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 computation 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 (LN) 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., 2024) 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., 2024), 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.

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 Jiang et al.'s 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 $n \bmod d$): $\sigma(A)_{i,j} = A_{i,[i+j]_d}$, $\tau(A)_{i,j} = A_{[i+j]_d,j}$, $\phi(A)_{i,j} = A_{i,[j+1]_d}$, $\psi(A)_{i,j} = A_{[i+1]_d,j}$. 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., 2024) 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., 2021) 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., 2023) 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., 2024) 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.

3 Batch Method

We propose the Batch Method, the first approach that achieves both faster computation and preserves the accuracy of the original model, outperforming existing distillation-based methods.

3.1 Batch Power-Max Function

The Batch Power-Max Function is introduced as an efficient alternative to softmax, formulated as:

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

Previous methods, such as MPCFormer and SecFormer, applied distillation-based training using the 2Quad function $\frac{(x+c)^2}{\sum(x+c)^2}$, whereas PowerSoftmax utilized the power function $\frac{x^p}{\sum x^p}$ with standard training. The Batch Power-Max Function leverages both the c parameter and higher-order terms in the numerator, allowing it to approximate softmax closely without requiring additional nonlinear function computations. In all implementations and experiments, the parameters were set as $p = 5$, $c = 5$.

For a mini-batch size B, the input tensor has dimensions $B \times h \times L \times L$. Similar to standard softmax, the function first sums over the last L-dimension, obtaining a tensor of shape $B \times h \times L \times 1$. Then, the maximum over the B-dimension is computed to form a denominator with dimensions $1 \times h \times L \times 1$. By distilling the model using this denominator as a reference within each mini-batch, the model inherently learns a normalization effect, ensuring consistent scaling across different inputs.

During inference, a different approach is taken to further optimize computation. The final denominator from the last step of training is stored as the running denominator (rd_p). Instead of recomputing the denominator during inference, the precomputed rd_p is directly substituted in the denominator, simplifying the function to $\frac{(x+c)^p}{\mathsf{rd}_p}$. This modification effectively eliminates division operations in inference, replacing them with a single plaintext multiplication, significantly improving computational efficiency while maintaining the accuracy of softmax.

3.2 Batch Layer Normalization

Batch LN is introduced as an alternative to standard LN, formulated as:

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

The input data has dimensions of $B \times L \times d_m$. Similar to standard LN, the function first computes the mean and variance along the last d_m-dimension, resulting in a tensor of shape $B \times L \times 1$. Then, the maximum over the B-dimension is computed to form a denominator with dimensions $1 \times L \times 1$.

During inference, the process follows the same approach as the Batch Power-Max Function. The final denominator from the last step of training is stored as the running denominator (rd_ℓ). Instead of recomputing the denominator during inference, the precomputed rd is directly substituted, simplifying the function to: $\gamma \cdot \frac{x-\mu}{R_d} + \beta$ This approach effectively reduces computational overhead by eliminating division operations in inference, replacing them with a single plaintext multiplication while maintaining the functionality of LN.

3.3 Training Method

Previous research demonstrated a distillation method that effectively captures the characteristics of the BERT model. In this approach, a pretrained 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. For the final prediction layer (4), soft cross-entropy is used as the loss function for classification tasks, while mean squared error (MSE) is applied for regression tasks. The remaining three positions (1, 2, and 3) all use MSE loss. The training process is structured in two stages: in the first stage, the losses from the embedding layer (1), attention matrix (2), and hidden states (3) are minimized to mimic the intermediate outputs of the teacher model. In the second stage, the final prediction layer (4) is optimized to achieve high accuracy.

However, when the proposed batch method is combined with existing distillation techniques, an issue arises where the range of intermediate tensor values becomes unstable. In a standard plaintext environment, computational limitations do not constrain operations, meaning that even if feature explosion occurs—causing the intermediate tensor values to grow excessively—the final accuracy may remain unaffected. Additionally, even if some data points experience extreme value growth that renders computations infeasible, the model may still be selected as optimal based on accuracy if other data points are processed correctly.

