS requires a total of $4 \times 11 + 3 = 47$ bootstrapping stages, as bootstrapping is performed both before and after each layer normalization operation. Given that the 128×768 input matrices are encrypted in a row-wise manner, each bootstrapping stage involves 128 bootstrapping operations, resulting in a total of $47 \times 128 = 6016$ bootstrappings. For a fair comparison, we estimate the runtime of these operations using the

Table 7: Performance for the baseline and our method.

Dataset	#Test	Metric	Unencrypted				Encrypted
			Baseline	G	G-LN	G-LN-S	
MPRC	408	Accuracy	85.29	85.54	85.54	85.78	**84.80**
		F1-score	89.90	90.05	90.05	90.24	**89.49**
RTE	277	Accuracy	72.20	71.48	71.84	72.20	**71.12**
SST-2	872	Accuracy	91.51	91.40	91.40	91.63	**90.71**

same HE library as in our implementation, yielding approximately 1.02 sec $\times$ 6016 $\approx$ 1.7 hours. According to the reported results from NEXUS, bootstrapping accounts for 62.3% of the total runtime, allowing us to estimate a latency of 2.7 hours for a single prediction and a throughput of ~6 predictions per hour. Due to their evaluation strategy, a large number of operations are required for a single prediction, resulting in high latency compared to our approach.

7.3.2 Accuracy. Table 7 presents the inference performance under different settings: (a) Baseline: the pre-trained model without any approximation; (b) G: the baseline model with GELU replaced by our GELU approximation; (c) G-LN: the model with GELU and layer normalization replaced by our approximations; (d) G-LN-S: the full approximation model, including the softmax operation replaced by our approximation model; and (e) Encrypted: the full approximation applied to encrypted data. We found that the full approximation model outperforms the baseline, demonstrating the effectiveness of our approximation. Overall, the accuracy achieved by THOR matches that of the plaintext model, with only a 1% drop compared to the baseline. We attribute this slight degradation not to the approximations of the non-linear functions, but rather to computational error introduced by homomorphic operations.

8 Conclusion

In this study, we present THOR, a fast and secure transformer inference system on encrypted data. THOR leverages efficient homomorphic matrix multiplication algorithms, achieving substantial speedups in attention mechanisms compared to prior methods. With recent advances in algorithmic efficiency of HE and implementation optimization through hardware accelerators, THOR's performance can be further improved. The proposed cryptographic techniques for homomorphic matrix multiplication and non-linear function evaluation are applicable the other transformer-based tasks [28] and decoder models such as GPT. Moreover, our fundamental developments can be extended to other privacy-preserving machine learning applications, including neural networks [18, 19] and transfer learning [21].

References

[1] Yoshimasa Akimoto, Kazuto Fukuchi, Youhei Akimoto, and Jun Sakuma. 2023. Privformer: Privacy-preserving transformer with MPC. In *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*. 392–410.

[2] Ahmad Al Badawi, Louie Hoang, Chan Fook Mun, Kim Laine, and Khin Mi Mi Aung. 2020. Privft: Private and fast text classification with homomorphic encryption. *IEEE Access* 8 (2020), 226544–226556.

[3] Youngjin Bae, Jung Hee Cheon, Guillaume Hanrot, Jai Hyun Park, and Damien Stehlé. 2024. Plaintext-ciphertext matrix multiplication and FHE bootstrapping: fast and fused. In *Advances in Cryptology – CRYPTO 2024*. 387–421.

[4] Bert. 2019. https://github.com/google-research/bert. Google.

[5] Jean-Philippe Bossuat, Rosario Cammarota, Ilaria Chillotti, Benjamin R. Curtis, Wei Dai, Huijing Gong, Erin Hales, Duhyeong Kim, Bryan Kumara, Changmin Lee, Xianhui Lu, Carsten Maple, Alberto Pedrouzo-Ulloa, Rachel Player, Yuriy Polyakov, Luis Antonio Ruiz Lopez, Yongsoo Song, and Donggeon Yhee. 2024. Security Guidelines for Implementing Homomorphic Encryption. Cryptology ePrint Archive, Paper 2024/463.

[6] Jingwei Chen, Linhan Yang, Chen Yang, Shuai Wang, Rui Li, Weijie Miao, Wenyuan Wu, Li Yang, Kang Wu, and Lizhong Dai. 2024. Homomorphic matrix operations under bicyclic encoding. *IEEE Transactions on Information Forensics and Security* (2024).

