THOR: Secure Transformer Inference with Homomorphic Encryption

Jungho Moon
Hanyang University
email: moonjungho@hanyang.ac.kr

Dongwoo Yoo
Yonsei University
email: aydw0507@yonsei.ac.kr

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

Miran Kim
Hanyang University
email: miran@hanyang.ac.kr

***Abstract*—As language models are increasingly deployed in cloud environments, privacy concerns have become a significant issue. To address this, we design THOR, a secure inference framework for transformer models on encrypted data. Specifically, we first propose new fast matrix multiplication algorithms based on diagonal-major order encoding and extend them to parallel matrix computation through the compact ciphertext packing technique. Second, we design efficient protocols for secure computations of four non-linear functions such as softmax, LayerNorm, GELU, and Tanh, by integrating advanced underlying approximation methods with tailored optimizations. Our matrix multiplication algorithms reduce the number of key-switching operations in the linear layers of the attention block in the BERT-base model by up to 14.5x, compared to the state-of-the-art HE-based secure inference protocol (Park et al., Preprint). Combined with cryptographic optimizations, our experimental results demonstrate that THOR provides secure inference for the BERT-base model with a latency of 10.43 minutes on a single GPU, while maintaining comparable inference accuracy on the MRPC dataset.**

1. Introduction

The rapid advancement of *transformer models* has changed the way machines understand and generate sequence data, such as human language. These models are mainly based on *attention mechanism* [1], which allows the model to focus dynamically on different entities in the input sequence. This capability enables transformers to weigh the importance of various entities in a sequence, regardless of their position, thereby capturing long-range dependencies more effectively than traditional recurrent neural networks (RNNs) and long short-term memory networks (LSTMs). The introduction of BERT (Bidirectional Encoder Representations from Transformers) [2] exemplified the power of transformer models, leading to significant advancements in various natural language processing tasks, including question answering, named entity recognition, and translation. However, substantial computational demands of deep learning pose challenges for practical applications. As a result, deploying transformer models in a cloud environment is a natural choice.

However, this approach has significant concerns when dealing with sensitive data, such as protected health information. The main issue is the potential exposure of such data during inference. Traditional methods of model deployment on cloud computing platforms, such as OpenAI's GPT-4o [3] or Meta's LLaMA [4], do not adequately address these privacy concerns, highlighting the need for new approaches to ensure data confidentiality.

To address privacy concerns, several cryptographic techniques such as secure multi-party computation (MPC), homomorphic encryption (HE), or their combination, have been adopted to secure transformer inference. In their scenario, the service provider holds a fine-tuned transformer model and provides inference services to clients whose input data is sensitive. Recent studies [5, 6, 7, 8, 9, 10] utilize oblivious transfer-based or secret sharing-based MPC techniques to protect private information across all layers, including the clients' input. Although these interactive approaches achieve faster execution times, they still incur high communication costs due to the inherent overhead of MPC.

Several studies have explored secure transformer inference under HE. Although HE-based methods come with a substantial performance overhead, they allow the service provider (or server) to perform computations on encrypted data without decryption. THE-X [11] is the first HE-based transformer inference system; however, it sacrifices accuracy by employing HE-compatible replacements for non-linear functions. Most notably, its protocol delegates the computations of ReLU to the client, thereby revealing the comparison results involved in softmax and GELU approximations. Two recent works [12, 13] are HE-based non-interactive systems that perfectly protect data privacy during transformer inference. However, their works focus on the BERT-tiny model (a distilled student model of BERT with 4.4 million parameters), which limits the scalability and efficiency of matrix multiplication algorithms when

extended to larger models. In addition, they do not fully utilize all plaintext slots during matrix multiplications, leading to substantial performance overhead. Furthermore, their non-linear approximation methods are restricted to a small domain, reducing effectiveness for larger models. On the other hand, NEXUS [14] evaluated the BERT-base model (110M parameters [2]). Although NEXUS is capable of making inferences with high throughput, it requires a substantial number of operations even for a single prediction, resulting in high latency. This work employs various packing methods, such as component-wise, row/column-wise, and diagonal-wise packings; however, the details of changes in the packing structure during secure inference are not fully explained. In particular, the output of one encoder layer is used as the input for the plaintext-ciphertext matrix multiplication in the subsequent attention layer, requiring substantial computational costs to convert it into component-wise format.

Our contributions. The main challenges to achieving accurate and low-latency inference for large transformer models under HE are efficient homomorphic matrix multiplications required for attention mechanisms and secure computation of non-linear operations. In this study, we present THOR, a secure Transformer inference framework that leverages HomomOrphic encRyption to ensure data privacy. To the best of our knowledge, this is the first fully non-interactive secure transformer inference framework for the BERT-base model. Our contributions are three-fold:

Homomorphic matrix multiplication. We propose computation-efficient homomorphic matrix multiplication algorithms optimized for transformer architectures, aimed at reducing the computational overhead associated with encrypted operations. In a nutshell, we formulate matrix multiplication as entrywise additions and multiplications along matrix diagonals. In the context of HE, we leverage all plaintext slots by compactly packing multiple diagonal vectors and processing computations over packed ciphertext. Based on this insight, we introduce a plaintext-ciphertext matrix multiplication algorithm that performs parallel matrix multiplications between submatrices and aggregates the intermediate results to produce the final output. We also design a ciphertext-ciphertext matrix multiplication along with a lazy rotation optimization that reduces the number of rotations on input ciphertexts. Instead of performing rotations directly on the inputs, we defer these operations to the intermediate results by applying rotations of the same amounts. As a result, we can reduce the number of ciphertext rotations by roughly half, thus enhancing the computational efficiency.

Accurate and efficient non-linear homomorphic computations. Homomorphic evaluation of non-linear functions is an important factor in achieving accurate inference. We first propose an efficient method for evaluating the softmax function over large intervals using a *two-phase method*: approximating the exponential function within a small domain and then performing repeated squaring-and-normalization operations. Specifically, we adopt Goldschmidt's iterative method to approximate the inverse and adaptively adjust the relaxation coefficient to achieve faster convergence. This structured approach enables efficient computation while preserving high precision, even on large intervals. We also apply the adaptive iterative method to efficiently compute the inverse square root for the LayerNorm evaluation. Another contribution is the evaluation of the GELU function. We approximate it using a composition of two low-degree polynomials and leverage the Paterson-Stockmeyer algorithm for efficient polynomial evaluation.

Enc-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 $n = 128$ tokens with the BERT-base model with a latency of 10.43 minutes on a single GPU, while maintaining a comparable inference accuracy.

2. Preliminaries

Notation. The binary logarithm will be denoted by $\log(\ )$. 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.

2.1. Homomorphic Encryption

Homomorphic Encryption (HE) is a cryptographic primitive that allows computations to be performed directly on encrypted data, without the need for decryption. Since Gentry's breakthrough on HE [15], numerous HE schemes with various properties have been proposed. Among them, the Cheon-Kim-Kim-Song (CKKS) scheme [16] has received significant attention due to its support for approximate arithmetic on encrypted complex numbers. The CKKS scheme also allows operations to be executed in a single instruction multiple data (SIMD) manner, enabling efficient parallel computation of large-scale datasets. Specifically, a plaintext vector $\mathbf{m} \in \mathbb{C}^s$ is first transformed into a plaintext polynomial, which is then encrypted as a ciphertext, where $s = N/2$ represents the number of plaintext slots for a ring dimension N.

The CKKS scheme supports the following basic arithmetic and advanced operations:

Vectorized arithmetic operations: Element-wise addition and multiplication can be performed between two ciphertexts (Add and Mult), or between a ciphertext and a plaintext (SAdd and SMult)

Rotation: Given a ciphertext ct representing $\mathbf{m} \in \mathsf{C}^s$, it returns a ciphertext representing the left rotation $^r(\mathbf{m})$ by r amounts. We abuse the same notation for the operation on ciphertexts.
Complex conjugation: This operation applies element-wise complex conjugation to a ciphertext.
Key-switching: One can transform a ciphertext with a secret key s into a ciphertext that encrypts the (approximately) same message but with a different key s'. For example, ciphertext-ciphertext multiplication involves a key switching operation to obtain a normal ciphertext with respect to the original secret key after performing the tensor product of the inputs.
Bootstrapping: The bootstrapping operations [17] refreshes a ciphertext by increasing its coefficient modulus, allowing further operations without significant loss of precision.

2.2. Transformer

The transformer architecture [1] is designed to handle sequence-to-sequence tasks effectively, particularly in natural language understanding (NLU) and generation. The transformer is based on an encoder-decoder architecture, with both parts having a similar structure. Therefore, we mainly focus on the encoder below. The transformer takes as input $X \in \mathsf{R}^{n \times d}$, where n indicates the number of tokens and d denotes the model dimension.

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. Multi-head attention operates as follows: (i) The input sequence $X \in \mathsf{R}^{n \times d}$ is linearly projected to H heads so that $Q_h = XW_{Q;h}$; $K_h = XW_{K;h}$; and $V_h = XW_{V;h}$ with the parameter matrices $W_{Q;h}, W_{K;h} \in \mathsf{R}^{d \times d_k}$, and $W_{V;h} \in \mathsf{R}^{d \times d_v}$ for $h \in [H]$. (ii) The scaled dot-product attention is computed in each head:

$$\mathrm{Att}_h = \mathsf{Softmax}\left(\frac{Q_h K_h^{\intercal}}{\sqrt{d_k}}\right) V_h.$$

(iii) The results of multiple heads are concatenated and linearly transformed by $W^O \in \mathsf{R}^{Hd_v \times d}$ to produce the final result. Following the parameters from [1], we use $d_k = d_v = d/H$. A feed-forward layer consists of two linear transformations with a GELU (Gaussian Error Linear Unit function) activation in between.

Pooler and classifier. The first token from the L encoder layers (corresponding to the first row of the result matrix) 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 1 (Upper diagonal vector)**.** *For a matrix $A = (a_{ij}) \in \mathsf{R}^{a \times b}$ and $0 \le k < \min\{a, b\}$, the k-th upper diagonal vector $\mathsf{U}_k(A) \in \mathsf{R}^{\max\{a,b\}}$ is defined by*

$$\mathsf{U}_k(A)[t] = A[t \bmod a, k + t \bmod b]$$

for $0 \le t < \max\{a, b\}$.

Definition 2 (Lower diagonal vector)**.** *For a matrix $B = (b_{ij}) \in \mathsf{R}^{b \times c}$ and $0 \le \ell < \min\{b, c\}$, the ℓ-th lower diagonal vector $\mathsf{L}_\ell(B) \in \mathsf{R}^{\max\{b,c\}}$ is defined by*

$$\mathsf{L}_\ell(B)[t] = B[\ell + t \bmod b, t \bmod c]$$

for $0 \le t < \max\{b, c\}$.

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

Lemma 1. *For any $a, b \in \mathsf{R}^n$ and $r \in \mathsf{Z}$, we have $\langle a, b\rangle = \langle {}^r(a), {}^r(b)\rangle$. That is, ${}^r(\cdot) : \mathsf{R}^n \to \mathsf{R}^n$ is an isometry on the real inner product space R^n.*

Lemma 2. *For any $A \in \mathsf{R}^{a \times b}$ and $k \in \mathsf{Z} \setminus [0, \min\{a, b\})$, we have $\mathsf{U}_k(A) = \mathsf{L}_k(A^{\intercal})$.*

Lemma 3. *For $m \le n$, we have*

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

Lemma 4. *For a square matrix $A \in \mathsf{R}^{n \times n}$ and any $t \in [n]$, we have $(\mathsf{U}_k(A)[t])_{k \in [n]} = A_t \in \mathsf{R}^n$.*

3.2. Upper-lower Matrix Multiplication

We first propose matrix multiplication algorithms for the case when the upper and lower diagonal vectors of two input matrices are provided. In a nutshell, it follows from Lemma 1 that given two square matrices of A and B of size n, for any $r, t \in [n]$, we have $\langle {}^t(A_{r+t}), {}^t(B^t)\rangle = \langle A_{r+t}, B^t\rangle = C[r+t, t]$, which corresponds to the t-th entry of $\mathsf{L}_r(C)$. In a SIMD fashion, ${}^t(A_{r+t})$ is obtained from the t-th entries of ${}^r(\mathsf{U}_{\ell-r}(A))$ for $\ell \in [n]$. Similarly, ${}^t(B^t)$ is obtained from the t-th entries of $\mathsf{L}_\ell(B)$ for $\ell \in [n]$. Thus, this can be formulated as the following propositions.