In a HE environment, however, this characteristic poses a critical challenge. When intermediate tensor values grow too large in an HE model, errors in the bootstrapping process amplify, the accuracy of nonlinear function approximations degrades, and

computations become infeasible beyond a certain threshold. Therefore, maintaining a stable range for intermediate tensor values is essential in HE-based models. To address this issue, this study modifies the distillation method from previous research in three key ways.

One-step Distillation Training. Instead of separately optimizing each loss function in a two-step process, we integrate all losses into a single unified loss function and apply a one-step distillation training approach. In the conventional method, while minimizing the losses from (1), (2), and (3) in the first step helped stabilize intermediate tensors, the subsequent optimization of the final prediction layer (4) in the second step could lead to weight adjustments that caused feature explosion. By training all four losses simultaneously, our approach prevents drastic fluctuations in intermediate tensor values during training, ensuring greater stability throughout the process.

Additional Loss Function To further regulate intermediate tensor values, we introduce an additional loss function at a new position—right before the scaling and shifting operations in Batch LN, after the denominator has been computed. Batch LN is one of the primary causes of feature explosion since it processes the residual connection values obtained by summing the outputs of preceding attention or feed-forward layers. Given that Batch LN is applied twice per layer across 12 layers, it is executed a total of 24 times, amplifying the cumulative effect of small variations in tensor values. To mitigate this issue, we incorporate an additional loss term at this critical point, ensuring that intermediate tensor values remain stable throughout training. This loss is optimized using MSE loss and is learned within the same one-step distillation process, further enhancing the overall stability of the model.

Additional Parameter l A new adjustment parameter, l, was introduced in the denominator of Batch LN:

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

The parameter l is set to a value greater than 1 but close to 1, helping to mitigate the risk of feature explosion by adjusting the output of Batch LN. A larger l applies stronger regulation, but if set too high, it can hinder learning and negatively impact final model performance. Therefore, in this study, l was fine-tuned within the range of 1.0 to 1.5 to determine the optimal value. The impact of the proposed batch method and the effect of the adjustment parameter l on feature explosion are further analyzed in Appendix D.

4 Minimax Composition for Pseudo-Sign Function

Lee et al. (Lee et al., 2021) 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* $\mathrm{erf}(x) = \frac{2}{\pi}\int_0^x e^{-t^2} dt$ *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 $p_g \circ p_{\text{com}} = p_g \circ p_{t-1} \circ \cdots \circ p_0$ provides a high-accuracy approximation of the pseudo-sign function $f(x)$.

Algorithm 1: MiniCompPseudoSign(f, δ)

Input: A pseudo-sign function f with domain $[-1, 1]$, minimax approximation bound δ

Output: Polynomials $p_0, \cdots, p_{n-1}$ such that $\|p_{n-1} \circ \cdots \circ p_0 - f\|_{\infty,[-1,1]} \leq \delta$

1. Identify $\gamma \in (0, 1)$ such that $f(\gamma) = 1 - \delta/2$.
2. Compute $p_0, \cdots, p_{t-2}, \bar{p}_{t-1}$ using $\{p_0, \cdots, p_{t-2}, \bar{p}_{t-1}\} \leftarrow \text{MiniCompSign}\left(\gamma, \frac{\delta/4}{1-\delta/4}\right)$,
3. Update p_{t-1} by scaling: $p_{t-1} \leftarrow \left(1 - \frac{\delta}{4}\right) \bar{p}_{t-1}$.
4. Define γ' 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 $\tilde{p}_{\text{scale}}(x)$.
5. Define $g(x) = f \circ \tilde{p}^{-1}_{\text{scale}}(x)$ on $[-1, 1]$. Use Remez algorithm to approximate g with a minimax polynomial p_g:
$$p_g \leftarrow \text{Remez}(g, \delta/2).$$
6. 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 MiniCompPseudoSign algorithm in Algorithm 1 through the following theorem, which is specified and proven in Appendix B.

Theorem 4.1. *Let f be a pseudo-sign function and $0 < \delta < 1$. Assume that there exists $0 < \gamma < 1$ 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., 2023) 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 $\{m_i \in \mathbb{R}^n\}_{1\le i\le \frac{d_1 d_2}{n}}$ 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, $\{ct_i\}_{1\le i\le \mathsf{mid}}$, and ensure that the columns of the output matrix are packed into $\mathsf{ed} = \frac{d_1 d_3}{n}$ ciphertexts, $\{ct'_i\}_{1\le i\le \mathsf{ed}}$. 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.