[7] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. 2018. Bootstrapping for approximate homomorphic encryption. In *Advances in Cryptology – EUROCRYPT 2018*. 360–384.

[8] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. 2019. A Full RNS Variant of Approximate Homomorphic Encryption. In *Selected Areas in Cryptography – SAC 2018 (LNCS, Vol. 11349)*. Springer, 347–368.

[9] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2017. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology – ASIACRYPT 2017*. 409–437.

[10] Wonhee Cho, Guillaume Hanrot, Taeseong Kim, Minje Park, and Damien Stehlé. 2024. Fast and accurate homomorphic softmax evaluation. In *Proceedings of the ACM SIGSAC conference on computer and communications security.*

[11] DESILO. 2023. Liberate.FHE. https://github.com/Desilo/liberate-fhe.

[12] Ye Dong, Wen-jie Lu, Yancheng Zheng, Haoqi Wu, Derun Zhao, Jin Tan, Zhicong Huang, Cheng Hong, Tao Wei, and Wenguang Chen. 2023. PUMA: Secure inference of LLaMA-7B in five minutes. arXiv:2307.12533.

[13] Shai Halevi and Victor Shoup. 2014. Algorithms in HElib. In *Advances in Cryptology–CRYPTO 2014*. 554–571.

[14] Shai Halevi and Victor Shoup. 2018. Faster homomorphic linear transformations in HElib. In *Advances in Cryptology–CRYPTO 2018*. 93–120.

[15] Meng Hao, Hongwei Li, Hanxiao Chen, Pengzhi Xing, Guowen Xu, and Tianwei Zhang. 2022. Iron: Private inference on transformers. In *Advances in Neural Information Processing Systems*, Vol. 35. 15718–15731.

[16] Xiaoqian Jiang, Miran Kim, Kristin Lauter, and Yongsoo Song. 2018. Secure outsourced matrix computation and application to neural networks. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. ACM.

[17] Chao Jin, Mohamed Ragab, and Khin Mi Mi Aung. 2020. Secure transfer learning for machine fault diagnosis under different operating conditions. In *Provable and Practical Security*. 278–297.

[18] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. 2018. GAZELLE: A low latency framework for secure neural network inference. In *27th USENIX security symposium (USENIX security 18)*. 1651–1669.

[19] Miran Kim, Xiaoqian Jiang, Kristin Lauter, Elkhan Ismayilzada, and Shayan Shams. 2022. Secure human action recognition by encrypted neural network inference. *Nature communications* 13, 1 (2022), 4799.

[20] Joon-Woo Lee, HyungChul Kang, Yongwoo Lee, Woosuk Choi, Jieun Eom, Maxim Deryabin, Eunsang Lee, Junghyun Lee, Donghoon Yoo, Young-Sik Kim, and Jong-Seon No. 2022. Privacy-preserving machine learning with fully homomorphic encryption for deep neural network. *IEEE Access* 10 (2022), 30039–30054.

[21] Seewoo Lee, Garam Lee, Jung Woo Kim, Junbum Shin, and Mun-Kyu Lee. 2023. HETAL: Efficient privacy-preserving transfer learning with homomorphic encryption. In *Proceedings of the 40th International Conference on Machine Learning*.

[22] Baiyu Li and Daniele Micciancio. 2021. On the security of homomorphic encryption on approximate numbers. In *Advances in Cryptology – EUROCRYPT*.

[23] Wen-jie Lu, Zhicong Huang, Zhen Gu, Jingyu Li, Jian Liu, Cheng Hong, Kui Ren, Tao Wei, and Wenguang Chen. 2025. BumbleBee: Secure two-party inference framework for large transformers. In *Proceedings of the 32nd Annual Network and Distributed System Security Symposium*.

[24] Jungho Moon, Zhanibek Omarov, Donghoon Yoo, Yongdae An, and Heewon Chung. 2024. Adaptive successive over-relaxation method for a faster iterative approximation of homomorphic operations. Cryptology ePrint Archive.

[25] Qi Pang, Jinhao Zhu, Helen Möllering, Wenting Zheng, and Thomas Schneider. 2024. BOLT: Privacy-preserving, accurate and efficient inference for transformers. In *2024 IEEE Symposium on Security and Privacy (SP)*. 4753–4771.

