Powerformer: Efficient and High-Accuracy Privacy-Preserving Language Model with Homomorphic Encryption

Dongjin Park[1], Eunsang Lee[2†], Joon-Woo Lee[1†],

[1]Chung-Ang University [2]Sejong University,

Correspondence: Eunsang Lee (eslee3209@sejong.ac.kr) and Joon-Woo Lee (jwlee2815@cau.ac.kr)

Abstract

We propose *Powerformer*, an efficient homomorphic encryption (HE)-based privacy-preserving language model (PPLM) designed to reduce computational overhead while maintaining model performance. Powerformer incorporates three key techniques to optimize encrypted computations: 1) A novel distillation technique that replaces softmax and layer normalization with computationally efficient power and linear functions, ensuring no performance degradation while enabling seamless encrypted computation. 2) A pseudo-sign composite approximation method that accurately approximates GELU and tanh functions with minimal computational overhead. 3) A homomorphic matrix multiplication algorithm specifically optimized for Transformer models, enhancing efficiency in encrypted environments. By integrating these techniques, Powerformer based on the BERT-base model achieves a 45% reduction in computation time compared to the state-of-the-art HE-based PPLM without any loss in accuracy.

1 Introduction

As AI services continue to expand, many companies now offer machine learning as a service (MLaaS). However, there are growing concerns about potential privacy breaches when clients entrust their sensitive data to a server. To address this issue, there has been increasing interest in privacy-preserving language models (PPLMs), which utilize data encryption. In particular, PPLMs leveraging homomorphic encryption (HE) enable the client to send encrypted data to the server, where all processing is conducted without decryption. The server subsequently returns the encrypted output to the client. This approach drastically lowers the client's computational load while minimizing exposure of client data or model information. Consequently, HE-based PPLMs have emerged as a promising solution that preserves data privacy and AI capabilities in MLaaS environments.

Because HE supports only arithmetic operations, performing non-polynomial operations within HE-based PPLMs is challenging. To address this, various techniques (Zhang et al., 2024; Cho et al., 2024) have been proposed to accurately approximate non-polynomial functions using arithmetic operations, but most rely on high-degree polynomials, which significantly increases computation time. Some studies (Zimerman et al., 2024b; Rho et al., 2025) have attempted to fine-tune models by replacing certain non-polynomial functions with arithmetic-friendly alternatives, yet this consistently leads to reduced inference accuracy. Consequently, finding an HE-based transformer implementation that maintains accuracy while improving speed remains a major challenge. Recently, THOR (Moon et al., 2024b), the fastest end-to-end HE-based transformer model, was reported to require 10.43 minutes of inference time for a BERT-base model on a single GPU. For real-world use, research to further shorten the runtime is essential.

In this study, we propose a PPLM model, *Powerformer*, designed to effectively reduce the inference time of HE-based Transformer models. Powerformer integrates: (1) a novel model tuning approach that replaces softmax and layer normalization (LN) with simple arithmetic operations without compromising accuracy, (2) efficient approximation techniques for GELU and tanh, and (3) a homomorphic matrix computation method optimized for Transformer models. As a result, we achieved a 45.0% reduction in inference time compared to the state-of-the-art PPLM model THOR, successfully lowering the BERT-base PPLM inference time to 5.74 minutes under the same environment.

Detailed implementations can be checked from https://github.com/thrudgelmir/Powerformer.

2 Related Work

2.1 Homomorphic Encryption

HE is a cryptographic algorithm designed to perform arbitrary arithmetic operations directly on encrypted data, and we use the RNS-CKKS scheme (Cheon et al., 2017, 2019), one of the most widely used HE schemes for real number computations. The RNS-CKKS scheme encrypts a vector u of length n (referred to as a slot) in $\mathbb{R}^n$ or $\mathbb{C}^n$. For simplicity, we denote its ciphertext as $[u]$. Under this scheme, the following homomorphic operations are defined and satisfy the corresponding properties: $[u] \oplus [v] = [u + v]$, $[u] \odot v = u \odot [v] = [u \odot v]$, $[u] \otimes [v] = [u \odot v]$, $\mathsf{Rot}([u]; r) = [\rho(u; r)]$, $\mathsf{Multi}([u]) = [\mathrm{i} \cdot u]$, $\mathsf{Conj}([u]) = [\bar{u}]$, where $u \odot v$ and $\bar{u}$ represent elementwise multiplication and complex conjugation, respectively, and $\rho(v; r)$ is defined as $(v_r, v_{r+1}, \ldots, v_{n-1}, v_0, \ldots, v_{r-1})$, denoting a left cyclic shift by r positions. Further details can be found in Section A.2.

2.2 Advanced Homomorphic Operations

One of the most widely used HE-based ciphertext-ciphertext matrix multiplication (CCMM) algorithms is the Jiang et al. method (Jiang et al., 2018). This approach leverages the following transformed matrix multiplication equation $A \cdot B = \sum_{\ell=0}^{d-1} (\phi^\ell \circ \sigma(A)) \odot (\psi^\ell \circ \tau(B))$, where $d \times d$ matrix permutations σ, τ, ϕ, ψ are defined as follows ($[n]_d$ denotes n mod d): $\sigma(A)_{i,j} = A_{i,[i+j]_d}, \tau(A)_{i,j} = A_{[i+j]_d,j}, \phi(A)_{i,j} = A_{i,[j+1]_d}, \psi(A)_{i,j} = A_{[i+1]_d,j}$

BOLT (Pang et al., 2024) proposed a matrix multiplication algorithm based on column packing, which demonstrates fast performance in ciphertext-plaintext matrix multiplication (CPMM). Recently, THOR (Moon et al., 2024b) introduced a new type of matrix multiplication algorithm based on diagonal packing, which outperforms BOLT.

Several studies have focused on efficiently approximating nonlinear functions to enable stable and computationally efficient homomorphic operations. Lee et al. (Lee et al., 2022) proposed an approach for efficient ReLU approximation by decomposing the sign function into a composition of multiple low-degree polynomials, ensuring optimal efficiency under HE. For GELU approximation, methods used in BumbleBee (Lu et al., 2025) and PUMA models (Dong et al., 2023) approximate the Gaussian CDF by dividing the function into multiple segments and computing separate polynomial approximations for each region.

2.3 Privacy-Preserving Language Model

Various PPLM models have been proposed to implement transformers in HE environments. The-X (Chen et al., 2022) replaced softmax with a machine learning model, but offloaded the ReLU operation to the client side, preventing it from being considered a fully comprehensive solution. Meanwhile, NEXUS (Zhang et al., 2024) was the first to achieve end-to-end HE inference for transformers, and the recent state-of-the-art HE-based transformer model THOR (Moon et al., 2024b) introduced diagonally packed matrix multiplication to accelerate end-to-end HE inference. Nonetheless, there remains significant room for speed optimization due to the high computational cost of non-polynomial operations like softmax and LN.

Because softmax relies on costly division and exponentiation, various approximation techniques have been explored to replace it. MPC-based PPLM models such as MPCFormer (Li et al., 2023) and SecFormer (Luo et al., 2024) employed the 2Quad function, $\frac{(x+c)^2}{\sum (x+c)^2}$, alongside distillation to reduce computational overhead, but suffered from accuracy degradation. Power-Softmax (Zimerman et al., 2024a) similarly replaced the exponential function in softmax with a power function and applied conventional fine-tuning, but encountered about a 1% drop in accuracy. While these methods successfully remove the exponentiation step, they do not entirely eliminate division, leaving it as a bottleneck in HE-based PPLM models. LN also presents another bottleneck for HE inference. The-X (Chen et al., 2022) proposed an LN distillation technique, replacing LN with a linear layer, but still observed a 1.71% accuracy drop even in a small model like BERT-Tiny. Therefore, further research is essential to replace softmax and LN with HE-friendly operations without sacrificing accuracy.

2.4 Distillation of Bert Model

Creating lightweight models through knowledge distillation is widely utilized in machine learning, and the situation is no different when constructing HE-friendly models. Studies such as MPCFormer and SecFormer are representative examples, both employing the TinyBERT distillation approach (Jiao et al., 2020). TinyBERT demonstrated a distillation method that effectively captures the characteristics of the BERT model. In this approach, a pre-trained BERT model serves as the teacher, and loss functions are applied at four key positions: (1) the embedding layer, (2) the attention matrix in each Transformer layer, (3) the hidden states after each Transformer layer, and (4) the final prediction layer.

The training process is structured in two stages: In the first stage, mean squared error (MSE) from (1),(2) and (3) are minimized to mimic the intermediate output of the teacher model. In the second stage, the loss from (4) is minimized to achieve higher performance. Soft cross-entropy with the teacher model's output is used as the loss function for classification tasks, while MSE with label is applied for regression tasks.

3 Batch Method

We propose Batch Method, a normalization framework that enables fixed normalization behavior by utilizing batch-level statistics during training. Rather than dynamically computing normalization terms for each input, Batch Method extracts representative values across the batch, specifically, maximum values, and reuses them as constants during inference.

The key rationale behind using maximum values is that they provide a consistently large normalization anchor, which helps suppress activation explosion in deep networks. Importantly, the model is trained to mimic the normalization effect of the teacher model via distillation, ensuring that the outputs align without requiring expensive division operations.

3.1 Batch Power-Max Function

We propose Batch Power-Max (BPMax) as a replacement for softmax, where the exponential function is substituted with a power-based form $(x+c)^p$, as similarly adopted in MPCFormer, SecFormer ($(x+c)^2$) and PowerSoftmax (x^p).

BPMax modifies the denominator to be independent of each input by applying Batch Method:

$$\text{BPMax}(x) = \frac{(x+c)^p}{\max_i \sum_l (x_{i,j,k,l}+c)^p} \rightarrow \frac{(x+c)^p}{R_d}$$

During training, the denominator is computed per batch; during inference, it is replaced with a fixed constant R_d computed in advance.

3.2 Batch Layer Normalization

We propose Batch Layer Normalization (Batch LN) as a replacement for conventional LN. As with BPMax, Batch Method is applied:

$$\text{Batch LN}(x) = \gamma \cdot \frac{x-\mu}{\max_i \sqrt{\frac{1}{d_m}\sum_k (x_{i,j,k}-\mu)^2}} + \beta$$

$$\rightarrow \gamma \cdot \frac{x-\mu}{l \cdot R_d} + \beta$$

We additionally include a parameter l in the denominator. Its effect and rationale are discussed in detail in the next section.

As with BPMax, the network is retrained to internalize the normalization behavior guided by these fixed constants.

3.3 Training of Batch Method

Training with fixed normalization constants can destabilize intermediate representations. In particular, we observed cases where the model achieved high prediction accuracy, yet the overall loss became unstable or even diverged. This occurs because a small number of activations do not normalize properly, resulting in excessively large values that dominate the loss calculation. We refer to this phenomenon as the feature explosion under fixed normalization.

To address this issue, we propose a unified training strategy that combines one-step distillation with an additional loss term, while also introducing an extra scaling parameter.

The extra scaling parameter l serves as a conservative scaling factor in the denominator, slightly expanding the normalization scale to suppress the influence of outliers during residual accumulation, thus promoting stable convergence under fixed normalization.

In contrast to TinyBERT's original two-step distillation, one-step distillation integrates all losses into a single unified loss function, encouraging the network to maintain consistent normalization effects across all layers and closely mimic the original model's normalization.

Furthermore, an additional loss term is applied in Batch LN, immediately after the normalization stage, before scaling and biasing, to directly constrain raw activation magnitudes. This helps stabilize the training process by preventing extreme activation values from overwhelming or destabilizing the training signal.

This training approach is essential for maintaining numerical stability in Batch Method, allowing fixed normalization constants to remain compatible with both accurate final outputs and well-behaved intermediate activations.

4 Minimax Composition for Pseudo-Sign Function

Lee et al. (Lee et al., 2022) effectively approximated the ReLU function for HE operations using the minimax composition method, which enables efficient computation of the sign function. However, in Transformers, the GELU function and $\tanh$ function are used more frequently than the ReLU function. To compute the GELU function, it is necessary to accurately approximate the cumulative distribution function $\Phi(x)$ of the Gaussian distribution. Functions such as $\Phi(x)$ and $\tanh(x)$, which are commonly used in Transformers, exhibit behavior similar to the sign function for inputs with large absolute values. However, for inputs close to zero, these functions exhibit unique characteristics, which often determine the performance of each activation function.

4.1 Pseudo-Sign Function and Minimax Composite Polynomial for Sign Function

We devise a method to extend the minimax composition technique to approximate $\Phi(x)$. Our observation is that when the sign function is approximated using the minimax composition method over an approximation interval with a sufficiently large ϵ, the resulting approximation is monotonic and its function values remain within the range $[-1, 1]$ over the interval $[-\epsilon, \epsilon]$. If we can compose this approximation with a simple function to achieve the desired shape of $\Phi(x)$ within the interval $[-\epsilon, \epsilon]$, we can compute $\Phi(x)$ with almost the same computational complexity as the original minimax composition method. Furthermore, since $\Phi(x)$ approaches ± 1 relatively gradually, it is acceptable to approximate the sign function for a relatively large ϵ, making this approach both practical and efficient.