Proposition 1. *For $n \ge m$, suppose that n is divisible by m. Given two matrices $A \in \mathsf{R}^{n \times m}$ and $B \in \mathsf{R}^{m \times n}$, for $r \in [n]$, the r-th lower diagonal vector of the matrix AB can be computed as follows:*

$$\mathsf{L}_r(AB) = \sum_{\ell \in [m]} {}^r(\mathsf{U}_{[\ell-r]_m}(A)) \cdot \mathsf{L}_\ell(B). \tag{1}$$

Proof. See Appendix A.1. □

Proposition 2. *For $n \ge m$, suppose that n is divisible by m. Given two matrices $A \in \mathsf{R}^{m \times n}$ and $B \in \mathsf{R}^{n \times n}$, for*

$r \in [m]$, *the r-th lower diagonal vector of the matrix AB can be computed as follows:*

$$\mathsf{L}_r(AB) = \sum_{\ell \in [n]} \rho^{r}(\mathsf{U}^{0}_{\ell+r}(A)) \odot \mathsf{L}_{\ell}(B);$$

where the upper diagonal vector $\mathsf{U}^0_k(A) \in \mathsf{R}^n$ is defined by

$$\mathsf{U}^0_k(A) = \begin{cases} \mathsf{U}_k(A) & \text{if } k < m; \\ \mathsf{mask}_{k \bmod m}(\mathsf{U}_{(k \bmod m)}(A)) & \text{if } m \le k < n. \end{cases}$$

Proof. See Appendix A.2. □

3.3. Lower-lower Matrix Multiplication

We now propose matrix multiplication algorithms for the case when the lower diagonal vectors of two input matrices are provided.

Definition 3. *For a matrix $A \in \mathsf{R}^{n \times m}$ with $n \ge m$, the lower diagonal vector $\mathsf{L}^0_k(A) \in \mathsf{R}^n$ is defined by*

$$\mathsf{L}^0_k(A) = \begin{cases} \mathsf{L}_k(A) & \text{if } k < m, \\ \mathsf{mask}_{k \bmod m}(\mathsf{L}_{(k \bmod m)}(A)) & \text{if } m \le k < n. \end{cases}$$

Corollary 1. *For $n \ge m$, suppose that n is divisible by m. Given two matrices $A \in \mathsf{R}^{n \times m}$ and $B \in \mathsf{R}^{m \times n}$, for $r \in [n]$, the r-th lower diagonal vector of the matrix AB can be computed as follows:*

$$\mathsf{L}_r(AB) = \sum_{\ell \in [m]} \rho^{\ell}(\mathsf{L}^0_{[n-\ell+r]_n}(A)) \odot \mathsf{L}_{\ell}(B). \qquad (2)$$

Proof. See Appendix A.3. □

Corollary 2. *For $n \ge m$, suppose that n is divisible by m. Given two matrices $A \in \mathsf{R}^{m \times n}$ and $B \in \mathsf{R}^{n \times n}$, for $r \in [m]$, the r-th lower diagonal vector of the matrix AB can be computed as follows:*

$$\mathsf{L}_r(AB) = \sum_{\ell \in [n]} \rho^{\ell}(\mathsf{L}_{[m-\ell+r]_m}(A)) \odot \mathsf{L}_{\ell}(B).$$

Proof. See Appendix A.4. □

4. Homomorphic Matrix Multiplication

In this section, we explain how to evaluate the proposed matrix multiplication algorithms using a combination of compact ciphertext packing and SIMD techniques. The main goal of our algorithms is to minimize the number of key-switching operations, particularly rotations, which is the dominant process in homomorphic operations.

4.1. Matrix Multiplication on Packed Ciphertexts

We introduce a compact packing method that packs multiple diagonal vectors of an input matrix into a single ciphertext, ensuring that the output diagonals of the matrix product are efficiently packed within the resulting ciphertext. Furthermore, multiple diagonals from different matrices fit into a single plaintext to maximize the slot usage, enabling parallel matrix computations in a SIMD manner.

4.1.1. Compact Matrix Packing. For simplicity of presentation, we assume that the matrix $A \in \mathsf{R}^{n \times m}$ is provided in plaintext, while the matrix $B \in \mathsf{R}^{m \times n}$ is provided in encrypted form, with $n \ge m$. If the number of plaintext slots s is equal to n, each n-dimensional diagonal vector of A and B can be encoded in a single ciphertext. Since it requires m scalar multiplications to compute one diagonal vector of the output matrix by evaluating Eq. (1), the total computation requires mn scalar multiplications in total.

We now assume that the slot parameter s is larger than n. A straightforward approach is to pad zeros after the n slots and process with the algorithm as described above. However, this method is space-inefficient since the slots after n remain unused. To address this inefficiency, we aim to produce multiple diagonal vectors of AB within a single ciphertext by utilizing all available slots. This can be achieved by compactly packing multiple diagonal input vectors and performing SIMD computations over the packed ciphertexts. Specifically, given that $s = cn$ for some integer c, we assume that c many lower diagonal vectors of B are packed into a single ciphertext. Let $\mathsf{ct}.\mathsf{B}_j$ be an encryption of the stacked diagonals $(\mathsf{L}_{cj}(B), \mathsf{L}_{cj+1}(B), \ldots, \mathsf{L}_{c(j+1)-1}(B))$ for $j \in [m/c]$. Then, for $r \in [n/c]$, an encryption of $(\mathsf{L}_{cr}(C), \mathsf{L}_{cr+1}(C), \ldots, \mathsf{L}_{c(r+1)-1}(C))$ can be obtained by evaluating

$$\sum_{\ell \in [c], j \in [m/c]} \mathsf{SMult}(\rho^{\ell}(\mathsf{ct}.\mathsf{B}_j), \mathsf{pt}.\mathsf{A}_{j,\ell,r}); \qquad (3)$$

where $\mathsf{pt}.\mathsf{A}_{j,\ell,r}$ denotes a plaintext of the internally rotated $(\ell + cj)$-th upper diagonal $(\rho^{cr}(\mathsf{U}_{\ell+cj}(A)), \rho^{cr+1}(\mathsf{U}_{\ell+cj}(A)), \ldots, \rho^{c(r+1)-1}(\mathsf{U}_{\ell+cj}(A)))$. Although the number of rotations is increased, the total number of scalar multiplications is reduced to mn/c.

4.1.2. Parallel Matrix Multiplications. Assume that s is divisible by n and let $\tilde{s} = s/n$. Given that $m \ge n$, we aim to perform matrix multiplications of $A^{(z)} \in \mathsf{R}^{n \times m}$ and $B^{(z)} \in \mathsf{R}^{m \times n}$ for $z \in [\tilde{s}]$. Let $A = \{A^{(z)}\}_{z \in [\tilde{s}]}$ and $B = \{B^{(z)}\}_{z \in [\tilde{s}]}$. For $k \in [m]$, we define the k-th interlaced upper diagonal vector $\widetilde{\mathsf{U}}_k(A) \in \mathsf{R}^s$ as

$$\widetilde{\mathsf{U}}_k(A)[t] = \mathsf{U}_k(A^{(t \bmod \tilde{s})})[\lfloor t/\tilde{s} \rfloor c] \qquad (4)$$

for $t \in [s]$. In other words, the entries of the upper diagonal vectors $\mathsf{U}_k(A^{(z)})$ are interlaced within the slots. The ℓ-th interlaced lower diagonal vector $\widetilde{\mathsf{L}}_\ell(B) \in \mathsf{R}^s$ can be defined in a similar way. We then perform $\tilde{s}$-many matrix multiplications to obtain the matrix $C^{(z)} = A^{(z)}B^{(z)}$ in one shot by slightly modifying the single matrix multiplication algorithm in Eq. (1). Specifically, the r-th interlaced lower diagonal vector of the output matrices $C^{(z)}$ can be computed as follows: $\widetilde{\mathsf{L}}_r(C) = \sum_{\ell \in [m]} \rho^{r\tilde{s}}(\widetilde{\mathsf{U}}_{\ell+r}(A)) \odot \widetilde{\mathsf{L}}_\ell(B)$.

In practice, the BERT-base model has $H = 12$ heads, so H many matrix multiplications are executed in parallel. Given that each diagonal vector of the intermediate matrix in transformer has a size of $n = 128$, we get a power of two integer c that is less than $s/(nH) = 2^{15}/(2^7 \cdot 12)$, resulting in $c = 16$.

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 large 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 $n \geq d$ (see Section 6.2 for details). Without loss of generality, we assume that d is divisible by n and denote $\tilde{d} = d/n$; otherwise, zeros can be padded. Additionally, we assume $s = cd$, where c is an integer that divides n, and we denote $\tilde{n} = n/c$.

4.2.1. Extension of Basic Matrix Multiplication. One approach is to modify the basic upper-lower matrix multiplication algorithms in Section 3.2 to leverage the d-dimensional diagonal vectors of A and B. For $r \in [n]$, the r-th lower diagonal of C can be computed as $\mathsf{L}_r(C) = \sum_{\ell \in [d]} \rho^r(\mathsf{U}_{\ell-r})(A) \cdot \mathsf{L}^{\ell}_0(B)$, where $\mathsf{L}^{\ell}_0(B) \in \mathbb{R}^d$ is defined by

$$\mathsf{L}^{\ell}_0(B) = \begin{cases} \mathsf{L}_\ell(B) & \text{if } \ell < n; \\ \prod_{b=\lfloor \ell/n \rfloor c} (\mathsf{L}_{(\ell \bmod n)}(B)) & \text{if } n \leq \ell < d. \end{cases}$$

As described in Section 4.1.1, we assume that c-many d-dimensional diagonal vectors of B are fully packed into a ciphertext. Let $\mathsf{ct}.\mathsf{B}_j$ be an encryption of $(\mathsf{L}_{cj}(B), \mathsf{L}_{cj+1}(B), \ldots, \mathsf{L}_{c(j+1)-1}(B))$ for $j \in [\tilde{n}]$. We first need rotations within diagonal vectors to obtain encryptions of $\mathsf{L}^{\ell}_0(B)$. Specifically, we generate a ciphertext $\mathsf{ct}.\mathsf{B}_{\tilde{n}t+j}$ of

$$\begin{aligned} &(\rho^t(\mathsf{L}_{cj}(B)), \rho^t(\mathsf{L}_{cj+1}(B)), \ldots, \rho^t(\mathsf{L}_{c(j+1)-1}(B))) \\ &= (\mathsf{L}^{\ell}_{nt+cj}(B), \mathsf{L}^{\ell}_{nt+cj+1}(B), \ldots, \mathsf{L}^{\ell}_{nt+c(j+1)-1}(B)) \end{aligned}$$

for $t \in [1, \tilde{d}]$. That is, we need to rotate within a subslot, but the CKKS scheme only supports 1-dimensional rotations. To achieve this, we perform an *internal rotation*[1], which requires two homomorphic rotations and two scalar multiplications. Since we repeat this operation for each rotation amount $t \in [1, \tilde{d}]$ on $\tilde{n}$ many input ciphertexts $\mathsf{ct}.\mathsf{B}_j$, this step requires $2(\tilde{d}-1)\tilde{n} = 2d/c$ rotations and $2(\tilde{d}-1)\tilde{n} = 2d/c$ scalar multiplications.

Similar to Eq. (3), we obtain ciphertexts for each set of c-many output diagonals by evaluating

$$\sum_{\ell \in [c], j \in [d/c]} \mathsf{SMult}(\rho^{d-\ell}(\mathsf{ct}.\mathsf{B}_j), \mathsf{pt}.\mathsf{A}_{j,\ell,r});$$

for $r \in [\tilde{n}]$, using appropriate plaintexts $\mathsf{pt}.\mathsf{A}_{j,\ell,r}$ from the matrix A. For each $j \in [d/c]$, the ciphertext $\mathsf{ct}.\mathsf{B}_j$ is rotated $(c-1)$ times, requiring $(c-1)(d/c) = d$ rotations. In conclusion, this algorithm takes approximately $2d/c + d = O(d)$ rotations and $2d/c + c(d/c)\tilde{n} = 2d/c + dn/c$ scalar multiplications.

1. For example, given that a ciphertext representing the vector $(u_0, u_1, u_2, u_3, v_0, v_1, v_2, v_3)$, it can be rotated to the left by one and right by three. Each ciphertext is applied by the mask to extract $(u_1, u_2, u_3, 0, v_1, v_2, v_3, 0)$ and $(0, 0, 0, u_0, 0, 0, 0, v_0)$. At the end, we get a ciphertext of $(u_1, u_2, u_3, u_0, v_1, v_2, v_3, v_0)$ by summing the two intermediate ciphertexts.

