# STIP: Efficient and Secure Non-Interactive Transformer Inference via Compact Packing

Zihao Wang[1], Rongmao Chen[1], Xinwen Gao[1], Yi Wang[1]
Lin Liu[1], Zixin Lan[1], Zhaoyu Wang[2], Shaojing Fu[1]
Qiong Wang[1], Xinyi Huang[3]
[1]National University of Defense Technology
[2]Shanghai Jiao Tong University
[3]Nanjing University of Aeronautics and Astronautics

**Abstract**

Secure Transformer Inference (STFI) for LLMs aims to protect user inputs and model parameters. Fully Homomorphic Encryption (FHE) enables non-interactive STFI without intermediate communication, but its computational overhead remains substantial. Advancements have accelerated inference by optimizing packing strategies and reducing rotations. However, several challenges persist, including excessive rotations in ciphertext-ciphertext matrix multiplications (CCMMs), low input/output projection throughput, and expensive maximum/inverse operations, as well as wasted storage slots and inflated ciphertext counts due to sparse packing. Therefore, we propose STIP, an efficient and secure non-interactive transformer inference framework incorporating three novel packing strategies: (1) Real-Imaginary Hybrid Packing (RIHP) halves the rotation costs of CCMMs by enabling the simultaneous computation of two outputs within the real and imaginary components; (2) Double-Head Packing (DHP) maps adjacent heads to the real and imaginary components, doubling the throughput of attention projections; and (3) Adaptive Multi-Column Packing (AMCP) packs multiple feature columns into a single ciphertext, maximizing slot occupancy to reduce the total ciphertext count and enhance computational parallelism. For nonlinear layers, we employ the Gaussian Kernel instead of Softmax, eliminating the need for maximum and inverse operations, supported by a column-packed RIHP-based L2-norm algorithm. We reformulate the first Layer Normalization into an inverse-free form by exploiting scale-invariance. Compared with Euston (S&P' 26), STIP requires 20.6× fewer rotations and achieves a 34.7× attention-layer speedup. End-to-end experiments on BERT-base show that STIP achieves an 8.5× speedup over Euston at batch size $t = 32$, and 2.3×, 4.8×, and 2.7× over MOAI (ICLR' 26), THOR (CCS' 25), and Powerformer (ACL' 25) at $t = 256$, respectively.

## 1 Introduction

Modern transformer-based Large Language Models (LLMs) [1], such as BERT [2], GPT [3], and LLaMA [4], have emerged as the cornerstone of modern artificial intelligence, powering applications from medical diagnosis to financial risk assessment. Platforms like OpenAI's ChatGPT and Google's Gemini deploy models via cloud APIs to deliver scalable services. Consequently, users rely on these centralized systems to process highly sensitive data [5].

Unfortunately, this cloud-centric model introduces significant privacy risks for both parties [6, 7]. For clients, transmitting sensitive data, such as medical notes or financial records in plaintext, exposes users to the risk of unauthorized access and data leakage [8]. For servers,

proprietary model parameters require protection against disclosure beyond what inference outputs reveal [9].

Secure transformer inference (STFI) has emerged as a promising paradigm to protect the data privacy of both clients and servers [10]. Such a two-party cryptographic protocol must ensure that the client's input remains confidential from the server while the server's model parameters are protected from the client. Early research on STFI focused on interactive protocols, typically employing Multi-Party Computation (MPC), Oblivious Transfer (OT), Secret Sharing (SS), or hybrid protocols. However, state-of-the-art (SOTA) interactive protocols, such as BOLT [11], SIGMA [12], BumbleBee [13], BLB [14], and SHAFT [15], require transmitting dozens of GB of data between the client and server. This massive communication overhead makes interactive inference difficult to scale in practice. Furthermore, frequent interactions introduce synchronization delays that prevent full exploitation of hardware parallelism.

Consequently, recent works including NEXUS (NDSS' 25) [16], Powerformer (ACL' 25) [17], THOR (CCS' 25) [18], MOAI (ICLR' 26) [19], and Euston (S&P' 26) [20] have pivoted toward non-interactive inference based on Fully Homomorphic Encryption (FHE). In this paradigm, the client sends an encrypted request and receives an encrypted response. By computing on encrypted data without intermediate communication, these methods enable lower network latency [21] and better hardware acceleration [22].

Currently, MOAI and Euston achieve rapid non-interactive batch inference [23]. By adopting a column-wise packing strategy (packing each column of the input matrix into a distinct ciphertext), they eliminate rotations in both nonlinear layers and plaintext-ciphertext matrix multiplications (PCMMs), thereby achieving rapid inference speeds. However, this layout imposes severe penalties on other critical components.

① **Rotation-intensive CCMMs.** The attention layer involves ciphertext-ciphertext matrix multiplications (CCMMs) between encrypted intermediate matrices. Under Single Instruction Multiple Data (SIMD) packing, CCMMs require rotations to align ciphertext slots before element-wise multiplication and accumulation, and each rotation incurs an expensive key-switching operation. Euston computes the CCMM between the matrices $\boldsymbol{Q} \in \mathbb{R}^{m \times d'}$ and $\boldsymbol{K}^\top \in \mathbb{R}^{d' \times m}$ requiring $2(m-1)d'$ rotations. By adopting the interleaved packing from [24, 25], MOAI requires $(m-1)d'$ rotations. For the CCMM of the activation output $\boldsymbol{S} \in \mathbb{R}^{m \times m}$ with $\boldsymbol{V} \in \mathbb{R}^{m \times d'}$, Euston incurs $2(m-1)d'$ rotations, whereas MOAI requires $2\sqrt{m}d' + m$ rotations by adopting [26]. Collectively, these CCMM-induced rotations dominate the total number of rotations in both schemes, thereby constituting the primary bottleneck.

② **Throughput bottlenecks in linear projections.** The input and output projections within the attention layer are computation-intensive PCMMs. Current schemes perform these projections sequentially for each attention head. This serial execution introduces a linear dependency on the number of heads, constraining the projection throughput and causing inference latency to scale with model complexity.

③ **Severe slot underutilization and excessive ciphertexts at small batch sizes.** Efficient inference at modest batch sizes, such as $t = 32$ evaluated by both NEXUS and Euston, is important when clients have limited inputs available or need to bound the waiting time for batch formation [27, 28]. However, the single-column packing adopted by MOAI and Euston relies on large batches to fully exploit SIMD parallelism. For BERT-base with sequence length $m = 128$ and $2^{15}$ slots per ciphertext, $t = 32$ utilizes only 12.5% of the slots. At fixed parameters, homomorphic operations process entire ciphertexts regardless of slot occupancy. Consequently, smaller batches process fewer inputs at the same computational cost, reducing amortized throughput. The sparse layout also requires 2304 ciphertexts to represent the $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$ matrices at $t = 32$ (compared with 384 in STIP), imposing a substantial burden on GPU memory. This dependence on large

batches limits the practicality of sparse packing for workloads with few available inputs or tight limits on batch-formation delay.

④ **Limited-scalability Softmax and high-cost Layer Normalization.** The column-wise packing inherently fragments the elements of an attention row across multiple distinct ciphertexts. Unlike row-wise schemes that can compute the row maximum to stabilize Softmax, MOAI and Euston are forced to substitute it with a predetermined constant. This static approach often leads to exponential overflow or underflow, thereby distorting the attention distribution and pushing values outside the convergence interval of homomorphic inverse approximations. Furthermore, their standard Layer Normalization (LN) and Softmax formulations necessitate explicit inverse computations. This operation is computationally expensive, consuming substantial multiplicative depths and requiring extensive key-switching operations.

### 1.1 Our Contributions

In this paper, we present **STIP**, an efficient and secure non-interactive transformer inference via compact packing. Built upon the residue number system (RNS) variant of Cheon-Kim-Kim-Song (CKKS) scheme [29, 30], STIP addresses the bottlenecks of computation and memory of SOTA schemes.

First, we unlock complex-domain parallelism via imaginary-slot multiplexing. We propose two *Real-Imaginary Hybrid Packing (RIHP)* strategies to reduce the rotation cost of CCMMs ($\boldsymbol{QK}^\top, \boldsymbol{SV}$). The first strategy exploits the diagonal structure of attention scores by encoding the key matrix ($\boldsymbol{K}$) to simultaneously compute two output diagonals in the real and imaginary components. The second strategy leverages feature dimension folding by encoding query ($\boldsymbol{Q}$) and key ($\boldsymbol{K}$) matrices, enabling concurrent feature-product accumulation, while similarly folding $\boldsymbol{V}$ features for parallel $\boldsymbol{SV}$ computation. We introduce *Double-Head Packing (DHP)* for input and output projections, which maps adjacent attention heads to the real and imaginary components for simultaneous processing and doubles their throughput. Second, to resolve slot underutilization in column-wise packing, we introduce *Adaptive Multi-Column Packing (AMCP)*. It compacts multiple feature columns into a single ciphertext, ensuring high slot occupancy even at low batch sizes. Third, we redesign nonlinear functions for stability and efficiency. We replace Softmax with a Gaussian Kernel (GK), utilizing the second RIHP strategy to halve the rotation overhead in L2-norm computation. Furthermore, we transform the first LN into an inverse-free variant, exploiting scale-invariance to absorb the scaling factor into the second standard LN and eliminate high-depth inverse operations. Our contributions are as follows:

① **Compact packing strategies.** RIHP and DHP exploit complex-domain parallelism. RIHP reduces the rotation overhead for CCMMs ($\boldsymbol{QK}^\top, \boldsymbol{SV}$) by approximately 50%, and DHP doubles the throughput of input and output projections. AMCP improves slot occupancy and reduces ciphertext counts by packing multiple feature columns together, supporting parallel computation across the transformer.

② **Efficient and stable nonlinear layers.** We eliminate the dependence of nonlinear layers on maximum and inverse evaluations, which minimizes multiplicative depth and key-switching operations. We substitute Softmax with the GK attention and introduce a column-packed L2-norm algorithm tailored for column-packed matrices. Our second RIHP strategy further reduces the rotation overhead of the L2-norm computation by approximately 50%. We propose an inverse-free LN by exploiting scale-invariance.

③ **Enhanced inference efficiency.** Compared with Euston, STIP requires 20.6× fewer rotations and achieves a 34.7× attention-layer speedup. End-to-end GPU evaluations on BERT-base demonstrate that STIP outperforms Euston by 8.5× at batch size $t = 32$, and MOAI, THOR, and Powerformer by 2.3×, 4.8×, and 2.7× at $t = 256$, respectively.

## 2 Preliminaries

**Table 1:** Frequently used notation.

| Notation | Description |
|---|---|
| $t$ | batch size |
| $m, d, d'$ | dimensions |
| $H$ | the number of attention heads |
| $q$ | modulus |
| $N$ | polynomial degree |
| $S$ | SIMD slots |
| $\boxplus$ | homomorphic addition |
| $\boxtimes$ | homomorphic multiplication |
| $\boxminus$ | homomorphic subtraction |
| Conj | homomorphic conjugate |
| RotL/RotR | left rotation/right rotation |
| $\boldsymbol{x} = (x_1, x_2, ..., x_m)$ | vector |
| $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_m]$ | matrix |
| $[\![\boldsymbol{x}]\!]$ | ciphertext |

### 2.1 Transformer Architecture

A transformer layer processes an input embedding matrix sequentially through an attention layer and a feed-forward network (FFN) [1]. In multi-head attention, the input is projected into several attention heads, whose outputs are then concatenated and linearly transformed. Separate LN operations are applied around both the attention layer and the FFN. After these stages, the model produces a logits vector through a final output projection, and the final prediction is obtained by selecting the entry with the highest value.

**Attention Layer.** For an input embedding matrix $\boldsymbol{X} \in \mathbb{R}^{m \times d}$ (for a batch size of $t$, $\boldsymbol{X} \in \mathbb{R}^{tm \times d}$), it is multiplied with the three projection matrices $\boldsymbol{W}_Q \in \mathbb{R}^{d \times d}$, $\boldsymbol{W}_K \in \mathbb{R}^{d \times d}$, and $\boldsymbol{W}_V \in \mathbb{R}^{d \times d}$ to obtain a query matrix $\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}_Q$, a key matrix $\boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}_K$, and a value matrix $\boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}_V$. Under multi-head attention, the query, key, and value matrices are partitioned into $H$ heads. For each head $h \in [H]$, $\boldsymbol{Q}^{(h)}, \boldsymbol{K}^{(h)}, \boldsymbol{V}^{(h)} \in \mathbb{R}^{m \times d'}$, where $d' = d/H$.