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

for $1 \le \ell \le \mathsf{ed}$, 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 $N_1 N_2 = \frac{n}{d_1}$.

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

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 ciphertext-ciphertext matrix multiplication (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:

$$R_i[j] = \begin{cases} 1\colon [j]_4 \le 4 - i \\ 0\colon else \end{cases} \quad L_i[j] = \begin{cases} 1\colon [j]_4 < i \\ 0\colon else. \end{cases}$$

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

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

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, which refers to the total count of non-scalar multiplications (relinearizations), rotations, conjugations, and other operations requiring a key-switch. In this table, the number of key-switches is based on the BERT-base model that we target. Our algorithm requires $32\% \sim 36\%$ fewer key-switches for CPMM and 22% fewer key-switches for CCMM compared to THOR, a state-of-the-art technique.

6 Experiment Results

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.

Table 1: Comparison of key-switch counts in different homomorphic matrix multiplication methods.

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**

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. All experiments were conducted on a single NVIDIA A100 GPU.

6.1 Plaintext Results

	Baseline	**Distill**	**Loss**
RTE	71.48	73.29	6.55
MRPC	85.54	87.25	5.72
SST-2	92.66	92.20	4.22

Table 2: Fine-tuned distillation results on downstream tasks.

Table 2 presents the results of knowledge distillation training. The *Baseline* represents the performance of the original BERT model (teacher model) fine-tuned for downstream tasks, while *Distill* refers to the performance of the model after completing distillation training with Batch PowerMax and Batch LN applied. Each task was evaluated using three different random seeds, and the highest recorded performance among them is reported. Additionally, the *Loss* value indicates the loss of the *Distill* model that achieved the reported performance. Experimental results show that the proposed distilled model achieved an average accuracy improvement of 1.02% compared to the original model while maintaining the *Loss* value

within a stable range. This indicates that the distilled model preserves the expressiveness of the baseline model without encountering the feature explosion problem while maintaining a HE-friendly structure. Detailed experimental results for each task can be found in the Appendix, where the complete set of results is provided.

6.2 Ciphertext Results

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

Table 3: End-to-end inference results for the RTE task using an encrypted model.

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
LN2	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 without Pooler & Classification	344.32	623.39	279.07

Table 4: Breakdown of the execution time(sec) compared to THOR.

Table 3 presents the results of the HE experiment conducted on the distilled model from Table 2. *Plaintext* represents the performance of the *Distill* model evaluated in a plaintext environment, while *Ciphertext* represents the performance of the Powerformer model evaluated in an encrypted environment. The Powerformer model is an end-to-end HE model that incorporates all the proposed techniques in this paper. Experiments were conducted using the model that achieved the highest accuracy on the RTE task, and the results confirmed that there was no performance difference between the plaintext and ciphertext environments. Additionally, *Max Diff* represents the maximum difference between the output values in both environments, recording an extremely small value of 0.019. This demonstrates that the Powerformer model can robustly handle various nonlinear functions, suggesting its potential for high performance not only in classification tasks but also in regression problems.

Table 4 presents the performance breakdown, analyzing computation time for each layer in HE experiment from Table 3. Matrix operation time varies significantly depending on the computation level, making it difficult to assess performance improvements solely from this table. For example, FC1 and FC2 perform identical operations, but FC2 runs at a lower level, making it 15.44 seconds faster. Despite its higher computational workload, FC2 also runs 13.88 seconds faster than the Attention Layer due to its lower execution level. This suggests that execution time is influenced more by computation level than workload—lower levels speed up processing but require additional bootstrapping.

The proposed model focuses on minimizing bootstrapping, which results in computations being performed at relatively higher levels. Consequently, the reduction in per-layer computation time may appear minor compared to THOR. However, as shown in Table 1, key switching count comparisons confirm a significant decrease in overall computational workload. We applied nonlinear function replacement techniques, including Batch PowerMax, Batch LN, and the minimax composition of GELU and tanh. These not only reduced per-layer computation time but also significantly lowered bootstrapping overhead. Ultimately, the model reduces both bootstrapping frequency and required computation levels. This led to a 70% reduction in bootstrapping time, which previously accounted for over half of THOR's total computation, and an overall computation time reduction of about 45%.