Definition 4.1. *A function f satisfying the following conditions is defined as a* pseudo-sign function*: f is a monotonically increasing function, satisfying $f(-x) = -f(x)$ for all x and $\lim_{x\to\infty} f(x) = 1$.*

Example 4.1. *The error function* $\operatorname{erf}(x) = \frac{2}{\pi}\int_0^x e^{-t^2}\,dt$ *and the hyperbolic tangent* $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ *is pseudo-sign functions.*

We first observe the approximated minimax composite polynomial for sign function to approximate the pseudo-sign function $f(x)$, especially the "don't care" part of the approximated composite polynomial. Any (ϵ, δ)-approximate minimax composite polynomial for the sign function, denoted as $p_{\text{com}}(x) = (p_{t-1} \circ \cdots \circ p_0)(x)$, can be shown to increase monotonically within the interval $[-\epsilon, \epsilon]$ and maps to the range $[-1 + \delta, 1 - \delta]$. Therefore, the minimax composite polynomial for the sign function can itself be regarded as a pseudo-sign function. It is important to note that the shape of the curve within $[-\epsilon, \epsilon]$ varies depending on the specific pseudo-sign function.

Specifically, $p_{\text{com}}(x)$ increases monotonically near the origin over the interval $[-\epsilon', \epsilon']$ ($\epsilon < \epsilon'$), where its range of $p_{\text{com}}(x)$ within $[-\epsilon', \epsilon']$ is contained in $[-1 - \delta, 1 + \delta]$, satisfying $p_{\text{com}}(-\epsilon') = -1 - \delta$ and $p_{\text{com}}(\epsilon') = 1 + \delta$. Now, define a scaled function $h(x) = \frac{1}{1+\delta} p_{\text{com}}(x)$ over $[-\epsilon', \epsilon']$. Since $h(x)$ is monotonically increasing, it has an inverse. By defining $g(x) = f \circ h^{-1}(x)$ on the interval $[-1, 1]$, $g(x)$ is also monotonically increasing and maps to the range $[-1, 1]$. This function $g(x)$ can be closely approximated by a low-degree minimax polynomial, denoted as $p_g(x)$. Using $p_g(x)$, the composite polynomial $p_g \circ (\frac{1}{1+\delta} \cdot p_{\text{com}}) = p_g \circ (\frac{1}{1+\delta} \cdot p_{t-1}) \circ p_{t-2} \circ \cdots \circ p_0$ provides a high-accuracy approximation of the pseudo-sign function $f(x)$.

Algorithm 1: $\mathsf{MiniCompPseudoSign}(f, \delta)$

```
Input: A pseudo-sign function $f$ with domain
    $[-1, 1]$, minimax approximation bound $\delta$
Output: Polynomials $p_0, \cdots, p_{n-1}$ such that
    $\|p_{n-1} \circ \cdots \circ p_0 - f\|_{\infty,[-1,1]} \le \delta$
Identify $\gamma \in (0, 1)$ such that $f(\gamma) = 1 - \delta/2$.
Compute $p_0, \cdots, p_{t-2}, \bar{p}_{t-1}$ using
  $\{p_0, \cdots, p_{t-2}, \bar{p}_{t-1}\} \leftarrow$
  $\mathsf{MiniCompSign}\left(\gamma, \frac{\delta/4}{1-\delta/4}\right)$,
Update $p_{t-1}$ by scaling: $p_{t-1} \leftarrow \frac{1}{1+\delta'} \cdot \tilde{p}_{t-1}$, where
  $\delta'$ is the minimax error of $p_{t-1} \circ \cdots \circ p_0$ in the
  domain $[-1, -\gamma] \cup [\gamma, 1]$.
Define $\gamma'$ as the smallest positive $x$ such that
  $p_{\text{scale}}(x) := p_{t-1} \circ \cdots \circ p_0(x) = 1$. Restrict the
  domain of $p_{\text{scale}}(x)$ to $[-\gamma', \gamma']$, and denote the
  resulting function as $\tilde{p}_{\text{scale}}(x)$.
Define $g(x) = f \circ \tilde{p}_{\text{scale}}^{-1}(x)$ on $[-1, 1]$. Use Remez
  algorithm to approximate $g$ with a minimax
  polynomial $p_g \leftarrow \mathsf{Remez}(g, \delta/2)$.
Let $n = t + 1$, and define $p_{n-1} := p_g$. Output the
  polynomial sequence $\{p_0, \cdots, p_{n-1}\}$.
```

We can verify the correctness of the $\mathsf{MiniCompPseudoSign}$ algorithm in Algorithm 1 through the following theorem, which is specified and proven in Appendix B. Figure 1 shows the schematic representation of the $\mathsf{MiniCompPseudoSign}$ algorithm.

Theorem 4.1. *Let f be a pseudo-sign function and $0 < \delta < 1$. Assume that there exists*

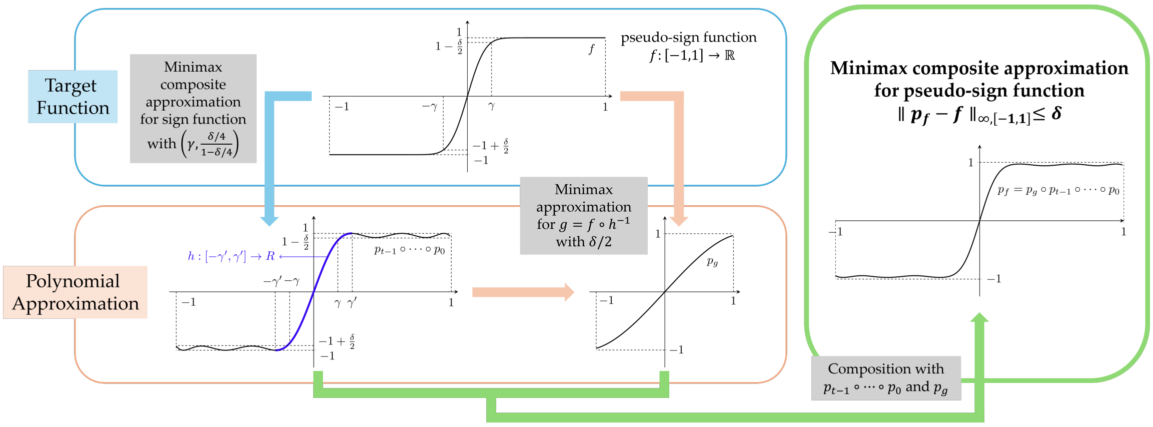
Target Function
Minimax composite approximation for sign function with $\left(\gamma, \frac{\delta/4}{1-\delta/4}\right)$
pseudo-sign function $f: [-1,1] \to \mathbb{R}$
$1 - \frac{\delta}{2}$
1
f
-1
$-\gamma$
γ
1
$-1 + \frac{\delta}{2}$
-1
Polynomial Approximation
$h : [-\gamma', \gamma'] \to R$
$1 - \frac{\delta}{2}$
$p_{t-1} \circ \cdots \circ p_0$
$-\gamma' -\gamma$
γ γ'
$-1 + \frac{\delta}{2}$
Minimax approximation for $g = f \circ h^{-1}$ with $\delta/2$
p_g
Minimax composite approximation for pseudo-sign function
$\| p_f - f \|_{\infty,[-1,1]} \leq \delta$
$p_f = p_g \circ p_{t-1} \circ \cdots \circ p_0$
Composition with $p_{t-1} \circ \cdots \circ p_0$ and p_g

Figure 1: Schematic representation of the pseudo-sign composite approximation.

$0 < \gamma < 1$ *such that* $f(\gamma) = 1 - \delta/4$. *Then, the polynomials* $p_0, \cdots, p_{n-1}$ *obtained through* MiniCompPseudoSign(f, δ) *satisfy the following condition:* $\|p_{n-1} \circ \cdots \circ p_0 - f\|_{\infty,[-1,1]} \leq \delta$.

4.2 Efficiency of Pseudo-Sign Composite Approximation

The Bumblebee (Lu et al., 2025) and PUMA (Dong et al., 2023) methods for approximating the GELU function were adopted by NEXUS (Zhang et al., 2024) for use in HE-based PPLMs. Their approach segments the function into four regions, requiring three separate sign function evaluations to compute the indicator functions. Each sign function is approximated using composite polynomials with a maximum error of $\delta/2$ and a don't-care region half-width of $\delta/2$. Afterward, each segmented function is approximated with a maximum error of $\delta/2$.

In contrast, our approach requires only a single sign function approximation with a maximum error of $\delta/4$ and a significantly wider don't-care region half-width of γ. The function $g(x)$ is then approximated with a maximum error of $\delta/2$, leading to a more efficient composition. This reduces the number of sign function evaluations from three to one while significantly expanding the don't-care region, thereby lowering the polynomial degree required for the overall approximation.

For instance, when $\delta = 2^{-10}$, the NEXUS model requires computing three composite polynomials of degrees $(3, 5, 5, 5, 5, 5, 5)$ and two third-degree polynomials. However, our method achieves the same accuracy using only one composite polynomial of degrees $(5, 5, 5, 7)$. Similarly, when $\delta = 2^{-13}$, NEXUS computes three composite polynomials of degrees $(3, 3, 5, 5, 5, 5, 5, 5, 5)$ and two third-degree polynomials, whereas our approach only requires a single composite polynomial of degrees $(3, 3, 5, 5, 13)$. As a result, our method reduces computational cost by approximately 4.7× and depth by approximately 1.8× compared to NEXUS (Zhang et al., 2024).

5 Optimized Homomorphic Matrix Operation

5.1 Optimized Ciphertext-Plaintext Matrix Multiplication

We propose a fast CPMM algorithm based on column packing. Let $\{m_i \in \mathbb{R}^n\}_{1 \leq i \leq \frac{d_1 d_2}{n}}$ be the plaintext vectors storing $A \in \mathbb{R}^{d_1 \times d_2}$ via column packing (Section A.3), which we denote by $[A]_C$. For matrices $A \in \mathbb{R}^{d_1 \times d_2}$ and $B \in \mathbb{R}^{d_2 \times d_3}$, our algorithm computes the ciphertexts of $[AB]_C$ from the ciphertexts of $[A]_C$. We pack the columns of the input matrix into a total of $\mathsf{mid} = \frac{d_1 d_2}{n}$ ciphertexts, $\{ct_i\}_{1 \leq i \leq \mathsf{mid}}$, and ensure that the columns of the output matrix are packed into $\mathsf{ed} = \frac{d_1 d_3}{n}$ ciphertexts, $\{ct'_i\}_{1 \leq i \leq \mathsf{ed}}$. By noting that each column of the output matrix can be expressed as a linear combination of the columns of the input matrix, we derive the following equation.

$$ct'_\ell = \sum_{1 \leq j \leq \mathsf{mid}} \sum_{0 \leq i < \frac{n}{d_1}} \mathsf{Rot}(ct_j; id_1) \odot d_i^{j,\ell} e \quad (1)$$

for $1 \leq \ell \leq \mathsf{ed}$, where $d_i^{j,\ell} \in \mathbb{R}^n$ stores the elements of matrix B appropriately. By applying the baby-step giant-step technique to Equation 1, we can transform it into the following form for some N_1, N_2 satisfying $N_1 N_2 = \frac{n}{d_1}$.

$$ct'_\ell = \sum_{0 \leq p < N_2} \mathsf{Rot}\Big(\sum_{1 \leq j \leq mid} \sum_{0 \leq q < N_1} \mathsf{Rot}(ct_j; qd_1) \odot \rho(d^{j,\ell}_{pN_1+q}; -pN_1 d_1); pN_1 d_1\Big) \quad (2)$$

for $1 \le \ell \le \mathsf{ed}$. When N_1 and N_2 satisfy $\mathsf{mid} \cdot N_1 = \mathsf{ed} \cdot N_2$, the algorithm needs about $2\sqrt{\frac{n}{d_1} \cdot \mathsf{mid} \cdot \mathsf{ed}}$ rotations.

Speedup via Complex Numbers By leveraging complex numbers, we propose a method that further reduces rotations. For any two ciphertexts $ct'_{\ell_1}, ct'_{\ell_2}$, we can utilize complex numbers to compute the equation all at once as follows: $ct' = \sum_{1 \le j \le \mathsf{mid}} \sum_{0 \le i < \frac{n}{d_1}} \mathsf{Rot}(ct_j; id_1) \odot (d_i^{j,\ell_1} + \mathrm{i} d_i^{j,\ell_2})$. By using the Extract algorithm (Section C.5), which seperates the real and imaginary parts, we obtain $ct'_{\ell_1}, ct'_{\ell_2} = \mathsf{Extract}(ct')$. By grouping the ed output ciphertexts in pairs and combining their expressions, ed is updated as $\mathsf{ed} \leftarrow \lceil \mathsf{ed}/2 \rceil$, which leads to a reduction in the number of rotations.

Meanwhile, it is also possible to combine two input ciphertexts using complex numbers. For matrices $A, B \in \mathbb{R}^{d_1 \times d_2/2}, C, D \in \mathbb{R}^{d_2/2 \times d_3}$, the product of $[A|B]$ and $\begin{bmatrix} C \\ D \end{bmatrix}$ is $AC + BD$. This computation can instead be performed by extracting the real part of $(A + B\mathrm{i})(C - D\mathrm{i})$. In this case, the value of mid is updated as $\mathsf{mid} \leftarrow \lceil \mathsf{mid}/2 \rceil$, and accordingly, the number of rotations also decreases.