The attention output of head $h$ is computed as:

$$\text{Atten}^{(h)} = \text{Softmax}(\boldsymbol{Q}^{(h)}(\boldsymbol{K}^{(h)})^\top / \sqrt{d'})\boldsymbol{V}^{(h)}. \tag{1}$$

The GK attention is calculated using [31, 32]:

$$\begin{aligned} &\text{Atten}(\boldsymbol{Q}^{(h)}, \boldsymbol{K}^{(h)}, \boldsymbol{V}^{(h)}) = S(\boldsymbol{Q}^{(h)}, \boldsymbol{K}^{(h)})\boldsymbol{V}^{(h)}, \\ &S(\boldsymbol{Q}^{(h)}, \boldsymbol{K}^{(h)})_{ij} = \exp(-\frac{1}{2\sqrt{d'}}\|\boldsymbol{Q}^{(h)}_{i,:} - \boldsymbol{K}^{(h)}_{j,:}\|_2^2), i, j \in [m], \end{aligned} \tag{2}$$

where $\|\cdot\|_2$ is L2-norm, $\boldsymbol{Q}_{i,:}$ and $\boldsymbol{K}_{j,:}$ mean the $i$-th and $j$-th row of $\boldsymbol{Q}$ and $\boldsymbol{K}$, and exp is the exponential operation.

**LN.** For an input $\boldsymbol{x} \in \mathbb{R}^n$, LN computes the mean $\mu$ and standard deviation $\sigma$.

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i, \quad \sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2}. \tag{3}$$

The normalized output $\boldsymbol{y}$ is computed by scaling with a learnable parameter $\boldsymbol{\gamma} \in \mathbb{R}^n$ and shifting with a learnable parameter $\boldsymbol{\beta} \in \mathbb{R}^n$.

$$\boldsymbol{y} = \boldsymbol{\gamma} \cdot (\boldsymbol{x} - \mu)/\sigma + \boldsymbol{\beta}. \tag{4}$$

**Feed-Forward Network.** An FFN consists of two linear transformations with an activation function. To enable scale propagation within FFN, the ReLU activation is employed.

$$\mathrm{FFN}(\boldsymbol{X}) = \mathrm{ReLU}(\boldsymbol{X}\boldsymbol{W}_1 + \boldsymbol{b}_1)\boldsymbol{W}_2 + \boldsymbol{b}_2. \tag{5}$$

### 2.2 Fully Homomorphic Encryption

RNS-CKKS [30] enables approximate homomorphic computation over the polynomial ring $R_Q = \mathbb{Z}_Q[x]/(x^N + 1)$, where $N$ is a power of two, $Q = \prod_{i=0}^{L} q_i$ is the product of distinct primes $q_i$, and $L$ is the multiplicative depth. In this leveled scheme, a ciphertext at level $k$ is associated with modulus $Q_k = \prod_{i=0}^{k} q_i$. The noise is managed via rescaling, which reduces the modulus by the last prime $q_i$. For deep circuits, bootstrapping consumes $K$ levels to refresh the ciphertext, leaving $L - K$ levels available for subsequent operations. RNS-CKKS supports SIMD by packing vectors via the mapping $\mathbb{C}^{N/2} \to R_Q$, enabling parallel arithmetic on $S = N/2$ slots.

We summarize below the homomorphic operations employed in this paper:

**Homomorphic Addition.** Homomorphic addition allows the computation of the sum of two encrypted vectors. Given two ciphertexts $[\![\boldsymbol{a}]\!]$ and $[\![\boldsymbol{b}]\!]$, homomorphic addition is $[\![\boldsymbol{a}]\!] \boxplus [\![\boldsymbol{b}]\!]$.

**Homomorphic Multiplication.** Homomorphic multiplication computes the element-wise product of two encrypted vectors. Given two ciphertexts $[\![\boldsymbol{a}]\!]$ and $[\![\boldsymbol{b}]\!]$, homomorphic multiplication is $[\![\boldsymbol{a}]\!] \boxtimes [\![\boldsymbol{b}]\!]$. It consumes one level of multiplicative depth, requiring rescaling to manage the ciphertext scale and a relinearization step to recover the ciphertext size.

**Homomorphic Rotation.** The rotation is a key primitive for summing elements, which is computationally expensive because it involves the key-switching operation.

① Left rotation: $\textsc{RotL}_k([\![\boldsymbol{a}]\!])$ shifts the vector $[\![\boldsymbol{a}]\!]$ to the left by $k$ positions. If $[\![\boldsymbol{a}]\!] = (a_0, a_1, \ldots, a_{S-1})$, the result corresponds to an encryption of $(a_k, \ldots, a_{S-1}, a_0, \ldots, a_{k-1})$.

② Right rotation: $\textsc{RotR}_k([\![\boldsymbol{a}]\!])$ shifts the vector $[\![\boldsymbol{a}]\!]$ to the right by $k$ positions.

**Homomorphic Conjugate.** Given a ciphertext $[\![\boldsymbol{a}]\!]$ encrypting a complex vector $\boldsymbol{a} = \boldsymbol{x} + i\boldsymbol{y}$, the homomorphic conjugate computes the complex conjugate of each element, denoted as $\textsc{Conj}([\![\boldsymbol{a}]\!]) = [\![\boldsymbol{x} - i\boldsymbol{y}]\!]$. This operation is computationally expensive as it requires key-switching.

**Homomorphic Sign Function.** For the ReLU evaluation, we adopt the approximate homomorphic ReLU from [33]. The sign function is approximated by a composition of two minimax polynomials [34, 35]:

$$\textsc{Sgn}(x) = p_2 \left( p_1(x) \right), \tag{6}$$

where $\deg(p_1, p_2) = \{7, 15\}$ and the precision parameter is $\alpha = 8$. The approximation is evaluated over $[-1, -2^{-\alpha}] \cup [2^{-\alpha}, 1]$, and the input is scaled to $[-1, 1]$.

## 3 Overview

### 3.1 System Model and Threat Model

Following prior works [16, 17, 18, 19, 20], we consider a transformer inference scenario where a client transmits encrypted input $\boldsymbol{X} \in \mathbb{R}^{tm \times d}$ to a server, and the server generates responses

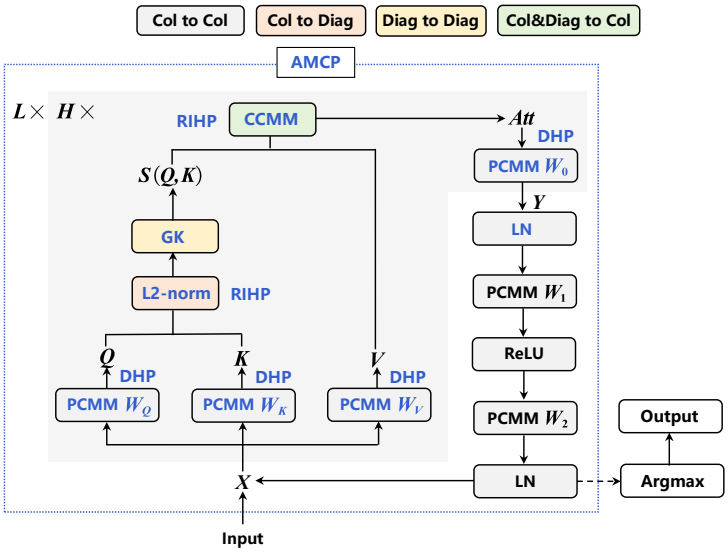



**Figure 1:** A simple workflow of STIP. The improvements made by STIP are highlighted in blue.

$\mathcal{M}(\boldsymbol{X})$ with model parameter $\mathcal{M}$. We assume that both the client and the server are semi-honest adversaries who adhere to the protocol specifications while seeking additional information about the transformer model or the user's data. Model extraction [36] against a plaintext model through repeated output queries is outside our scope.

### 3.2 Workflow

As depicted in Figure 1, STIP orchestrates a dynamic transition of packing methods between layers.

**Input Projection (Col→Col).** The workflow initiates with the column-packed (Col) input $\boldsymbol{X}$. We employ PCMM operations with weight matrices ($\boldsymbol{W}_{Q,K,V}$) to generate query, key, and value matrices while maintaining the column layout. To accelerate this phase, DHP pairs adjacent heads into real and imaginary components, effectively doubling the throughput (cf. Section 4.2).

**Attention Mechanism (Col→Diag→Diag).** The workflow transitions to a diagonal-packed (Diag) layout to efficiently handle the attention mechanism. We design a column-packed L2-norm algorithm that converts $\boldsymbol{Q}$ and $\boldsymbol{K}$ from column to diagonal packing, utilizing the RIHP to halve the number of rotations (cf. Section 4.1 and 5.1). Subsequently, the output of the L2-norm computation is processed by a GK function. This rotation-free GK layer yields the diagonal-packed attention score matrix $\boldsymbol{S}(\boldsymbol{Q}, \boldsymbol{K})$ (cf. Section 5.1).

**Attention Output (Col&Diag→Col).** This stage generates the column-packed attention $\boldsymbol{Att}$ by aggregating the diagonal-packed scores $\boldsymbol{S}(\boldsymbol{Q}, \boldsymbol{K})$ with the column-packed value matrix $\boldsymbol{V}$. The RIHP strategy and Baby-Step Giant-Step (BSGS) algorithm are employed to minimize rotation overheads in this CCMM (cf. Appendix A).

**Output Projection and Feed-Forward Network (Col → Col).** The final stage processes the attention $\boldsymbol{Att}$ through the output projection and FFN via PCMMs ($\boldsymbol{W}_{0,1,2}$), where DHP doubles the throughput of the output projection (cf. Section 4.2). Within this block, the first LN is reformulated to eliminate inverse operations, while the second follows the standard protocol (cf. Section 5.2).

**Compact Packing across the Transformer Layer.** We employ AMCP to pack multiple feature columns per ciphertext throughout the transformer layer. During the attention layer, the

packed columns belong to distinct heads. After output projection, they represent distinct output features. The compact layout continues through LN and FFN, reducing intermediate ciphertext counts and enhancing computational parallelism (cf. Section 4.3).

# 4 Efficient Matrix Multiplication and Compact Packing Strategy

**Table 2:** Amortized number of key-switching operations required for computing attention scores $\{\boldsymbol{Q}^{(h)}(\boldsymbol{K}^{(h)})^{\mathsf{T}}\}_{h\in[H]}$ per batch, where $\boldsymbol{Q}^{(h)}, \boldsymbol{K}^{(h)} \in \mathbb{R}^{m\times d'}$. Parameters correspond to the BERT-base model ($m = 128, d = 768, d' = 64, H = 12$) with $N = 2^{16}$ and $t = N/2m = 256$. We set $a = 64$ and $c = 16$, consistent with Powerformer and THOR, respectively. "†" indicates the first RIHP. "‡" indicates the second RIHP.

| Scheme | Key switch | |
|---|---|---|
| | Equation | Number |
| Euston | $((m-1)d'+m)H/(N/4m)$ | 774 |
| THOR | $\frac{m}{2c}(\frac{m}{2}+7)+\frac{m}{8}(\log c+3)+\frac{m}{2}$ | 460 |
| Powerformer | $\approx(17+14\sqrt{2})\sqrt{a}+3a$ | 486 |
| MOAI | $((m-1)d'+m)H/(N/2m)$ | 387 |
| **STIP**† | $((d'+2)m/2)H/(N/2m)$ | **198** |
| **STIP**‡ | $((m-1)(d'/2)+2m)H/(N/2m)$ | **203** |

## 4.1 Real-Imaginary Hybrid Packing

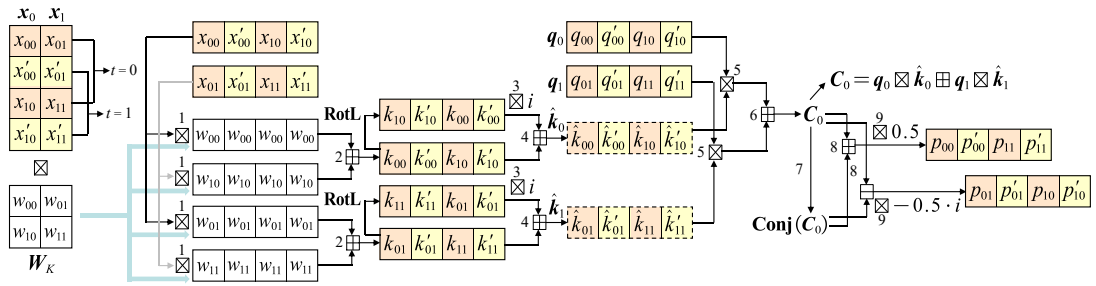



**Figure 2:** An example of the first RIHP strategy. We set dimensions $m = d = d' = 2$, batch size $t = 2$, and slots $S = 4$. The dashed box indicates the utilization of the imaginary component. "**Conj**" indicates the conjugate. Under the interleaved batching, the rotational step size is $\phi \cdot \varphi$, where $\phi$ is the original step size and $\varphi = S/m$. The numbered operations are executed sequentially.