[26] Dongjin Park, Eunsang Lee, and Joon-Woo Lee. 2024. Powerformer: Efficient privacy-preserving transformer with batch rectifier-power max function and optimized homomorphic attention.

[27] Jai Hyun Park. 2025. Ciphertext-ciphertext matrix multiplication: Fast for large matrices. In *Advances in Cryptology–CRYPTO 2025*. 153–180.

[28] Donghwan Rho, Taeseong Kim, Minje Park, Jung Woo Kim, Hyunsik Chae, Ernest K Ryu, and Jung Hee Cheon. 2025. Encryption-friendly LLM architecture. In *13th International Conference on Learning Representations*.

[29] Lorenzo Rovida and Alberto Leporati. 2024. Transformer-based language models and homomorphic encryption: An intersection with BERT-tiny. In *Proceedings of the 10th ACM International Workshop on Security and Privacy Analytics*. 3–13.

[30] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the Advances in neural information processing systems.*

[31] Jiawen Zhang, Jian Liu, Xinpeng Yang, Yinghao Wang, Kejia Chen, Xiaoyang Hou, Kui Ren, and Xiaohu Yang. 2025. Secure transformer inference made

non-interactive. In *Network and Distributed System Security Symposium*.

[32] Itamar Zimerman, Moran Baruch, Nir Drucker, Gilad Ezov, Omri Soceanu, and Lior Wolf. 2024. Converting transformers to polynomial form for secure inference over homomorphic encryption. In *International Conference on Machine Learning*.

A Proofs of Main Results

A.1 Proof of Proposition 3.6

Proof. For $\ell \in [m]$ and $r, t \in [n]$, the elements $A[r+t, \ell+t]$ and $B[\ell+t, t]$ correspond to the t-th entries of $\rho^r(\mathsf{U}_{\ell-r}(A))$ and $\mathsf{L}_\ell(B)$, respectively. Therefore, by Eq. (1), the t-th entry of the vector $\sum_{\ell\in[m]} \rho^r(\mathsf{U}_{\ell-r}(A)) \odot \mathsf{L}_\ell(B)$ is as follows:

$$\begin{aligned}\sum_{\ell\in[m]} \rho^r(\mathsf{U}_{\ell-r}(A))[t] \cdot \mathsf{L}_\ell(B)[t] &= \sum_{\ell\in[m]} A[r+t, \ell+t] \cdot B[\ell+t, t] \\ &= AB[r+t, t] \\ &= \mathsf{L}_r(AB)[t].\end{aligned}$$

□

A.2 Proof of Proposition 3.7

Proof. We first consider the t-th entry of $\rho^r(\mathsf{U}'_{\ell-r}(A))$ for $r \in [m]$ and $\ell, t \in [n]$. If $\ell - r < m$, this entry corresponds to the t-th entry of $\rho^r(\mathsf{U}_{\ell-r}(A))$, which is equal to $A[r+t \pmod m, \ell+t \pmod n]$. Otherwise, suppose that $\ell - r = mQ + R$ for some integers $Q \in \mathbb{Z}$ and $R \in [m]$. Then we have

$$\begin{aligned}\rho^r(\mathsf{U}'_{\ell-r}(A)) &= \rho^r(\rho^{m\cdot\lfloor(\ell-r)/m\rfloor}(\mathsf{U}_{(\ell-r) \bmod m}(A)))) \\ &= \rho^{r+mQ}(\mathsf{U}_R(A)).\end{aligned}$$

Thus, its t-th entry is $A[r+t \pmod m, \ell+t \pmod n]$. Similarly, the element $B[\ell+t \pmod n, t \pmod n]$ corresponds to the t-th entry of $\mathsf{L}_\ell(B)$. Therefore, the t-th entry of $\sum_{\ell\in[n]} \rho^r(\mathsf{U}'_{\ell-r}(A)) \odot \mathsf{L}_\ell(B)$ is $\sum_{\ell\in[n]} A[r+t, \ell+t] \cdot B[\ell+t, t] = \mathsf{L}_r(AB)[t]$. □