7 Conclusion

We proposed *Powerformer*, an efficient HE-based PPLM designed to reduce computation 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 reduced computation time significantly compared to the leading HE-based PPLM while maintaining the same level of accuracy.

Limitations

This model assumes a semi-honest security model, meaning that both the client and server follow the agreed-upon protocol. 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, an MPC-based PPLM model would be required instead, 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. Notably, 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.

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: 25th International Conference, Calgary, AB, Canada, August 15–17, 2018, Revised Selected Papers 25*, pages 347–368. Springer.

Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2017. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, 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 on ACM SIGSAC Conference on Computer and Communications Security*, 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*, pages 1209–1222.

Eunsang Lee, Joon-Woo Lee, Jong-Seon No, and Young-Sik Kim. 2021. Minimax approximation of sign function by composite polynomial for homomorphic comparison. *IEEE Transactions on Dependable and Secure Computing*, 19(6):3711–3727.

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*.

Wen-jie Lu, Zhicong Huang, Zhen Gu, Jingyu Li, Jian Liu, Cheng Hong, Kui Ren, Tao Wei, and WenGuang Chen. 2023. Bumblebee: Secure two-party inference framework for large transformers. *Cryptology ePrint Archive*.

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, Dongwoo Yoo, Xiaoqian Jiang, and Miran Kim. 2024. Thor: Secure transformer inference with homomorphic encryption. *Cryptology ePrint Archive*.

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

Donghwan Rho, Taeseong Kim, Minje Park, Jung Woo Kim, Hyunsik Chae, Jung Hee Cheon, and Ernest K Ryu. 2024. Encryption-friendly llm architecture. *arXiv preprint arXiv:2410.02486*.

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. *Cryptology ePrint Archive*.

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*.

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, layer normalization, feed-forward network, and layer normalization.

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:

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

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, layer normalization 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, layer normalization 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 1 shows the architecture of one encoder block in the BERT-base model.

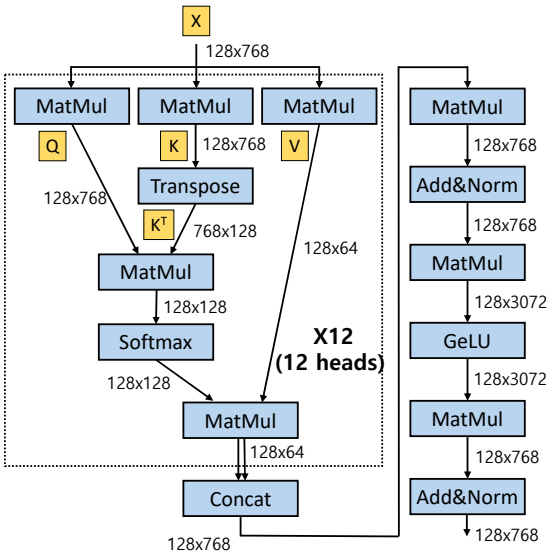

Figure 1: Overview of one encoder block of BERT-base Transformer architecture.

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 $v = (v_0, \ldots, v_{n-1}) \in \mathbb{C}^n$, 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 $(v_r, v_{r+1}, \ldots, v_{n-1}, v_0, \ldots, v_{r-1})$. If $\mathsf{ct1}$ 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: $[\sigma(A)]_C = \sum_{0 \le i < d} b_i \odot \rho([A]_C; di)$, $[\tau(A)]_C = \sum_{-d < i < d} a_i \odot \rho([A]_C; i)$, $[\phi^\ell(A)]_C = \rho([A]_C; d\ell)$, $[\psi^\ell(A)]_C = c_\ell \rho([A]_C; \ell) + c'_\ell \rho([A]_C; \ell - d)$. 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 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.