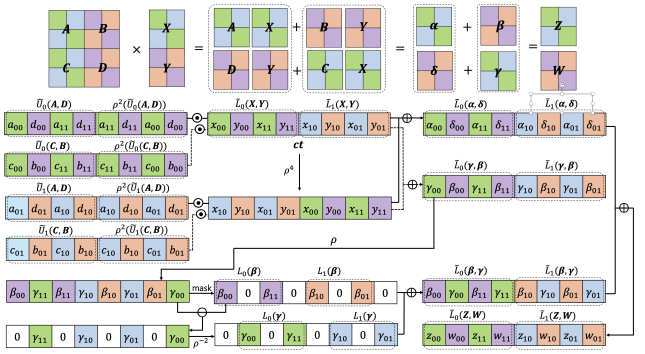

Figure 1. Diagonal block plaintext-ciphertext matrix multiplication. The ciphertext ct represents a stacked vector of $\tilde{L}_0(X, Y)$ and $\tilde{L}_1(X, Y)$. For simplicity, we do not include the complexification.

Algorithm 1 PLAINTEXT-CIPHERTEXT MATMUL

Input: Plaintexts $\mathsf{pt}.\mathsf{A}_{i,j,\ell,r}$ of the internally rotated upper diagonal vectors of $A \in \mathbb{R}^{d \times d}$; ciphertexts $\mathsf{ct}.\mathsf{B}_j$ of the lower-diagonally stacked vectors of $B \in \mathbb{R}^{d \times n}$ for $i \in [d/n]$; $j, r \in [n/c]$; $\ell \in [c]$

Output: Ciphertexts $\mathsf{ct}.\mathsf{C}_r$ of the lower-diagonally stacked vectors of C for $r \in [n/c]$

$\tilde{d} \leftarrow d/n$, $\tilde{n} \leftarrow n/c$
for $i = 0$ to $\tilde{d}-1$ **do**
  for $r = 0$ to $\tilde{n}-1$ **do**
    for $j = 0$ to $\tilde{n}-1$, $\ell = 0$ to $c-1$ **do**
      $\mathsf{ct}_{i,j,\ell,r} \leftarrow \mathsf{SMult}(\rho^{d-\ell}(\mathsf{ct}.\mathsf{B}_j), \mathsf{pt}.\mathsf{A}_{i,j,\ell,r})$
    $\mathsf{ct}_{i,r} \leftarrow \sum_{\ell \in [c], j \in [\tilde{n}]} \mathsf{ct}_{i,j,\ell,r}$
for $r = 0$ to $\tilde{n}-1$ **do**
  $\mathsf{ct}.\mathsf{C}_r \leftarrow \mathsf{SMult}(\mathsf{ct}_{0,r}, \mathsf{pt}_0)$
  for $i = 1$ to $\tilde{d}-1$ **do**
    $\mathsf{ct}_{i,r,R} \leftarrow \mathsf{SMult}(\mathsf{ct}_{i,r}, \mathsf{pt}_i)$
    $\mathsf{ct}_{i,r,L} \leftarrow \mathsf{ct}_{i,r} - \mathsf{ct}_{i,r,R}$
    $\mathsf{ct}^{\ell}_{i,r} \leftarrow \rho^{d(\tilde{d}-i)}(\mathsf{ct}_{i,r,R}) + \rho^{di}(\mathsf{ct}_{i,r,L})$
    $\mathsf{ct}.\mathsf{C}_r \leftarrow \mathsf{ct}.\mathsf{C}_r + \mathsf{ct}^{\ell}_{i,r}$
return $\mathsf{ct}.\mathsf{C}_r$

4.2.2. Diagonal Block Matrix Computation. To further enhance efficiency, we partition the two matrices into $n \times n$ blocks and perform the matrix product using the n-dimensional diagonal vectors from the submatrices. Specifically, 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 $i, k \in [\tilde{d}]$. Then, the submatrices in the resulting matrix C are calculated by blockwise multiplication, where $C_k = \sum_i A_{ik}B_k$ for $k \in [\tilde{d}]$. This blockwise computation is efficiently implemented using HE through the following three steps: (i) perform parallel matrix multiplications between submatrices in the same diagonal direction, (ii) rotate the intermediate results by an appropriate amount to align the partial results for final submatrix output into the same slots, and (iii) sum the aligned values to obtain the final blockwise product.

A concrete example is provided in Fig. 1, and the detailed procedure is described in Algorithm 1. First, for each

TABLE 1. COMPARISON OF PLAINTEXT-CIPHERTEXT MATRIX MULTIPLICATION METHODS OF AB WHERE $A \in \mathbb{R}^{d \times d}$ AND $B \in \mathbb{R}^{d \times n}$ WITH $n \leq d$ AND $s = cd$.

Method	#Smult	#Rot
Basic	$\frac{2d}{c} + \frac{dn}{c}$	$d + \frac{2d}{c}$
DiagBlock	$\frac{d}{c} + \frac{dn}{c}$	$n + \frac{2d}{c}$

$\ell \in [n]$, the ℓ-th lower diagonal vectors from the matrices $B_0, \ldots, B_{d-1}$ are interlaced to form the vector $\widetilde{L}_\ell(B) \in \mathbb{R}^d$, as shown in Eq. (4). Next, c such vectors are concatenated to be fully-packed into a single ciphertext. Specifically, for $j \in [\hat{n}]$, we define

$$\widetilde{L}_{[cj:c(j+1)}(B) = (\widetilde{L}_{cj}(B), \ldots, \widetilde{L}_{c(j+1)-1}(B)) \in \mathbb{R}^s$$

and denote the corresponding ciphertext as $\mathsf{ct}.B_j$. Similarly, we define $A^{(i)} = \{A_{i,0}, A_{i+1,1}, \ldots, A_{i+d-1,d-1}\}$ as the set of submatrices in the i-th diagonal order for $i \in [\hat{d}]$. The plaintext of their interlaced compactly packed upper diagonal vectors for $A^{(i)}$ is denoted as $\mathsf{pt}.A_{i,j,\ell,r}$ for $\ell \in [c]$, and $j, r \in [\hat{n}]$, as explained in Section 4.1.1.

Let $C_{i,k} = A_{i,k}B_k$ and $C^{(i)} = \{C_{i,0}, C_{i+1,1}, \ldots, C_{i+d-1,d-1}\}$. For each $i \in [\hat{d}]$ and $r \in [\hat{n}]$, we begin by computing the concatenated lower diagonal vectors $\widetilde{L}_{[cr:c(r+1)}(C^{(i)})$ (lines 5 and 6 in Algorithm 1). The parallel submatix multiplication requires $(c-1)\hat{n} \cdot n$ rotations and $n \cdot \hat{d}\hat{n} = dn/c$ scalar multiplications. Similar to an internal rotation operation, these results are internally rotated within the output diagonals of the same order, yielding the product result $\{C_{0,\hat{d}-i}, \ldots, C_{i,0}, \ldots, C_{\hat{d}-1,\hat{d}-1-i}\}$ (lines 10 to 12). This process involves $2(\hat{d}-1)\hat{n} \cdot 2d/c$ rotations and $(\hat{d}-1)\hat{n} \cdot d/c$ scalar multiplications to compute all the aligned results of the submatrix products. Finally, each lower diagonal vector of the intermediate results is added to produce the final output (line 13). We note that this ciphertext $\mathsf{ct}.C_r$ represents the plaintext vector of $\widetilde{L}_{[cr:c(r+1)}(C)$ where $C = \{C_0, C_1, \ldots, C_{d-1}\}$.

Although additional computations are required to rotate plaintext slots for interactions between values of different submatrices, intermediate submatrix products can be parallelized in a SIMD manner. Consequently, the number of rotations is reduced to approximately $n + 2d/c = O(n)$, enabling the block matrix multiplication algorithm to outperform the naive method. We present 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 directly applied in a similar manner to the plaintext-ciphertext matrix multiplication method. However, it is very challenging to generate encryptions of the plaintexts $\mathsf{pt}.A_{j,\ell,r}$ as explain in Section 4.1.1 from the stacked upper diagonal vectors of A. For example, consider the product AB with $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times n}$. Suppose that we are given the diagonally stacked ciphertext representing $(\mathsf{U}_{cj}(A), \mathsf{U}_{cj+1}(A), \ldots, \mathsf{U}_{c(j+1)-1}(A))$ for $j \in [\hat{m}]$, where $\hat{m} = m/c$. We apply the the internal rotation method to generate ciphertexts that are internally rotated by $r \in [1, n)$ positions for each input ciphertext. This procedure requires $2(n-1)\hat{m}$ rotations and scalar multiplications. After that, it takes roughly mn scalar multiplications to generate the packed ciphertexts in the form of $(\rho^{\ell}(\mathsf{U}_k(A)), \rho^{\ell+1}(\mathsf{U}_k(A)), \ldots, \rho^{c(\ell+1)-1}(\mathsf{U}_k(A)))$ for each $k \in [m]$ and $\ell \in [n/c]$. Instead of this naive extension, we leverage both the lower diagonals of the input matrices and perform the matrix computation in parallel to enhance performance.

4.3.1. Packed Lower-lower Matrix Multiplication. Suppose that we are given the ciphertext $\mathsf{ct}.A_j$ representing the lower-diagonally stacked vectors $(\mathsf{L}_{cj}(A), \mathsf{L}_{cj+1}(A), \ldots, \mathsf{L}_{c(j+1)-1}(A))$ for $j \in [\hat{m}]$. First, two rotated ciphertexts are generated as follows:

$$\mathsf{ct}_{j,\ell} = \rho^{n\lceil \ell \rceil_c + \ell}(\mathsf{ct}.A_j)$$
$$\mathsf{ct}^0_{j,\ell} = \rho^{n(\lceil \ell \rceil_c + 1) + \ell}(\mathsf{ct}.A_j)$$

Then $\mathsf{ct}_{j,\ell}$ is applied by two masks $\mathsf{m}_{\ell,0}$ and $\mathsf{m}_{\ell,1}$ to extract the first $(n-\ell)$ entries of $\rho^{\ell}(\mathsf{L}_{cj+r}(A))$ for each $\lceil \ell \rceil_c \leq r < c$ and $0 \leq r < \lceil \ell \rceil_c$, respectively. Similarly, $\mathsf{ct}^0_{j,\ell}$ is masked to extract the last ℓ entries of each $\rho^{\ell}(\mathsf{L}_{cj+r}(A))$ using $\mathsf{m}_{\ell,2}$ and $\mathsf{m}_{\ell,3}$. This procedure requires approximately $2\hat{m}n$ rotations and $4\hat{m}n$ scalar multiplications. Next, these ciphertexts are appropriately added together to obtain the internally rotated ciphertext $\mathsf{ct}.A_{j,\ell}$ representing $(\rho^{\ell}(\mathsf{L}_{cj}(A)), \rho^{\ell}(\mathsf{L}_{cj+1}(A)), \ldots, \rho^{\ell}(\mathsf{L}_{c(j+1)-1}(A)))$, i.e.,

$$\mathsf{ct}.A_{j,\ell} = (\mathsf{ct}_{j-1,\ell} \cdot \mathsf{m}_{\ell,0} + \mathsf{ct}_{j,\ell} \cdot \mathsf{m}_{\ell,1}) + (\mathsf{ct}^0_{j-1,\ell} \cdot \mathsf{m}_{\ell,2} + \mathsf{ct}^0_{j,\ell} \cdot \mathsf{m}_{\ell,3}). \quad (5)$$

We now assume that the ciphertext $\mathsf{ct}.B_\ell$ contains c copies of the ℓ-th lower diagonal vector of B for $\ell \in [n]$. Corollary 2 implies that the diagonally stacked ciphertext of $(\mathsf{L}_{cj}(C), \mathsf{L}_{cj+1}(C), \ldots, \mathsf{L}_{c(j+1)-1}(C))$ can be obtained by evaluating

$$\mathsf{ct}.C_j = \sum_{\ell \in [n]} \mathsf{Mult}(\mathsf{ct}.A_{j,\ell}, \mathsf{ct}.B_\ell). \quad (6)$$

4.3.2. Lazy Rotation Technique. We introduce the lazy rotation optimization, which reduces the required number of rotations by roughly a factor of two. The main observation is that the same rotation amount ℓ is used for the ciphertexts $\mathsf{ct}.A_{j,\ell}$ across different j. This consistency in rotation amounts simplifies the computation and enables efficient parallel processing.