For computing the attention score $\boldsymbol{QK}^{\top}$, the SOTA CCMM in MOAI evaluates $m$ diagonals individually: $[\![\boldsymbol{P}_r]\!] = \sum_{j=0}^{d'-1}[\![\boldsymbol{q}_j]\!] \boxtimes \text{RoTL}_r([\![\boldsymbol{k}_j]\!])$ for $r \in [0, m-1]$, where $\boldsymbol{q}_j$ and $\boldsymbol{k}_j$ denote the $j$-th feature column of the query and key matrices. However, this method incurs substantial overheads due to the extensive number of rotations (see Table 2).

To decrease the overhead of computing the attention score, we introduce two RIHP strategies. We present the first strategy in this section. The second strategy is described in Appendix A, and the details of the second strategy applied to the $\boldsymbol{SV}$ computation are also described in Appendix A. Algorithm 1 presents the pseudocode of the first strategy, and Figure 2 illustrates an example of the first strategy, where we adopt the interleaved batching in [19, 24, 25] for packing multi-batch inputs, which enables efficient intra-batch rotations. The pipeline comprises three stages:

**Algorithm 1** The First RIHP Strategy

**Input:** Input matrix $\boldsymbol{X} = \{\boldsymbol{x}_l\}_{l=0}^{d-1}$, Weights $\boldsymbol{W}_O = \{w_{l,j}^O\}_{l\in[d],j\in[d']}$, $\boldsymbol{O} \in \{\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}\}$, $\boldsymbol{o} \in \{\boldsymbol{q}, \boldsymbol{k}, \boldsymbol{v}\}$.
**Output:** Encrypted diagonals of $\boldsymbol{QK}^\top$: $\{[\![\boldsymbol{P}_r]\!]\}_{r=0}^{m-1}$.
**if** Layer $= 0$ **then**
  **for** $l = 0, 1, \ldots, d-1$ **do**
    $\hat{\boldsymbol{x}}_l \leftarrow \boldsymbol{x}_l + i \cdot \textsc{RotL}_{m/2}(\boldsymbol{x}_l)$
  **end for**
  **for** $j = 0, 1, \ldots, d'-1$ **do**
    $[\![\hat{\boldsymbol{k}}_j]\!] \leftarrow \sum_{l=0}^{d-1} w_{l,j}^K \cdot [\![\hat{\boldsymbol{x}}_l]\!]$
    $[\![\boldsymbol{q}_j]\!] \leftarrow \sum_{l=0}^{d-1} w_{l,j}^Q \cdot [\![\boldsymbol{x}_l]\!]$
    $[\![\boldsymbol{v}_j]\!] \leftarrow \sum_{l=0}^{d-1} w_{l,j}^V \cdot [\![\boldsymbol{x}_l]\!]$
  **end for**
**else**
  **for** $j = 0, 1, \ldots, d'-1$ **do**
    $[\![\boldsymbol{o}_j]\!] \leftarrow \sum_{l=0}^{d-1} w_{l,j}^O \cdot [\![\boldsymbol{x}_l]\!]$
    $[\![\hat{\boldsymbol{k}}_j]\!] \leftarrow [\![\boldsymbol{k}_j]\!] \boxplus i \cdot \textsc{RotL}_{m/2}([\![\boldsymbol{k}_j]\!])$
  **end for**
**end if**
**for** $r = 0, 1, \ldots, m/2-1$ **do**
  $[\![\boldsymbol{C}_r]\!] \leftarrow \sum_{j=0}^{d'-1} ([\![\boldsymbol{q}_j]\!] \boxtimes \textsc{RotL}_r([\![\hat{\boldsymbol{k}}_j]\!]))$
  $[\![\boldsymbol{P}_r]\!] \leftarrow \frac{1}{2} \boxtimes ([\![\boldsymbol{C}_r]\!] \boxplus \textsc{Conj}([\![\boldsymbol{C}_r]\!]))$
  $[\![\boldsymbol{P}_{r+m/2}]\!] \leftarrow -\frac{i}{2} \boxtimes ([\![\boldsymbol{C}_r]\!] \boxminus \textsc{Conj}([\![\boldsymbol{C}_r]\!]))$
**end for**
**return** $\{[\![\boldsymbol{P}_r]\!]\}_{r=0}^{m-1}$

***Phase 1: Input Projection with Hybrid Key Matrix.*** This phase generates the encrypted query and value matrices, while constructing an encrypted hybrid key matrix.

**Key Innovation:** We introduce a hybrid packing format for the key matrix $\boldsymbol{K}$, denoted as $[\![\hat{\boldsymbol{K}}]\!] = [[\![\hat{\boldsymbol{k}}_0]\!], \ldots, [\![\hat{\boldsymbol{k}}_{d'-1}]\!]]$, where each slot encodes original data in the real part and rotated data in the imaginary part: $\hat{\boldsymbol{k}}_j = \boldsymbol{k}_j + i \cdot \textsc{RotL}_{m/2}(\boldsymbol{k}_j)$. This enables the halved-rotation attention mechanism in ***Phase 2***. The implementation strategy distinguishes between the first layer and subsequent layers:

*First Layer:* Given the input $\boldsymbol{X} \in \mathbb{R}^{m\times d}$, the client encrypts the column-wise input $[\![\boldsymbol{X}]\!] = [[\![\boldsymbol{x}_0]\!], ..., [\![\boldsymbol{x}_{d-1}]\!]]$ for the query and value matrices. For the key matrix, the client constructs a hybrid plaintext $\hat{\boldsymbol{X}} = \boldsymbol{X} + i \cdot \textsc{RotL}_{m/2}(\boldsymbol{X})$, yielding $[\![\hat{\boldsymbol{X}}]\!] = [[\![\hat{\boldsymbol{x}}_0]\!], ..., [\![\hat{\boldsymbol{x}}_{d-1}]\!]]$. The client sends $[\![\hat{\boldsymbol{X}}]\!]$ and $[\![\boldsymbol{X}]\!]$ to the server.

Upon receiving these ciphertexts, the server computes the input projection via PCMM with plaintext weights. Let $\boldsymbol{W}_O \in \mathbb{R}^{d\times d'}(\boldsymbol{O} \in \{\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}\})$ be the weight matrix with $w_{l,j}^O$ representing the weight at row $l$ and column $j$. Specifically, the $j$-th column of the hybrid key matrix $[\![\hat{\boldsymbol{K}}]\!]$ is:

$$[\![\hat{\boldsymbol{k}}_j]\!] = \sum\nolimits_{l=0}^{d-1} w_{l,j}^K \cdot [\![\hat{\boldsymbol{x}}_l]\!]. \tag{7}$$

Similarly, the query and value projections, $[\![\boldsymbol{Q}]\!]$ and $[\![\boldsymbol{V}]\!]$, are obtained by applying the weight matrices $\boldsymbol{W}_Q$ and $\boldsymbol{W}_V$ to the encrypted input.

*Subsequent Layers:* For subsequent layers, the input $[\![\boldsymbol{X}]\!]$ is the encrypted output of the previous layer. The column-wise encrypted packing of $\boldsymbol{K}$ is $[\![\boldsymbol{K}]\!] = [[\![\boldsymbol{k}_0]\!], [\![\boldsymbol{k}_1]\!], ..., [\![\boldsymbol{k}_{d'-1}]\!]]$. We compute the standard key vectors $[\![\boldsymbol{k}_j]\!]$:

$$[\![\boldsymbol{k}_j]\!] = \sum\nolimits_{l=0}^{d-1} w_{l,j}^K \cdot [\![\boldsymbol{x}_l]\!]. \tag{8}$$

Then, to enable RIHP strategy, we apply a homomorphic rotation to construct the hybrid key matrix $[\![\hat{\boldsymbol{K}}]\!]$:

$$[\![\hat{\boldsymbol{k}}_j]\!] = [\![\boldsymbol{k}_j]\!] \boxplus i \cdot \textsc{RotL}_{m/2}([\![\boldsymbol{k}_j]\!]). \tag{9}$$

***Phase 2: Halved-Rotation CCMM.*** We propose a compact CCMM to compute the attention scores $\boldsymbol{P} = \boldsymbol{Q}\boldsymbol{K}^\top$.

**Key Innovation:** Instead of computing $m$ separate diagonals, we leverage the complex nature of CKKS to pack two diagonals into one ciphertext.

For $r \in \{0, \ldots, m/2-1\}$, the $r$-th hybrid ciphertext $[\![\boldsymbol{C}_r]\!]$ is computed as:

$$[\![\boldsymbol{C}_r]\!] = \sum\nolimits_{j=0}^{d'-1} [\![\boldsymbol{q}_j]\!] \boxtimes \textsc{RotL}_r([\![\hat{\boldsymbol{k}}_j]\!]). \tag{10}$$

Due to the linearity of rotation and the hybrid structure of $[\![\hat{\boldsymbol{K}}]\!]$, $[\![\boldsymbol{C}_r]\!]$ simultaneously stores the $r$-th real diagonal and the $(r + m/2)$-th imaginary diagonal:

$$[\![\boldsymbol{C}_r]\!] = [\![\boldsymbol{P}_r]\!] \boxplus i \cdot [\![\boldsymbol{P}_{r+m/2}]\!]. \tag{11}$$

**Complexity Gain:** This technique requires only $md'/2$ multiplications and $(m/2-1)d'$ rotations, reducing the computational cost by **50%** compared to the MOAI approach.

***Phase 3: Conjugate Unpacking.*** Since the subsequent Softmax function is nonlinear and requires pure real inputs, we must separate the packed diagonals. We utilize the homomorphic conjugate operation ($\textsc{Conj}$) for efficient extraction:

$$[\![\boldsymbol{P}_r]\!] = 0.5 \boxtimes ([\![\boldsymbol{C}_r]\!] \boxplus \textsc{Conj}([\![\boldsymbol{C}_r]\!])), \tag{12}$$

$$[\![\boldsymbol{P}_{r+m/2}]\!] = -0.5i \boxtimes ([\![\boldsymbol{C}_r]\!] \boxminus \textsc{Conj}([\![\boldsymbol{C}_r]\!])). \tag{13}$$

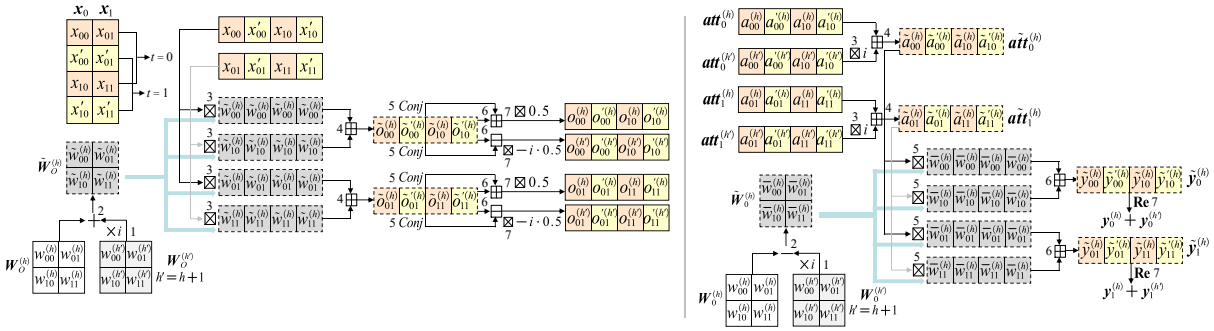


**Figure 3:** Examples of DHP. The left side is the input projection, and the right side is the output projection. We set dimensions $m = d = d' = 2$, batch size $t = 2$, and slots $S = 4$. The dashed box indicates use of the imaginary component. "**Re**" denotes extraction of the real part.

## 4.2 Double-Head Packing

For input and output projections in the attention layer, MOAI and Euston utilize column packing to compute PCMMs. However, both schemes process a single attention head per ciphertext, limiting throughput (see Table 3).