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'

B Details for Pseudo-Sign Composite Approximation

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

$$\|p_{t-1} \circ \cdots \circ p_0 - \mathsf{sign}\|_{\infty,[-1,-\gamma]\cup[\gamma,1]} \leq \frac{\delta/4}{1-\delta/4}.$$

We define a scaled composite polynomial $p_{\text{scale}}(x) = (1 - \frac{\delta}{4})p_{\text{com}}(x)$, which satisfies the following conditions:

$$\left\| p_{\text{scale}} - \left(1 - \frac{\delta}{4}\right) \cdot \mathsf{sign} \right\|_{\infty,[-1,-\gamma]\cup[\gamma,1]} \leq \frac{\delta/4}{1-\delta/4} \cdot \left(1 - \frac{\delta}{4}\right) = \frac{\delta}{4}$$

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 $g = f \circ \tilde{p}^{-1}_{\text{scale}} : [-1, 1] \to [-1, 1]$. 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 $0 \leq x \leq \gamma$, then $f(x) = g(p_{\text{scale}}(x))$ and $p_f(x) = p_g(p_{\text{scale}}(x))$ as $x \leq \gamma < \gamma'$. Since $p_{\text{scale}}(x) \in [-1, 1]$, it follows that

  $$|p_f(x) - f(x)| = |p_g(p_{\text{scale}}(x)) - g(p_{\text{scale}}(x))| \leq \delta/2 < \delta.$$

  Thus, $\|p_f - f\|_{\infty,[0,\gamma]} < \delta$ is satisfied.

- If $\gamma \leq x \leq 1$, then $\|p_{\text{scale}}(x) - (1 - \delta/4)\| \leq \delta/4$, which implies $1 - \delta/4 \leq p_{\text{scale}}(x) \leq 1$. Define $\tilde{x} = \tilde{p}^{-1}_{\text{scale}} \circ p_{\text{scale}}(x)$. By definition, we have $\gamma \leq \tilde{x} \leq x \leq 1$. Since $f(x)$ has a range within $[1 - \delta/2, 1]$ for $x \in [\gamma, 1]$, it follows that $|f(\tilde{x}) - f(x)| \leq \delta/2$. For $\gamma \leq \tilde{x} \leq \gamma'$, 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

  $$|p_f(x) - f(\tilde{x})| = |p_f(\tilde{x}) - f(\tilde{x})| = |p_g(p_{\text{scale}}(\tilde{x})) - g(p_{\text{scale}}(\tilde{x}))|.$$

  Since $p_{\text{scale}}(\tilde{x}) \in [-1, 1]$, it follows that

  $$|p_f(x) - f(\tilde{x})| = |p_g(p_{\text{scale}}(\tilde{x})) - g(p_{\text{scale}}(\tilde{x}))| \leq \delta/2.$$

  Finally, combining these results, we obtain

  $$|p_f(x) - f(x)| \leq |p_f(x) - f(\tilde{x})| + |f(\tilde{x}) - f(x)| \leq \delta/2 + \delta/2 = \delta.$$

  Thus, $\|p_f - f\|_{\infty,[\gamma,1]} \leq \delta$ 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

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

□

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

$\{ct_i\}_{1\le i\le \frac{d_1d_2}{n}}$, 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 $\{m^{(i)}\}_{1\le i\le \frac{d_1d_2}{n}}$ by $[A]^k_C$. When $d_1d_2 = n$, $[A]^k_C$ 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 2 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 $[AB]^k_C$ from the ciphertexts of $[A]^k_C$. The columns of the input matrix are packed into a total of $\mathsf{mid} = \frac{d_1d_2}{n}$ ciphertexts, $\{ct_i\}_{1\le i\le \mathsf{mid}}$, and the columns of the output matrix are packed into a total of $\mathsf{ed} = \frac{d_1d_3}{n}$ 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 $N_1N_2 = \frac{n}{d_1}$:

$$ct'_\ell = \sum_{0\le p<N_2} \mathsf{Rot}\Big(\sum_{1\le j\le \mathsf{mid}} \sum_{0\le q<N_1} \mathsf{Rot}(ct_j; qd_1) \odot \rho(d'^{j,\ell}_{pN_1+q}; -pN_1d_1); pN_1d_1\Big). \quad (7)$$

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 **do**
        $ct' \leftarrow ct' + ct_j^{(q)} \odot \rho(d'^{j,\ell}_{pN_1+q}; -pN_1d_1)$
      end
    end
    $ct'_\ell \leftarrow ct'_\ell + \mathsf{Rot}(ct'; pN_1d_1)$
  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.