We observe from Eq. (5) that the ciphertext $\mathsf{ct}.A_{j,\ell}$ can be computed as

$$\begin{aligned}\mathsf{ct}.A_{j,\ell} &= \mathsf{ct}_{j-1,\ell} \cdot \mathsf{m}_{\ell,0} + \mathsf{ct}_{j,\ell} \cdot \mathsf{m}_{\ell,1} + \rho^{n}(\mathsf{ct}_{j-1,\ell}) \cdot \mathsf{m}_{\ell,2} + \rho^{n}(\mathsf{ct}_{j,\ell}) \cdot \mathsf{m}_{\ell,3} \\ &= \mathsf{ct}_{j-1,\ell} \cdot \mathsf{m}_{\ell,0} + \mathsf{ct}_{j,\ell} \cdot \mathsf{m}_{\ell,1} + \rho^{n}(\mathsf{ct}_{j-1,\ell} \cdot \rho^{n}(\mathsf{m}_{\ell,2}) + \mathsf{ct}_{j,\ell} \cdot \rho^{n}(\mathsf{m}_{\ell,3})).\end{aligned}$$

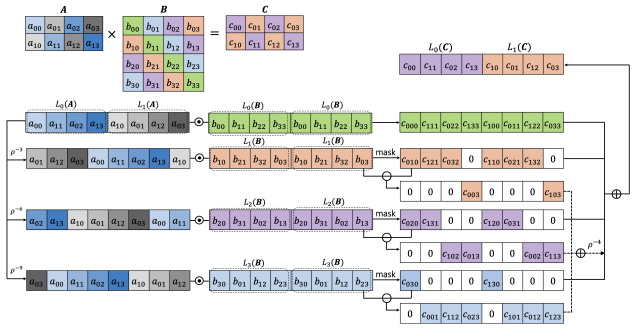

Figure 2. Ciphertext-ciphertext matrix multiplication. We denote $c_{ijk} = a_{ij}b_{jk}$. For simplicity, we do not include the complexification.

The first equality follows from the fact that $\mathsf{ct}^{0}_{j,\ell} = \rho^{n}(\mathsf{ct}_{j,\ell})$. By substituting this into Eq. (6), the ciphertext $\mathsf{ct}{:}\mathsf{C}_j$ can be expressed as follows:

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

Let $\mathsf{ct}{:}\mathsf{C}_{j,\ell} = \mathsf{Mult}(\mathsf{ct}_{j,\ell}; \mathsf{ct}{:}\mathsf{B}_\ell)$. Then the first term can be expressed as follows:

$$\sum_{\ell \in [n]} (\mathsf{Mult}(\mathsf{ct}_{j\ 1,\ell}; \mathsf{ct}{:}\mathsf{B}_\ell)\ \ \sigma_{\ell,0} + \mathsf{Mult}(\mathsf{ct}_{j,\ell}; \mathsf{ct}{:}\mathsf{B}_\ell)\ \ \sigma_{\ell,1}) = \sum_{\ell \in [n]} (\mathsf{ct}{:}\mathsf{C}_{j\ 1,\ell}\ \ \sigma_{\ell,0} + \mathsf{ct}{:}\mathsf{C}_{j,\ell}\ \ \sigma_{\ell,1}).$$

The second term can be expressed as follows:

$$\rho^{n}(\sum_{\ell \in [n]} \mathsf{Mult}(\mathsf{ct}_{j\ 1,\ell}\ \ \rho^{n}(\ \ \sigma_{\ell,2}) + \mathsf{ct}_{j,\ell}\ \ \rho^{n}(\ \ \sigma_{\ell,3}); \mathsf{ct}{:}\mathsf{B}_\ell)) = \rho^{n}(\sum_{\ell \in [n]} \mathsf{ct}{:}\mathsf{C}_{j\ 1,\ell}\ \ \rho^{n}(\ \ \sigma_{\ell,2}) + \mathsf{ct}{:}\mathsf{C}_{j,\ell}\ \ \rho^{n}(\ \ \sigma_{\ell,3}))$$

The first equation follows from the fact that $\mathsf{ct}{:}\mathsf{B}_\ell = \rho^{n}(\mathsf{ct}{:}\mathsf{B}_\ell)$.

We visualize the high-level idea of the lazy rotation technique in Fig. 2, and the detailed procedure is provided in Algorithm 2. The cost of the lazy rotation method is as follows. First, we apply one rotation to compute $\mathsf{ct}_{j,\ell}$ for each $j \in [m]$ and $1 \le \ell < n$ (line 5). Next, we perform mn homomorphic multiplications between $\mathsf{ct}_{j,\ell}$ and $\mathsf{ct}{:}\mathsf{B}_\ell$ (line 6). After that, we carry out $4mn$ scalar multiplications of the resulting ciphertexts with the masks $\sigma_{\ell,0}$; $\sigma_{\ell,1}$; $\rho^{n}(\ \ \sigma_{\ell,2})$; and $\rho^{n}(\ \ \sigma_{\ell,3})$ (lines 9 to 11). We note that the number of scalar multiplications can be reduced by $3mn$ using the fact that $\mathsf{ct}{:}\mathsf{C}_{j,\ell}\ \ \rho^{n}(\ \ \sigma_{\ell,3}) = \mathsf{ct}{:}\mathsf{C}_{j,\ell}\ \ \mathsf{ct}{:}\mathsf{C}_{j,\ell}\ \ \sigma_{\ell,0}\ \ \mathsf{ct}{:}\mathsf{C}_{j,\ell}\ \ \sigma_{\ell,1}\ \ \mathsf{ct}{:}\mathsf{C}_{j,\ell}\ \ \rho^{n}(\ \ \sigma_{\ell,2})$ (line 12). Finally, the output ciphertexts are appropriately summed with rotations by n positions (lines 14 to 16).

The main advantage of the lazy rotation technique is the reduction in the number of rotations from $2mn$ to mn.

Algorithm 2 CIPHERTEXT-CIPHERTEXT MATMUL

Input: Ciphertexts $\mathsf{ct}{:}\mathsf{A}_j$ of the lower-diagonally stacked vectors of $A \in \mathsf{R}^{m\ \ n}$ for $j \in [m/c]$; ciphertexts $\mathsf{ct}{:}\mathsf{B}_\ell$ of the replicated lower diagonal vectors of $B \in \mathsf{R}^{n\ \ n}$ for $\ell \in [n]$

Output: Ciphertexts $\mathsf{ct}{:}\mathsf{C}_j$ of the lower-diagonally stacked vectors of C

1: $m\ \ m/c$
2: **for** $j = 0$ to $m\ \ 1$ **do**
3: $\quad \mathsf{ct}{:}\mathsf{C}_j\ \ \mathsf{Mult}(\mathsf{ct}{:}\mathsf{A}_j; \mathsf{ct}{:}\mathsf{B}_0)$
4: $\quad$ **for** $\ell = 1$ to $n\ \ 1$ **do**
5: $\quad\quad \mathsf{ct}_{j,\ell}\ \ \ \rho^{n\lceil \ell/c \rceil - \ell}(\mathsf{ct}{:}\mathsf{A}_j)$
6: $\quad\quad \mathsf{ct}{:}\mathsf{C}_{j,\ell}\ \ \mathsf{Mult}(\mathsf{ct}_{j,\ell}; \mathsf{ct}{:}\mathsf{B}_\ell)$
7: **for** $\ell = 1$ to $n\ \ 1$ **do**
8: $\quad$ **for** $j = 0$ to $m\ \ 1$ **do**
9: $\quad\quad \mathsf{ct}_{j,\ell;0}\ \ \mathsf{SMult}(\mathsf{ct}{:}\mathsf{C}_{j,\ell}; \ \ \sigma_{\ell,0})$
10: $\quad\quad \mathsf{ct}_{j,\ell;1}\ \ \mathsf{SMult}(\mathsf{ct}{:}\mathsf{C}_{j,\ell}; \ \ \sigma_{\ell,1})$
11: $\quad\quad \mathsf{ct}_{j,\ell;2}\ \ \mathsf{SMult}(\mathsf{ct}{:}\mathsf{C}_{j,\ell}; \ \ \rho^{n}(\ \ \sigma_{\ell,2}))$
12: $\quad\quad \mathsf{ct}_{j,\ell;3}\ \ \mathsf{ct}{:}\mathsf{C}_{j,\ell}\ \ \ \mathsf{ct}_{j,\ell;0}\ \ \ \mathsf{ct}_{j,\ell;1}\ \ \ \mathsf{ct}_{j,\ell;2}$
13: **for** $j = 0$ to $m\ \ 1$ **do**
14: $\quad \mathsf{ct}{:}\mathsf{C}^{0}_j\ \ \sum_{1 \le \ell < n}(\mathsf{ct}_{j\ 1,\ell;0} + \mathsf{ct}_{j,\ell;1})$
15: $\quad \mathsf{ct}{:}\mathsf{C}^{00}_j\ \ \sum_{1 \le \ell < n}(\mathsf{ct}_{j\ 1,\ell;2} + \mathsf{ct}_{j,\ell;3})$
16: $\quad \mathsf{ct}{:}\mathsf{C}_j\ \ \mathsf{ct}{:}\mathsf{C}_j + \mathsf{ct}{:}\mathsf{C}^{0}_j + \ \ \rho^{n}(\mathsf{ct}{:}\mathsf{C}^{00}_j)$
17: **return** $\mathsf{ct}{:}\mathsf{C}_j$

TABLE 2. COMPARISON OF CIPHERTEXT-CIPHERTEXT MATRIX MULTIPLICATION METHODS OF AB WHERE $A \in \mathsf{R}^{m\ \ n}$ AND $B \in \mathsf{R}^{n\ \ n}$ WITH $n\ \ m$ AND $s = cn$.

Method	Cplx	#Smult	#Mult		#Rot
			#TProd	#KS	
Basic	✗	$\frac{4mn}{c}$	$\frac{mn}{c}$	$\frac{m}{c}$	$\frac{2mn}{c}$
Lazy rotations	✗	$\frac{3mn}{c}$	$\frac{mn}{c}$	$\frac{2m}{c}$	$\frac{mn}{c}$
	✓	$\frac{3mn}{2c}$	$\frac{mn}{2c}$	$\frac{2m}{c}$	$\frac{m(n+2)}{2c}$

When combined with the lazy relinearization technique (will be explained in Section 4.5), the number of key-switching operations during homomorphic multiplication is doubled to $2m$ compared to the naive approach. However, this increase is relatively small compared to the reduction in rotations, so it does not result in significant efficiency degradation. We also remark that the ciphertext-ciphertext matrix multiplication can be parallelized in a SIMD manner without incurring additional costs, as explained in Section 4.1.2.

4.4. Row-wise Computation over Diagonal Representation

Assume that the upper diagonal vectors of A are given in encrypted form, where $\mathsf{ct}{:}\mathsf{A}_k$ denotes the encryption of the k-th upper diagonal vector $\mathsf{U}_k(A)$. It follows from Lemma 4 that the vector formed by the t-th entries of the upper diagonal vectors corresponds to the t-th row of the matrix

TABLE 3. COMPARISON OF HOMOMORPHIC MATRIX MULTIPLICATIONS. THE CONCRETE NUMBERS ARE BASED ON THE BERT-BASE PARAMETERS: $n = 128$, $d = 768$, $d_k = 64$, $H = 12$, AND $s = 2^{15}$.

Operation	Method	# KeySwitch	
		Equation	Number
$\{XW_{Q;h}; XW_{K;h}; XW_{V;h}\}_{h\in[H]}$ $(X \in \mathbb{R}^{n \times d}; W \in \mathbb{R}^{d \times d_k})$	BOLT	$\frac{63nd}{s} + (\frac{s}{64n} - 1)\frac{nd_k}{s} - 3H$	297
	NEXUS	$6dd_kH$	3538944
	Powerformer	$\left(\sqrt{3n+d_k} + (\sqrt{3n+d_k} + \log(\frac{n}{d_k}))\frac{3H}{2}\right)\frac{d}{n}$	2612
	THOR	$\frac{n}{2} + 6(\frac{d}{2d_k} - 1)\frac{d_k}{c} + 3\frac{d_k}{c}$	180
$\{Q_hK_h^T\}_{h\in[H]}$ $(Q_h; K_h \in \mathbb{R}^{n \times n/2})$	BOLT	$(2n - 2 + n\log(\frac{n}{2}))H$	12264
	NEXUS	n^2H	196608
	Powerformer	$(\frac{5n}{2} + 4\sqrt{n})\frac{H}{2}$	2196
	THOR	$(\frac{6n}{c} + 2) + \frac{n}{8}(\log c + 3) + \frac{2n}{c} + 2(n - 1) + \frac{3n}{2c}$	444
$\{A_hV_h\}_{h\in[H]}$ $(A_h \in \mathbb{R}^{n \times n}; V_h \in \mathbb{R}^{n \times n/2})$	BOLT	$(2n - 1)\log n - H$	21420
	NEXUS		
	Powerformer	$(4n + 3\sqrt{14n} + 8\sqrt{n} + 2\sqrt{2n})\frac{H}{2}$	4614
	THOR	$\frac{n}{4}(\log c + 3) + \frac{n}{c}(1 + \frac{n+2}{4})$	492
$\mathsf{Concat}(\mathrm{Att}_0; \cdots; \mathrm{Att}_{H-1})W^O$ $(\mathrm{Att}_h \in \mathbb{R}^{n \times n/2}; W^O \in \mathbb{R}^{d \times d})$	BOLT	$\frac{31nd}{s} + (\frac{s}{32n} - 1)\frac{nd_k}{s} - H$	177
	NEXUS	$2d^2H$	14155776
	Powerformer	$4\sqrt{n}(\frac{d}{n})^2$	1656
	THOR	$\frac{d_k}{2} + 2(\frac{d}{n} - 1)\frac{n}{c}$	118

A. Therefore, if the target function $f(X)$ is applied row by row to A and can be approximated by a function $p(X)$, this function can be used to approximate f for each row of A. Specifically, let $= f(A)$ be the row-wise application of f to A. The homomorphically evaluated result $p(\mathsf{ct}:A_k)$ corresponds to an approximate result for the upper diagonal vectors of . This approach can be extended to a packed implementation, making it applicable for approximations such as the softmax and LayerNorm in secure transformer inference.