By applying the proposed techniques, the number of rotations used by CPMM is reduced to $\sqrt{5/9}$, $\sqrt{2/3}$, $\sqrt{1/2}$, and $\sqrt{1/2}$ times the original values, respectively, for each of the following: (1) computing the query, key, and value matrices, (2) multiplying the output projection matrix, (3) the first feed-forward network, and (4) the second feed-forward network.

5.2 Optimized Ciphertext-Ciphertext Matrix Multiplication

5.2.1 Square Matrix Multiplication

Jiang et al. (Jiang et al., 2018) proposed a CCMM algorithm using row packing. By simply swapping its two inputs, we can obatin a column packing version (see Section A.3 for details).

We also propose a technique to optimize this column packing-based CCMM algorithm. Let the input matrices A and B both be of size $d \times d$, and let $n = d^2$. Suppose the constant vectors $R_i, L_i \in \mathbb{R}^n$ for $0 \le i < d$ are defined as follows:

$$R_i[j] = \begin{cases} 1\colon [j]_4 \ge i \\ 0\colon else \end{cases} \quad L_i[j] = \begin{cases} 1\colon [j]_4 < i \\ 0\colon else. \end{cases}$$

for $0 \le j < n$. Let $A' = \sigma(A)$ and $B' = \tau(B)$. Then, the following equation holds:

$$[\sum_{0 \le \ell < d} \phi_\ell(A') \odot \psi_\ell(B')]_C = \sum_{0 \le j < N_2} \rho(\sum_{0 \le i < N_1} \rho([A']_C; id) \odot \rho([B']_C \odot R_{N_1 j + i} + \rho([B']_C; -d) \odot L_{N_1 j + i}; N_1 j + i - N_1 j d); N_1 j d).$$

for N_1, N_2 satisfying $N_1 N_2 = d$, which corresponds to an efficient CCMM algorithm. When $N_1 = N_2 = \sqrt{d}$, a total of $d + 2\sqrt{2d} + 5\sqrt{d}$ key-switches are required, which is much fewer than the $4d + 2\sqrt{2d} + 2\sqrt{d}$ in Jiang et al.'s algorithm.

5.2.2 Multi-Head Attention

In this section, we first discuss block-wise matrix operations. For a k satisfying $k \mid d_1$ and $k \mid d_2$, let us partition the $d_1 \times d_2$ matrix A into $k \times k$ blocks $A^{i,j}$. Suppose that the matrix operations σ, τ, ϕ, ψ are defined for $k \times k$ matrices. We define the operations $\tilde{\sigma}, \tilde{\tau}, \tilde{\phi}, \tilde{\psi}$, which apply σ, τ, ϕ, ψ blockwise to each block $A^{i,j}$ of the entire matrix A. Then, we found that by using the new packing method, the *modified column packing* (defined in Section C.1), we can obtain homomorphic algorithms for $\tilde{\sigma}, \tilde{\tau}, \tilde{\phi}, \tilde{\psi}$ on $n = d_1 d_2$, each of which has the same computational complexity as the corresponding homomorphic algorithm for σ, τ, ϕ, ψ on $n = k^2$. The CCMM algorithm discussed in Section 5.2.1 can also be naturally extended to a blockwise (for k) CCMM algorithm for a $d_1 \times d_2$ matrix without any additional overhead (see Section C.4 for details).

By appropriately utilizing blockwise operations for $k = 64$, we can implement all the CCMM operations required for the entire multi-head attention. In addition, we propose a method to reduce computation by making use of complex number components. For example, if we need to compute $\tilde{\sigma}(A)$ and $\tilde{\sigma}(B)$, we can instead compute $\tilde{\sigma}(A + B\mathrm{i})$ and then separate the real and imaginary parts, thereby reducing the number of calls to $\tilde{\sigma}$. The same idea applies to $\tilde{\tau}$ and the blockwise transpose algorithm. Moreover, if we need to compute the products AB and AC, we can reduce the number of multiplication algorithms by computing $A(B + C\mathrm{i})$ instead. Additionally, if we need to compute $AB + CD$, we can compute $(A + C\mathrm{i})(B - D\mathrm{i})$ and extract the real part. By combining these ideas, the final optimized algorithm is presented in Section C.5.

5.3 Microbenchmarks

Table 1 presents the microbenchmark results for the homomorphic matrix multiplication algorithms.

One effective metric for estimating computational complexity is the number of key-switches(KS), which refers to the total count of non-scalar multiplications (relinearizations), rotations, conjugations, and other operations requiring a KS. In this table, the number of KS is based on the BERT-base model that we target. Our algorithm requires $32\% \sim 36\%$ fewer KS for CPMM and 22% fewer KS for CCMM compared to THOR, a state-of-the-art technique. A detailed ablation study of our algorithm can be found in Section D.

Operation	Method	#key-switch
$\times W_Q, W_K, W_V$	NEXUS	3538944
	BOLT	288
	THOR	180
	Powerformer	**122**
QK^T $\& \times V$	NEXUS	-
	BOLT	33684
	THOR	936
	Powerformer	**731**
$\times W^O$	NEXUS	14155776
	BOLT	168
	THOR	118
	Powerformer	**75**

Table 1: Comparison of KS counts in different homomorphic matrix multiplication methods.

6 Experiments

Model and Dataset In this study, we utilized a BERT-base model with a sequence length of $L = 128$ and conducted experiments on the RTE, MRPC, and SST-2 tasks from the GLUE benchmark (Wang, 2018).

Hyperparameter In standard training, early stopping was applied at 10 epochs, while in knowledge distillation training, early stopping was set at 20 epochs. This was determined based on the point at which no further performance improvement was observed. Other hyperparameters were fixed, with a batch size of 64 and a learning rate of 5×10^{-5}, to ensure a consistent and fair comparison of relative model performance.

Seeds All experiments were conducted three times each, using seeds 0, 42, and 777.

HE Environment Powerformer is built on the GPU version of the Liberate.FHE library with a slot size of 2^{15}. Our HE setting ensures a 128-bit security level, and 11 multiplicative levels are available before bootstrapping(BTS). All experiments were conducted on a single NVIDIA A100 GPU.

6.1 Plaintext Evaluation

As our training method is distillation-based, each baseline in the table corresponds to the accuracy of a fine-tuned BERT-base model used as the teacher.

6.1.1 Analysis of BPmax Parameters (p, c)

Table 2 presents the average accuracy across the three tasks under various parameters p and c. The results show that both p and c positively influence the polynomial replacement of softmax. We propose the combination $p = 5, c = 5$, which is the only combination that exceeds the baseline (82.86%).

p \ c	1	3	5	7
1	69.79	72.77	73.64	73.85
3	66.78	81.54	80.97	79.50
5	56.21	82.62	**83.11**	82.44
7	51.28	82.06	82.07	82.51

Table 2: Average accuracy for different (p, c) combinations (%).

6.1.2 Analysis of Batch LN Parameter (l)

Table 3 presents the accuracy and feature explosion rates for l values ranging from 1.0 to 1.3 in increments of 0.1. The results indicate that increasing l slightly reduces overall accuracy while significantly mitigating the risk of feature explosion. This suggests that although the l parameter is effective in preventing feature explosion, careful tuning is required to avoid performance degradation. We therefore recommend fine-tuning l rather than relying on a fixed value.

Task	Baseline	1.0	1.1	1.2	1.3
RTE	69.43	70.52	69.92	68.95	70.28
MRPC	87.01	86.76	86.85	86.19	86.11
SST-2	92.13	92.09	91.93	91.86	91.82
Average	82.86	83.12	82.90	82.33	82.74
f.e rate	–	44.44	22.22	0.00	0.00

Table 3: Effect of parameter l (%). "f.e." denotes feature explosion.

6.1.3 Ablation Study of the Proposed Training Strategy

Table 4 presents the accuracy and feature explosion rates from the ablation study of our proposed training method. In the table, the original distillation approach is denoted as "2 step", while one-step distillation is represented as "1 step Loss X" and "1 step Loss O", depending on whether the additional loss function is applied. The final configuration proposed in this paper corresponds to "1 step Loss O".

The summarized results show that one-step distillation improves both accuracy and stability compared to the "2 step" baseline. In particular, the version with the additional loss function ("1 step Loss O") achieves higher accuracy and further suppresses feature explosion than the version without it.

This training strategy not only reduces computational overhead but also ensures stable and accurate learning. Combined with the previously introduced l parameter, it plays a key role in effectively mitigating feature explosion.

Task	Baseline	1-step O	1-step X	2-step
RTE	69.43	70.52	68.11	69.49
MRPC	87.01	86.76	86.19	83.42
SST-2	92.13	92.05	91.93	87.08
Average	82.86	**83.11**	82.08	80.00
f.e rate	–	0.00	33.33	66.67

Table 4: Comparison of training methods (%).

6.1.4 Ablation Study of the Batch Method

Table 5 presents the ablation results for the BPMax function and Batch LN. We compare three configurations against the baseline: BPMax alone ("BPMax"), Batch LN alone ("Batch LN"), and both applied together ("Both"). The results show that each method individually, as well as their combination, improves accuracy over the baseline. These findings suggest that BPMax and Batch LN are effective as independent, general-purpose components, and highlight their potential as HE-friendly solutions to the conventional scaling problem.

Task	Baseline	BPMax	Batch LN	Both
RTE	69.43	69.07	71.48	70.52
MRPC	87.01	87.83	87.91	86.76
SST-2	92.13	92.13	92.05	91.97
Average	82.86	**83.01**	**83.81**	**83.08**

Table 5: Ablation study of Batch Methods (%).

6.1.5 Evaluation on Longer Contexts

We conducted additional experiments on longer sequence using the BoolQ task from the SuperGLUE (Sarlin et al., 2020) benchmark, with a sequence length of 256 and a batch size of 32. Table 6 presents the results. Unlike previous tasks with a sequence length of 128, where no accuracy drop was observed, a slight decrease of 0.77% was noted.

We attribute this performance degradation to the l parameter used in normalization. Figure 2 illustrates the occurrence of feature explosion across different values of l. In contrast to previous tasks where feature explosion disappeared at $l = 1.2$, in the long-sequence setting, stability was only achieved from $l = 1.9$ onward. This indicates that normalization becomes more unstable as the sequence length increases, but the issue can be resolved by simply increasing the parameter l.

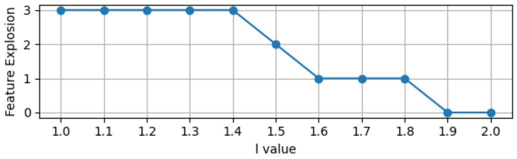

Figure 2: Feature explosion count for different l values.

Method	0	42	777	Average
Baseline	74.98	76.70	77.34	76.34
Ours	74.98	75.47	76.24	75.57

Table 6: Accuracy across random seeds (%).

6.2 Ciphertext Evaluation

BTS is the most computationally expensive operation in HE, and is triggered once a certain multiplicative depth is consumed (THOR: 13, Ours: 11). Although KS is less costly than BTS, it remains the dominant source of latency during ciphertext multiplications and rotations.

Therefore, we compare the number of KS and depth consumption (which determines the need for BTS) to estimate the relative computational cost of each method.

6.2.1 Efficiency of the Batch Method

Table 7 presents the number of KS and depth required for the softmax approximation method of THOR and our BPMax. For softmax, our method requires only about 7.5% of the KS and 10% of the depth used by THOR.

Method	#KS	Depth
BPmax	18	3
THOR	240	30

Table 7: Comparison of KS and depth of softmax.

Table 8 shows the corresponding results for LN, comparing the approximation method from THOR and our Batch LN. Although the THOR paper does not provide explicit iteration details - only referencing the use of (Moon et al., 2024a) to compute the inverse square root - our method achieves the same objective using only 12.5% to 25% of key switching operations and 6. 25% to 12. 5% of depth, depending on the assumed number of iterations.

Method	#KS	Depth
Batch LN	8	1

	THOR							
Iteration	7	8	9	11	12	13	14	15
Depth	8	10	10	12	12	14	14	16
#KS	40	48	48	56	56	64	64	72

Table 8: Comparison of KS and depth of LN.

In conclusion, our Batch Method performs both softmax and LN approximations with a combined computational cost of under 10% compared to THOR.

6.2.2 End-to-end Runtime Analysis

Table 9 presents the results of a single RTE task conducted under HE. "Plaintext" refers to the model accuracy in the plaintext environment, while "Ciphertext" represents the performance of the Powerformer model evaluated under encryption.

The Powerformer model is an end-to-end HE-compatible architecture that integrates all techniques proposed in this paper. We report results using the model that achieved the highest accuracy on the RTE task. The experiment confirms that there is no performance gap between the plaintext and ciphertext settings. As an additional measure, "Max Diff" denotes the maximum difference observed between output values in both environments, with an extremely small deviation of 0.019.

These findings demonstrate that the Powerformer model can robustly approximate various nonlinear functions, highlighting its potential not only for classification tasks but also for regression problems under HE.

	Plaintext	Ciphertext	Max Diff	Time (s)
Value	73.29	73.29	0.019	344.52

Table 9: End-to-end HE inference result. for the RTE task using an encrypted model.