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

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:

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

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 $\mathsf{mid} = 3$ and $\mathsf{ed} = 12$, 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 $2\sqrt{\frac{n}{L} \cdot \mathsf{mid} \cdot 6}$, which is $\sqrt{2}$ times fewer than the $2\sqrt{\frac{n}{L} \cdot \mathsf{mid} \cdot 12}$ required by Algorithm 5.

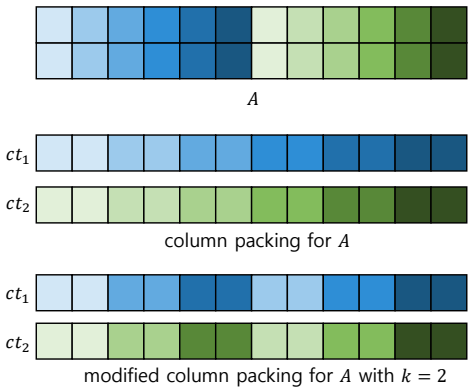

Figure 2: Column packing and modifid column packing with $k = 2$ for $d_1 = 2, d_2 = 12$, and $n = 12$.

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:

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

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

$$R_i[j] = \begin{cases} 1\colon [j]_4 \le 4 - i \\ 0\colon else \end{cases} \quad L_i[j] = \begin{cases} 1\colon [j]_4 < i \\ 0\colon else. \end{cases}$$

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 follow-

ing 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).

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'

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 $n = d_1 d_2$. 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 $[\tilde{\sigma}(A)]_C^k = \sum_{0\le i<k} b_i' \odot \rho([A]_C^k; \frac{d_1 d_2}{k} i)$ where each vector $b_i' \in \mathbb{R}^{d_1 d_2}$ is obtained by splitting $b_i \in \mathbb{R}^{k^2}$ into chunks of size k and repeating each chunk $\frac{d_1 d_2}{k^2}$ times. Similarly, just as τ for a $k \times k$ matrix is expressed as $[\tau(A)]_C = \sum_{-k<i<k} a_i \odot \rho([A]_C; i)$, $\tilde{\tau}$ for a $d_1 \times d_2$ matrix can be written as $[\tilde{\tau}(A)]_C^k = \sum_{-k<i<k} a_i' \odot \rho([A]_C^k; i)$. 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 $[\tilde{\psi}^i(A)]_C^k = \bar{c}_i \odot \rho([A]_C^k; i) + \bar{c}_i' \odot \rho([A]_C^k; i-k)$. 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 3 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 in Section 5.2.1, with the only difference being that $n = d_1 d_2$. Then, the following holds.

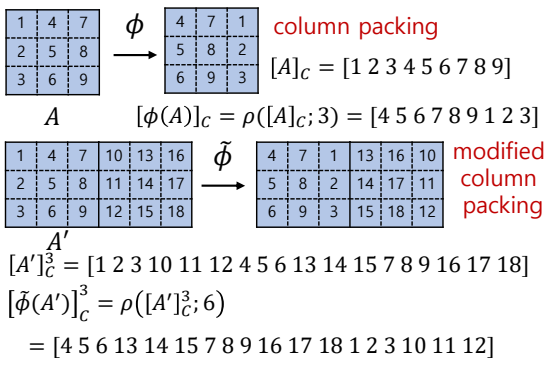

Figure 3: An algorithm for $\tilde{\phi}$ using modified column packing

$$
\begin{aligned}
&[\sum_{0\le\ell<d} \tilde{\phi}_\ell(A') \odot \tilde{\psi}_\ell(B')]^k_C \\
&= \sum_{0\le\ell<d} \rho([A']^k_C; \frac{d_1d_2}{k}\ell) \odot [\tilde{\psi}_\ell(B')]^k_C \qquad (11)\\
&= \sum_{0\le j<N_2} \sum_{0\le i<N_1} \rho([A']^k_C; \frac{d_1d_2}{k}(N_1j+i)) \\
&\quad \odot [\tilde{\psi}_{N_1i+j}(B')]^k_C \\
&= \sum_{0\le j<N_2} \rho(\sum_{0\le i<N_1} \rho([A']^k_C; \frac{d_1d_2}{k}i) \odot \\
&\quad \rho([\tilde{\psi}_{N_1j+i}(B')]^k_C; -\frac{d_1d_2}{k}N_1j); \frac{d_1d_2}{k}N_1j) \\
&= \sum_{0\le j<N_2} \rho(\sum_{0\le i<N_1} \rho([A']^k_C; \frac{d_1d_2}{k}i) \odot \\
&\rho([B']^k_C \odot R'_{N_1j+i} + \rho([B']^k_C; -k) \odot L'_{N_1j+i}; \\
&N_1j+i-\frac{d_1d_2}{k}N_1j); \frac{d_1d_2}{k}N_1j).
\end{aligned}
$$