4.5. Optimization

Lazy homomorphic operations. We apply a lazy rescaling technique during plaintext-ciphertext multiplications by adjusting a scaling factor only once within the resulting ciphertext after multiple multiplications and additions (e.g., lines 5 and 6 in Algorithm 1). Additionally, we rely on a lazy key-relinearization technique during ciphertext multiplications, which enables tensor products between ciphertexts and summation over the extended ciphertexts, along with only a single relinearization via key-switching operation. For example, we perform key-switching operations after the computations in lines 14 and 15 of Algorithm 2. Consequently, by using this optimized approach, we reduce the number of key-switching operations during homomorphic multiplications from mn to $2m$ (Table 2).

Complexification. For simplicity of presentation, we focus on Proposition 1 given that m is even. Eq. (1) can be reformulated as follows:

$$\begin{aligned}
\mathsf{L}_r(C) &= \sum_{\ell \in 2[m=2]} \tau^r(\mathsf{U}_{[\ell \cdots r]_m}(A)) \cdot \mathsf{L}_\ell(B) \\
&\quad + \tau^r(\mathsf{U}_{[\ell+m=2 \cdots r]_m}(A)) \cdot \mathsf{L}_{\ell+m=2}(B)) \\
&= \sum_{\ell \in 2[m=2]} \mathsf{Re}(\tau^r(\mathsf{U}_{[\ell \cdots r]_r}(A)) \cdot \mathsf{L}_\ell(B)) \\
&= \mathsf{Re}(\sum_{\ell \in 2[m=2]} \tau^r(\mathsf{U}_{[\ell \cdots r]_m}(A)) \cdot \mathsf{L}_\ell(B));
\end{aligned}$$

where $\overline{\mathsf{U}}_k(A) = \mathsf{U}_k(A) - \sqrt{-1}\mathsf{U}_{k+m=2}(A)$ and $\overline{\mathsf{L}}_\ell(B) = \mathsf{L}_\ell(B) + \sqrt{-1}\mathsf{L}_{\ell+m=2}(B)$. Since $\mathsf{Re}(2x) = x + \mathsf{conj}(x)$, we have $\mathsf{L}_r(AB) = X + \mathsf{conj}(X)$ where

$$X = \sum_{\ell \in 2[m=2]} \tau^r(\overline{\mathsf{U}}_{[\ell \cdots r]_m}(A)/2)) \cdot \overline{\mathsf{L}}_\ell(B).$$

Thus, the number of scalar multiplications is reduced by half. A similar approach can be applied to all the proposed matrix multiplication algorithms.

4.6. Comparison

In Table 3, we compare the computational complexity of the proposed matrix multiplication algorithms for multi-head attention in the BERT-base model with BOLT [6], NEXUS [14], and Powerformer [13]. Since the key-switching operation turns out to be the dominant process in homomorphic operations [18], we count the number of the required key-switching operations during ciphertext multiplications and rotations in each matrix computation. To

ensure a fair comparison, we assume that all methods use the same HE parameters, with the ring dimension of $N = 2^{16}$ and the number of plaintext slots $s = 2^{15}$. Specifically, THOR uses the diagonal packing parameter $c = 16$ as noted in Section 4.1.2. THOR consistently outperforms the state-of-the-art HE-based approaches, BOLT and Powerformer, across all matrix operations.

5. Secure Non-linear Computations

5.1. Softmax

For a vector $\mathbf{x} = (x_i) \in \mathbb{R}^n$, the softmax function is defined as

$$\mathsf{Softmax}(\mathbf{x}) = \left(\frac{\exp(x_i)}{\sum_{j \in [n]} \exp(x_j)} \right)_{i \in [n]} .$$

5.1.1. Our Approach at a High Level. To approximate the exponential function e^x over a given interval $[a, b]$, we can directly approximate it by minimizing the least squares error. However, for larger intervals, direct polynomial approximation becomes less effective due to the rapid growth of the exponential function. To address this issue, we approximate the scaled exponential function $e^{x/r}$ over the scaled interval, and the result is then raised to the power of r.

After computing the exponential function, we aggregate the results and compute their inverse using Goldschmidt's division algorithm [19]. Specifically, Goldschmidt's algorithm uses an iterative process of repeatedly refining the inverse approximation using the update formula $f(x) = x(2 - x)$. The Adaptive Successive Over-Relaxation (aSOR) method [20] enhances this process by dynamically adjusting relaxation factors k_i in each iteration. These factors are applied as $x_{i+1} = k_i x_i (2 - k_i x_i)$, where $k_i = 2/(1 + \epsilon_i)$ is chosen to maximize the rate of convergence by efficiently reducing the error range. Here, ϵ_i represents the minimum value in the current iteration. We consistently apply this approach to the inverse operations throughout the paper.

5.1.2. The Two-phase Method. A large input range can introduce another significant computational challenges beyond stability issues, as it requires a substantial number of fractional bits. This makes it computationally intensive to maintain a necessary high precision within ciphertext. To address this issue and achieve accurate softmax results efficiently, we propose a new *two-phase method* which consists of the following steps.

We begin by shifting the input vector to enhance numerical stability during exponentiation. Given that the elements of the input vector are within the range $[a, b]$, each element is adjusted by subtracting the constant b. As a result, the shifted elements are within the interval $[-M, 0]$, where $M = b - a$. This adjustment prevents the exponential of large positive numbers, making it more manageable to approximate numerically. We then introduce three scaling factors: r_0, r_1, and r_2. The factor r_0 is selected to define the range for the exponential approximation, r_1 is used to manage the power of the initial scaled exponentiation, and r_2 determines the number of repeated squaring-and-normalization operations. In practice, r_2 is calculated as $r_2 = 2^{\lceil \log_2(M/(r_0 r_1)) \rceil}$.

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

Input: $\mathbf{x} = (x_i) \in [-M, 0]^n$, $r_1 \in \mathbb{Z}^+$, $r_2 \in \mathbb{Z}^+$
Output: An approximated softmax of $\mathbf{x}$

1: $z_i \leftarrow (\mathsf{AExp}(x_i/(r_0 r_1)))^{r_1}$; $i = 0 :: n-1$
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 $\log(r_2)$ **do**
5: $\quad z_i \leftarrow (y_i^{(j-1)})^2$; $i = 0 :: n-1$
6: $\quad S^{(j)} \leftarrow \sum_{i \in [n]} z_i$
7: $\quad y_i^{(j)} \leftarrow z_i \cdot \mathsf{AInv}(S^{(j)})$; $i = 0 :: n-1$
8: **return** $\mathbf{y} = (y_i^{(\log(r_2))})$

> **Phase 1:** The input vector is scaled by a factor of $1/(r_0 r_1 r_2)$. Then each element of the scaled input vector lies within the interval $[-M/(r_0 r_1 r_2), 0] \subseteq [-r_0, 0]$, enabling stable exponential computations in the subsequent steps. Specifically, the exponentials of the scaled inputs are approximated to a small-degree polynomial using the least squares method. Then, we compute the r_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 numerical stability and accuracy, the probabilities are refined through an iterative process of squaring and normalization for $\log r_2$ iterations.

Algorithm 3 provides an explicit description of our softmax approximation. Here, AExp and AInv refer to the approximated exponential and inverse functions. The output $\mathbf{y}^{(0)} = (y_i^{(0)})$ from the first phase is an approximation of $\mathsf{Softmax}(\mathbf{x}/r_2)$ over $[-r_0, 0]^n$ (line 3). Since

$$\mathsf{Softmax}(2\mathbf{x})_i = \frac{y_i^2}{\sum_{j \in [n]} y_j^2}$$

for $\mathbf{y} = (y_i) = \mathsf{Softmax}(\mathbf{x})$, the j-th update $\mathbf{y}^{(j)} = (y_i^{(j)})$ in the second phase is approximate to $\mathsf{Softmax}(2^j \mathbf{x}/r_2)$. After $\log r_2$ iterations, this process yields an approximate result of $\mathsf{Softmax}(\mathbf{x})$.

5.2. LayerNorm

In LayerNorm, the normalized output is computed as follows: for $i \in [n]$ and $j \in [d]$,

$$\mathsf{LayerNorm}(X)_{i,j} = \gamma_j \cdot \frac{X_{i,j} - \mu_i}{\sqrt{\sigma_i^2 + \varepsilon}} + \beta_j,$$

where $X_{i,j}$ is the input vector, μ_i is the mean, σ_i^2 is the variance, ε is a small constant for numerical stability, and

$_j$ and $_j$ are parameters that scale and shift the normalized output, respectively. We integrate the Goldschmidt's algorithm with the aSOR technique to approximate the inverse square root operation. Goldschmidt's inverse square root algorithm refines the approximation iteratively using the update formula $f(x) = x(3 \quad x)^2$=4. Similar to the softmax approximation, aSOR is applied to these factors as $x_{i+1} = k_i x_i(3 \quad k_i x_i)^2$=4, where k_i is selected to maximize the rate of convergence by reducing the error range efficiently.

5.3. GELU & Tanh

We employ a two-step approach to approximate the GELU and Tanh functions using a composition of low-degree polynomials. We first derive a small-degree polynomial $f_1(x)$ that approximates the function by minimizing the least squares error. While this polynomial effectively captures the overall behavior of the input function, it may lack precision in certain regions, especially near the boundaries. To enhance the accuracy, the output of $f_1(x)$ is used as the input for the polynomial $f_2(x)$. This second polynomial is optimized through least squares fitting to refine the approximation, correcting any residual errors. The resulting composite function $f_2(f_1(x))$ offers a close approximation of the target function over large ranges, providing an efficient and accurate approximation method. We apply the Paterson-Stockmeyer algorithm to efficiently evaluate the polynomials in terms of both running time and depth consumption. In our implementation, GELU is approximated using degree-31 and degree-27 polynomials, while Tanh is approximated using two degree-15 polynomials.

6. Secure Transformer Inference

6.1. System Setup & Threat Model

THOR is a secure inference protocol for transformer models. Our system involves two parties: the client and the service provider (or server). The service provider holds a fine-tuned transformer model, while the client seeks inference services from this model. We use BERT to demonstrate the efficiency of our system. Given that the original BERT model is trained on a public dataset, its tokenization process is robust and effectively maps sentences into a BERT-optimized space. We freeze the tokenization process during private fine-tuning.

The threat model of this paper is similar to that of previous private transformer inference works of NEXUS and Powerformer. We assume that the server is honest-but-curious. In other words, it follows the defined protocol precisely but may attempt to infer information from legitimately received messages during the interaction. Our security goal is to ensure that the server does not learn any information about the client's private input.

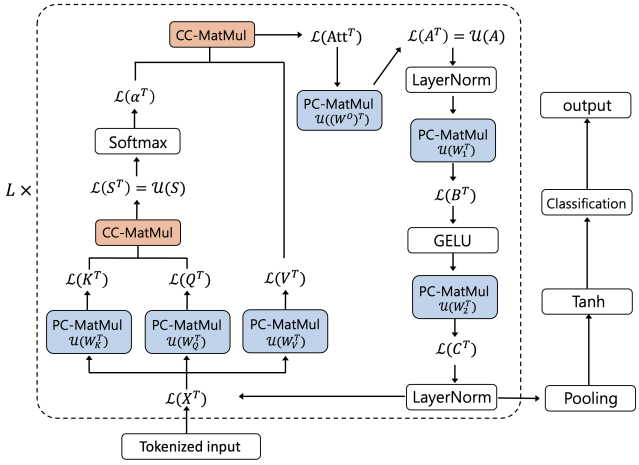

Figure 3. Overview of secure transformer inference in BERT.

6.2. THOR

Figure 3 provides an overview of secure transformer inference within the BERT architecture. In this setup, the client first computes the embeddings for a tokenized sentence using the embedding tables. The resulting output is then encrypted and securely fed into the transformer model for inference.