To enhance the computational parallelism of the multi-head attention projection, we introduce a DHP strategy. By exploiting the complex-valued slots of CKKS ciphertexts, we process two attention heads ($h$ and $h+1$) simultaneously within one ciphertext. This strategy is applied to the input projection ($\boldsymbol{X} \to \boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$) and the output projection ($\boldsymbol{Att} \to \boldsymbol{Y}$). Figure 3 illustrates an example of DHP. The pipeline comprises four stages:

**Table 3:** Amortized number of key-switching operations required for computing input projections $\{\boldsymbol{X}\boldsymbol{W}_O^{(h)}\}_{h\in[H]}$ per batch, where $\boldsymbol{X} \in \mathbb{R}^{m\times d}, \boldsymbol{W}_O^{(h)} \in \mathbb{R}^{d\times d'}$. Parameters are identical to Table 2. "Throughput" refers to the number of heads processed for each input projection.

| Scheme | Key switch | | Throughput |
|---|---|---|---|
| | Equation | Number | |
| NEXUS | $2md/(N/2m)$ | 768 | 1× |
| THOR | $\frac{m}{2} + 3(H-2)\frac{d'}{c} + \frac{3d'}{c}$ | 196 | 1× |
| Powerformer | $\approx \sqrt{\frac{5}{9}} \cdot 2\sqrt{\frac{md(3d)}{N/2}}$ | 124 | 1× |
| Euston/MOAI | 0 | 0 | 1× |
| **STIP** | $(3d/2)/(N/(2m))$ | **4.5** | **2**× |

***Phase 1: Offline Weight Pre-coding.*** We pair adjacent attention heads $(h, h+1)$ for $h \in \{0, 2, \ldots, H-2\}$. Let $\boldsymbol{W}_O^{(h)}$ be the real-valued weights ($\boldsymbol{O} \in \{\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}\}$). For the input projection, we construct complex weight matrices $\tilde{\boldsymbol{W}}_O^{(h)}$:

$$\tilde{\boldsymbol{W}}_O^{(h)} = \boldsymbol{W}_O^{(h)} + i \cdot \boldsymbol{W}_O^{(h+1)}. \tag{14}$$

For the output projection, we construct the weights using a conjugate structure to facilitate efficient aggregation:

$$\tilde{\boldsymbol{W}}_0^{(h)} = \boldsymbol{W}_0^{(h)} - i \cdot \boldsymbol{W}_0^{(h+1)}. \tag{15}$$

This preprocessing is performed offline and incurs zero online computational overhead.

***Phase 2: Parallel Input Projection.*** For the column-wise input $[\![\boldsymbol{X}]\!] = [[\![\boldsymbol{x}_0]\!], ..., [\![\boldsymbol{x}_{d-1}]\!]]$, we compute the projections $[\![\tilde{\boldsymbol{O}}^{(h)}]\!] = [[\![\tilde{\boldsymbol{o}}_0^{(h)}]\!], [\![\tilde{\boldsymbol{o}}_1^{(h)}]\!], ..., [\![\tilde{\boldsymbol{o}}_{d'-1}^{(h)}]\!]]$ ($\boldsymbol{o} \in \{\boldsymbol{q}, \boldsymbol{k}, \boldsymbol{v}\}$) for both heads in a single PCMM with the pre-coded weights $\tilde{w}_{l,j}^{(h)} \in \tilde{\boldsymbol{W}}_O^{(h)}$:

$$[\![\tilde{\boldsymbol{o}}_j^{(h)}]\!] = \sum_{l=0}^{d-1} \tilde{w}_{l,j}^{(h)} \cdot [\![\boldsymbol{x}_l]\!]. \tag{16}$$

Due to the linearity of the operation, the resulting ciphertext $[\![\tilde{\boldsymbol{O}}^{(h)}]\!]$ naturally encodes the projections for head $h$ in the real part and head $h+1$ in the imaginary part:

$$\tilde{\boldsymbol{O}}^{(h)} = \boldsymbol{X}\boldsymbol{W}_{O,h} + i \cdot \boldsymbol{X}\boldsymbol{W}_{O,h+1}. \tag{17}$$

This step effectively doubles the throughput of PCMM.

***Phase 3: Unpacking for Attention Mechanism.*** To prevent cross-term interference arising from the attention score calculation ($\boldsymbol{Q}\boldsymbol{K}^\top$) on complex-packed data, we unpack the heads prior to the attention block. Using the homomorphic conjugate operation (Conj), we extract the individual heads:

$$[\![\boldsymbol{O}^{(h)}]\!] = 0.5 \boxtimes ([\![\tilde{\boldsymbol{O}}^{(h)}]\!] \boxplus \textsc{Conj}([\![\tilde{\boldsymbol{O}}^{(h)}]\!])), \tag{18}$$

$$[\![\boldsymbol{O}^{(h+1)}]\!] = -0.5i \boxtimes ([\![\tilde{\boldsymbol{O}}^{(h)}]\!] \boxminus \textsc{Conj}([\![\tilde{\boldsymbol{O}}^{(h)}]\!])). \tag{19}$$

Following unpacking, the attention mechanism is applied to obtain
$[\![\boldsymbol{Att}^{(h)}]\!] = [[\![\boldsymbol{att}_0^{(h)}]\!], [\![\boldsymbol{att}_1^{(h)}]\!], ..., [\![\boldsymbol{att}_{d'-1}^{(h)}]\!]]$ and
$[\![\boldsymbol{Att}^{(h+1)}]\!] = [[\![\boldsymbol{att}_0^{(h+1)}]\!], [\![\boldsymbol{att}_1^{(h+1)}]\!], ..., [\![\boldsymbol{att}_{d'-1}^{(h+1)}]\!]]$.

***Phase 4: Parallel Output Projection.*** After computing the attention outputs for all $H$ heads, we aggregate them into $H/2$ complex pairs. For each pair index $h \in \{0, 2, \ldots, H-2\}$, we pack the outputs of head $h$ and $h+1$ into a complex ciphertext:

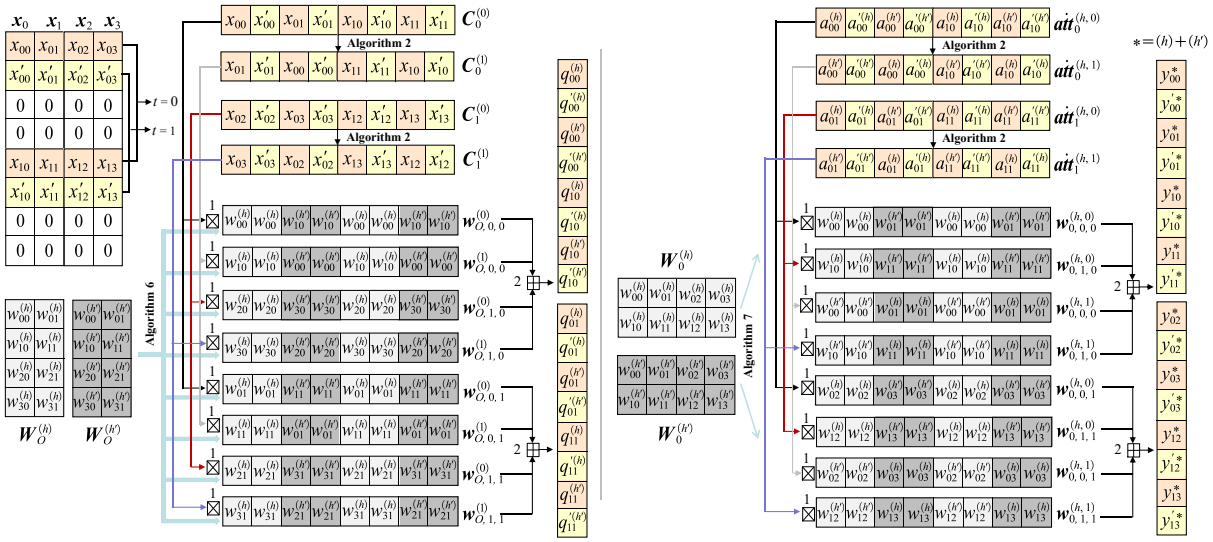



**Figure 4:** Examples of AMCP. The left side corresponds to the input projection, and the right side corresponds to the output projection. We set dimensions $m = d' = 2, d = 4$, batch size $t = 2$, slots $S = 8$, head number $H = 2$, and packing factor $k = 2$.

$$\begin{aligned}[\![\tilde{\boldsymbol{Att}}^{(h)}]\!] &= [\![\boldsymbol{Att}^{(h)}]\!] \boxplus i \boxtimes [\![\boldsymbol{Att}^{(h+1)}]\!] \\ &= [[\![\tilde{\boldsymbol{att}}_0^{(h)}]\!], [\![\tilde{\boldsymbol{att}}_1^{(h)}]\!], ..., [\![\tilde{\boldsymbol{att}}_{d'-1}^{(h)}]\!]].\end{aligned} \tag{20}$$

The output $\boldsymbol{Y}$ is the sum of projections from all heads:

$$\boldsymbol{Y} = \sum\nolimits_{h=0}^{H-1} \boldsymbol{Att}^{(h)} \boldsymbol{W}_0^{(h)} = [\sum\nolimits_{h=0}^{H-1} \boldsymbol{y}_0^{(h)}, ..., \sum\nolimits_{h=0}^{H-1} \boldsymbol{y}_{d-1}^{(h)}]. \tag{21}$$

We compute $\boldsymbol{Y}$ by aggregating the products of the complex pairs and their corresponding pre-coded conjugate weights $\tilde{\boldsymbol{W}}_0^{(h)} = \boldsymbol{W}_0^{(h)} - i \cdot \boldsymbol{W}_0^{(h+1)}$:

$$\tilde{\boldsymbol{Y}} = \sum\nolimits_h (\tilde{\boldsymbol{Att}}^{(h)} \times \tilde{\boldsymbol{W}}_0^{(h)}), h \in \{0, 2, \ldots, H-2\}. \tag{22}$$

For $\bar{w}_{l,j}^{(h)} \in \tilde{\boldsymbol{W}}_0^{(h)}$, the value of the $j$-th column of $[\![\tilde{\boldsymbol{Att}}^{(h)} \times \tilde{\boldsymbol{W}}_0^{(h)}]\!]$ is:

$$[\![\tilde{\boldsymbol{y}}_j^{(h)}]\!] = \sum\nolimits_{l=0}^{d'-1} \bar{w}_{l,j}^{(h)} \cdot [\![\tilde{\boldsymbol{att}}_l^{(h)}]\!], j \in \{0, 1, \ldots, d-1\}. \tag{23}$$

Due to the conjugate structure of the weights, the real part of each term in the summation corresponds to the combined contribution of the two heads in that pair:

$$\begin{aligned}\mathrm{Re}(\tilde{\boldsymbol{Att}}^{(h)} \times \tilde{\boldsymbol{W}}_0^{(h)}) &= \boldsymbol{Att}^{(h)} \boldsymbol{W}_0^{(h)} + \boldsymbol{Att}^{(h+1)} \boldsymbol{W}_0^{(h+1)} \\ &= [\boldsymbol{y}_0^{(h)} + \boldsymbol{y}_0^{(h+1)}, ..., \boldsymbol{y}_{d-1}^{(h)} + \boldsymbol{y}_{d-1}^{(h+1)}].\end{aligned} \tag{24}$$

$\mathrm{Re}(\cdot)$ denotes the real part. Consequently, the real part of the accumulated ciphertext $[\![\tilde{\boldsymbol{Y}}]\!]$ yields the correct multi-head attention output $\boldsymbol{Y}$. We perform a final real-part extraction:

$$[\![\boldsymbol{Y}]\!] = 0.5 \boxtimes ([\![\tilde{\boldsymbol{Y}}]\!] \boxplus \textsc{Conj}([\![\tilde{\boldsymbol{Y}}]\!])) = [[\![\boldsymbol{y}_0]\!], [\![\boldsymbol{y}_1]\!], ..., [\![\boldsymbol{y}_{d-1}]\!]]. \tag{25}$$

This step recovers the multi-head attention output $\boldsymbol{Y}$, restoring the column-packed format for the next transformer block. Although conjugation adds a small key-switching overhead (Table 3), DHP halves the PCMMs in both projections, yielding savings that far outweigh this overhead.

### 4.3 Adaptive Multi-Column Packing

The column packing of MOAI and Euston suffers from severe slot underutilization when the column length $L = t \cdot m$ is less than the slots $S$. This waste leads to constrained parallel efficiency and substantial ciphertext numbers.

To overcome these limitations, we propose AMCP, which packs multiple feature columns into a single ciphertext. By adopting an interleaved batching strategy [19, 24, 25], AMCP achieves simultaneous parallelism across different batches and feature columns within a single ciphertext, while facilitating internal rotations.

The design challenge of AMCP lies in adapting attention, LN, and FFN to the multi-column layout. Attention requires structuring the input projection for multi-column packing and the output projection for multi-head aggregation while preserving compact outputs (Figure 4). LN requires aggregating packed columns to compute row-wise means and variances, and FFN requires aligning packed columns with weights. This section details these computations.

**Parallel Multi-Column Packing Strategy.** We apply a *grouped packing* strategy to the input $\boldsymbol{X} \in \mathbb{R}^{tm \times d}$. Define $k$ as the packing factor, which is the number of feature columns per grouped ciphertext. $d$ columns of the input matrix $\boldsymbol{X}$ are partitioned into $G = d/k$ groups, indexed by $g \in \{0, \ldots, G-1\}$. We select the packing factor compatible with the attention-head grouping: $k = \max\{\kappa \in \text{Divisors}(H) \mid \kappa \cdot L \leq S\}$. Divisors$(H)$ is the set of factors of the head count $H$. LN and FFN use the same $k$ to group feature columns.