Table 10 presents the layer-wise performance breakdown for the HE experiment in Table 9. Since matrix operation time scales nearly linearly with ciphertext level, this table may not fully capture performance gains. As shown in Table 1, our matrix operations have a lower computational complexity but may exhibit longer runtimes due to their execution at higher levels—a consequence of our level selection strategy designed to minimize BTS. In contrast, non-linear function replacements, including BPMax, Batch LN, and minimax-based GELU/tanh, significantly reduce both per-layer runtime and BTS overhead. Overall, the model achieves a 70% reduction in BTS time—previously the dominant cost in THOR—and a 45% reduction in total computation time.

Operation	Ours	THOR	Diff
Attention layer	57.65	49.77	-7.88
Attention score	28.76	32.53	3.77
Softmax	0.75	15.53	14.78
Attention heads	18.95	20.63	1.68
Multi-head attention	22.54	27.43	4.89
LayerNorm1	0.37	7.13	6.76
FC1	59.21	49.80	-9.41
GELU	8.31	29.42	21.11
FC2	43.77	49.19	5.42
LayerNorm2	0.30	4.10	3.80
Pooler & Classification	0.20	2.70	2.50
Bootstrappings	103.72	337.86	234.14
Total	344.52	626.09	281.57
Total w/o Pooler & Class.	344.32	623.39	279.07

Table 10: Breakdown of execution time (in seconds) compared to THOR.

7 Conclusion

We proposed *Powerformer*, an efficient HE-based PPLM designed to reduce computational overhead while maintaining model performance. To minimize computational overhead while preserving model accuracy, our work introduced a novel distillation framework for softmax and LN, an optimized approximation method for GELU and tanh, and a highly efficient matrix multiplication algorithm tailored for transformer models. By incorporating these methods, it significantly reduced the computation time compared to the leading HE-based PPLM while maintaining the same level of accuracy.

Limitations

This model assumes a semi-honest security model, which means that both the client and the server follow the protocol agreed upon. This assumption is standard for all HE-based PPLM models, as homomorphic encryption itself is designed within the semi-honest framework. If the possibility of a malicious client or server deviating from the protocol were considered, instead an MPC-based PPLM model would be required, which would lead to an extreme increase in computational resource requirements. However, even under the semi-honest assumption, HE-based PPLM models can still adequately ensure data privacy in cloud AI systems. In particular, even if the server does not fully adhere to the protocol, it cannot extract any meaningful information from the client's data due to the inherent security properties of HE. Given that there is no strong incentive for the server to act maliciously in a practical setting, assuming a semi-honest security model remains a realistic and reasonable approach.

Acknowledgements

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education(No. NRF-2022R1I1A1A01068284) and the Institute of Information & Communi cations Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.RS-2024-00398360, Development of a user-friendly and efficiency-optimized real-time homomorphic statistical analysis processing platform & No.RS-2024-00399401, Development of post-quantum security infrastructure transition and comprehensive quantum security verification technology)

References

Tianyu Chen, Hangbo Bao, Shaohan Huang, Li Dong, Binxing Jiao, Daxin Jiang, Haoyi Zhou, Jianxin Li, and Furu Wei. 2022. THE-X: Privacy-preserving transformer inference with homomorphic encryption. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 3510–3520.

Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. 2019. A full RNS variant of approximate homomorphic encryption. In *Selected Areas in Cryptography–SAC 2018, Calgary, AB, Canada, August 15–17*, pages 347–368. Springer.

Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2017. Homomorphic encryption for arithmetic of approximate numbers. In *ASIACRYPT 2017, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*, pages 409–437. Springer.

Wonhee Cho, Guillaume Hanrot, Taeseong Kim, Minje Park, and Damien Stehlé. 2024. Fast and accurate homomorphic softmax evaluation. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS 2024)*, pages 4391–4404.

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

Xiaoqian Jiang, Miran Kim, Kristin Lauter, and Yongsoo Song. 2018. Secure outsourced matrix computation and application to neural networks. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security (CCS 2018)*, pages 1209–1222.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2020. TinyBERT: Distilling BERT for natural language understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4163–4174.

Eunsang Lee, Joon-Woo Lee, Jong-Seon No, and Young-Sik Kim. 2022. Minimax approximation of sign function by composite polynomial for homomorphic comparison. *IEEE Transactions on Dependable and Secure Computing*, 19(6):3711–3727.

Joon-Woo Lee, Eunsang Lee, Yongwoo Lee, Young-Sik Kim, and Jong-Seon No. 2021. High-precision bootstrapping of rns-ckks homomorphic encryption using optimal minimax polynomial approximation and inverse sine function. In *EUROCRYPT 2021*, pages 618–647. Springer.

Dacheng Li, Hongyi Wang, Rulin Shao, Han Guo, Eric Xing, and Hao Zhang. 2023. MPCFormer: Fast, performant and private transformer inference with mpc. In *International Conference on Learning Representations (ICLR 2023)*.

Wen-jie Lu, Zhicong Huang, Zhen Gu, Jingyu Li, Jian Liu, Cheng Hong, Kui Ren, Tao Wei, and WenGuang Chen. 2025. Bumblebee: Secure two-party inference framework for large transformers. *Network and Distributed System Security Symposium (NDSS 2025)*.

Jinglong Luo, Yehong Zhang, Zhuo Zhang, Jiaqi Zhang, Xin Mu, Hui Wang, Yue Yu, and Zenglin Xu. 2024. Secformer: Fast and accurate privacy-preserving inference for transformer models via smpc. In *Findings of the Association for Computational Linguistics (ACL 2024)*, pages 13333–13348.

Jungho Moon, Zhanibek Omarov, Donghoon Yoo, Yongdae An, and Heewon Chung. 2024a. Adaptive successive over-relaxation method for a faster iterative approximation of homomorphic operations. *Cryptology ePrint Archive, 2024/1366.*

Jungho Moon, Dongwoo Yoo, Xiaoqian Jiang, and Miran Kim. 2024b. THOR: Secure transformer inference with homomorphic encryption. *Cryptology ePrint Archive, 2024/1881.*

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

Donghwan Rho, Taeseong Kim, Minje Park, Jung Woo Kim, Hyunsik Chae, Jung Hee Cheon, and Ernest K Ryu. 2025. Encryption-friendly llm architecture. In *International Conference on Learning Representations (ICLR 2025).*

Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. 2020. SuperGlue: Learning feature matching with graph neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2020).*

Alex Wang. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461.*

Jiawen Zhang, Xinpeng Yang, Lipeng He, Kejia Chen, Wen-jie Lu, Yinghao Wang, Xiaoyang Hou, Jian Liu, Kui Ren, and Xiaohu Yang. 2024. Secure transformer inference made non-interactive. *Network and Distributed System Security Symposium (NDSS 2025).*

Itamar Zimerman, Allon Adir, Ehud Aharoni, Matan Avitan, Moran Baruch, Nir Drucker, Jenny Lerner, Ramy Masalha, Reut Meiri, and Omri Soceanu. 2024a. Power-softmax: Towards secure llm inference over encrypted data. *arXiv preprint arXiv:2410.09457.*

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

A Extended Preliminaries

A.1 Transformer

In this paper, we focus on homomorphically implementing a Transformer-based model, BERT (Bidirectional Encoder Representations from Transformers) using the RNS-CKKS scheme (specifically, BERT-base model). The BERT-base model consists of 12 identical encoder blocks, where each encoder block sequentially performs multi-head attention, LN, feed-forward network, and LN.

First, the input sentence is tokenized, and each token undergoes an embedding process to become a fixed-size vector. After embedding, we obtain the $L \times d_m$ matrix X, which serves as the input to the first encoder block. The multi-head attention mechanism has h heads, and for each head, the query, key, and value matrices are computed by multiplying the input matrix X with the corresponding weight matrices. If the query, key, and value weight matrices for head j ($j = 0, 1, \cdots, h-1$) are denoted as $W_Q^{(j)}$, $W_K^{(j)}$, and $W_V^{(j)} \in \mathbb{R}^{d_m \times d_m/h}$, respectively, the following matrix multiplications need to be performed:

$$Q^{(j)} = XW_Q^{(j)}, K^{(j)} = XW_K^{(j)}, V^{(j)} = XW_V^{(j)}. \tag{3}$$

For each head, the following $L \times L$ matrix is computed:

$$\frac{Q^{(j)}K^{(j)T}}{\sqrt{d/2}} \tag{4}$$

Next, apply softmax and multiply by $V^{(j)}$ to obtain the following $L \times d_m/h$ matrix:

$$Y_j = \mathsf{softmax}\left(\frac{Q^{(j)}K^{(j)T}}{\sqrt{d_m/2}}\right)V^{(j)}. \tag{5}$$

The Y_j matrices for the multiple heads are concatenated horizontally to form the $L \times d_m$ matrix $Y = [Y_0|Y_1|\cdots|Y_{h-1}]$. After that, the weight matrix W^O is multiplied on the right, and according to the skip connection, matrix X is added, resulting in $YW^O + X$, which completes the multi-head attention process.

Next, LN is performed to obtain the matrix Y. In the subsequent feed-forward network, the weight matrix $W_{F1} \in \mathbb{R}^{d_m \times d_h}$ is first multiplied to obtain YW_{F1}, followed by applying GELU and then multiplying by the second weight matrix $W_{F2} \in \mathbb{R}^{d_h \times d_m}$ on the right. After that, LN is performed. The process described so far constitutes one encoder layer, and the BERT model repeats this encoder layer several times with the same structure, though with different weight parameters. In this paper, our homomorphic implementation focuses on the BERT-base model, which has parameters $L = 128, d_m = 768$, $h = 12$, and $d_h = 3072$. Figure 3 shows the architecture of one encoder block in the BERT-base model.

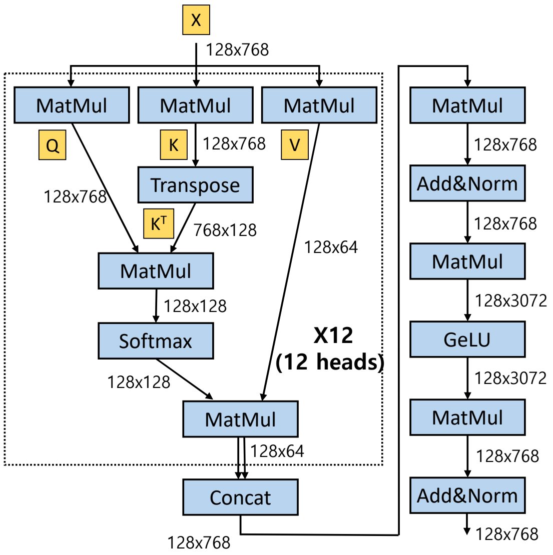

Figure 3: Overview of one encoder block of BERT-base Transformer architecture.

A.2 Homomorphic Encryption

HE is a cryptographic algorithm designed to perform arbitrary arithmetic operations directly on encrypted data. The CKKS HE scheme is optimized for real-number computations, which are widely used in AI tasks, making it a key technique for implementing HE-based privacy-preserving machine learning models. The CKKS scheme enables the encryption of a vector of length n, where the elements are either real or complex numbers. Specifically, given a vector $v = (v_0, \dots, v_{n-1}) \in \mathbb{C}^n$, it produces a corresponding ciphertext ct. Several operations can be performed directly on these ciphertexts, including addition, plaintext multiplication, ciphertext multiplication, rotation, and conjugation.

For two vectors v and w of length n, the operations $v + w$, $v \cdot w$, and $\bar{v}$ correspond to elementwise addition, multiplication, and conjugation, respectively. Additionally, the cyclic left shift of v by r positions, denoted as $\rho(v; r)$, is given by $(v_r, v_{r+1}, \dots, v_{n-1}, v_0, \dots, v_{r-1})$. If ct_1 and ct_2 represent the ciphertexts of vectors v_1 and v_2, respectively, the corresponding homomorphic operations function as follows:

- Addition: $\mathsf{Add}(\mathsf{ct}_1, \mathsf{ct}_2) = \mathsf{ct}_{\mathsf{add}}$, where ctadd decrypts to $v_1 + v_2$. This operation can be written as $\mathsf{ct}_1 + \mathsf{ct}_2$.
- Plaintext Multiplication: $\mathsf{PMult}(\mathsf{ct}_1, v_2) = \mathsf{ct}_{\mathsf{pmult}}$, where $\mathsf{ct}_{\mathsf{pmult}}$ decrypts to $v_1 \cdot v_2$. It can be expressed as $v_2 \cdot \mathsf{ct}_1$.
- Ciphertext Multiplication: $\mathsf{CMult}(\mathsf{ct}_1, \mathsf{ct}_2) = \mathsf{ct}_{\mathsf{cmult}}$, where $\mathsf{ct}_{\mathsf{cmult}}$ decrypts to $v_1 \cdot v_2$. This can be written as $\mathsf{ct}_1 \cdot \mathsf{ct}_2$.
- Rotation: $\mathsf{Rot}(\mathsf{ct}_1; r) = \mathsf{ct}_{\mathsf{rot}}$, where $\mathsf{ct}_{\mathsf{rot}}$ decrypts to $\rho(v_1, r)$.
- Multiplication by i: $\mathsf{Multi}(ct_1) = ct_{\mathsf{multi}}$, where ct_{multi} decrypts to $\mathrm{i} \cdot v_1$
- Conjugation: $\mathsf{Conj}(\mathsf{ct}_1) = \mathsf{ct}_{\mathsf{conj}}$, where $\mathsf{ct}_{\mathsf{conj}}$ decrypts to $\bar{v}_1$.