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

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 $[A]^k_C$ and outputs the ciphertexts of $[A^{BT}]^k_C$.

Algorithm 7: BlockSigma

Input: ct
Output: ct'

$ct' \leftarrow ct_{zero}$
for $i \leftarrow 0$ **to** $N_1 - 1$ **do**
  $ct^{(i)} \leftarrow \mathsf{Rot}(ct; \frac{d_1d_2}{k}i)$
end
for $j \leftarrow 0$ **to** $N_2 - 1$ **do**
  $ct'' \leftarrow ct_{zero}$
  for $i \leftarrow 0$ **to** $N_1 - 1$ **do**
    $ct'' \leftarrow ct'' + \rho(b'_{N_1j+i}; -\frac{d_1d_2}{k}N_1j) \odot ct^{(i)}$
  end
  $ct' \leftarrow ct' + \mathsf{Rot}(ct''; \frac{d_1d_2}{k}N_1j)$
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**
    $ct'' \leftarrow ct'' + \rho(a'_{N_1j+i-k+1}; -N_1j + k - 1) \odot ct^{(i)}$
  end
  $ct' \leftarrow ct' + \mathsf{Rot}(ct''; N_1j - k + 1)$
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 $a + bi$ 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

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'

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'

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 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 Full Experiment Result

Table 5 presents the complete results of the distillation experiments. The *Baseline* represents the performance of the original BERT model (teacher model) fine-tuned for downstream tasks, while *Distill* refers to the performance of the model after undergoing distillation training with batch powermax and batch layer normalization applied. The *Loss* represents the loss value during the distillation process.

The occurrence of feature explosion can be identified by examining the loss value. At the loss computation points, the MSE between the distilled model's outputs and the teacher model's outputs is incorporated into the loss. If feature explosion occurs, this value increases excessively. In gen-

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

Algorithm 14: Optimized CCMM algorithm for multi-head attention

Input: $\{ct'_{qi}\}_{1\le i\le 3}$, $\{ct'_{ki}\}_{1\le i\le 3}$, and $\{ct'_{vi}\}_{1\le i\le 3}$
Output: ct_{qkv1}, ct_{qkv2}, and ct_{qkv3}

$\hat{ct}_{k2} \leftarrow ct'_{k2} + \mathsf{Multi}(ct'_{k3})$
$ct'_{k1} \leftarrow \mathsf{BlockTrans}(ct'_{k1})$
$\hat{ct}_{k2} \leftarrow \mathsf{BlockTrans}(\hat{ct}_{k2})$
$ct'_{k2}, ct'_{k3} \leftarrow \hat{ct}_{k2}$
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**
  $\hat{ct}_{qki} \leftarrow \mathsf{BlockMult}'(ct'_{qi}, \hat{ct}_{ki})$
  $ct_{qki,1}, ct_{qki,2} \leftarrow \mathsf{SplitPaste}(\hat{ct}_{qki})$
  $ct_{qki,1} \leftarrow \mathsf{Soft}(ct_{qki,1})$
  $ct_{qki,2} \leftarrow \mathsf{Soft}(ct_{qki,2})$
  $\hat{ct}_{qki} \leftarrow ct_{qki,1} + \mathsf{Multi}(ct_{qki,2})$
end
$ct'_{v1,1}, ct'_{v1,2} \leftarrow \mathsf{BlockTau}(ct'_{v1})$
$\hat{ct}_{v2} \leftarrow \mathsf{BlockTau}(ct'_{v2} + \mathsf{Multi}(ct'_{v3}))$
$ct'_{v2}, ct'_{v3} \leftarrow \mathsf{Extract}(\hat{ct}_{v2})$
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}

eral, loss values exceeding a single-digit number indicate the presence of feature explosion. Additionally, as the l value increases from 1.0 to 1.5, the frequency of feature explosion occurrences decreases, with counts of 6, 1, 2, 1, 1, and 0. This trend demonstrates the effectiveness of introducing the l parameter in mitigating feature explosion.