For BERT-base ($H = 12$), this constraint ($\kappa \in \{1, 2, 3, 4, 6, 12\}$) ensures that the number of ciphertext groups $G = d/k$ is a multiple of the head dimension $d'$. For instance, with $S = 2^{15}, m = 128, t = 32$, we derive $k = 6$, which compresses the query matrix $\boldsymbol{Q}$ from 768 to 128 ciphertexts. Since $2d' = 128$, the layout is perfectly aligned: each of the first 64 ciphertexts holds one feature column from each of the first 6 heads, while the subsequent 64 ciphertexts cover the remaining 6 heads. This uniform interleaving is essential for unified parallel processing.

**Interleaved Batching.** Consider a grouped ciphertext with $k$ feature columns and batch size $t$, forming $W = k \cdot t$ data streams $\{\boldsymbol{v}^{(0)}, \ldots, \boldsymbol{v}^{(W-1)}\}$. These streams are logically ordered: indices 0 to $t-1$ cover all batches of the first feature column, followed by indices $t$ to $2t-1$ for the second.

We construct the ciphertext $[\![\boldsymbol{C}_g]\!]$ by interleaving the elements of $W$ streams. Let $v_s^{(\zeta)}$ denote the element at sequence position $s$ of the $\zeta$-th stream. Specifically, we arrange elements based on their sequence index $s \in [0, m-1]$, inserting zero padding after the active streams to align the total width to the stride $\varphi = S/m$:

$$[\![\boldsymbol{C}_g]\!] = [\underbrace{v_0^{(0)}, \ldots, v_0^{(W-1)}, 0, \ldots}_{\textit{Stride } \varphi \text{ (Position 0)}}, \underbrace{v_1^{(0)}, \ldots, v_1^{(W-1)}, 0, \ldots}_{\textit{Stride } \varphi \text{ (Position 1)}}, \ldots]. \tag{26}$$

This structure implies that:

① Adjacent ciphertext slots store data from consecutive batches or adjacent feature columns (vary $\zeta$, fix $s$).

② Data from the same stream is distributed with a fixed stride $\varphi$, ensuring that the physical distance between sequence elements remains constant.

For example, as shown in Figure 4, the first grouped ciphertext with two feature columns $\boldsymbol{x}_0, \boldsymbol{x}_1$ forms data streams: $\boldsymbol{v}^{(0)} = \{x_{00}, x_{10}\}, \boldsymbol{v}^{(1)} = \{x'_{00}, x'_{10}\}, \boldsymbol{v}^{(2)} = \{x_{01}, x_{11}\}, \boldsymbol{v}^{(3)} = \{x'_{01}, x'_{11}\}$. The sequence of ciphertext $[\![\boldsymbol{C}_g]\!]$ is $x_{00}, x'_{00}, x_{01}, x'_{01}, x_{10}, x'_{10}, x_{11}, x'_{11}$.

**Interleaved Input Projection.** The input projection aims to compute $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$ by summing the products of input columns and their corresponding weights. However, since the ciphertext

**Algorithm 2** Iterative Group-Wise Rotation

**Input:** Ciphertext $[\![\boldsymbol{C}_g]\!]$, batch size $t$, packing factor $k$, active width $W = k \cdot t$, total slots $S$, sequence length $m$.
**Output:** Aligned ciphertexts $[[\![\boldsymbol{C}_g^{(0)}]\!], \ldots, [\![\boldsymbol{C}_g^{(k-1)}]\!]]$.
$\varphi \leftarrow S/m$
$[\![\boldsymbol{C}_g^{(0)}]\!] \leftarrow [\![\boldsymbol{C}_g]\!]$
**for** $r = 1, 2, \ldots, k-1$ **do**
    $u \leftarrow (\underbrace{1, \ldots, 1}_{W-rt}, \underbrace{0, \ldots, 0}_{rt}, \underbrace{0, \ldots, 0}_{\varphi - W})$
    $\boldsymbol{u} \leftarrow (u, u, \ldots, u)$
    $v \leftarrow (\underbrace{0, \ldots, 0}_{W-rt}, \underbrace{1, \ldots, 1}_{rt}, \underbrace{0, \ldots, 0}_{\varphi - W})$
    $\boldsymbol{v} \leftarrow (v, v, \ldots, v)$
    $[\![\boldsymbol{C}_g^{(r)}]\!] \leftarrow (\textsc{RotL}_{rt}([\![\boldsymbol{C}_g]\!]) \boxtimes \boldsymbol{u})$
        $\boxplus (\textsc{RotR}_{W-rt}([\![\boldsymbol{C}_g]\!]) \boxtimes \boldsymbol{v})$
**end for**
**return** $[[\![\boldsymbol{C}_g^{(0)}]\!], [\![\boldsymbol{C}_g^{(1)}]\!], \ldots, [\![\boldsymbol{C}_g^{(k-1)}]\!]]$

$[\![\boldsymbol{C}_g]\!]$ interleaves $k$ distinct feature columns within a stride of $\varphi$, we must align these input columns to their target weights via rotation.

We generate $k$ aligned states $\{[\![\boldsymbol{C}_g^{(r)}]\!]\}_{r=0}^{k-1}$ using Algorithm 2. Each shift $r$ is computed directly from the original ciphertext by combining a left rotation of $rt$ slots and a right rotation of $W - rt$ slots with disjoint masks. The masks confine the cyclic shift to the $W$ active slots within each stride. Thus, the aligned states do not depend on one another.

**Input Weight Construction and Aggregation.** A detailed construction is provided in Appendix B.

**Rotation Step Scaling for CCMMs.** When applying AMCP to CCMMs, rotation indices must be scaled to account for the stride $\varphi = S/m$. Since logically adjacent elements are separated by $\varphi$ slots in the packed ciphertext, any logical rotation of step $\phi$ required for standard operations maps to a scaling rotation of $\phi \cdot \varphi$.

**Interleaved Output Projection.** After the attention mechanism, the output attention $[\![\dot{\boldsymbol{Att}}^{(h)}]\!] = [[\![\dot{\boldsymbol{att}}_0^{(h)}]\!], [\![\dot{\boldsymbol{att}}_1^{(h)}]\!], ..., [\![\dot{\boldsymbol{att}}_{d'-1}^{(h)}]\!]]$ retains the AMCP structure. Each ciphertext $[\![\dot{\boldsymbol{att}}_i^{(h)}]\!]$ packs $k$ attention-output columns, one from each of $k$ distinct heads. Unlike standard column packing, this distribution requires a specialized weight construction and a rotation-based accumulation to fuse the dispersed attentions.

**Output Weight Construction and Aggregation.** A detailed construction is provided in Appendix B.

**LN and FFN on Compact Outputs.** The compact outputs retain $k$ feature columns per ciphertext. LN computes each row's mean and variance by summing values across packed columns, while FFN PCMMs use group-wise rotations with repacked weights. Appendix B details the computations.

### 4.4 Integration of RIHP, DHP, and AMCP

Appendix C presents the workflow for combining the RIHP with DHP, and for integrating RIHP, DHP, and AMCP.

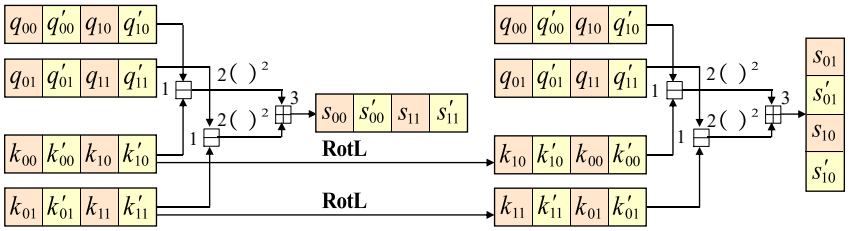



**Figure 5:** Examples of L2-norm under column packing. We set dimensions $m = d = d' = 2$ and batch size $t = 2$.

# 5 Efficient Nonlinear Layers

## 5.1 Gaussian Kernel-based Attention

To ensure the numerical stability and accelerate the attention mechanism, we employ the GK instead of Softmax. This reformulation eliminates the need for maximum value search and replaces the dynamic inverse with a constant.

**Formulation.** The attention score $\boldsymbol{S}(\boldsymbol{Q}, \boldsymbol{K})$ is computed by Eq. (2). Since the Euclidean distance is non-negative, this formulation restricts outputs to the $(0, 1]$ range, eliminating the need for maximum value required by Softmax to prevent overflow. The GK uses a fixed scale $1/(2\sqrt{d'})$ and does not require the dynamic normalization used by Softmax, thereby eliminating the associated inverse evaluation.

**Efficient L2-Norm under Column Packing.** The prior work [37] adopts a row-packing layout to compute the GK, obtaining L2-norm via intra-row rotation and summation. However, applying this calculation method directly to column packing is computationally prohibitive.

We first address the structural incompatibility of row-based expansion by proposing a direct squared difference accumulation. Figure 5 computes the $r$-th diagonal of the squared distance matrix between $\boldsymbol{Q}$ and $\boldsymbol{K}$, for $r \in \{0, \ldots, m-1\}$:

$$[\![\boldsymbol{P}_r]\!] = \sum\nolimits_{j=0}^{d'-1} [[\![\boldsymbol{q}_j]\!] \boxminus \textsc{RotL}_r([\![\boldsymbol{k}_j]\!])]^2, \tag{27}$$

where $\boldsymbol{q}_j$ and $\boldsymbol{k}_j$ denote the $j$-th feature columns of the query and key matrices. The number of rotations matches the attention score computed by MOAI.

To further minimize cost, we introduce the second RIHP to L2-norm computation with the squared expansion in Eq. (27): $\sum_{j=0}^{d'-1} [[\![\boldsymbol{q}_j]\!]^2 \boxplus (\textsc{RotL}_r([\![\boldsymbol{k}_j]\!]))^2 \boxminus 2 \boxtimes [\![\boldsymbol{q}_j]\!] \boxtimes \textsc{RotL}_r([\![\boldsymbol{k}_j]\!])]$. Algorithm 3 provides the second RIHP strategy to efficiently compute L2-norm, reducing the key-switch numbers from $(m-1)d' + m$ to $(m+1)d'/2 + 3m + 1$. Appendix D provides the pseudocode for the GK function and exponential function.

**Depth Analysis.** The GK requires five levels: one for the scalar PCMM and four for the exponential computation. Absorbing the scalar into the ciphertext scale eliminates its rescale, reducing the depth to four levels, compared to ten levels for Softmax in MOAI and Euston.

## 5.2 Inverse-Free Layer Normalization

LN computes $\boldsymbol{y} = \gamma \cdot (\boldsymbol{x} - \mu)/\sigma + \boldsymbol{\beta}$, which involves calculating the inverse of the standard deviation $(1/\sigma)$. We propose a scale propagation strategy, eliminating the need for inverse computation. This section describes the implementation for the post-layer normalization architecture. The implementation for the pre-layer normalization architecture is detailed in Appendix E.

**Algorithm 3** L2-Norm Computation with the Second RIHP

**Input:** Query $[\![\boldsymbol{Q}]\!] = \{[\![\boldsymbol{q}_j]\!]\}_{j=0}^{d'-1}$, key $[\![\boldsymbol{K}]\!] = \{[\![\boldsymbol{k}_j]\!]\}_{j=0}^{d'-1}$.
**Output:** Encrypted diagonals $\{[\![\boldsymbol{P}_r]\!]\}_{r=0}^{m-1}$.
$[\![\boldsymbol{S}_Q]\!] \leftarrow 0, [\![\boldsymbol{S}_K]\!] \leftarrow 0$
**for** $j = 0, \ldots, d'/2 - 1$ **do**
  $[\![\hat{\boldsymbol{q}}_j]\!] \leftarrow [\![\boldsymbol{q}_j]\!] \boxplus i \cdot [\![\boldsymbol{q}_{j+d'/2}]\!]$
  $[\![\hat{\boldsymbol{k}}_j]\!] \leftarrow [\![\boldsymbol{k}_j]\!] \boxplus i \cdot [\![\boldsymbol{k}_{j+d'/2}]\!]$
  $[\![\hat{\boldsymbol{k}}'_j]\!] \leftarrow \textsc{Conj}([\![\hat{\boldsymbol{k}}_j]\!])$
  $[\![\boldsymbol{S}_Q]\!] \leftarrow [\![\boldsymbol{S}_Q]\!] \boxplus [\![\hat{\boldsymbol{q}}_j]\!] \boxtimes \textsc{Conj}([\![\hat{\boldsymbol{q}}_j]\!])$
  $[\![\boldsymbol{S}_K]\!] \leftarrow [\![\boldsymbol{S}_K]\!] \boxplus [\![\hat{\boldsymbol{k}}_j]\!] \boxtimes [\![\hat{\boldsymbol{k}}'_j]\!]$
**end for**
**for** $r = 0, \ldots, m - 1$ **do**
  $[\![\boldsymbol{C}_r]\!] \leftarrow \sum_{j=0}^{d'/2-1} [\![\hat{\boldsymbol{q}}_j]\!] \boxtimes \textsc{RotL}_r([\![\hat{\boldsymbol{k}}'_j]\!])$
  $[\![\boldsymbol{T}_r]\!] \leftarrow [\![\boldsymbol{S}_Q]\!] \boxplus \textsc{RotL}_r([\![\boldsymbol{S}_K]\!]) \boxminus ([\![\boldsymbol{C}_r]\!] \boxplus [\![\boldsymbol{C}_r]\!])$
  $[\![\boldsymbol{P}_r]\!] \leftarrow ([\![\boldsymbol{T}_r]\!] \boxplus \textsc{Conj}([\![\boldsymbol{T}_r]\!])) \boxtimes 0.5$
**end for**
**return** $\{[\![\boldsymbol{P}_r]\!]\}_{r=0}^{m-1}$

**Inverse Elimination via Scale Invariance.** To bypass the inverse square root, we reformulate the first LN to propagate the scaling factor. Euston implements LN with the minimal multiplicative level. The standard LN in Euston is:

$$y_i = \gamma \cdot \frac{x_i - \mu}{\sigma} + \beta = \gamma \frac{z_i \sqrt{n\lambda}}{\sqrt{(\lambda \sum_{i=1}^{n} z_i^2 - 1) + 1}} + \beta, \tag{28}$$

where $z_i = x_i - \mu = \frac{1}{n}\sum_{j=1}^{n}(x_i - x_j)$, and $\lambda$ constrains $\sum_{i=1}^{n} z_i^2$ within $[0, 2]$ for square root approximations. However, computing (28) requires 9 multiplicative levels.