We consider the computation of $C = AB$, where the matrix B can be provided in the clear. According to the original matrix multiplication algorithm for $C = AB$, the encrypted lower diagonal vectors of C are computed using the rotated ciphertexts of the upper diagonal vectors of A and the plaintexts of the lower diagonal vectors of B. For square matrix multiplication as in Eq. (1), this process requires n^2 rotations to compute $^r(\mathsf{U}\cdot_r(A))$ from the encryptions of $\mathsf{U}_k(A)$. Alternatively, the encrypted lower diagonal vectors of the transposed matrix $C^|$ can be obtained using the rotated plaintexts of the upper diagonal vectors of $B^|$ and the ciphertexts of the lower diagonal vectors of $A^|$. This approach can be implemented without rotations by precomputing $^r(\mathsf{U}\cdot_r(B^|))$ in cleartext. As a result, the transposed approach achieves better computation efficiency than the original method. Throughout the paper, the THOR protocol relies on this encoding format in attention mechanism during transformer inference.

Attention layer: query, key, and value. Given the weight matrices $W_{Q;h}$ 2 $\mathsf{R}^{d \quad d_k}$ for h 2 $[H]$, each query matrix is computed as $Q_h = XW_{Q;h}$ 2 $\mathsf{R}^{n \quad d_k}$, and these matrices are horizontally stacked to form the query matrix Q 2 $\mathsf{R}^{n \quad d}$. In other words, we have $Q^| = W_Q^| X^|$, where $W_Q^|$ 2 $\mathsf{R}^{d \quad d}$ is a matrix formed by vertically stacking $W_{Q;h}$'s for h 2 $[H]$. This formulation enables secure computation using the diagonal block matrix multiplication method described in Section 4.2.2. This process generates the compactly packed ciphertexts $\mathsf{ct}{:}\mathsf{Q}_j$ for j 2 $[d_k$=c$]$, which correspond to the stacked lower diagonals of $Q^|$. We repeat this procedure to generate ciphertexts $\mathsf{ct}{:}\mathsf{K}_j$ and $\mathsf{ct}{:}\mathsf{V}_j$ for j 2 $[d_k$=c$]$, which represent the lower diagonals of the transposed key matrix

$K^{\intercal} = \{K_h^{\intercal} = W_{K;h}^{\intercal} X^{\intercal}\}_{h \in [H]}$ and the transposed value matrix $V^{\intercal} = \{V_h^{\intercal} = W_{V;h}^{\intercal} X^{\intercal}\}_{h \in [H]}$, respectively.

Attention score. As mentioned in Section 4.2.2, the resulting ciphertext $\mathsf{ct}:\mathsf{K}_j$ corresponds to the plaintext vector of $(\widetilde{\mathsf{L}}_{cj}(K^{\intercal}); \widetilde{\mathsf{L}}_{cj+1}(K^{\intercal}); \cdots; \widetilde{\mathsf{L}}_{c(j+1)-1}(K^{\intercal}))$, where each lower diagonal entries of $K_h^{\intercal}$ are interlaced within $\widetilde{\mathsf{L}}_k(K^{\intercal})$. Using the property in Lemma 3 that $\mathsf{L}_{\ell}(A) = \mathsf{L}_{n+\ell}(\mathsf{U}_{m-\ell}(A)) = \mathsf{L}_{n+\ell}(\mathsf{L}_{m-\ell}(A^{\intercal}))$ for $A \in \mathsf{R}^{n \times d_k}$, we first securely compute the lower diagonals of K_h's, denoted the output ciphertexts as $\mathsf{ct}:\mathsf{K}'_j$ for $j \in [d_k/c]$ (see Appendix B.1 for details). On the other hand, the ciphertexts of $\mathsf{ct}:\mathsf{Q}_j$ are transformed to contain c copies of the lower diagonal vectors of $Q_h^{\intercal}$'s, denoted as the output ciphertexts $\mathsf{ct}:\mathsf{Q}'_{\ell}$ for $\ell \in [d_k]$ (see Appendix B.2 for more detail). Finally, to compute the transposed dot-product $S_h^{\intercal} = K_h Q_h^{\intercal} \in \mathsf{R}^{n \times n}$ in parallel, we apply the ciphertext-ciphertext matrix multiplication on $\mathsf{ct}:\mathsf{K}'_j$ and $\mathsf{ct}:\mathsf{Q}'_{\ell}$, yielding the ciphertexts $\mathsf{ct}:\mathsf{S}_r$ for $r \in [n/c]$. These ciphertexts represent the stacked lower diagonals of $S_h^{\intercal}$'s, which are equivalent to its upper diagonals of S_h's.

Soft weights. Let $\mathcal{A}_h = \mathsf{Softmax}(S_h) \in \mathsf{R}^{n \times n}$ be the row-wise application of the softmax operation of the matrix S_h. The row-wise homomorphic evaluation described in Section 4.4 can extended to compute on compactly packed upper diagonals across multiple input matrices. Specifically, for an approximation p of the softmax function, the homomorphically evaluated result of $p(\mathsf{ct}:\mathsf{S}_0; \mathsf{ct}:\mathsf{S}_1; \cdots; \mathsf{ct}:\mathsf{S}_{n/c-1})$ represents an approximate result of the upper diagonal vectors of $\mathcal{A}_h$'s.

Attention heads. From the softmax evaluation, we obtain compactly packed ciphertexts of $\mathsf{ct}:\mathcal{A}_r$ for $r \in [n/c]$, which represent the lower diagonal vectors of $\mathcal{A}_h^{\intercal}$. Let $\mathrm{Att}_h = \mathcal{A}_h V_h \in \mathsf{R}^{n \times d_k}$ be the h-th context matrix. First, we transform $\mathsf{ct}:\mathcal{A}_r$ to contain c copies of the lower diagonal vectors of $\mathcal{A}_h^{\intercal}$'s, resulting in the output ciphertexts $\mathsf{ct}:\mathcal{A}'_{\ell}$ for $\ell \in [n]$. We then perform ciphertext-ciphertext matrix multiplication on the ciphertexts $\mathsf{ct}:\mathsf{V}_j$ and $\mathsf{ct}:\mathcal{A}'_{\ell}$. The resulting ciphertexts, $\mathsf{ct}:\mathrm{Att}_j$ for $j \in [d_k/c]$, correspond to the compactly packed lower diagonals of $\mathrm{Att}_h^{\intercal}$.

Multi-head attention. The multi-head attention is computed as $A = \mathsf{Concat}(\mathrm{Att}_0; \cdots; \mathrm{Att}_{H-1}) W^O \in \mathsf{R}^{n \times d}$ with $W^O \in \mathsf{R}^{d \times d}$. Here, the projection parameter W^O can be considered as a horizontal concatenation of $W_{O;h} \in \mathsf{R}^{d_k \times d}$ for $h \in [H]$. Let $H_h = \mathrm{Att}_h W_{O;h} \in \mathsf{R}^{n \times d}$. Then the multi-head attention A can be computed by summing H_h over $h \in [H]$.

Let Att be the matrix obtained by vertically stacking the matrices $\mathrm{Att}_h^{\intercal}$. We apply diagonal block matrix multiplication on $\mathsf{ct}:\mathrm{Att}_j$ to compute $(W^O)^{\intercal} \cdot \mathrm{Att} \in \mathsf{R}^{d \times n}$, resulting in a matrix formed by vertically concatenating $H_h^{\intercal}$'s. Finally, we sum up the output ciphertexts $\mathsf{ct}:\mathsf{H}_r$ over $r \in [d/c]$, which correspond to the upper diagonals of the matrix A.

LayerNorm1. We start with computation of the row-wise mean of the matrix A, given the encryptions of its 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 computation, involving an approximation of the inverse square root, is performed entrywise over the encrypted data. In the end, we obtain the ciphertexts $\mathsf{ct}:\mathsf{A}_r$ for $r \in [d/c]$, which correspond to the upper diagonal vectors of the normalized matrix A.

FC1. Let $B = AW_1$ with $W_1 \in \mathsf{R}^{d \times fd}$. The weight matrix W_1 is partitioned into f submatrices $W_{1;t} \in \mathsf{R}^{d \times d}$. Let $B_t = AW_{1;t} \in \mathsf{R}^{n \times d}$. We apply diagonal block plaintext-ciphertext multiplication on $\mathsf{ct}:\mathsf{A}_r$ to compute the compactly packed lower diagonals of $B_t^{\intercal}$, denoted as $\mathsf{ct}:\mathsf{B}_{t;r}$ for $r \in [d/c]$.

GELU. The approximation of the GELU is performed entrywise over the encrypted data, enabling it to be implemented using the SIMD homomorphic operations on the input ciphertexts, denoted as $\mathsf{ct}:\mathsf{G}_{t;r}$.

FC2. For $t \in [f]$, let $C_t = G_t W_{2;t}$, where $G_t = \mathsf{GELU}(B_t)$, $W_{2;t} \in \mathsf{R}^{d \times d}$, and $C = \sum_{t \in [f]} C_t \in \mathsf{R}^{n \times d}$. We apply diagonal block matrix multiplication on $\mathsf{ct}:\mathsf{G}_{t;r}$ to compute the compactly packed lower diagonals of $C_t^{\intercal}$. Finally, we sum up the ciphertexts $\mathsf{ct}:\mathsf{C}_{t;r}$ over $t \in [f]$ to generate the ciphertexts $\mathsf{ct}:\mathsf{C}_r$, which correspond to the lower diagonals of $C^{\intercal}$.

LayerNorm2. Similar operations can be applied to obtain the normalized ciphertexts $\mathsf{ct}:\mathsf{X}_r$ for $r \in [d/c]$, which represent the lower diagonals of $X^{\intercal}$. These are used as input for the next attention layer.

Pooler & classification. Finally, the dense layers are computed on $\mathsf{ct}:\mathsf{X}_r$ by applying the plaintext-ciphertext multiplication algorithm.

We remark that our architecture is carefully designed to eliminate the need for repacking or reshaping the output from one encoder layers to match the input format of the subsequent layer. This seamless alignment enables our matrix multiplication algorithms to perform efficiently during secure transformer inference without incurring additional computational costs.

7. Evaluation

7.1. Experimental Setup

BERT Model and Dataset. We follow the BERT-base model with the number of encoding blocks $L = 12$, the number of multi-heads $H = 12$, the embedding dimension $d = 768$, the number of input tokens $t = 128$. 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 $4d$ (i.e., $f = 4$). We follow the default fine-tuning hyper-parameters in [21], e.g., batch size 16, learning rate $5 \cdot 10^{-5}$ and epoch 5.

We use the MRPC (Microsoft Research Paraphrase Corpus) dataset [22] from the GLUE tasks as our dataset. We performed fine-tuning on the training set and evaluated accuracy on the official validation set.

Libraries and Configurations. THOR is evaluated using the Liberate.FHE library [23], which implements the RNS variant of the CKKS scheme [24]. The ciphertext dimension

TABLE 4. SOFTMAX COMPARISON BETWEEN [26] AND OUR METHOD.

Method	# Iter.	Depth	# Mult	Precision (bits)
[26]	5	40	106	11
Ours	3	30	30	8

was set as $N = 2^{16}$ to support the bootstrapping operation, providing plaintext slots of length $s = 2^{15}$. The ciphertext modulus q was chosen to ensure a 128-bit security level [25]. Under these HE parameters, 13 multiplicative levels are available before bootstrapping is needed. All experiments were conducted on an Intel Platinum 8462 CPU at 2.8GHz and a single NVIDIA A100 GPU.

7.2. Micro-benchmark Evaluation of Softmax

In Table 4, we compare our softmax approximation algorithm with that of Cho et al. [26] in terms of the number of iterations (i.e., the number of approximate normalization operations), multiplicative depth, and the number of homomorphic multiplications required for softmax evaluation excluding initial exponentiation computations. We also measured the worst-case accuracy of the approximations. The input data for this evaluation was extracted from 12,800 samples from the third layer of the MRPC dataset, with $x_i \in [-70, 70]$ and $n = 128$.

It follows from [26, Theorem 3.4] that the softmax approximation requires $k = \lceil \log 140 - \log \ln 128 \rceil = 5$ iterations of repeated normalization-and-squaring operations. Since their experimental results were based on random inputs, we made slight adjustments to the inverse square root approximations used in the normalization procedure to fit the MRPC dataset. Specifically, the first and last iterations require depths of 7 and 11 for the inverse square roots, while the intermediate iterations require a depth of 4. In contrast, our algorithm achieves similar accuracy with only 3 iterations of squaring and normalization by the inverse root approximation. For this evaluation, the scaling factors are chosen as follows: $\alpha_0 = \frac{140}{8} = 17.5$, $\alpha_1 = 2$, and $\alpha_2 = 4$.

Since each iteration in both algorithms requires a single bootstrapping, the reduced number of iterations in our method lowers the required bootstrapping operations, thereby reducing the overall computational time and resource usage. Additionally, we optimize depth consumption and minimize multiplications for normalization by applying the aSOR-based Goldschmidt inverse algorithm. The experimental results show that our approach achieves substantial efficiency gains without compromising accuracy, making it suitable for applications that requires rapid and resource-efficient computation, particularly in Machine Learning as a Service (MLaaS) models.