A.3 Homomorphic Matrix Multiplication

In this section, we first define column packing. Let $\{ct_i\}_{1 \le i \le d_1 d_2 / n}$ denote the ciphertexts obtained by column-packing the matrix $A \in \mathbb{R}^{d_1 \times d_2}$. For simplicity, assume $d_1 \mid n$ and $n \mid d_1 d_2$. If $m^{(i)} \in \mathbb{R}^n$ is the decrypted vector of ct_i, then for $0 \le j < n$, we have

$$m^{(i)}[j] = A_{[j]_{d_1},\, \frac{n}{d_1}(i-1) + \left\lfloor \frac{j}{d_1} \right\rfloor}.$$

We denote by $[A]_C$ the set of plaintext vectors $\{m^{(i)} \in \mathbb{R}^n\}_{1 \le i \le d_1 d_2 / n}$ that store the matrix A in a column-packed manner. If $n = d_1 d_2$, then $[A]_C$ is simply $m^{(1)}$.

For some constant vectors $a_i, b_i, c_\ell, c'_\ell$, the following equations hold:

$$\begin{aligned}
[\sigma(A)]_C &= \sum_{0 \le i < d} b_i \odot \rho([A]_C; di), [\tau(A)]_C \\
&= \sum_{-d < i < d} a_i \odot \rho([A]_C; i), [\phi^\ell(A)]_C \\
&= \rho([A]_C; d\ell), [\psi^\ell(A)]_C \\
&= c_\ell \rho([A]_C; \ell) + c'_\ell \rho([A]_C; \ell - d).
\end{aligned}$$

Then, these equations naturally lead to a homomorphic CCMM algorithm for column packing.

Now, we describe the CCMM algorithm of Jiang et al. (Jiang et al., 2018) under the column packing approach. Suppose we have $d \times d$ matrices A and B. First, Algorithm 2 takes as input the ciphertexts of $[A]_C$ and outputs the ciphertexts of $[\sigma(A)]_C$. Algorithm 3 takes as input the ciphertexts of $[A]_C$ and outputs the ciphertexts of $[\tau(A)]_C$. In Algorithm 2, we use N_1, N_2 satisfying $N_1 N_2 = d$, and typically set $N_1 = N_2 \approx \sqrt{d}$. In Algorithm 3, we have $N_1 N_2 = 2d - 1$ and typically set $N_1 = N_2 \approx \sqrt{2d - 1}$. With these choices, the two algorithms respectively require about $2\sqrt{d}$ and $2\sqrt{2d}$ rotations.

Algorithm 2: Sigma

```
Input: $ct$
Output: $ct'$
$ct' \leftarrow ct_{zero}$
for $i \leftarrow 0$ to $N_1 - 1$ do
    $ct^{(i)} \leftarrow \mathsf{Rot}(ct; di)$
end
for $j \leftarrow 0$ to $N_2 - 1$ do
    $ct'' \leftarrow ct_{zero}$
    for $i \leftarrow 0$ to $N_1 - 1$ do
        $ct'' \leftarrow ct'' + \rho(b_{N_1 j+i}; -dN_1 j) \odot ct^{(i)}$
    end
    $ct' \leftarrow ct' + \mathsf{Rot}(ct''; dN_1 j)$
end
return $ct'$
```

Algorithm 3: Tau

```
Input: $ct$
Output: $ct'$
$ct' \leftarrow ct_{zero}$
for $i \leftarrow 0$ to $N_1 - 1$ do
    $ct^{(i)} \leftarrow \mathsf{Rot}(ct; i)$
end
for $j \leftarrow 0$ to $N_2 - 1$ do
    $ct'' \leftarrow ct_{zero}$
    for $i \leftarrow 0$ to $N_1 - 1$ do
        $ct'' \leftarrow ct'' + \rho(a_{N_1 j+i-d+1}; -N_1 j + d - 1) \odot ct^{(i)}$
    end
    $ct' \leftarrow ct' + \mathsf{Rot}(ct''; N_1 j - d + 1)$
end
return $ct'$
```

Algorithm 4: CCMM algorithm for column packing (Jiang et al., 2018)

```
Input: $ct_1$ and $ct_2$
Output: $ct'$
$ct' \leftarrow ct_{zero}$
$ct_1 \leftarrow \mathsf{Sigma}(ct_1)$
$ct_2 \leftarrow \mathsf{Tau}(ct_2)$
for $i \leftarrow 0$ to $d - 1$ do
    $ct'_1 \leftarrow \mathsf{Rot}(ct_1; di)$
    $ct'_2 \leftarrow c_i \odot \mathsf{Rot}(ct_2; i) + c'_i \odot \mathsf{Rot}(ct_2; i - d)$
    $ct' \leftarrow ct' + ct'_1 \otimes ct'_2$
end
return $ct'$
```

Algorithm 4 takes as input the ciphertexts of $[A]_C$ and $[B]_C$ and outputs the ciphertexts of $[AB]_C$. It can be carried out using approximately $3d + 2\sqrt{d} + 2\sqrt{2d}$ rotations and d non-scalar multiplications.

B Details for Pseudo-Sign Composite Approximation

In Algorithm 1, two algorithms are used as subroutines: $\mathsf{MiniCompSign}(\epsilon, \delta)$ and $\mathsf{Remez}(f, \delta)$.

The algorithm $\mathsf{MiniCompSign}(\epsilon, \delta)$ outputs a composite polynomial $p_{t-1} \circ \cdots \circ p_0$ that satisfies $\|p_{t-1} \circ \cdots \circ p_0(x) - \mathsf{sign}(x)\|_{\infty, I_\epsilon} \leq \delta$ over the domain $I_\epsilon = [-1, -\epsilon] \cup [\epsilon, 1]$, while minimizing the total number of ciphertext-ciphtertext homomorphic multiplication operations. This algorithm was proposed by Lee et al. (Lee et al., 2022) to compute the sign function efficiently and is proven to find a composite polynomial with the minimal homomorphic multiplication operations. For a detailed description of this algorithm, refer to (Lee et al., 2022).

The algorithm $\mathsf{Remez}(f, \delta)$ finds a polynomial p of the minimum possible degree such that $\|p - f\|_\infty \leq \delta$ over a bounded closed domain $D \subset \mathbb{R}$, where $f : D \to \mathbb{R}$. The original Remez algorithm takes a function f and a fixed polynomial degree d as input and outputs the degree-d polynomial that minimizes the maximum approximation error, which is called the minimax polynomial. When combined with a binary search over the degree, this can be transformed into the form of the $\mathsf{Remez}(f, \delta)$ algorithm as used here. Details on this algorithm and its implementation can be found in (Lee et al., 2021).

The following theorem is essential for proving the correctness of the pseudo-sign composite approximation.

Theorem B.1. *For any (ϵ, δ)-approximate minimax composite polynomial for sign function $p(x) = (p_{t-1} \circ \cdots \circ p_0)(x)$, there exists ϵ' such that $\epsilon < \epsilon' < 1$ and $f(\epsilon') = -f(-\epsilon') = 1 + \delta$ holds, and $p(x)$ monotonously increase in the interval $[-\epsilon', \epsilon']$.*

Proof. We prove this by mathematical induction. First, we show that the theorem holds for a single minimax polynomial p_0. Then, assuming that the minimax composite polynomial $p_{n-2} \circ \cdots \circ p_0$ satisfies the given property, we prove that the minimax composite polynomial $p_{n-1} \circ \cdots \circ p_0$, obtained by composing p_{n-1}, also satisfies the same property.

Let us first verify whether the theorem holds for a single minimax polynomial. It is well known that the minimax polynomial of an odd function is also an odd function. If the degree of the minimax polynomial p_0 is $d = 2\ell - 1$, then this polynomial minimizes $\|p - \mathsf{sign}\|_{\infty,D}$ among all polynomials of degree at most $d + 1 = 2\ell$ on the domain $D = [-1, -\epsilon] \cup [\epsilon, 1]$. According to the Chebyshev alternation theorem, the number of local extreme points of $p(x) - \mathsf{sign}(x)$ within D must be $d + 3 = 2\ell + 2$. However, a polynomial of degree $d = 2\ell - 1$ over $\mathbb{R}$ can have at most $d - 1 = 2\ell - 2$ local extreme points. On D, the boundary points of D can also be local extreme points. Thus, to satisfy the Chebyshev alternation theorem, all four boundary points of D and the local extreme points of $\mathbb{R}$ must lie in D, and these points must all be distinct. Consequently, $p(x) - \mathsf{sign}(x) = p(x) - 1$ cannot have extreme points within $[0, \epsilon]$, meaning that $p(x)$ must be monotonic in this interval. Since $0 = p(0) \leq 1 - \delta \leq p(\epsilon)$, it follows that $p(x)$ is monotonically increasing in $[0, \epsilon]$. Additionally, since $x = \epsilon$ is not a local extreme point over $\mathbb{R}$, the sign of the derivative near $x = \epsilon$ cannot change. Thus, $p(x)$ must continue increasing, and $x = \epsilon$ is a local minimum point, satisfying $p(\epsilon) - 1 = -\delta$.

Let ϵ' denote the smallest local extreme point greater than ϵ. By the Chebyshev alternation theorem, $x = \epsilon'$ must be a local maximum point, so $p(\epsilon') - 1 = \delta$. This implies that $p(x) - 1$ must be an increasing function on $[\epsilon, \epsilon']$. Consequently, $p(x)$ is monotonically increasing on $[0, \epsilon']$. Since $p(x)$ is an odd function, it follows that $p(x)$ is also monotonically increasing on $[-\epsilon', \epsilon']$. Thus, we conclude that the theorem holds for a single minimax polynomial.

Let us prove the second inductive step. Assume that $\tilde{p} = p_{n-2} \circ \cdots \circ p_0$ satisfies the theorem. This polynomial is also an $(\epsilon, \tilde{\delta})$-approximate minimax composite polynomial for some $\tilde{\delta}$, meaning there exists $\tilde{\epsilon}'$ such that $\tilde{p}(x)$ is monotonically increasing on $[-\tilde{\epsilon}', \tilde{\epsilon}']$ and satisfies $\tilde{p}(\tilde{\epsilon}') = 1 + \tilde{\delta}$. By the definition of minimax composition, the approximation domain of p_{n-1} is $D_{n-1} = [-1 - \tilde{\delta}, -1 + \tilde{\delta}] \cup [1 - \tilde{\delta}, 1 + \tilde{\delta}]$. Since p_{n-1} is a single minimax polynomial, from the result of the first inductive step, there exists ϵ'' such that $1 - \tilde{\delta} < \epsilon'' < 1 + \tilde{\delta}$, and $p_{n-1}(x)$ is monotonically increasing on $[-\epsilon'', \epsilon'']$, satisfying $p_{n-1}(\epsilon'') = 1 + \delta$. Also, there must exist ϵ' within $[0, \tilde{\epsilon}']$ such that $\tilde{p}(\epsilon') = \epsilon''$. As $[-\epsilon', \epsilon'] \subset [-\tilde{\epsilon}', \tilde{\epsilon}']$, $\tilde{p}(x)$ is monotonically increasing within $[-\epsilon', \epsilon']$, and $p_{n-1} \circ \tilde{p} = p_{n-1} \circ \cdots \circ p_0$ is also monotonically increasing within $[-\epsilon', \epsilon']$. Additionally, since $p_{n-1}(\tilde{p}(\epsilon')) = p_{n-1}(\epsilon'') = 1 + \delta$, the second inductive condition is satisfied. Thus, the theorem is proven. □

The core principle of Algorithm 1 is as follows. Without loss of generality, let us fix the approximation interval to $[-1, 1]$ and assume that the pseudo-sign function f is approximated within this interval. The goal is to find a polynomial $p(x)$ such that $\|f(x) - p(x)\|_{\infty,[-1,1]} < \delta$. The pseudo-sign function considered in this method converges rapidly to ±1, resulting in intervals sufficiently close to ±1 being long enough to matter. At the same time, the transition regions where $f(x)$ approaches ±1 cannot be ignored and must be accurately approximated. Therefore, it is reasonable to assume that there exists a $\gamma \in (0, 1]$ such that $f(\gamma) = 1 - \delta/2$. Given this, consider a $(\gamma, \frac{\delta/4}{1-\delta/4})$-approximate minimax composite polynomial for the sign function, denoted as $p_{\text{com}}(x)$. Then, we have

$$\|p_{t-1} \circ \cdots \circ p_0 - \mathsf{sign}\|_{\infty,[-1,-\gamma]\cup[\gamma,1]} \leq \delta' \leq \frac{\delta/4}{1 - \delta/4},$$

where δ' is the real minimax error. We define a scaled composite polynomial $p_{\text{scale}}(x) = \frac{1}{1+\delta'} \cdot p_{\text{com}}(x)$, which satisfies the following conditions:

$$\begin{aligned}
&\left\| p_{\text{scale}} - \left(1 - \frac{\delta}{4}\right) \cdot \mathsf{sign} \right\|_{\infty,[-1,-\gamma]\cup[\gamma,1]} \\
&\leq \left\| p_{\text{scale}} - \frac{1}{1+\delta'} \cdot \mathsf{sign} \right\|_{\infty,[-1,-\gamma]\cup[\gamma,1]} \\
&\quad + \left\| \left(\frac{1}{1+\delta'} - \left(1 - \frac{\delta}{4}\right) \right) \cdot \mathsf{sign} \right\|_{\infty,[-1,-\gamma]\cup[\gamma,1]} \\
&\leq \frac{\delta'}{1+\delta'} + \left(\frac{1}{1+\delta'} - \left(1 - \frac{\delta}{4}\right) \right) \leq \frac{\delta}{4}
\end{aligned}$$

Next, we approximate f within the interval $[0, \gamma]$. By Theorem 4.1, there exists a $\gamma' > 0$ such that $p_{\text{scale}}(x)$ is a monotonically increasing function on $[-\gamma', \gamma']$ and satisfies $p_{\text{scale}}(\gamma') = 1$. Let $\tilde{p}_{\text{scale}}(x)$ denote the restriction of $p_{\text{scale}}(x)$ to the domain $[-\gamma', \gamma']$. Since $\tilde{p}_{\text{scale}}(x)$ is monotonically increasing, it has an inverse function. Using this inverse, we can define $g = f \circ \tilde{p}_{\text{scale}}^{-1} : [-1, 1] \to [-1, 1]$. We refer to $g(x)$ as the *transformation function*. This function is smooth and can be approximated by a single minimax polynomial $p_g(x)$ such that

$$\|p_g - g\|_{\infty,[-1,1]} \leq \frac{\delta}{2}.$$

Finally, we can approximate $f(x)$ using the composite polynomial

$$p_f(x) = p_g \circ p_{\text{scale}}(x).$$

This construction ensures that $f(x)$ is approximated with high accuracy while maintaining the desired properties of the pseudo-sign function within the given interval. The specific approximation method is detailed in Algorithm 1.