Our strategy multiplies the entire equation by $v = \sqrt{(\lambda \sum_{i=1}^{n} z_i^2 - 1) + 1}$, yielding a scaled output $\hat{y}_i$:

$$\hat{y}_i = \gamma z_i \sqrt{n\lambda} + \beta \sqrt{(\lambda \sum_{i=1}^{n} z_i^2 - 1) + 1} = \gamma z_i \sqrt{n\lambda} + \beta v. \tag{29}$$

This effectively converts the inverse into a multiplication, treating $v$ as a scaling factor to the output data.

**Integration with Subsequent Layers.** The scaled output $\hat{\boldsymbol{y}}$ is then processed by the subsequent linear layer. To maintain mathematical consistency, the scaling factor $v$ must be preserved. While the weight multiplication $\boldsymbol{W}\hat{\boldsymbol{y}}$ naturally scales $v$, the bias term $\boldsymbol{b}$ must be adjusted to match this scale. We achieve this by multiplying the bias by $v$:

$$\hat{\boldsymbol{z}} = \boldsymbol{W}\hat{\boldsymbol{y}} + v \cdot \boldsymbol{b} = v(\boldsymbol{W}\boldsymbol{y} + \boldsymbol{b}). \tag{30}$$

$\hat{\boldsymbol{z}}$ represents the scaled input for the second LN. Crucially, this scaling factor naturally cancels out in the subsequent LN due to its scale-invariance property. By substituting the scaled input $\hat{z}_i = v z_i$ into Eq. (28), the expansion factor $v$ appears in both the numerator and denominator:

$$\frac{\hat{z}_i \sqrt{n\lambda}}{\sqrt{(\lambda \sum_{i=1}^{n} \hat{z}_i^2 - 1) + 1}} = \frac{v z_i \sqrt{n\lambda}}{\sqrt{v^2 \lambda \sum_{i=1}^{n} z_i^2}} = \frac{z_i \sqrt{n\lambda}}{\sqrt{\lambda \sum_{i=1}^{n} z_i^2}}. \tag{31}$$

This confirms that the expansion factor $v$ is eliminated by the division. Thus, the method removes the inverse operation from the first LN without affecting the correctness.

Appendix D presents the pseudocode for computing this inverse-free LN and the square root evaluation.

**Scale-Preserving Activation Function.** To support this inverse-free formulation, we ensure that the subsequent nonlinear activation functions satisfy the homogeneity property (i.e., $f(\alpha x) = \alpha f(x)$ for $\alpha > 0$). We adopt ReLU, which guarantees that the scaling factor propagates correctly through the FFN. Notably, as shown in Table 7 and 11, replacing GELU/SwiGLU with ReLU (including the substitution of Softmax with GK) preserves the model's accuracy [38].

**Depth Analysis.** By eliminating the inverse approximation and reducing Euston's square-root depth (cf. Appendix D), the depth of the first LN drops from nine to three (one for centered squaring, one for the square root, and one for PCMM). The square-root optimization also reduces the second standard LN from nine to eight levels.

# 6 Performance Evaluation

## 6.1 Theoretical Comparisons

**Table 4:** The number of rotations required for one transformer layer under batch size $t = 32$: (1) Input projection $\boldsymbol{X}\boldsymbol{W}_O$; (2) Attention scores computation $\boldsymbol{Q}\boldsymbol{K}^\mathsf{T}$; (3) Activation (Softmax/GK); (4) Activation output $\boldsymbol{S}$ multiplied by $\boldsymbol{V}$; (5) Output projection; (6) two LN operations; and (7) two FFN matrix multiplications. Concrete parameters are based on BERT-base model: $m = 128$, $d = 768$, $d' = 64$, $b = 3072$, $H = 12$, $N = 2^{16}$, $t = 32$. $k = 6$ is the packing factor in STIP. $G = d/k = 128$ is the number of grouped ciphertexts in STIP.

| ID | Scheme | Rotation Equation | Number |
|---|---|---|---|
| (1) | Euston | 0 | 0 |
| | STIP | $2(k-1)G$ | 1280 |
| (2) | Euston | $2(m-1)d'H$ | 195072 |
| | STIP | $(md'/2)(H/k)$ | 8192 |
| (3) | Euston | 0 | 0 |
| | STIP | 0 | 0 |
| (4) | Euston | $2(m-1)d'H$ | 195072 |
| | STIP | $(\sqrt{m}d'+m)(H/k)$ | 1704 |
| (5) | Euston | 0 | 0 |
| | STIP | $2(k-1)G$ | 1280 |
| (6) | Euston | 0 | 0 |
| | STIP | $8(k-1)$ | 40 |
| (7) | Euston | 0 | 0 |
| | STIP | $2(k-1)(d+b)/k$ | 6400 |
| Total | Euston | - | 390144 |
| | **STIP** | - | **18896** |

**Performance Comparison with Euston.** Table 4 presents a comparison of the number of rotations between Euston and STIP under batch size $t = 32$. By integrating RIHP, DHP, and AMCP, STIP substantially reduces the rotation complexity. The total number of rotations required for one transformer layer decreases from 390144 in Euston to 18896 in STIP, representing 20.6× fewer rotations. Specifically, the rotations required for $\boldsymbol{Q}\boldsymbol{K}^\top/\boldsymbol{S}\boldsymbol{V}$ decrease by 95.8%/99.1%. This reduction is primarily driven by AMCP, which parallelizes rotations across feature columns from different heads, and RIHP, which decreases the rotations required by CCMMs. Although

**Table 5:** Amortized number of key-switching operations incurred by matrix multiplications in one transformer layer under batch size $t = 256$: (1) Input projection $\boldsymbol{XW}_O$; (2) Attention scores computation $\boldsymbol{QK}^\mathsf{T}$; (3) Activation output $\boldsymbol{S}$ multiplied by $\boldsymbol{V}$; (4) Output projection; and (5) FFN matrix multiplications. The key-switching operations include rotations, conjugations, and relinearizations. The concrete parameters are $m = 128$, $d = 768$, $d' = 64$, $b = 3072$, $H = 12$, $N = 2^{16}$, $a = 64$, and $c = 16$.

| ID | Scheme | Key switch | |
|---|---|---|---|
| | | Equation | Number |
| (1) | NEXUS | $2md/(N/2m)$ | 768 |
| | THOR | $\frac{m}{2} + 3(H-2)\frac{d'}{c} + \frac{3d'}{c}$ | 196 |
| | Powerformer | $\approx \sqrt{\frac{5}{9}} \cdot 2\sqrt{\frac{md(3d)}{N/2}}$ | 124 |
| | Euston/MOAI | 0 | 0 |
| | **STIP** | $(3d/2)/(N/2m)$ | **4.5** |
| (2) | NEXUS | $m(d'+1)H/(N/4m)$ | 780 |
| | THOR | $\frac{m}{2c}\left(\frac{m}{2}+7\right) + \frac{m}{8}(\log_2 c + 3) + \frac{m}{2}$ | 460 |
| | Powerformer | $\approx (17 + 14\sqrt{2})\sqrt{a} + 3a$ | 486 |
| | Euston | $((m-1)d' + m)H/(N/4m)$ | 774 |
| | MOAI | $((m-1)d' + m)H/(N/2m)$ | 387 |
| | **STIP** | $((d'+2)m/2)H/(N/2m)$ | **198** |
| (3) | NEXUS | N/A | N/A |
| | THOR | $m(\log_2 c + 3)/4 + m\frac{m+6}{4c}$ | 492 |
| | Powerformer | $\approx (19 + 10\sqrt{2})\sqrt{a} + 3a$ | 457 |
| | Euston | $md'H/(N/4m)$ | 768 |
| | MOAI | $(3\sqrt{m}d' + m)H/(N/2m)$ | 108 |
| | **STIP** | $(\frac{3}{2}\sqrt{m}d' + m + d'/2)H/(N/2m)$ | **58** |
| (4) | NEXUS | $2d^2/(N/4m)$ | 9216 |
| | THOR | $\frac{d'}{2} + 2\left(\frac{H}{2} - 1\right)\frac{m}{c}$ | 112 |
| | Powerformer | $\approx \sqrt{\frac{2}{3}} \cdot 2\sqrt{\frac{md^2}{N/2}}$ | 78 |
| | Euston/MOAI | 0 | 0 |
| | **STIP** | $d/(N/2m)$ | **3** |
| (5) | NEXUS | $2m(d+b)/(N/4m)$ | 7680 |
| | THOR | $m + 2(b+d)/c$ | 608 |
| | Powerformer | $\approx 2\sqrt{2} \cdot \sqrt{\frac{mdb}{N/2}}$ | 272 |
| | Euston/MOAI | 0 | 0 |
| | **STIP** | 0 | **0** |
| Total | NEXUS | – | $> 18444$ |
| | THOR | – | 1868 |
| | Powerformer | – | 1417 |
| | Euston | – | 1542 |
| | MOAI | – | 495 |
| | **STIP** | – | **$\approx$ 264** |

**Table 6:** Amortized number of rotations incurred by nonlinear operations in one transformer layer under batch size $t = 256$: (1) Attention activation; and (2) two LN operations. The concrete parameters are $m = 128$, $d = 768$, $H = 12$, and $N = 2^{16}$.

| ID | Scheme | Rotation | |
|---|---|---|---|
| | | Equation | Number |
| (1) | NEXUS/THOR | $\lceil \log_2(m) \rceil \frac{m}{\frac{N}{4}/2^{\lceil \log_2(m) \rceil}} \cdot H$ | 84 |
| | Powerformer | 0 | 0 |
| | Euston/MOAI | 0 | 0 |
| | **STIP** | 0 | **0** |
| (2) | NEXUS/THOR | $\lceil \log_2(d) \rceil \frac{2d}{\frac{N}{4}/2^{\lceil \log_2(d) \rceil}}$ | 960 |
| | Powerformer | $2\log_2(N/(2m))$ | 16 |
| | Euston/MOAI | 0 | 0 |
| | **STIP** | 0 | **0** |
| Total | NEXUS/THOR | – | 1044 |
| | Powerformer | – | 16 |
| | Euston/MOAI | – | 0 |
| | **STIP** | – | **0** |

Algorithms 2 and 8 introduce a small number of additional rotations in PCMMs and LN, this overhead is negligible compared with the performance gains from AMCP's parallel computation.

**Performance Comparison with NEXUS, THOR, Powerformer, Euston, and MOAI.** Tables 5 and 6 separately report the key-switching operations incurred by matrix multiplications and the rotations incurred by nonlinear layers under batch size $t = 256$. Combining these matrix-multiplication key switches and nonlinear rotations gives 264 operations for STIP, with reductions of 98.6%, 90.9%, 81.6%, 82.9%, and 46.7% relative to NEXUS, THOR, Powerformer, Euston, and MOAI, respectively. We analyze the breakdown as follows.

*PCMMs:* NEXUS, THOR, and Powerformer incur substantial key-switching overheads in PCMMs because their packing layouts require slot alignment and accumulation operations. At $t = 256$, STIP, Euston, and MOAI require no rotations in these PCMMs. For STIP, DHP still incurs $(3d/2)/(N/(2m)) = 4.5$ amortized conjugations for unpacking all three input projections and $d/(N/(2m)) = 3$ for extracting the real output. This conjugation overhead is negligible compared with the computational savings achieved by halving the number of PCMMs with DHP.