7.3. End-to-enc Inference Evaluation

Performance Breakdown. Table 5 shows a detailed breakdown of the execution time for the BERT-base model. Due to the significant cost of bootstrapping, THOR's architecture is designed to perform bootstrapping operations when the number of intermediate or output ciphertexts is minimal. Nevertheless, it turns out that more than 53.96% of the total runtime is attributed to bootstrapping operations. Since the complexity of basic homomorphic operations, such as additions, multiplication, and rotations, scales almost linearly with the ciphertext level, we perform homomorphic matrix multiplications at the lowest possible level. The linear layers in the attention mechanism and the fully connected layers account for 36.70% of the total runtime. Our optimized matrix multiplication algorithms provide performance gains, resulting in a relatively lower runtime. While the total inference time of THOR is 10.43 minutes in a single GPU setting, NEXUS takes approximately 37.3 sec $\times$ 128 inputs $\approx$ 78.93 minutes with four A100 GPUs to process a single prediction request on the BERT-base model.

TABLE 5. BREAKDOWN OF THE EXECUTION TIME OF THOR.

Operation	Input	Time (sec)
Attention layer	$3 \times (\mathbb{R}^{128 \times 768} \times \mathbb{R}^{768 \times 64})$	49.77
Attention score	$12 \times (\mathbb{R}^{128 \times 64} \times \mathbb{R}^{64 \times 128})$	32.53
Softmax	$12 \times (\mathbb{R}^{128 \times 128})$	15.53
Attention heads	$12 \times (\mathbb{R}^{128 \times 128} \times \mathbb{R}^{128 \times 64})$	20.63
Multi-head attention	$\mathbb{R}^{128 \times 768} \times \mathbb{R}^{768 \times 768}$	27.43
LayerNorm1	$\mathbb{R}^{128 \times 768}$	7.13
FC1	$\mathbb{R}^{128 \times 768} \times \mathbb{R}^{768 \times 3072}$	49.80
GELU	$\mathbb{R}^{128 \times 3072}$	29.42
FC2	$\mathbb{R}^{128 \times 3072} \times \mathbb{R}^{3072 \times 768}$	49.19
LN2	$\mathbb{R}^{128 \times 768}$	4.10
Pooler & Classification	$\mathbb{R}^{128 \times 768}$	2.70
Bootstrappings	-	337.86
Total	-	104.35

TABLE 6. PERFORMANCE FOR THE BASELINE AND OUR METHOD.

Metrics	Baseline	Unencrypted			Encrypted
		G	G-LN	G-LN-S	
Accuracy	85.29	85.54	85.54	85.78	84.80
F1-score	89.90	90.05	90.05	90.24	89.49

Accuracy. In Table 6, we present the performance of the BERT-base model across five different settings to evaluate the effectiveness of our approximation components: (a) Baseline: fine-tuning the pre-trained model with no approximation, (b) G: fine-tuning the model with GELU replaced by our GELU approximation, (c) G-LN: additionally fine-tuning the model with layer normalization operation replaced by our layer normalization approximation, (d) G-LN-S: implementing full approximation, including the softmax replaced by our approximation model, and (e) Encrypted: applying the full approximation to encrypted data for secure inference. As in previous works, we report the accuracy and the F1 score. Overall, the accuracy achieved by THOR matches that of the plaintext model, with an accuracy drop of 0.49% compared with the baseline. We attribute this not to approximations of the non-linear functions but rather to computational error from homomorphic operations.

8. Related Work

8.1. Privacy-preserving Transformer Inference

Several studies have investigated how to preserve data privacy during transformer inference. In the MPC-based approach, MPCFormer [8] introduces a private inference system for the BERT-base model using secret-sharing techniques. More recently, PrivFormer [9] and PUMA [10] have proposed secure inference for transformers in a 3PC setting. In the hybrid approach using HE, Iron [5] and BOLT [6] present HE-based matrix multiplication algorithms that utilize ciphertext packing techniques. Despite these advancements in the protocol, they still incur a high communication cost; for example, BOLT requires 25.74 GB of communication for an end-to-end inference on the BERT-base model under one of the WAN settings.

In the HE-based approach, THE-X [11] proposed crypto-friendly approximation methods for non-linear functions in transformer inference. Specifically, THE-X replaces the softmax function with a combination of ReLU and a three-layer linear neural network, substitutes GELUs with ReLUs, and replaces LayerNorm with an affine transformation with learnable parameters. However, their scenario is different from ours: intermediate results of non-linear layers (input of ReLU) are sent to the client, who then performs computation on the decrypted results and re-encrypts them before sending them back to the server. This approach introduces two significant issues: (i) the frequent back-and-forth data exchange between the client and server increases communication costs significantly and (ii) the inputs to RELU are exposed to the client during this process, leading to potential privacy leakages. Recent works of [12, 13] have focused on the BERT-tiny model. Thus, cryptographic techniques including homomorphic matrix multiplication algorithms are not scalable to larger transformer models with high embedding dimensions and a large number of layers.

The intersection of HE and BERT was investigated in [27], where BERT embeddings are used for training a HE-based logistic regression model. However, since the primary focus is on implementing the logistic regression model, the BERT inference is performed by the client rather than the server, and only the text classification is conducted on encrypted data.

8.2. Homomorphic Matrix Multiplication

Various approaches have been investigated to optimize plaintext-ciphertext matrix multiplication (PC-MM). Gazelle [28] and Iron [5] rely on inner dot products of input matrices, resulting in a non-compact output packing on plaintext slots or coefficients after computation. BOLT [6], the current state-of-the-art for PCMM, still suffers from computational complexity in large transformer models. Furthermore, it aims to minimize the multiplicative depth for ciphertext-ciphertext matrix multiplication due to limitations in the HE parameters when integrated with MPC. As a result, the computation is suboptimal, and its computational complexity is significantly higher than ours.

The algorithms proposed in [12, 29] perform inner dot products over packed ciphertexts, requiring numerous slot permutations to accumulate values located in different plaintext slots. This approach relies on repetitive rotate-and-sum operations, leading to wasted slots in the resulting ciphertext after homomorphic matrix multiplication. Consequently, this increases computational overhead due to the need for repacking the non-compact output into multiple ciphertexts.

NEXUS [14] employs various packing methods similar to BOLT [6], including component-wise, row-wise, column-wise, and diagonal-wise packings. However, the details of changes in the packing structure during secure inference are not fully explained. On the other hand, PowerFormer [13] leverages the state-of-the-art ciphertext-ciphertext matrix multiplication algorithms of Jiang et al. [30]. Despite these algorithmic improvements, the matrix multiplication algorithms remain suboptimal for rectangular matrix multiplication, as they result in redundant entries after computation and these sparsely packed ciphertexts are then used as input for the subsequent matrix computations.

The recent work of Bae et al. [31] proposed the MaMBo (Matrix Multiplication Bootstrapping) framework, which performs PC-MM by leveraging multi-secret variants of RLWE to compactly represent multiple ciphertexts. However, this approach does not address the challenge of performing multiplication between ciphertexts, which is a necessary operation for tasks such as transformer inference. In contrast, THOR introduces a unified concept that not only retains the efficiency of multiplication between plaintext and ciphertext, but also supports multiplication between ciphertexts in an efficient manner, along with other complex operations required for implementing large language models in homomorphic encryption.

8.3. Softmax Computation

In NEXUS [14], the softmax is approximated using the Taylor series expansion of the exponential function and Goldschmidt's division algorithm. To improve numerical stability of the exponential function, the maximum value of each row in the input matrix is subtracted from each element. In their implementation, this maximum value is taken as a constant for efficiency, which is impractical in real-world applications. As explained in [32], this value should be computed through a substantial number of comparison operations on encrypted data, leading to significant computational overhead.

An alternative method for computing softmax on encrypted data is to replace it with a more HE-friendly function, avoiding unstable approximation methods. For instance, PowerFormer [13] replaced the softmax function with a ReLU-based function, while [33] substituted it with Gaussian kernels to enable secure computation without division. However, these methods achieve only comparable performance to the original softmax in small BERT models (e.g., two-layer BERT model) on encrypted data.

9. 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 a significant speedup in attention mechanisms compared to previous approaches. With recent advancements in algorithmic efficiency of HE and implementation optimization through hardware accelerators, we believe that performance can be significantly accelerated and parallelized, further enhancing THOR's performance. The cryptographic techniques in THOR are applicable to the other transformer encoders that utilize different nonlinear approximations from ours, such as [13, 33], as well as transformer decoders such as GPT [34]. Additionally, our fundamental developments can be extended to other privacy-preserving machine learning applications, including neural network models and transfer learning.

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

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

[3] Openai, "chatgpt," https://openai.com/blog/chatgpt, 2022.

[4] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "LLaMA: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[5] M. Hao, H. Li, H. Chen, P. Xing, G. Xu, and T. Zhang, "Iron: Private inference on transformers," *Advances in neural information processing systems*, vol. 35, pp. 15 718–15 731, 2022.

[6] Q. Pang, J. Zhu, H. Möllering, W. Zheng, and T. Schneider, "BOLT: Privacy-preserving, accurate and efficient inference for transformers," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 4753–4771.

[7] W.-j. Lu, Z. Huang, Z. Gu, J. Li, J. Liu, C. Hong, K. Ren, T. Wei, and W. Chen, "BumbleBee: Secure two-party inference framework for large transformers," *Manuscript, to appear in Network and Distributed System Security Symposium 2025*, p. available at https://eprint.iacr.org/2023/1678.

[8] D. Li, H. Wang, R. Shao, H. Guo, E. Xing, and H. Zhang, "MPCFORMER: Fast, performant and private transformer inference with MPC," in *The Eleventh International Conference on Learning Representations*, 2023.

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

[10] Y. Dong, W.-j. Lu, Y. Zheng, H. Wu, D. Zhao, J. Tan, Z. Huang, C. Hong, T. Wei, and W. Chen, "PUMA: Secure inference of LLaMA-7B in five minutes," *arXiv preprint arXiv:2307.12533*, 2023.

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

[12] L. Rovida and A. Leporati, "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*, 2024, pp. 3–13.

[13] D. Park, E. Lee, and J.-W. Lee, "Powerformer: Efficient privacy-preserving transformer with batch rectifier-power max function and optimized homomorphic attention," *Cryptology ePrint Archive, Paper 2024/1429*, 2024.

[14] J. Zhang, J. Liu, X. Yang, Y. Wang, K. Chen, X. Hou, K. Ren, and X. Yang, "Secure transformer inference made non-interactive," *Manuscript, to appear in Network and Distributed System Security Symposium 2025*, p. available at https://eprint.iacr.org/2024/136, 2025.

[15] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the forty-first annual ACM symposium on Theory of computing*, 2009, pp. 169–178.

[16] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *International Conference on the Theory and Application of Cryptology and Information Securitybossuat2024security*. Springer, 2017, pp. 409–437.

[17] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "Bootstrapping for approximate homomorphic encryption," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2018, pp. 360–384.

[18] M. Kim, D. Lee, J. Seo, and Y. Song, "Accelerating HE operations from key decomposition technique," in *Annual International Cryptology Conference*. Springer, 2023, pp. 70–92.

[19] R. E. Goldschmidt, "Applications of division by convergence," Ph.D. dissertation, Massachusetts Institute of Technology, 1964.

[20] J. Moon, Z. Omarov, D. Yoo, Y. An, and H. Chung, "Adaptive successive over-relaxation method for a faster iterative approximation of homomorphic operations," *Cryptology ePrint Archive, Paper 2024/1366*, 2024.

[21] Bert, https://github.com/google-research/bert, 2019, google.