A.3 Proof for Corollary 3.8

Proof. From Proposition 3.7, it suffices to show that $\rho^r(\mathsf{U}'_{\ell-r}(A)) = \rho^\ell(\mathsf{L}_{r-\ell}(A))$. We first assume that $\ell - r < m$. Then, by Lemma 3.4, we have

$$\begin{aligned}\rho^r(\mathsf{U}'_{\ell-r}(A)) &= \rho^r(\mathsf{U}_{\ell-r}(A)) \\ &= \rho^r(\rho^{\ell-r}(\mathsf{L}_{m-\ell+r}(A))) \\ &= \rho^\ell(\mathsf{L}_{[m-\ell+r]_m}(A)).\end{aligned}$$

Now, suppose that $\ell - r \geq m$. Then we have

$$\begin{aligned}\rho^r(\mathsf{U}'_{\ell-r}(A)) &= \rho^r\left(\rho^{m\cdot\lfloor(\ell-r)/m\rfloor}(\mathsf{U}_{[\ell-r]_m}(A))\right) \\ &= \rho^r\left(\rho^{m\cdot\lfloor(\ell-r)/m\rfloor+[\ell-r]_m}(\mathsf{L}_{m-[\ell-r]_m}(A)\right) \\ &= \rho^r(\rho^{\ell-r}(\mathsf{L}_{m-[\ell-r]_m}(A)) \\ &= \rho^\ell(\mathsf{L}_{[m-\ell+r]_m}(A)).\end{aligned}$$

□

A.4 Proof for Corollary 3.9

Proof. By definition of the extended lower diagonal vector of A, we have

$$\mathsf{L}'_{n-k}(A) = \rho^{m\cdot(n/m-1)}(\mathsf{L}_{m-k}(A)) = \rho^{n-m}(\mathsf{L}_{m-k}(A))$$

for $k \in [m]$. It follows from Lemma 3.4 that $\mathsf{U}_k(A) = \rho^{-m+k}(\mathsf{L}_{m-k}(A)) = \rho^k(\mathsf{L}'_{n-k}(A))$ for $k \in [m]$. Therefore, we have

$$\rho^r(\mathsf{U}_{\ell-r}(A)) = \rho^r(\rho^{\ell-r}(\mathsf{L}'_{n-(\ell-r)}(A))) = \rho^\ell(\mathsf{L}'_{n-\ell+r}(A)),$$

as desired. □

B Homomorphic Matrix Operations

B.1 Complexity Analysis of CC-MM

The upper-lower method proceeds in three steps: (a) Internal rotations are applied to the ciphertext of $(\mathsf{U}_{cj}(A)|\ldots|\mathsf{U}_{c(j+1)-1}(A))$ to generate encryptions of the vector $(\rho^r(\mathsf{U}_{cj}(A))|\rho^r(\mathsf{U}_{cj+1}(A))|\ldots|\rho^r(\mathsf{U}_{c(j+1)-1}(A)))$ for $r \in [1, n)$. This requires $2(n-1)m_c$ rotations and $(n-1)m_c$ plaintext multiplications. Next, we generate the required ciphertexts for A by performing $(c-1)m_c n$ rotations, each followed by appropriate masking operations, incurring additional $(c-1)m_c n$ plaintext multiplications. (b) Each ciphertext of $(\mathsf{U}_{c\ell}(B)|\ldots|\mathsf{U}_{c(\ell+1)-1}(B))$ is rotated from 1 to $c-1$, requiring $n_c \cdot (c-1)$ rotations. (c) The resulting ciphertexts for A are multiplied by the rotated ciphertexts of B, requiring nm_c tensor products and m_c key-switching operations. In total, this method requires roughly $2(n-1)m_c + (c-1)m_c n + n_c \cdot (c-1) \approx mn + n = (m+1)n$ rotations, $(n-1)m_c + (c-1)m_c n \approx mn$ plaintext multiplications, and $m_c n$ ciphertext multiplications.

The basic lower-lower method proceeds in three steps: (a) Rotations are applied to each input ciphertext $\mathsf{ct.A}_j$ to generate intermediate ciphertexts $\mathsf{ct}_{j,\ell}$ and $\mathsf{ct}'_{j,\ell}$ for each $j \in [m_c]$, requiring $2m_c n$ rotations. (b) We then apply the appropriate masking operations, which involve approximately $4m_c n$ plaintext multiplications. (c) Finally, we perform multiplications between the resulting ciphertexts $\mathsf{ct.A}_{j,\ell}$ and the ciphertexts of B. This step incurs nm_c tensor products, along with m_c key-switching operations.