Below is the formal proof of the main theorem.

Proof. (Proof of Theorem 4.1)

Since each function is odd, it suffices to check for positive inputs only.

- If $0 \le x \le \gamma$, then $f(x) = g(p_{\text{scale}}(x))$ and $p_f(x) = p_g(p_{\text{scale}}(x))$ as $x \le \gamma < \gamma'$. Since $p_{\text{scale}}(x) \in [-1, 1]$, it follows that
$$\begin{aligned}|p_f(x) - f(x)| &= |p_g(p_{\text{scale}}(x)) - g(p_{\text{scale}}(x))| \\ &\le \delta/2 < \delta.\end{aligned}$$
Thus, $\|p_f - f\|_{\infty,[0,\gamma]} < \delta$ is satisfied.

- If $\gamma \le x \le 1$, then $\|p_{\text{scale}}(x) - (1 - \delta/4)\| \le \delta/4$, which implies $1 - \delta/4 \le p_{\text{scale}}(x) \le 1$. Define $\tilde{x} = \tilde{p}_{\text{scale}}^{-1} \circ p_{\text{scale}}(x)$. By definition, we have $\gamma \le \tilde{x} \le x \le 1$. Since $f(x)$ has a range within $[1 - \delta/2, 1]$ for $x \in [\gamma, 1]$, it follows that $|f(\tilde{x}) - f(x)| \le \delta/2$. For $\gamma \le \tilde{x} \le \gamma'$, we know $f(\tilde{x}) = g \circ \tilde{p}_{\text{scale}}(\tilde{x}) = g \circ p_{\text{scale}}(\tilde{x})$. Furthermore, by the definition of $\tilde{x}$, we have $p_f(x) = p_g \circ p_{\text{scale}}(x) = p_g \circ p_{\text{scale}}(\tilde{x}) = p_f(\tilde{x})$. This allows us to deduce that
$$\begin{aligned}|p_f(x) - f(\tilde{x})| &= |p_f(\tilde{x}) - f(\tilde{x})| \\ &= |p_g(p_{\text{scale}}(\tilde{x})) - g(p_{\text{scale}}(\tilde{x}))|.\end{aligned}$$
Since $p_{\text{scale}}(\tilde{x}) \in [-1, 1]$, it follows that
$$|p_f(x) - f(\tilde{x})| = |p_g(p_{\text{scale}}(\tilde{x})) - g(p_{\text{scale}}(\tilde{x}))| \le \delta/2.$$
Finally, combining these results, we obtain
$$\begin{aligned}|p_f(x) - &f(x)| \\ &\le |p_f(x) - f(\tilde{x})| + |f(\tilde{x}) - f(x)| \\ &\le \delta/2 + \delta/2 = \delta.\end{aligned}$$
Thus, $\|p_f - f\|_{\infty,[\gamma,1]} \le \delta$ is satisfied.

Due to the odd-function property of $f(x)$, this result holds symmetrically for $x \in [-1, 0]$ as well. Therefore, combining the results for all intervals, we conclude that

$$\|p_f - f\|_{\infty,[-1,1]} \le \delta.$$

□

C Detailed Algorithms for Optimized Matrix Multiplication

C.1 Packing Method

In this paper, we use a new packing method called *modified column packing* instead of column packing. Suppose we have a matrix A of size $d_1 \times d_2$ and a natural number k satisfying $k|d_1$ and $k|\frac{n}{d_1}$. For simplicity, assume $n|d_1 d_2$. The modified column packing for k takes A as input and outputs $\{ct_i\}_{1 \le i \le \frac{d_1 d_2}{n}}$, where each ct_i encrypts a vector $m^{(i)} \in \mathbb{R}^n$ defined by

$$m^{(i)}[j] = A_{[j]_{d_1},\, k\left[\left\lfloor \frac{j}{d_1} \right\rfloor\right]_{\frac{n}{kd_1}} + \left\lfloor \frac{kj}{n} \right\rfloor + \frac{n}{d_1}(i-1)}$$

for $0 \le j < n$. We denote the set of vectors $\{m^{(i)}\}_{1 \le i \le \frac{d_1 d_2}{n}}$ by $[A]_C^k$. When $d_1 d_2 = n$, $[A]_C^k$ is simply $m^{(1)}$. Blockwise matrix operation algorithms based on this modified column packing method require fewer rotations compared to blockwise algorithms based on column packing. We ensure that any intermediate matrix computed during BERT model inference is always packed using modified column packing with the parameter $k = 64$. Figure 4 illustrates both the column packing and modified column packing methods.

C.2 Optimized CPMM

In this paper, we present a CPMM algorithm based on modified column packing. For the matrices $A \in \mathbb{R}^{d_1 \times d_2}$ and $B \in \mathbb{R}^{d_2 \times d_3}$, consider the situation of computing the matrix product AB. Let $k|d_1$, $k|d_2$, and $k|d_3$. The proposed algorithm computes the ciphertexts corresponding to $[AB]_C^k$ from the ciphertexts of $[A]_C^k$. The columns of the input matrix are packed into a total of $\mathsf{mid} = \frac{d_1 d_2}{n}$ ciphertexts, $\{ct_i\}_{1 \le i \le \mathsf{mid}}$, and the columns of the output matrix are packed into a total of $\mathsf{ed} = \frac{d_1 d_3}{n}$ ciphertexts, $\{ct'_i\}_{1 \le i \le \mathsf{ed}}$.

Based on the fact that each column of the output matrix can be expressed as a linear combination of the columns of the input matrix, the following equation can be derived:

$$ct'_\ell = \sum_{1 \le j \le \mathsf{mid}} \sum_{0 \le i < \frac{n}{d_1}} \mathsf{Rot}(ct_j; id_1) \odot d'^{j,\ell}_i. \quad (6)$$

for $1 \le \ell \le \mathsf{ed}$. Here, $d'^{j,\ell}_i \in \mathbb{R}^n$ stores the elements of matrix B appropriately. By applying the baby-step giant-step technique to the above equation, it can be transformed into the following form for some natural numbers N_1, N_2 satisfying $N_1 N_2 = \frac{n}{d_1}$:

$$ct'_\ell = \sum_{0\le p<N_2} \mathsf{Rot}\Big(\sum_{1\le j\le \mathsf{mid}} \sum_{0\le q<N_1} \mathsf{Rot}(ct_j; qd_1) \odot \rho(d'^{j,\ell}_{pN_1+q}; -pN_1 d_1); pN_1 d_1\Big). \tag{7}$$

for $1 \le \ell \le \mathsf{ed}$. Algorithm 5 is derived from the above equation. This algorithm uses $\mathsf{mid} \cdot N_1 + \mathsf{ed} \cdot N_2$ rotations, and when N_1 and N_2 satisfy $\mathsf{mid} \cdot N_1 = \mathsf{ed} \cdot N_2$, it requires approximately $2\sqrt{\frac{n}{d_1} \cdot \mathsf{mid} \cdot \mathsf{ed}}$ rotations.

Note that in the main text, our CPMM algorithm is described under column packing, whereas in the appendix, it is described under modified column packing. For both packing methods, Equation 6 remains valid (only the plaintext vectors $d'^{j,\ell}_i$ change). Hence, the algorithm's procedure and computational complexity are exactly the same.

Algorithm 5: CPMM algorithm

```
Input: $\{ct_j\}_{1\le j\le \mathsf{mid}}$
Output: $\{ct'_\ell\}_{1\le \ell\le \mathsf{ed}}$
for $j \leftarrow 1$ to mid do
    for $q \leftarrow 0$ to $N_1 - 1$ do
        $ct_j^{(q)} \leftarrow \mathsf{Rot}(ct_j; qd_1)$
    end
end
for $\ell \leftarrow 1$ to ed do
    $ct'_\ell \leftarrow ct_{zero}$
    for $p \leftarrow 0$ to $N_2 - 1$ do
        $ct' \leftarrow ct_{zero}$
        for $j \leftarrow 1$ to mid do
            for $q \leftarrow 0$ to $N_1 - 1$ do
                $ct' \leftarrow ct' + ct_j^{(q)} \odot$
                  $\rho(d'^{j,\ell}_{pN_1+q}; -pN_1 d_1)$
            end
        end
        $ct'_\ell \leftarrow ct'_\ell + \mathsf{Rot}(ct'; pN_1 d_1)$
    end
end
return $\{ct'_\ell\}_{1\le \ell\le \mathsf{ed}}$
```

In addition, we speed up Algorithm 5 by making appropriate use of complex numbers. The followings explain how complex numbers are utilized, depending on the specific case.

Q, K, V Calculation In the BERT-base model, from the input matrix $X \in \mathbb{R}^{L\times d_m}$, we need to compute $Q^{(j)} = XW_Q^{(j)}, K^{(j)} = XW_K^{(j)}, V^{(j)} = XW_V^{(j)}$ for $W_Q^{(j)}, W_K^{(j)}, W_V^{(j)} \in \mathbb{R}^{d_m \times d_m/h}$ where $0 \le j < h$. We have parameters $L = 128, d_m = 768$, and $h = 12$. This can be viewed as computing XW for one large matrix $W \in \mathbb{R}^{d_m \times 3d_m}$ obtained by concatenating all the smaller matrices. Consequently, it suffices to compute the following equation.

$$ct'_\ell = \sum_{1\le j\le \mathsf{mid}} \sum_{0\le i<\frac{n}{L}} \mathsf{Rot}(ct_j; iL) \odot d'^{j,\ell}_i \tag{8}$$

for $1 \le \ell \le \mathsf{ed}$. Here, $\mathsf{mid} = \frac{Ld_m}{n} = 3$ and $\mathsf{ed} = \frac{3Ld_m}{n} = 9$. We need to compute the expression for a total of $\mathsf{ed} = 9$ output ciphertexts. By making use of complex numbers, the expression for any two ciphertexts ct'_{ℓ_1} and ct'_{ℓ_2} can be computed at once as follows:

$$ct' = \sum_{1\le j\le \mathsf{mid}} \sum_{0\le i<\frac{n}{L}} \mathsf{Rot}(ct_j; iL) \odot (d'^{j,\ell_1}_i + id'^{j,\ell_2}_i) \tag{9}$$

Afterward, by using the Extract algorithm, which extracts the real and imaginary parts, we obtain $ct'_{\ell_1}, ct'_{\ell_2} = \mathsf{Extract}(ct')$. Thus, by pairing up 8 of the 9 output ciphertexts in twos, we only need to compute the expression for a total of 5 ciphertexts ($\mathsf{ed} = 5$). When using the baby-step giant-step algorithm, the number of rotations is approximately $2\sqrt{\frac{n}{L} \cdot 3 \cdot 5}$, which is $\sqrt{\frac{5}{9}}$ times the $2\sqrt{\frac{n}{L} \cdot 3 \cdot 9}$ required by Algorithm 5.

Multiplication with W^O or W_{F1} The concatenated attention matrix $Y \in \mathbb{R}^{L\times d_m}$ is multiplied by $W^O \in \mathbb{R}^{768\times 768}$. In the feed-forward network, $Y \in \mathbb{R}^{L\times d_m}$ is multiplied by $W_{F1} \in \mathbb{R}^{d_m \times d_h}$ (where $d_h = 3072$). In both cases, we can make use of complex numbers to combine the expressions for two ciphertexts in the same way, thereby reducing ed from $3 \to 2$ and from $12 \to 6$, respectively. Consequently, the number of rotations in each case is reduced to $\sqrt{\frac{2}{3}}$ and $\sqrt{\frac{1}{2}}$ times that of Algorithm 5.