[22] B. Dolan and C. Brockett, "Automatically constructing a corpus of sentential paraphrases," in *Third international workshop on paraphrasing (IWP2005)*, 2005.
[23] DESILO, "Liberate.FHE: A new FHE library for bridging the gap between theory and practice with a focus on performance and accuracy," https://github.com/Desilo/liberate-fhe, 2023.
[24] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "A full RNS variant of approximate homomorphic encryption," in *International Conference on Selected Areas in Cryptography*. Springer, 2018, pp. 347–368.
[25] J.-P. Bossuat, R. Cammarota, J. H. Cheon, I. Chillotti, B. R. Curtis, W. Dai, H. Gong, E. Hales, D. Kim, B. Kumara *et al.*, "Security guidelines for implementing homomorphic encryption," *Cryptology ePrint Archive*, 2024.
[26] W. Cho, G. Hanrot, T. Kim, M. Park, and D. Stehlé, "Fast and accurate homomorphic softmax evaluation," in *Proceedings of the 2024 ACM SIGSAC conference on computer and communications security*, 2024.
[27] G. Lee, M. Kim, J. H. Park, S.-w. Hwang, and J. H. Cheon, "Privacy-preserving text classification on BERT embeddings with homomorphic encryption," in *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2022, pp. 3169–3175.
[28] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *27th USENIX security symposium (USENIX security 18)*, 2018, pp. 1651–1669.
[29] E. Crockett, "A low-depth homomorphic circuit for logistic regression model training," *Cryptology ePrint Archive*, 2020.
[30] X. Jiang, M. Kim, K. Lauter, and Y. Song, "Secure outsourced matrix computation and application to neural networks," in *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, 2018, pp. 1209–1222.
[31] Y. Bae, J. H. Cheon, G. Hanrot, J. H. Park, and D. Stehlé, "Plaintext-ciphertext matrix multiplication and FHE bootstrapping: Fast and fused," in *Annual International Cryptology Conference*. Springer, 2024, pp. 387–421.
[32] S. Lee, G. Lee, J. W. Kim, J. Shin, and M.-K. Lee, "HETAL: Efficient privacy-preserving transfer learning with homomorphic encryption," in *International Conference on Machine Learning*. PMLR, 2023, pp. 19 010–19 035.
[33] D. Rho, T. Kim, M. Park, J. W. Kim, H. Chae, J. H. Cheon, and E. K. Ryu, "Encryption-friendly LLM architecture," *arXiv preprint arXiv:2410.02486*, 2024.
[34] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

Appendix A. Proofs of Main Results

A.1. Proof of Proposition 1

Proof. Let $C = AB \in \mathsf{R}^{n \times n}$. 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 the r-th shifted vector $\rho^{r}(\mathsf{U}_{\ell})$ and the lower diagonal vector $\mathsf{L}_{\ell}(B)$, respectively. Therefore, the t-th entry of the vector $\sum_{\ell \in [m]} \rho^{r}(\mathsf{U}_{\ell}) \odot \mathsf{L}_{\ell}(B)$ is as follows:

$$\begin{aligned}\Big(\sum_{\ell \in [m]} \rho^{r}(\mathsf{U}_{\ell}) \odot \mathsf{L}_{\ell}(B)\Big)[t] &= \sum_{\ell \in [m]} A_{r+t,\ell+t} \cdot B_{\ell+t,t} \\ &= C_{r+t,t} \\ &= \mathsf{L}_{r}(AB)[t].\end{aligned}$$

□

A.2. Proof of Proposition 2

Proof. Let $C = AB \in \mathsf{R}^{m \times n}$. We first consider the t-th entry of the r-th shifted vector $\rho^{r}(\mathsf{U}^{0}_{\ell,k})$ for $r \in [m]$ and $\ell, t \in [n]$ If $\ell - r < m$, this entry corresponds to the t-th entry of $\rho^{r}(\mathsf{U}_{\ell})$, which is equal to the element $A[r+t \pmod m, \ell+t \pmod n]$. Otherwise, suppose that $\ell - r = mQ + R$ for some integers $Q \in \mathbb{Z}$ and $R \in [m]$. In this case, the entry corresponds to the t-th entry of

$$\begin{aligned}\rho^{r}(\mathsf{U}^{0}_{\ell,k}) &= \rho^{r}(\rho^{mb(\ell-r)/mc}(\mathsf{U}_{((\ell-r) \bmod m)})) \\ &= \rho^{r}(\rho^{mQ}(\mathsf{U}_{R})) \\ &= \rho^{r+mQ}(\mathsf{U}_{R}).\end{aligned}$$

We note that

$$\begin{aligned}\rho^{r+mQ}(\mathsf{U}_{R}(A)) &= \rho^{r+mQ}(A_{0,R}, A_{0,R+1}, \ldots) \\ &= (A_{r,r+mQ+R}, A_{r+1,r+mQ+R+1}, \ldots) \\ &= (A_{r,\ell}, A_{r+1,\ell+1}, \ldots).\end{aligned}$$

Thus, the t-th entry of $\rho^{r+mQ}(\mathsf{U}_{R}(A))$ 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 the lower diagonal vector $\mathsf{L}_{\ell}(B)$. Therefore, for each $r \in [m]$, the t-th entry of $\sum_{\ell \in [n]} \rho^{r}(\mathsf{U}^{0}_{\ell,k}) \odot \mathsf{L}_{\ell}(B)$ is as follows:

$$\begin{aligned}\Big(\sum_{\ell \in [n]} \rho^{r}(\mathsf{U}^{0}_{\ell,k}) \odot \mathsf{L}_{\ell}(B)\Big)[t] &= \sum_{\ell \in [n]} A_{r+t,\ell+t} \cdot B_{\ell+t,t} \\ &= C_{r+t,t} \\ &= \mathsf{L}_{r}(AB)[t].\end{aligned}$$

□

A.3. Proof of Corollary 1

Proof. By definition 3, we have

$$\mathsf{L}^{0}_{n,k} = \rho^{m(n/m-1)}(\mathsf{L}_{m,k}) = \rho^{n-m}(\mathsf{L}_{m,k})$$

for $k \in [m]$. It follows from Lemma 3 that

$$\begin{aligned} \mathsf{U}_k(A) &= \quad^{m+k}(\mathsf{L}_{m\ \ k}(A)) \\ &= \quad^{m+k}(\quad^{n+m}(\mathsf{L}^0_{n\ \ k}(A))) \\ &= \quad^{k}(\mathsf{L}^0_{n\ \ k}) \end{aligned}$$

for $k \in [m]$. Therefore, we have

$$^{r}(\mathsf{U}_{\ell\ r}) = \ ^{r}(\quad^{\ell}\ ^{r}(\mathsf{L}^0_{n\ \ (\ell\ r)})) = \quad^{\ell}(\mathsf{L}^0_{n\ \ \ell+r}(A));$$

as desired.

□

A.4. Proof for Corollary 2

Proof. The upper diagonal vector $\mathsf{U}^0_k(A) \in \mathsf{R}^n$ is equal to $\mathsf{U}_k(A)$ when $\ell \quad r < m$. By Lemma 3, we have $^{r}(\mathsf{U}^0_{\ell\ r}(A)) = \ ^{r}(\mathsf{U}_{\ell\ r}(A)) = \ ^{r}(\quad^{\ell}\ ^{r}(\mathsf{L}_{m\ \ \ell+r}(A))) = \quad^{\ell}(\mathsf{L}_{[m\ \ \ell+r]_m}(A))$. Now, suppose that $\ell \quad r \quad m$. Then we have

$$\begin{aligned} ^{r}(\mathsf{U}^0_{\ell\ r}(A)) &= \ ^{r}\left(\quad^{m\ b(\ell\ r)=mc}(\mathsf{U}_{[\ell\ r]_m}(A))\right) \\ &= \ ^{r}\left(\quad^{m\ b(\ell\ r)=mc+[\ell\ r]_m}(\mathsf{L}_{m\ \ [\ell\ r]_m}(A)\right) \\ &= \ ^{r}(\quad^{\ell}\ ^{r}(\mathsf{L}_{m\ \ [\ell\ r]_m}(A)) \\ &= \quad^{\ell}(\mathsf{L}_{[m\ \ \ell+r]_m}(A)). \end{aligned}$$

□

Appendix B. Homomorphic Matrix Operations

B.1. Matrix Transposition

Algorithm 4 performs matrix transposition which transforms the upper-diagonally stacked vectors of a matrix B into the upper-diagonally stacked vectors of $B^{\mid}$. Equivalently, the output ciphertexts can be also considered as the lower-diagonally stacked vectors of B. We define two masking vectors $_{j;k;0};\ _{j;k;1} \in \mathsf{R}^s$ which have entries as follows:

$$_{j;k;0}[i] = \begin{cases} 1 & \text{if } a \quad i < a + n \quad b; \\ 0 & \text{otherwise;} \end{cases}$$

$$_{j;k;1}[i] = \begin{cases} 1 & \text{if } [a \quad b]_s \quad i < [a \quad b]_s + b; \\ 0 & \text{otherwise;} \end{cases}$$

where $a = [\ kn]_s$ and $b = [n \quad k \quad cj]_n$. This transposition procedure requires $(2n \quad 1)$ scalar multiplications and $(n \quad 1 + n/c)$ rotations in total. By leveraging both real and complex numbers for rotation, the total computational cost changes to $4n$ scalar multiplications and $\frac{(c+2)n}{2c}$ rotations.

Algorithm 4 TRANSPOSE

Input: Ciphertexts $\mathsf{ct}:\mathsf{B}_j$ of the upper-diagonally stacked vectors of $B \in \mathsf{R}^{n\ \ n}$ for $j \in [n/c]$
Output: Ciphertexts $\mathsf{ct}:\mathsf{B}^{\ell}$ of the upper-diagonally stacked vectors of $B^{\mid}$ for $\ell \in [n/c]$

for $j = 0$ to $n/c \quad 1$ **do**
  $\mathsf{ct}^0 \quad ^{[n\ \ cj]_n}(\mathsf{ct}:\mathsf{B}_j)$
  $l \quad j \pmod{n/c}$
  $\mathsf{ct}_{\ell;0} \quad \mathsf{SMult}(\mathsf{ct}^0;\ _{j;0;0})$
  $\mathsf{ct}_{\ell;1} \quad \mathsf{SMult}(\mathsf{ct}^0;\ _{j;0;1})$
for $j = 0$ to $n/c \quad 1$ **do**
  $\ell \quad n/c \quad 1 \quad j$
  for $k = 1$ to c **do**
    $\mathsf{ct}^0 \quad ^{(n\ \ cj \ \ k+2kn \ \ cn)}(\mathsf{ct}:\mathsf{B}_j)$
    $\mathsf{ct}_{\ell;0} \quad \mathsf{Add}(\mathsf{ct}_{\ell;0}; \mathsf{SMult}(\mathsf{ct}^0;\ _{j;k;0}))$
    $\mathsf{ct}_{\ell;1} \quad \mathsf{Add}(\mathsf{ct}_{\ell;0}; \mathsf{SMult}(\mathsf{ct}^0;\ _{j;k;1}))$
for $\ell = 0$ to $n/c \quad 1$ **do**
  $\mathsf{ct}:\mathsf{B}^{\ell} \quad \mathsf{Add}(\mathsf{ct}_{\ell;0};\ ^{n}(\mathsf{ct}_{\ell;1}))$
return $\mathsf{ct}:\mathsf{B}^{\ell}$

Algorithm 5 MERGE-COPY

Input: Ciphertexts $\mathsf{ct}:\mathsf{B}_j$ of the lower-diagonally stacked vectors of $B \in \mathsf{R}^{n\ \ n}$ for $j \in [n/c]$
Output: Ciphertexts $\mathsf{ct}:\mathsf{B}^{\ell}$ of the replicated lower diagonal vectors of B for $\ell \in [n]$

for $j = 0$ to $n/c \quad 1$ **do**
  for $k = 0$ to $c/2 \quad 1$ **do**
    $\mathsf{ct}_{j;k} \quad \mathsf{SMult}(\mathsf{ct}:\mathsf{B}_j;\ _k)$
    for $r = 0$ to $\log(c/2) \quad 1$ **do**
      $\mathsf{ct}_{j;k} \quad \mathsf{ct}_{j;k} + \ ^{2n\ 2^r}(\mathsf{ct}_{j;k})$
    $\mathsf{ct}:\mathsf{B}^0_{cj+2k} \quad \mathsf{SMult}(\mathsf{ct}_{j;k};\ _L)$
    $\mathsf{ct}:\mathsf{B}^0_{cj+2k+1} \quad \mathsf{ct}_{j;k} \quad \mathsf{ct}:\mathsf{B}_{cj+2k}$
  for $k = 0$ to $c \quad 1$ **do**
    $\mathsf{ct}:\mathsf{B}^0_{cj+k} \quad \mathsf{ct}:\mathsf{B}^0_{cj+k} + \ ^{n}(\mathsf{ct}:\mathsf{B}^0_{cj+k})$
return $\mathsf{ct}:\mathsf{B}^{\ell}$

B.2. Merge-and-copy

Algorithm 5 provides the detailed procedure for generating replicated lower diagonal vectors from the stacked lower diagonal vectors of a matrix B. We define two masking vectors $_k;\ _L \in \mathsf{R}^s$ which have entries as follows:

$$_k[i] = \begin{cases} 1 & 2kn \quad i < 2(k+1)n; \\ 0 & \text{otherwise.} \end{cases}$$

$$_L[i] = \begin{cases} 1 & 0 \quad i < n; \\ 0 & \text{otherwise.} \end{cases}$$

This merge-and-copy procedure requires n scalar multiplication and $\frac{n}{2}(\log c + 1)$ rotations in total.