B.2 Replication

Algorithm 4 describes how to generate replicated batched lower diagonal vectors $(\bar{\mathsf{L}}_\ell(B)|\ldots|\bar{\mathsf{L}}_\ell(B)|)$ from the multi-diagonal batched encodings of $B = (B^{(z)})_{z\in[H]}$. To this end, we define the masking vectors $v_k \in \mathbb{R}^s$ for $k \in [n/c]$ and $v \in \mathbb{R}^s$ as follows: for $i \in [s]$,

$$v_k[i] = \begin{cases} 1 & \text{if } 2knH \leq i < 2(k+1)nH, \\ 0 & \text{otherwise,} \end{cases}$$

$$v[i] = \begin{cases} 1 & \text{if } 2tH \leq i < (2t+1)nH, 0 \leq t < c/2, \\ 0 & \text{otherwise.} \end{cases}$$

The total complexity is n plaintext multiplications and $n/2(\log c + 1)$ rotations in total. By applying complexification to $\mathsf{ct.B}_j$ and $\mathsf{ct.B}_{j+n/2}$, the cost is reduced by half—$n/2$ plaintext multiplications and $n/4(\log c + 3)$ rotations.

B.3 Matrix Transposition

Algorithm 5 performs parallel matrix transposition that transforms the multi-upper-diagonal batched encodings of $B = (B^{(z)})_{z\in[H]}$ into those of their transposes $B^\intercal = (B^{(z)\intercal})_{z\in[H]}$. Equivalently, the resulting ciphertexts represent the multi-lower-diagonal batched encoded vectors of $B^{(z)}$. Assume that $s = cnH$ for some integer c. For $j \in [n/c]$ and $k \in [c]$, we define two masking vectors

Algorithm 4 Replication

Input: Ciphertexts $\mathsf{ct.B}_j$ of the multi-lower-diagonal batched encoded vectors of $(B^{(z)})_{z\in[H]}$ for $j \in [n/c]$ where $B^{(z)} \in \mathbb{R}^{n\times n}$
Output: Ciphertexts $\mathsf{ct.B}'_\ell$ of the replicated batched lower diagonal vectors of $(B^{(z)})_{z\in[H]}$ for $\ell \in [n]$

```
1:  for $j = 0$ to $n/c - 1$ do
2:      for $k = 0$ to $c/2 - 1$ do
3:          $\mathsf{ct}_{j,k} \leftarrow \mathtt{PMult}(\mathsf{ct.B}_j, v_k)$
4:          for $r = 0$ to $\log(c/2) - 1$ do
5:              $\mathsf{ct}_{j,k} \leftarrow \mathsf{ct}_{j,k} + \mathtt{Rot}(\mathsf{ct}_{j,k}; -2n \cdot 2^r H)$
6:          $\mathsf{ct.B}'_{cj+2k} \leftarrow \mathtt{PMult}(\mathsf{ct}_{j,k}, v)$
7:          $\mathsf{ct.B}'_{cj+2k+1} \leftarrow \mathsf{ct}_{j,k} - \mathsf{ct.B}'_{cj+2k}$
8:      for $k = 0$ to $c - 1$ do
9:          $\mathsf{ct.B}'_{cj+k} \leftarrow \mathsf{ct.B}'_{cj+k} + \mathtt{Rot}(\mathsf{ct.B}'_{cj+k}; -nH)$
10: return $\mathsf{ct.B}'_\ell$
```