Multiplication with W^O In multi-head attention, the concatenated result matrix $Y \in \mathbb{R}^{128\times 768}$ is multiplied by $W^O \in \mathbb{R}^{768\times 768}$. In this case, we need to compute Equation 8 for $1 \le \ell \le 3$, and as in the computation of Q, K, and V, we can use complex numbers to merge the expressions for two output ciphertexts into one. Consequently, the number of rotations is $2\sqrt{\frac{n}{L} \cdot \mathsf{mid} \cdot 2}$, which is $\sqrt{\frac{2}{3}}$ times the $2\sqrt{\frac{n}{L} \cdot \mathsf{mid} \cdot 3}$ required by Algorithm 5.

Multiplication with W_{F1} When multiplying $Y \in \mathbb{R}^{128\times 768}$ by $W_{F1} \in \mathbb{R}^{768\times 3072}$, the parameters are mid $=3$ and ed $=12$, so Equation 8 must be computed for the 12 ciphertexts ct'_ℓ (for $1 \le \ell \le 12$). By similarly utilizing complex numbers to pair these ciphertexts, we only need to compute it for the 6 ciphertexts. The number of rotations is $2\sqrt{\frac{n}{L}\cdot \text{mid} \cdot 6}$, which is $\sqrt{2}$ times fewer than the $2\sqrt{\frac{n}{L}\cdot \text{mid}\cdot 12}$ required by Algorithm 5.

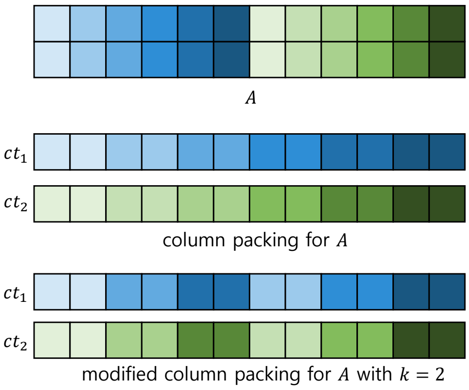

Figure 4: Column packing and modifid column packing with $k=2$ for $d_1=2, d_2=12$, and $n=12$.

C.3 CCMM for Square Matrix

In this section, we present a new algorithm that is faster than the CCMM algorithm of Jiang et al. We begin with the following Equation:

$$A \cdot B = \sum_{\ell=0}^{d-1} (\phi^\ell \circ \sigma(A)) \odot (\psi^\ell \circ \tau(B)).$$

First, we note that the operation ψ^i satisfies the following equation:

$$[\psi^i(A)]_C = \rho([A]_C \odot R_i + \rho([A]_C;-d) \odot L_i; i) \tag{10}$$

Here, the constant vectors $R_i, L_i \in \mathbb{R}^n$ for $0 \le i < d$ are defined as follows:

$$R_i[j] = \begin{cases} 1\colon [j]_4 \ge i \\ 0\colon else \end{cases} \quad L_i[j] = \begin{cases} 1\colon [j]_4 < i \\ 0\colon else. \end{cases}$$

for $0 \le j < n$.

Then, for $A' = \sigma(A)$, $B' = \tau(B)$, and natural numbers N_1, N_2 satisfying $N_1 N_2 = d$, the following equation holds.

$$\begin{aligned}
&[\sum_{0\le\ell<d} \phi_\ell(A') \odot \psi_\ell(B')]_C \\
&= \sum_{0\le\ell<d} \rho([A']_C;\ell d) \odot [\psi_\ell(B')]_C \\
&= \sum_{0\le j<N_2} \sum_{0\le i<N_1} \rho([A']_C;(N_1 j+i)d) \\
&\quad \odot [\psi_{N_1 j+i}(B')]_C \\
&= \sum_{0\le j<N_2} \rho(\sum_{0\le i<N_1} \rho([A']_C;id) \odot \\
&\quad \rho([\psi_{N_1 j+i}(B')]_C;-N_1 jd);N_1 jd) \\
&= \sum_{0\le j<N_2} \rho(\sum_{0\le i<N_1} \rho([A']_C;id) \odot \rho([B']_C \odot \\
&R_{N_1 j+i} + \rho([B']_C;-d) \odot L_{N_1 j+i}; \\
&N_1 j+i-N_1 jd);N_1 jd).
\end{aligned}$$

Algorithm 6 describes the CCMM algorithm based on the final equation above. It requires $2\sqrt{d}+2\sqrt{2d}+N_2+N_1+N_1N_2$ rotations, which becomes $d+2\sqrt{2d}+4\sqrt{d}$ when $N_1=N_2=\sqrt{d}$. Moreover, by using the lazy relinearization technique—where all non-scalar multiplication results in the loop over j are summed up first and then relinearized only once at the end—the total number of relinearizations used is $\sqrt{d}$. Consequently, the total number of key-switches is $d+2\sqrt{2d}+5\sqrt{d}$, which is smaller than the $4d+2\sqrt{2d}+2\sqrt{d}$ required by Algorithm 4 (Jiang et al., 2018).

C.4 Blockwise Matrix Multiplication

By using the modified column packing method, we can obtain homomorphic algorithms for $\tilde{\sigma}, \tilde{\tau}, \tilde{\phi}, \tilde{\psi}$ on $n=d_1d_2$. Each of these has the same computational complexity as its corresponding homomorphic algorithm for σ, τ, ϕ, ψ on $n=k^2$, respectively. In this section, we describe the algorithms for the proposed blockwise matrix operations. Let $A, B \in \mathbb{R}^{d_1\times d_2}$ and suppose we have a k such that $k|d_1$ and $k|d_2$. We define $\tilde{\sigma}$ and $\tilde{\tau}$ to be the operations that apply σ and τ, respectively, block by block. Also, let $A \boxdot B$ denote the result of the blockwise (with block size k) multiplication of A and B.

Similar to the equation $[\sigma(A)]_C = \sum_{0\le i<k} b_i \odot \rho([A]_C;ki)$, we have $[\tilde{\sigma}(A)]_C^k = \sum_{0\le i<k} b'_i \odot \rho([A]_C^k;\frac{d_1d_2}{k}i)$ where each vector $b'_i \in \mathbb{R}^{d_1d_2}$ is obtained by splitting $b_i \in \mathbb{R}^{k^2}$ into chunks of size k and repeating each chunk $\frac{d_1d_2}{k^2}$ times. Similarly, just

```
Algorithm 6: CCMM algorithm
  Input: $ct_1$ and $ct_2$
  Output: $ct'$
  $ct' \leftarrow ct_{zero}$
  $ct_1 \leftarrow \mathsf{Sigma}(ct_1)$
  $ct_2 \leftarrow \mathsf{Tau}(ct_2)$
  $ct_2' \leftarrow \mathsf{Rot}(ct_2; -d)$
  for $j \leftarrow 0$ to $N_2 - 1$ do
      $ct^{(j)} \leftarrow \mathsf{Rot}(ct_1; jd)$
  end
  for $i \leftarrow 0$ to $N_1 - 1$ do
      $ct'' \leftarrow ct_{zero}$
      for $j \leftarrow 0$ to $N_2 - 1$ do
          $ct'' \leftarrow$
            $ct'' + ct^{(j)} \otimes \mathsf{Rot}(ct_2 \odot R_{N_1 i+j} +$
            $ct_2' \odot L_{N_1 i+j}; N_1 i + j - N_1 id)$
      end
      $ct' \leftarrow ct' + \mathsf{Rot}(ct''; N_1 id)$
  end
  return $ct'$
```

as τ for a $k \times k$ matrix is expressed as $[\tau(A)]_C = \sum_{-k<i<k} a_i \odot \rho\big([A]_C; i\big)$, $\tilde{\tau}$ for a $d_1 \times d_2$ matrix can be written as $[\tilde{\tau}(A)]_C^k = \sum_{-k<i<k} a_i' \odot \rho\big([A]_C^k; i\big)$. Similarly, just as $[\phi^i(A)]_C = \rho([A]_C; ki)$, we have $[\tilde{\phi}^i(A)]_C^k = \rho([A]_C^k; \frac{d_1 d_2}{k} i)$. Likewise, just as $[\psi^i(A)]_C = c_i \odot \rho([A]_C; i) + c_i' \rho([A]_C; i-k)$, we have $[\tilde{\psi}^i(A)]_C^k = \bar{c}_i \odot \rho([A]_C^k; i) + c_i' \odot \rho([A]_C^k; i-k)$. Therefore, the blockwise operations share the same computational complexity (i.e., the same number of rotations) as the original operations, and the baby-step giant-step approach also retains the same complexity. For matrix transposition, the blockwise counterpart to $[A^T]_C = \sum_{i=-d+1}^{d-1} s_i \odot \rho([A]_C; (d-1)i)$ is $\sum_{i=-d+1}^{d-1} s_i' \odot \rho([A]_C^k; (2d-1)i)$, which likewise has the same computational cost. As an example, Figure 5 shows the algorithm for $\tilde{\phi}$ when using modified column packing.

Algorithm 6 can also be naturally extended to a blockwise (matrix multiplication for k) algorithm on a $d_1 \times d_2$ matrix. The constant vectors R_i' and L_i' used here are defined in the same way as R_i and L_i (in Section 5.2.1), respectively, with the only difference being that $n = d_1 d_2$. Then, the following holds.

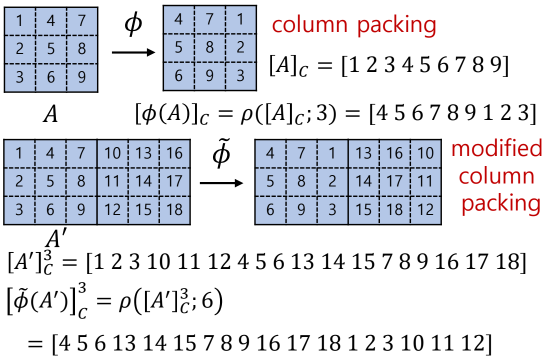
ϕ
column packing
$[A]_C$ = [1 2 3 4 5 6 7 8 9]
A
$[\phi(A)]_C = \rho([A]_C; 3)$ = [4 5 6 7 8 9 1 2 3]
$\tilde{\phi}$
modified column packing
A'
$[A']_C^3$ = [1 2 3 10 11 12 4 5 6 13 14 15 7 8 9 16 17 18]
$[\tilde{\phi}(A')]_C^3 = \rho([A']_C^3; 6)$
= [4 5 6 13 14 15 7 8 9 16 17 18 1 2 3 10 11 12]

Figure 5: An algorithm for $\tilde{\phi}$ using modified column packing

$$
\begin{aligned}
&[\sum_{0\le \ell<d} \tilde{\phi}_\ell(A') \odot \tilde{\psi}_\ell(B')]_C^k \\
&= \sum_{0\le \ell<d} \rho([A']_C^k; \frac{d_1 d_2}{k}\ell) \odot [\tilde{\psi}_\ell(B')]_C^k \qquad (11)\\
&= \sum_{0\le j<N_2} \sum_{0\le i<N_1} \rho([A']_C^k; \frac{d_1 d_2}{k}(N_1 j + i)) \\
&\qquad \odot [\tilde{\psi}_{N_1 i+j}(B')]_C^k \\
&= \sum_{0\le j<N_2} \rho(\sum_{0\le i<N_1} \rho([A']_C^k; \frac{d_1 d_2}{k} i) \odot \\
&\qquad \rho([\tilde{\psi}_{N_1 j+i}(B')]_C^k; -\frac{d_1 d_2}{k} N_1 j); \frac{d_1 d_2}{k} N_1 j) \\
&= \sum_{0\le j<N_2} \rho(\sum_{0\le i<N_1} \rho([A']_C^k; \frac{d_1 d_2}{k} i) \odot \\
&\rho([B']_C^k \odot R_{N_1 j+i}' + \rho([B']_C^k; -k) \odot L_{N_1 j+i}'; \\
&N_1 j + i - \frac{d_1 d_2}{k} N_1 j); \frac{d_1 d_2}{k} N_1 j).
\end{aligned}
$$

The above derivations for blockwise operations each correspond to their respective algorithms. Algorithm 7 takes as input the ciphertexts of $[A]_C^k$ and outputs the ciphertexts of $[\tilde{\sigma}(A)]_C^k$. Algorithm 8 takes as input the ciphertexts of $[A]_C^k$ and outputs the ciphertexts of $[\tilde{\tau}(A)]_C^k$. Finally, Algorithm 9 takes as input the ciphertexts of $[A]_C^k$ and $[B]_C^k$ and outputs the ciphertexts of $[A \boxdot B]_C^k$.

Algorithm 10 is essentially Algorithm 9 with only the BlockSigma and BlockTau steps removed, allowing these steps to be computed separately. Algorithm 10 is used in Algorithm 14. Algorithm 11 describes the blockwise transpose operation. Let A^{BT} denote the matrix obtained by transposing the matrix $A \in \mathbb{R}^{d_1 \times d_2}$ in blocks of size k. Then, Algorithm 11 takes as input the ciphertexts of $[A]_C^k$ and outputs the ciphertexts of $[A^{BT}]_C^k$.