*CCMMs Bottleneck:* CCMMs are the primary bottleneck for Euston, MOAI, and STIP. Euston requires 1,542 key-switching operations for $\boldsymbol{QK}^\top$ and $\boldsymbol{SV}$, while MOAI requires 495. STIP reduces this overhead to 256 through the RIHP strategies. Specifically, the key-switching cost of $\boldsymbol{QK}^\top$ decreases from 387 in MOAI to 198 in STIP, while that of $\boldsymbol{SV}$ decreases from 108 to 58. These results correspond to reductions of 48.8% and 46.3%, respectively.

*Nonlinear Layers:* NEXUS and THOR require 1,044 rotations for the attention activation and the two LN operations. Powerformer's BPMax requires no rotation, but its two Batch LN operations require 16 rotations. Euston, MOAI, and STIP incur no nonlinear rotations.

## 6.2 Experimental Setup

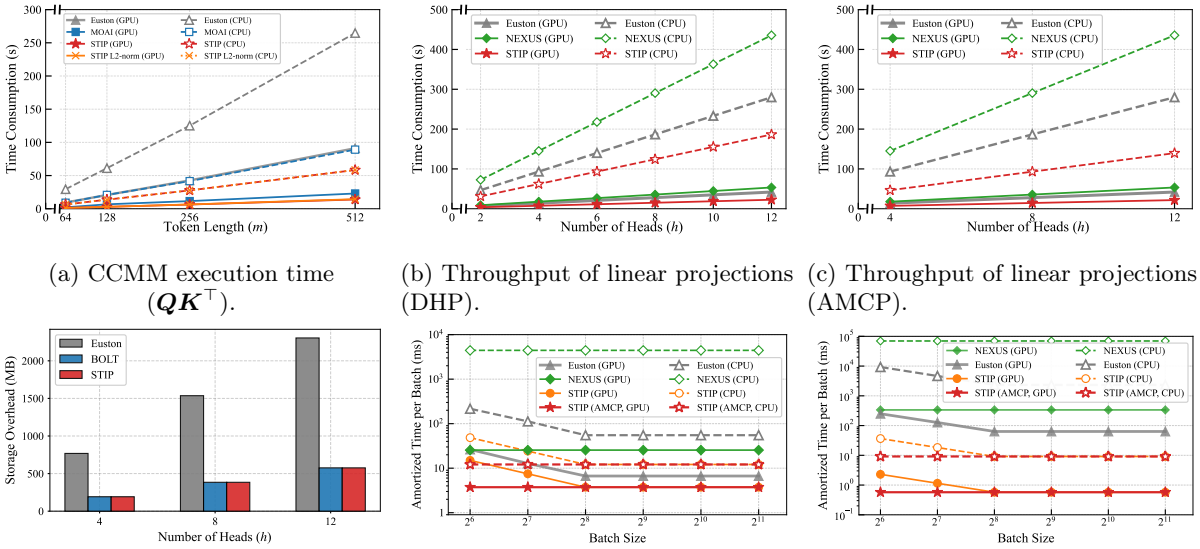
Building on the architectures of NEXUS[1] and Euston[2], we implemented STIP in C++ with support for both GPU and CPU execution. Similar to NEXUS and Euston, STIP implements the RNS-CKKS scheme using Phantom [39] for the GPU and Microsoft SEAL [40] for the CPU.

[1] https://github.com/zju-abclab/NEXUS
[2] https://github.com/FLL-Lab/Euston

Parameter Configuration: In the end-to-end inference, we configured the polynomial degree to $N = 2^{16}$ and utilized a 1763-bit composite modulus, which ensures a security level of 128 bits [41]. We adopted the bootstrapping used in Euston. In the micro-benchmarks, we configured $N = 2^{16}$. The composite modulus and modulus chain length are configured according to the micro-benchmark settings.

GPU Environment: GPU acceleration was evaluated using an NVIDIA A100 featuring 80 GB of VRAM, operating within the CUDA 12.2 runtime environment.

CPU Environment: Experiments were executed on an AMD EPYC 7542 processor (2.90 GHz). The system ran Ubuntu 22.04.1 LTS. The thread count was set at 32.




(a) CCMM execution time ($\boldsymbol{QK}^\top$).

(b) Throughput of linear projections (DHP).

(c) Throughput of linear projections (AMCP).

(d) Storage overhead of $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$.

(e) GK/Softmax amortized runtime.

(f) LN amortized runtime.

**Figure 6:** Micro-benchmark results in the GPU and CPU environments.

## 6.3 Micro-benchmarks

We evaluated STIP against other batch packing schemes via GPU and CPU micro-benchmarks on BERT-base model, as illustrated in Figure 6.

**CCMM for Attention Scores ($\boldsymbol{QK}^\top$).** This benchmark evaluates the execution time of CCMM for $\boldsymbol{QK}^\top$. We configured STIP with a modulus chain length of $L = 2$ to support the imaginary unit operation and conjugate unpacking, compared to MOAI ($L = 1$) and Euston ($L = 2$). As shown in Figure 6 (a), STIP significantly outperforms the baselines. Across varying token lengths, STIP achieves speedups of 1.86× on GPU and 1.5× on CPU compared to MOAI. Against Euston, STIP achieves speedups of 6.92× on GPU and 4.54× on CPU. Furthermore, we evaluated the computational cost of the L2-norm using the second RIHP strategy, where its execution time is nearly identical to the $\boldsymbol{QK}^\top$ computation in STIP.

**Throughput of Linear Projections.** We assess the throughput of linear projections ($\boldsymbol{XW}_O, O \in \{\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}\}$) using DHP and AMCP strategies separately. First, under standard settings ($m = 128, t = 256$), STIP with DHP consistently outperforms baselines with a higher modulus chain $L = 2$ ($L = 1$ for baselines) for conjugate unpacking. As shown in Figure 6 (b), STIP achieves speedups of 1.86×/2.36× on GPU and 1.50×/2.34× on CPU compared to Euston/NEXUS, respectively. Second, under a small batch size ($t = 64$), we configured STIP with a higher modulus chain length of $L = 2$ ($L = 1$ for baselines) to accommodate the additional

operations in Algorithm 2. As shown in Figure 6 (c), AMCP maximizes slot utilization, achieving speedups of 1.93×/2.46× on GPU and 2.00×/3.12× on CPU compared to Euston/NEXUS, respectively. Specifically, AMCP applies to the entire transformer. The cumulative performance benefits are demonstrated in our end-to-end experiments.

**Storage Overhead.** We evaluate the memory footprint for storing ciphertexts $(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V})$ with $m = 128$ and $t = 64$. Leveraging the AMCP strategy, STIP packs four feature columns per ciphertext, achieving SOTA memory efficiency comparable to BOLT. As shown in Figure 6 (d), for a 12-head configuration, STIP requires only 576 MB, while Euston requires 4× as much storage (2,304 MB).

**Efficiency of Nonlinear Layers.** For GK, STIP reduces the modulus chain to $L = 4$ ($L = 10/16$ for Euston/NEXUS), yielding 1.78×/6.96× speedups on GPU. For LN, our inverse-free approach reduces the depth to $L = 4$ ($L = 9/16$ for Euston/NEXUS), achieving a 110.29×/585.16× reduction in amortized execution time on GPU. Notably, AMCP is applied to both GK and LN to maintain constant low latency even at small batch sizes ($2^6$), eliminating the latency spikes observed in Euston.

**Table 7:** Inference accuracy evaluation on GLUE benchmark.

| Model | Dataset | Baseline | | | Ours | |
|---|---|---|---|---|---|---|
| | | Plaintext | NEXUS | Euston | plaintext | STIP |
| BERT-base | RTE | 70.76% | 70.60% | 70.76% | 72.20% | 72.06% |
| | SST-2 | 92.66% | 92.41% | 92.55% | 92.73% | 92.62% |
| | QNLI | 91.67% | 91.27% | 91.52% | 91.21% | 91.06% |
| LLaMA-3-8B | RTE | 82.75% | 81.24% | 81.56% | 82.46% | 81.32% |
| | SST-2 | 94.94% | 94.46% | 94.67% | 94.39% | 94.01% |
| | QNLI | 90.70% | 90.20% | 90.38% | 90.36% | 90.12% |

## 6.4 Accuracy

To evaluate inference accuracy, we assessed STIP on the RTE, SST-2, and QNLI datasets from the GLUE benchmark [42]. We employed BERT-base and LLaMA-3-8B fine-tuned for each task. STIP applies GK and ReLU in all transformer layers. As shown in Table 7, STIP exhibits acceptable deviations from the plaintext baseline and achieves comparable performance to Euston and NEXUS. We also evaluated the inference accuracy of GPT-2-1.5B in Table 11 of Appendix E.

## 6.5 End-to-end Inference Evaluation

**Comparisons with Euston.** Consistent with the evaluation methodologies in Euston[3], we analyze the runtime breakdown of STIP and Euston on BERT-base, LLaMA-3-8B, and GPT-2-1.5B, covering all transformer layers. Following the experimental setting in Euston, both frameworks are configured with $t = 32$ and $m = 128$. As detailed in Table 8, STIP demonstrates a significant advantage in end-to-end inference, achieving approximately 8.5× speedup over Euston. Table 10 in Appendix E extends this comparison to LLaMA-3-8B and GPT-2-1.5B, showing approximately 11.3× and 9.0× end-to-end speedups over Euston, respectively.

[3]We use the same batch size ($t = 32$), Phantom library, and measurement methodology as Euston, with the modulus chain set to the absolute depth consumed.

**Table 8:** Breakdown of execution time (s) on an A100 GPU compared to Euston under BERT-base with batch size $t = 32$.

| **STIP** | | | **Euston** | | |
|---|---|---|---|---|---|
| **Operation** | **Depth** | **Time** | **Operation** | **Depth** | **Time** |
| Input projection | 15→12 | 1.80 | Input projection | 17→16 | 15.72 |
| Attention score | 12→10 | 2.16 | Attention score | 16→14 | 92.88 |
| GK | 10→6 | 0.72 | Softmax | 14→4 | 7.68 |
| Attention output | 6→4 | 0.48 | Attention output | 4→2 | 90.48 |
| Output projection | 4→1 | 0.96 | Output projection | 2→1 | 5.76 |
| Bootstrappings | 1→5 | 23.04 | Bootstrappings | 1→10 | 188.76 |
| LN1 | 5→1 | 0.048 | LN1 | 10→1 | 6.00 |
| Bootstrappings | 1→11 | 30.48 | Bootstrappings | 1→15 | 222.12 |
| FC1 | 11→10 | 3.05 | FC1 | 15→14 | 20.76 |
| Activation layer | 10→2 | 4.92 | Activation layer | 14→2 | 87.36 |
| FC2 | 2→1 | 3.39 | FC2 | 2→1 | 20.88 |
| Bootstrappings | 1→11 | 30.48 | Bootstrappings | 1→10 | 188.76 |
| LN2 | 11→1 | 0.084 | LN2 | 10→1 | 6.00 |
| Bootstrappings | 1→15 | 35.40 | Bootstrappings | 1→17 | 242.04 |
| Classification | - | 3.5 | Classification | - | 3.5 |
| **Total** | - | **140.51** | **Total** | - | **1198.70** |

*Runtime breakdown analysis:* We further analyze the runtime breakdown reported in Table 8. The performance disparity is primarily driven by optimizations in the attention layer, nonlinear layer, and bootstrapping:

① Attention Layer: STIP exhibits lower latency across the attention layer, reducing the total execution time from Euston's 212.52 s to 6.12 s, achieving a 34.7× speedup. This overall reduction in execution time is driven by our AMCP, which enables a single ciphertext to process feature columns from multiple attention heads in parallel. Building on this foundation, the DHP strategy accelerates the throughput of input and output projections, while the RIHP strategy speeds up computations of the attention score and attention output.

② Nonlinear Layer: By replacing Softmax with the GK and leveraging AMCP for parallel execution, STIP reduces the runtime from 7.68 s to 0.72 s (10.7× speedup). Additionally, optimizations in LN1 eliminate inverse operations, reducing the runtime from 6.00 s to 0.048 s (125× speedup).

③ Bootstrapping Optimizations: STIP achieves a total bootstrapping time of 119.40 s, representing a 7.0× speedup over Euston's 841.68 s. AMCP maintains a compact ciphertext layout throughout the transformer layer, including attention layer, LN, and FFN computations. As a result, each of the four bootstrappings processes 128 ciphertexts at each transformer layer, versus 768 for Euston. The reduced depth of nonlinear layers further lowers bootstrapping costs.