Table 6 presents the experimental results evaluating the effectiveness of GK attention. The experiment was conducted on the BERT-base model with a fixed batch size of 16 and a learning rate of 5e-5. *Softmax* represents the results obtained using the standard softmax function, while *GK* denotes the results when applying GK attention.

The results show that GK attention leads to an average performance degradation of 11.72% and a maximum degradation of 17.69%, indicating that it is not a suitable replacement for the standard softmax function. Notably, this performance decline is more pronounced in certain tasks, demonstrating the limitations of GK attention in fully replacing softmax's stable probability distribution generation capability.

	Seed	0				42				777				
Task	l	Baseline	Distill	Loss	Diff	Baseline	Distill	Loss	Diff	Baseline	Distill	Loss	Diff	Feature Explosion
RTE	1	71.48	70.40	INF	-1.08	69.31	68.95	4.62	-0.36	67.87	71.84	4.84	3.97	1
	1.1	71.48	72.20	6.87	0.72	69.31	68.95	4.94	-0.36	67.87	73.29	1150.08	5.42	1
	1.2	71.48	71.84	5.24	0.36	69.31	69.68	5.68	0.36	67.87	72.20	6.17	4.33	0
	1.3	71.48	71.48	6.06	0.00	69.31	67.51	6.86	-1.81	67.87	73.65	5.73	5.78	0
	1.4	71.48	72.92	6.28	1.44	69.31	67.87	7.49	-1.44	67.87	73.29	6.55	5.42	0
	1.5	71.48	72.56	7.15	1.08	69.31	68.95	7.79	-0.36	67.87	72.56	7.66	4.69	0
MRPC	1	83.82	85.78	INF	1.96	85.54	85.29	INF	-0.25	85.05	87.50	2.45	2.45	3
	1.1	83.82	85.29	5.02	1.47	85.54	85.29	4.65	-0.25	85.05	87.25	5.72	2.21	0
	1.2	83.82	85.29	5.05	1.47	85.54	85.29	3487813	0.00	85.05	86.76	6.54	1.72	1
	1.3	83.82	85.78	5.53	1.96	85.54	85.78	5.83	-0.49	85.05	86.27	5.95	1.23	0
	1.4	83.82	85.29	6.96	1.47	85.54	85.29	6.74	-0.25	85.05	86.52	6.84	1.47	0
	1.5	83.82	85.05	7.89	1.23	85.54	85.05	7.70	-0.49	85.05	85.05	8.15	0.00	0
SST-2	1	92.66	91.97	3.16	-0.69	91.63	91.97	INF	0.00	91.97	91.97	INF	0.00	2
	1.1	92.66	92.09	3.47	-0.57	91.63	92.09	2.88	0.00	91.97	92.09	3.73	0.11	0
	1.2	92.66	92.20	2038.45	-0.46	91.63	92.20	3.31	0.00	91.97	92.20	4.22	0.23	1
	1.3	92.66	91.86	4.55	-0.80	91.63	91.86	INF	-0.11	91.97	91.86	4.34	-0.11	1
	1.4	92.66	91.86	INF	-0.80	91.63	91.86	5.36	-0.11	91.97	91.86	5.49	0.11	1
	1.5	92.66	91.86	6.36	-0.80	91.63	91.74	7.71	-0.23	91.97	91.74	7.71	-0.23	0

Table 5: Full distillation learning results across different seeds, tasks, and l parameters.

seed	0			42			777		
	Softmax	GK	Diff	Softmax	GK	Diff	Softmax	GK	Diff
RTE	70.76	53.07	-17.69	67.15	53.07	-14.08	67.15	52.71	-14.44
MRPC	85.29	70.59	-14.71	86.52	73.04	-13.48	86.03	69.61	-16.42
SST-2	91.06	84.86	-6.19	90.14	85.44	-4.70	90.14	86.35	-3.78

Table 6: Experimental results across different seeds