Algorithm 7: BlockSigma

```
Input: $ct$
Output: $ct'$
$ct' \leftarrow ct_{zero}$
for $i \leftarrow 0$ to $N_1 - 1$ do
    $ct^{(i)} \leftarrow \mathsf{Rot}(ct; \frac{d_1 d_2}{k} i)$
end
for $j \leftarrow 0$ to $N_2 - 1$ do
    $ct'' \leftarrow ct_{zero}$
    for $i \leftarrow 0$ to $N_1 - 1$ do
        $ct'' \leftarrow$
            $ct'' + \rho(b'_{N_1 j+i}; -\frac{d_1 d_2}{k} N_1 j) \odot ct^{(i)}$
    end
    $ct' \leftarrow ct' + \mathsf{Rot}(ct''; \frac{d_1 d_2}{k} N_1 j)$
end
return $ct'$
```

Algorithm 8: BlockTau

```
Input: $ct$
Output: $ct'$
$ct' \leftarrow ct_{zero}$
for $i \leftarrow 0$ to $N_1 - 1$ do
    $ct^{(i)} \leftarrow \mathsf{Rot}(ct; i)$
end
for $j \leftarrow 0$ to $N_2 - 1$ do
    $ct'' \leftarrow ct_{zero}$
    for $i \leftarrow 0$ to $N_1 - 1$ do
        $ct'' \leftarrow ct'' + \rho(a'_{N_1 j+i-k+1}; -N_1 j +$
            $k - 1) \odot ct^{(i)}$
    end
    $ct' \leftarrow ct' + \mathsf{Rot}(ct''; N_1 j - k + 1)$
end
return $ct'$
```

Algorithm 9: Optimized blockwise ciphertext-ciphertext matrix multiplication algorithm BlockMult

```
Input: $ct_1$ and $ct_2$
Output: $ct'$
$ct' \leftarrow ct_{zero}$
$ct_1 \leftarrow \mathsf{BlockSigma}(ct_1)$
$ct_2 \leftarrow \mathsf{BlockTau}(ct_2)$
$ct'_2 \leftarrow \mathsf{Rot}(ct_2; -k)$
for $i \leftarrow 0$ to $N_1 - 1$ do
    $ct^{(i)} \leftarrow \mathsf{Rot}(ct_1; \frac{d_1 d_2}{k} i)$
end
for $j \leftarrow 0$ to $N_2 - 1$ do
    $ct'' \leftarrow ct_{zero}$
    for $i \leftarrow 0$ to $N_1 - 1$ do
        $ct'' \leftarrow$
            $ct'' + ct^{(i)} \otimes \mathsf{Rot}(ct_2 \odot R'_{N_1 j+i} +$
            $ct'_2 \odot L'_{N_1 j+i}; N_1 j + i - \frac{d_1 d_2}{k} N_1 j)$
    end
    $ct' \leftarrow ct' + \mathsf{Rot}(ct''; \frac{d_1 d_2}{k} N_1 j)$
end
return $ct'$
```

C.5 CCMM for Multi-Head Attention

In this section, we address the optimization of the CCMM computations required to calculate multi-head attention. Specifically, these are the operations for the $Q^{(j)}K^{(j)T}$ multiplication and for multiplying the resulting matrix (after passing through the softmax) by $V^{(j)}$. We first introduce two component algorithms that make up this procedure. Algorithm 12 takes as input a ciphertext ct encrypting $a + b\mathrm{i}$ and outputs the ciphertexts ct_{real} and ct_{imag} encrypting a and b, respectively. This algorithm uses one key-switch for the conjugation operation. Any unnecessary level consumption arising from multiplying by 0.5 can be addressed by compensating for the 0.5 factor in the constants used immediately before or after this step. In addition, Algorithm 13 handles the operation of taking half of a matrix and copying it onto the other half. The constant vectors $y^{(0)}, y^{(1)} \in \mathbb{R}^n$ used in Algorithm 13 are defined as follows:

$$y^{(i)}[j] = \begin{cases} 1 - i & : [j]_{S_1} < S_1/2 \\ i & : else \end{cases}$$

for $0 \leq j < n$ and $0 \leq i \leq 1$.

For the CCMM computations in multi-head attention, we appropriately employ blockwise operations with $k = 64$. In addition, we use an idea that utilizes complex number components to reduce the computational cost. For example, if we need to compute $\sigma(A)$ and $\sigma(B)$, we can instead compute $\sigma(A + B\mathrm{i})$ and then separate the real and imaginary parts, reducing the number of calls to σ. The same approach applies to τ and the transpose algorithm. Furthermore, when we need to compute AB and AC, we can reduce the number of multiplication algorithms by computing $A(B + C\mathrm{i})$ instead. Additionally, if we need to compute $AB + CD$, we can replace it by computing $(A + C\mathrm{i})(B - D\mathrm{i})$ and extracting the real part. Algorithm 14 combines these ideas into a final optimized algorithm. Let $Q = [Q^{(0)}|Q^{(1)}|\cdots|Q^{(11)}]$, $K = [K^{(0)}|K^{(1)}|\cdots|K^{(11)}]$, and $V = [V^{(0)}|V^{(1)}|\cdots|V^{(11)}]$. Also, let the concatenated

Algorithm 10: Optimized blockwise ciphertext-ciphertext matrix multiplication algorithm BlockMult' (without Sigma and Tau)

```
Input: $ct_1$ and $ct_2$
Output: $ct'$
$ct' \leftarrow ct_{zero}$
$ct'_2 \leftarrow \mathsf{Rot}(ct_2; -k)$
for $i \leftarrow 0$ to $N_1 - 1$ do
    $ct^{(i)} \leftarrow \mathsf{Rot}(ct_1; \frac{d_1 d_2}{k} i)$
end
for $j \leftarrow 0$ to $N_2 - 1$ do
    $ct'' \leftarrow ct_{zero}$
    for $i \leftarrow 0$ to $N_1 - 1$ do
        $ct'' \leftarrow ct'' + ct^{(i)} \otimes \mathsf{Rot}(ct_2 \odot R'_{N_1 j+i} + ct'_2 \odot L'_{N_1 j+i}; N_1 j + i - \frac{d_1 d_2}{k} N_1 j)$
    end
    $ct' \leftarrow ct' + \mathsf{Rot}(ct''; \frac{d_1 d_2}{k} N_1 j)$
end
return $ct'$
```

Algorithm 11: BlockTrans

```
Input: $ct$
Output: $ct'$
$ct' \leftarrow ct_{zero}$
for $i \leftarrow 0$ to $N_1 - 1$ do
    $ct^{(i)} \leftarrow \mathsf{Rot}(ct; (2k-1)i)$
end
for $j \leftarrow 0$ to $N_2 - 1$ do
    $ct'' \leftarrow ct_{zero}$
    for $i \leftarrow 0$ to $N_1 - 1$ do
        $ct'' \leftarrow ct'' + \rho(s'_{N_1 j+i-k+1}; -(2k-1)(N_1 j - k + 1)) \odot ct^{(i)}$
    end
    $ct' \leftarrow ct' + \mathsf{Rot}(ct''; (2k-1)(N_1 j - k + 1))$
end
return $ct'$
```

Algorithm 12: Extract

```
Input: $ct$
Output: $ct_{real}, ct_{imag}$
$ct' \leftarrow \mathsf{Conj}(ct)$
$ct_{real} \leftarrow 0.5 \odot (ct + ct')$
$ct_{imag} \leftarrow -0.5 \odot \mathsf{Multi}(ct - ct')$
return $ct_{real}, ct_{imag}$
```

Algorithm 13: SplitPaste

```
Input: $ct$
Output: $ct'_1, ct'_2$
$ct'_1 \leftarrow ct \odot y^{(0)}$
$ct'_1 \leftarrow ct'_1 + \mathsf{Rot}(ct'_1; S_1/2)$
$ct'_2 \leftarrow ct \odot y^{(1)}$
$ct'_2 \leftarrow ct'_2 + \mathsf{Rot}(ct'_2; S_1/2)$
return $ct'_1, ct'_2$
```

attention matrix be $Y = [Y_0|Y_1|\cdots|Y_{11}] \in \mathbb{R}^{L \times d_m}$. Then, Algorithm 14 takes the ciphertexts of $[Q]_C^k, [K]_C^k, [V]_C^k$ as input and outputs the ciphertexts of $[Y]_C^k$.

In this algorithm, Soft is a function that briefly represents the softmax function; in practice, since we replace the softmax function with an component-wise function, this operation can be carried out as a component-wise operation without any packing concerns.

D Ablation Study on Optimized Homomorphic Matrix Multiplication

In Section 5, we presented a comparison of the number of key-switch operations between the proposed homomorphic matrix multiplication algorithm and existing methods such as NEXUS (Zhang et al., 2024), BOLT (Pang et al., 2024), and THOR (Moon et al., 2024b). In this section, we provide a more detailed ablation study to specifically analyze the performance improvements contributed by each proposed technique. The proposed CPMM is based on the BOLT algorithm, which utilizes column packing. Table 11 presents the ablation study results for CPMM. Similarly, our proposed CCMM builds upon the algorithm introduced by Jiang et al. (Jiang et al., 2018). Table 12 shows the ablation study results for CCMM.

Algorithm 14: Optimized CCMM algorithm for multi-head attention

```
Input: $\{ct'_{qi}\}_{1\leq i\leq 3}$, $\{ct'_{ki}\}_{1\leq i\leq 3}$, and
       $\{ct'_{vi}\}_{1\leq i\leq 3}$
Output: $ct_{qkv1}, ct_{qkv2}$, and $ct_{qkv3}$
$\hat{ct}_{k2} \leftarrow ct'_{k2} + \mathsf{Multi}(ct'_{k3})$
$ct'_{k1} \leftarrow \mathsf{BlockTrans}(ct'_{k1})$
$\hat{ct}_{k2} \leftarrow \mathsf{BlockTrans}(\hat{ct}_{k2})$
$ct'_{k1} \leftarrow \mathsf{BlockTau}(ct'_{k1})$
$\hat{ct}_{k2} \leftarrow \mathsf{BlockTau}(\hat{ct}_{k2})$
$ct'_{k2}, ct'_{k3} \leftarrow \mathsf{Extract}(\hat{ct}_{k2})$
for $i \leftarrow 1$ to $3$ do
    $ct_{ki,1}, ct_{ki,2} \leftarrow \mathsf{SplitPaste}(ct'_{ki})$
    $\hat{ct}_{ki} \leftarrow ct'_{ki,1} + \mathsf{Multi}(ct'_{ki,2})$
end
$\hat{ct}_{q2} \leftarrow ct'_{q2} + \mathsf{Multi}(ct'_{q3})$
$ct'_{q1} \leftarrow \mathsf{BlockSigma}(ct'_{q1})$
$ct'_{q2}, ct'_{q3} \leftarrow \mathsf{Extract}(\mathsf{BlockSigma}(\hat{ct}_{q2}))$
for $i \leftarrow 1$ to $3$ do
    $\hat{ct}_{qki} \leftarrow \mathsf{BlockMult'}(ct'_{qi}, \hat{ct}_{ki})$
    $ct_{qki,1}, ct_{qki,2} \leftarrow \mathsf{Extract}(\hat{ct}_{qki})$
    $ct_{qki,1} \leftarrow \mathsf{Soft}(ct_{qki,1})$
    $ct_{qki,2} \leftarrow \mathsf{Soft}(ct_{qki,2})$
    $\hat{ct}_{qki} \leftarrow ct_{qki,1} + \mathsf{Multi}(ct_{qki,2})$
end
$ct'_{v1} \leftarrow \mathsf{BlockTau}(ct'_{v1})$
$\hat{ct}_{v2} \leftarrow \mathsf{BlockTau}(ct'_{v2} + \mathsf{Multi}(ct'_{v3}))$
$ct'_{v2}, ct'_{v3} \leftarrow \mathsf{Extract}(\hat{ct}_{v2})$
for $i \leftarrow 1$ to $3$ do
    $ct'_{vi,1}, ct'_{vi,2} \leftarrow \mathsf{SplitPaste}(ct'_{vi})$
    $\hat{ct}_{vi} \leftarrow ct'_{vi,1} - \mathsf{Multi}(ct'_{vi,2})$
    $\hat{ct}_{qki} \leftarrow \mathsf{BlockSigma}(\hat{ct}_{qki})$
    $\hat{ct}_{qkvi} \leftarrow \mathsf{BlockMult'}(\hat{ct}_{qki}, \hat{ct}_{vi})$
    $ct_{qkvi}, \# \leftarrow \mathsf{Extract}(\hat{ct}_{qkvi})$
end
return $ct_{qkv1}, ct_{qkv2}$, and $ct_{qkv3}$
```

CPMM	$\times W_Q, W_K, W_V$	$\times W_O$	$\times W_{F1}$	$\times W_{F2}$	total	ratio
	$(128 \times 768,$ $768 \times 768) \times 3$	$128 \times 768,$ 768×768	$128 \times 768,$ 768×3072	$128 \times 3072,$ 3072×768		
BOLT	288	168	324	324	1104	100
with optimization	166	96	192	192	646	58.5
with complex	122	75	135	132	464	42.03

Table 11: Ablation study on CPMM

CCMM	$Q \times K^T$	$\times V$	total	ratio
	$(128 \times 768,$ 768×128	$128 \times 128,$ 128×768		
Jiang et al.	1770	1770	3540	100%
with optimization	1098	1098	2196	62.03%
with lazy relin.	762	762	1524	43.05%
with complex	380	351	731	20.65%

Table 12: Ablation study on CCMM