Algorithm 5 Matrix Transpose

Input: Ciphertexts $\mathsf{ct.B}_j$ of the multi-upper-diagonal batched encoded vectors of $(B^{(z)})_{z\in H}$ for $j \in [n/c]$ where $B^{(z)} \in \mathbb{R}^{n\times n}$
Output: Ciphertexts $\mathsf{ct.C}_\ell$ of the multi-lower-diagonal batched encoded vectors of $(B^{(z)})$ for $\ell \in [n/c]$

```
1:  for $j = 0$ to $n/c - 1$ do
2:      $\mathsf{ct}_{\mathrm{rot}} \leftarrow \mathtt{Rot}(\mathsf{ct}_j; [-cj]_n H)$
3:      $\ell := [-j]_{(n/c)}$
4:      $\mathsf{ct}_{\ell,0} \leftarrow \mathtt{PMult}(\mathsf{ct}_{\mathrm{rot}}, \mu_{j,0,0})$
5:      $\mathsf{ct}_{\ell,1} \leftarrow \mathtt{PMult}(\mathsf{ct}_{\mathrm{rot}}, \mu_{j,0,1})$
6:  for $j = 0$ to $n/c - 1$ do
7:      $\ell := n/c - 1 - j$
8:      for $k = 1$ to $c - 1$ do
9:          $\mathsf{ct}_{\mathrm{rot}} \leftarrow \mathtt{Rot}(\mathsf{ct}_j; (n - cj - k + 2kn - cn)H)$
10:         $\mathsf{ct}_{\ell,0} \leftarrow \mathtt{Add}(\mathsf{ct}_{\ell,0}, \mathtt{PMult}(\mathsf{ct}_{\mathrm{rot}}, \mu_{j,k,0}))$
11:         $\mathsf{ct}_{\ell,1} \leftarrow \mathtt{Add}(\mathsf{ct}_{\ell,1}, \mathtt{PMult}(\mathsf{ct}_{\mathrm{rot}}, \mu_{j,k,1}))$
12:     $\mathsf{ct.C}_\ell \leftarrow \mathtt{Add}(\mathsf{ct}_{\ell,0}, \mathtt{Rot}(\mathsf{ct}_{\ell,1}; -nH)$
13: return $\mathsf{ct.C}_\ell$
```

$\mu_{j,k,0}, \mu_{j,k,1}$ as follows: for $i \in [s]$,

$$\mu_{j,k,0}[i] = \begin{cases} 1 & \text{if } aH \le i < (a + n - b)H, \\ 0 & \text{otherwise,} \end{cases}$$

$$\mu_{j,k,1}[i] = \begin{cases} 1 & \text{if } [(a - b)]_s H \le i < ([a - b]_s + b)H, \\ 0 & \text{otherwise,} \end{cases}$$

where $a = [-kn]_s$ and $b = [n-k-cj]_n$. The transposition procedure requires $2n$ plaintext multiplications and $n + n/c$ rotations in total.

C Non-linear Approximation Details

- **Exponential.** Let $e(x) = \sum e_i x^i$ denote the polynomial approximation of the exponential function with coefficients

$$\begin{aligned}(e_i) = (&9.9999 \times 10^{-1}, 9.9999 \times 10^{-1}, 5.0002 \times 10^{-1}, 1.6667 \times 10^{-1}\\ &4.1653 \times 10^{-2}, 8.3316 \times 10^{-3}, 1.3920 \times 10^{-3}, 1.9871 \times 10^{-4}\\ &2.4471 \times 10^{-5}, 2.7286 \times 10^{-6}, 2.9466 \times 10^{-7}, 2.6438 \times 10^{-8},\\ &1.4896 \times 10^{-9}, 1.2104 \times 10^{-10}, 2.0777 \times 10^{-11}, 1.3376 \times 10^{-12}).\end{aligned}$$

- **Inverse.** Given $a_0 = x \in (0, 1)$ and $b_0 = 1$, we iteratively update the values: $a_{i+1} = k_i a_i(2 - k_i a_i)$ and $b_{i+1} = k_i b_i(2 - k_i a_i)$ for $i = 0, 1, \ldots, T - 1$, where k_i is a predetermined relaxation factor. The inverse of x is then approximated by b_T. The relaxation factors used in the approximation are $(k_i) = (2.00, 2.00, 1.98, 1.94, 1.79, 1.46, 1.12, 1.98, 1.94, 1.79, 1.46, 1.12, 1.01)$.
- **Inverse square root.** Given $a_0 = x \in (0, 1)$ and $b_0 = 1$, we iteratively compute $a_{i+1} = k_i a_i(3 - k_i a_i)^2/4$, and $b_{i+1} = \sqrt{k_i} b_i(3 - k_i a_i)/2$ for $i = 0, 1, \ldots, T-1$, where $k_i = 3(1-\sqrt{\epsilon_i}+\epsilon_i)/(1+\epsilon_i+\epsilon_i^2)$ and $\epsilon_i = k_{i-1}\epsilon_{i-1}(3 - k_{i-1}\epsilon_{i-1})^2/4$. The inverse square root of x is then approximated by b_T. The relaxation factors are specified as $(k_i) = (2.6374, 2.1722, 1.5135, 1.0907)$.
- **GELU & Tanh.** Let $f_1 = \sum f_{1,i} x^i$ and $f_2 = \sum f_{2,i} x^i$ denote the polynomial approximations for the GELU function, and let $g_1 = \sum g_{1,i} x^i$ and $g_2 = \sum g_{2,i} x^i$ denote the polynomial approximations for the Tanh function:

$$\begin{aligned}(f_{1i}) = (&-1.0624 \times 10^{-5}, 1.6445 \times 10^{-4}, -5.8353 \times 10^{-4}, -3.8091 \times 10^{-4},\\ &2.2443 \times 10^{-3}, 8.9230 \times 10^{-3}, -1.0528 \times 10^{-2}, -1.9183 \times 10^{-2},\\ &-2.0463 \times 10^{-1}, 4.5401 \times 10^{-1}, -5.4076 \times 10^{-1}, 5.6775 \times 10^{0},\\ &-1.3643 \times 10^{1}, 1.8257 \times 10^{1}, -8.4885 \times 10^{1}, 1.2869 \times 10^{2},\\ &3.6672 \times 10^{2}, -1.0140 \times 10^{3}, -1.2628 \times 10^{2}, 2.2173 \times 10^{3},\\ &-9.9542 \times 10^{2}, -2.3106 \times 10^{3}, 1.7358 \times 10^{3}, 1.2738 \times 10^{3},\\ &-1.2784 \times 10^{3}, -3.6678 \times 10^{2}, 4.7966 \times 10^{2}, 4.9461 \times 10^{1},\\ &-9.0675 \times 10^{1}, -2.3652 \times 10^{0}, 8.7431 \times 10^{0}, 1.6284 \times 10^{-2}).\end{aligned}$$

$$\begin{aligned}(f_{2i}) = (&-1.7027 \times 10^{2}, 6.8108 \times 10^{1}, 1.7920 \times 10^{3}, -6.8162 \times 10^{2},\\ &-8.4926 \times 10^{3}, 3.0563 \times 10^{3}, 2.3958 \times 10^{4}, -8.1044 \times 10^{3},\\ &-4.4815 \times 10^{4}, 1.4130 \times 10^{4}, 5.8620 \times 10^{4}, -1.7037 \times 10^{4},\\ &-5.5133 \times 10^{4}, 1.4553 \times 10^{4}, 3.7787 \times 10^{4}, -8.8767 \times 10^{3},\\ &-1.8951 \times 10^{4}, 3.8497 \times 10^{3}, 6.9417 \times 10^{3}, -1.1690 \times 10^{3},\\ &-1.8466 \times 10^{3}, 2.4169 \times 10^{2}, 3.5445 \times 10^{2}, -3.2450 \times 10^{1},\\ &-4.9192 \times 10^{1}, 2.5812 \times 10^{0}, 5.7839 \times 10^{0}, -9.4517 \times 10^{-2}).\end{aligned}$$

$$\begin{aligned}(g_{1i}) = (&-7.1453 \times 10^{3}, -7.7652 \times 10^{1}, 2.7428 \times 10^{4}, 2.4515 \times 10^{2},\\ &-4.2579 \times 10^{4}, -3.0195 \times 10^{2}, 3.4219 \times 10^{4}, 1.8299 \times 10^{2},\\ &-1.5116 \times 10^{4}, -5.6410 \times 10^{1}, 3.5876 \times 10^{3}, 8.1760 \times 10^{0},\\ &-4.1334 \times 10^{2}, -4.2902 \times 10^{-1}, 1.9506 \times 10^{1}, 2.0620 \times 10^{-3}).\end{aligned}$$

$$\begin{aligned}(g_{2i}) = (&-9.0257 \times 10^{-3}, -1.1232 \times 10^{-4}, 1.0876 \times 10^{-1}, 7.9679 \times 10^{-4},\\ &-5.4133 \times 10^{-1}, -1.4287 \times 10^{-3}, 1.4648 \times 10^{0}, -2.2242 \times 10^{-3},\\ &-2.4326 \times 10^{0}, 1.1738 \times 10^{-2}, 2.7497 \times 10^{-1}, -1.7763 \times 10^{-2},\\ &-2.3893 \times 10^{0}, 1.3019 \times 10^{-2}, 2.0287 \times 10^{0}, -4.0844 \times 10^{-3}).\end{aligned}$$