**Comparisons with MOAI, THOR, and Powerformer.** Consistent with the evaluation methodologies in MOAI[4], we analyze the runtime breakdown of STIP, MOAI, THOR, and Powerformer on BERT-base. As detailed in Table 9, STIP achieves approximately 2.3×, 4.8×, and 2.7× speedups over MOAI, THOR, and Powerformer at batch size $t = 256$. Depth transitions for THOR and Powerformer are omitted because their original papers do not report them.

The breakdown against MOAI highlights the contributions of RIHP, DHP, and the nonlinear optimizations. RIHP reduces CCMM rotations, while DHP improves projection throughput.

[4]We use the same batch size ($t = 256$), Phantom library, and measurement methodology as MOAI.

They lower attention-score and input/output-projection times from MOAI's 14.07/17.02 s to 6.50/7.45 s. GK and LN cut their combined runtime from 2.49 s to 0.78 s while lowering the required multiplicative depth. This lower depth requirement also reduces bootstrapping time from 227.84 s to 86.34 s.

**Table 9:** Breakdown of execution time (s) on an A100 GPU compared to MOAI, THOR, and Powerformer under BERT-base with batch size $t = 256$.

| Operation | STIP | MOAI | THOR | Powerformer |
|---|---|---|---|---|
| | Time (Depth) | Time (Depth) | Time | Time |
| Input projection | 7.20 (13 → 11) | 16.54 (15 → 14) | 49.77 | 57.65 |
| Attention score | 6.50 (11 → 9) | 14.07 (14 → 13) | 16.25 | 28.76 |
| GK / Softmax | 0.74 (9 → 5) | 2.19 (13 → 3) | 15.53 | 0.75 |
| Attention output | 0.20 (5 → 3) | 0.29 (3 → 2) | 13.08 | 18.95 |
| Output projection | 0.25 (3 → 1) | 0.48 (2 → 1) | 27.43 | 22.54 |
| LN1 | 0.01 (4 → 1) | 0.15 (21 → 1) | 7.13 | 0.37 |
| FC1 | 17.51 (11 → 10) | 16.18 (10 → 9) | 49.8 | 59.21 |
| Activation layer | 3.96 (10 → 2) | 3.3 (9 → 2) | 29.42 | 8.31 |
| FC2 | 2.88 (2 → 1) | 2.88 (2 → 1) | 49.19 | 43.77 |
| LN2 | 0.03 (9 → 1) | 0.15 (21 → 1) | 4.1 | 0.3 |
| Bootstrappings | 86.34 (1 → 37) | 227.84 (1 → 67) | 337.86 | 103.72 |
| Classification | 0.7 | 0.7 | 2.7 | 0.2 |
| **Total** | **126.32** | **284.77** | **602.26** | **344.53** |

# 7 Related Work

## 7.1 Packing Method and Matrix Multiplication

**Single-Sequence Packing Methods.** Schemes like THOR and Powerformer operate in a single-input style. They pack data into a single ciphertext to leverage SIMD parallelism. While this approach achieves much lower absolute latency for an isolated request, its runtime typically increases linearly with the number of sequence tokens. Furthermore, these methods suffer from spatial misalignment, necessitating rotation-intensive PCMMs and nonlinear aggregation. Specifically, THOR's diagonal strategy incurs key-switching overheads for non-square projections. Regarding complex-domain packing, early work [43] utilized the imaginary part for tensors. While Powerformer also utilizes the complex domain, our optimization goals differ fundamentally. For optimized CCMMs, based on [44], Powerformer uses complex conjugates to aggregate independent matrix products, which leaves the rotation complexity unchanged. In contrast, our RIHP provides an intrinsic algorithmic optimization, fundamentally reducing the key-switching overhead of $\boldsymbol{QK}^\top$ by approximately 50%. Regarding optimized PCMMs, Powerformer reduces rotations in compact packing by pairing vectors, yet the rotational overhead remains. Conversely, our DHP exploits the complex domain within a rotation-free column-wise strategy, processing two heads simultaneously to double throughput.

**Batch Packing Methods.** Schemes like NEXUS, MOAI, and Euston support batch inference, processing multiple sequences in a single execution. Although this batch style incurs a longer wall-clock time, it utilizes larger batch sizes to amortize the cost of expensive homomorphic operations, significantly reducing the average per-sequence latency. Moreover, MOAI and Euston achieve rotation-free PCMMs and nonlinear layers by packing each feature column into a distinct ciphertext. However, this incurs severe slot underutilization, low throughput and an

excessive number of ciphertexts. Furthermore, computing CCMMs still necessitates substantial homomorphic rotations, dominating the total rotation overhead.

### 7.2 Optimization of Nonlinear Layers

**Softmax Optimizations.** Current strategies face three limitations. Fixed-maximum simplification (NEXUS, MOAI, Euston) uses a predetermined constant to avoid maximum search but introduces severe numerical instability (overflow/underflow) and difficulty in evaluating row-wise maxima in sparse layouts. Iterative precision (THOR) maintains precision but relies on computationally intensive iterative algorithms for inverse approximation, resulting in excessive multiplicative depth and key-switching operations. Polynomial Replacement (Powerformer) replaces Softmax with BPMax via knowledge distillation, treating non-linearity as a black box and limiting theoretical robustness.

**LN Optimizations.** Optimization efforts generally fall into two categories. Direct Homomorphic Approximation (NEXUS, THOR, MOAI, Euston) frameworks evaluate the standard LN function, which is bottlenecked by the high multiplicative depth required for the inverse square root. Distillation-based Replacement (Powerformer) replaces dynamic statistics with learned constants via knowledge distillation. It necessitates a data-dependent re-training process, limiting its generality for various data and models.

## 8 Conclusion

In this paper, we presented STIP, a comprehensive framework designed to overcome the critical computational and storage bottlenecks in secure non-interactive transformer inference. By leveraging RIHP and DHP, we halve the rotation overhead in CCMMs and the execution time of input and output projections. Concurrently, our AMCP strategy resolves severe slot wastage to maximize parallel throughput while significantly reducing the ciphertext footprint. We replace Softmax with GK to ensure efficiency and stability by eliminating the need for both maximum value search and inverse operation. Additionally, we reformulate the first LN into an inverse-free form to reduce the multiplicative depth and expensive key-switch operations. Compared with Euston, STIP requires 20.6× fewer rotations per transformer layer and achieves a 34.7× attention-layer speedup. Experimental evaluations on BERT-base demonstrate that STIP achieves an 8.5× end-to-end speedup over Euston at batch size $t = 32$, and 2.3×, 4.8×, and 2.7× over MOAI, THOR, and Powerformer at $t = 256$, respectively.

## References


[1] A. Vaswani *et al.*, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5998–6008.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics: Human Lang. Technol. (NAACL-HLT)*, 2019, pp. 4171–4186.

[3] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI Blog*, vol. 1, no. 8, p. 9, 2019.

[4] H. Touvron *et al.*, "LLaMA: Open and efficient foundation language models," 2023, arXiv:2302.13971.

[5] C. Chen, L. Wei, J. Xie, and Y. Shi, "Privacy-preserving machine learning based on cryptography: a survey," *ACM Trans. Knowl. Discovery Data*, vol. 19, no. 4, pp. 1–33, 2025.

[6] R. Lehmkuhl, P. Mishra, A. Srinivasan, and R. A. Popa, "MUSE: Secure inference resilient to malicious clients," in *Proc. USENIX Secur. Symp. (USENIX Security)*, 2021, pp. 2201–2218.

[7] B. Rao, J. Zhang, D. Wu, C. Zhu, X. Sun, and B. Chen, "Privacy inference attack and defense in centralized and federated learning: a comprehensive survey," *IEEE Trans. Artif. Intell.*, vol. 6, no. 2, pp. 333–353, 2025.

[8] J. Kwesi, J. Cao, R. Manchanda, and P. Emami-Naeini, "Exploring user security and privacy attitudes and concerns toward the use of general-purpose LLM chatbots for mental health," in *Proc. USENIX Secur. Symp. (USENIX Security)*, 2025, pp. 6007–6024.

[9] F. Bourse, R. Del Pino, M. Minelli, and H. Wee, "FHE circuit privacy almost for free," in *Advances in Cryptology—CRYPTO 2016*, 2016, pp. 62–89.

[10] Y. Chen *et al.*, "SoK: Private transformer-based model inference," in *Proc. USENIX Secur. Symp. (USENIX Security)*, 2026, pp. 1127–1146.

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

[12] K. Gupta *et al.*, "SIGMA: Secure GPT inference with function secret sharing," *Proc. Privacy Enhancing Technol.*, vol. 2024, no. 4, pp. 61–79, 2024.

[13] W.-J. Lu *et al.*, "BumbleBee: Secure two-party inference framework for large transformers," in *Proc. Netw. Distrib. Syst. Secur. Symp. (NDSS)*, 2025.

[14] T. Xu *et al.*, "Breaking the layer barrier: Remodeling private transformer inference with hybrid CKKS and MPC," in *Proc. USENIX Secur. Symp. (USENIX Security)*, 2025, pp. 2653–2672.

[15] A. Y. L. Kei and S. S. M. Chow, "SHAFT: Secure, handy, accurate, and fast transformer inference," in *Proc. Netw. Distrib. Syst. Secur. Symp. (NDSS)*, 2025.

[16] J. Zhang *et al.*, "Secure transformer inference made non-interactive," in *Proc. Netw. Distrib. Syst. Secur. Symp. (NDSS)*, 2025.

[17] D. Park, E. Lee, and J.-W. Lee, "Powerformer: Efficient and High-Accuracy Privacy-Preserving Language Model with Homomorphic Encryption," in *Proc. 63rd Annu. Meeting Assoc. Comput. Linguistics*, 2025, pp. 11090–11111.

[18] J. Moon, D. Yoo, X. Jiang, and M. Kim, "THOR: Secure transformer inference with homomorphic encryption," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2025, pp. 3765–3779.

[19] L. Zhang *et al.*, "MOAI: Module-optimizing architecture for non-interactive secure transformer inference," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2026.

[20] X. Gao, S. Fu, L. Liu, Z. Liu, Y. Luo, and Y. Wang, "Euston: Efficient and user-friendly secure transformer inference with non-interactivity," in *Proc. IEEE Symp. Secur. Privacy (SP)*, 2026, pp. 1443–1462.

[21] W. Zeng *et al.*, "Towards efficient privacy-preserving machine learning: A systematic review from protocol, model, and system perspectives," 2025, arXiv:2507.14519.

[22] A. Viand, P. Jattke, M. Haller, and A. Hithnawi, "HECO: Fully homomorphic encryption compiler," in *Proc. USENIX Secur. Symp. (USENIX Security)*, 2023, pp. 4715–4732.

[23] Microsoft, "Batch inference," Microsoft Research, 2023.

[24] A. Adir, E. Aharoni, N. Drucker, R. Levy, H. Shaul, and O. Soceanu, *Homomorphic Encryption for Data Science (HE4DS)*. Springer, 2024.

[25] E. Aharoni *et al.*, "HeLayers: A tile tensors framework for large neural networks on encrypted data," *Proc. Privacy Enhancing Technol.*, vol. 2023, no. 1, pp. 325–342, 2023.

[26] S. Halevi and V. Shoup, "Bootstrapping for HElib," *J. Cryptol.*, vol. 34, no. 1, Art. no. 7, 2021.

[27] G.-I. Yu, J. S. Jeong, G.-W. Kim, S. Kim, and B.-G. Chun, "Orca: A distributed serving system for Transformer-Based generative models," in *Proc. 16th USENIX Symp. Oper. Syst. Des. Implement. (OSDI)*, 2022, pp. 521–538.

[28] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica, "Clipper: A low-latency online prediction serving system," in *Proc. 14th USENIX Symp. Netw. Syst. Des. Implement. (NSDI)*, 2017, pp. 613–627.

[29] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Advances in Cryptology—ASIACRYPT 2017: 23rd Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, 2017, pp. 409–437.

[30] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "A full RNS variant of approximate homomorphic encryption," in *Selected Areas in Cryptography—SAC 2018: 25th Int. Conf.*, 2019, pp. 347–368.

[31] J. Lu *et al.*, "SOFT: Softmax-free transformer with linear complexity," in *Proc. Adv. Neural Inf. Process. Syst.*, 2021, pp. 21297–21309.

[32] Y. Chen, Q. Zeng, H. Ji, and Y. Yang, "Skyformer: Remodel self-attention with Gaussian kernel and Nyström method," in *Proc. Adv. Neural Inf. Process. Syst.*, 2021, pp. 2122–2135.

[33] E. Lee *et al.*, "Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions," in *Proc. 39th Int. Conf. Mach. Learn.*, 2022, pp. 12403–12422.

[34] E. Lee, J.-W. Lee, J.-S. No, and Y.-S. Kim, "Minimax approximation of sign function by composite polynomial for homomorphic comparison," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 6, pp. 3711–3727, 2022.

[35] J. H. Cheon, D. Kim, and D. Kim, "Efficient homomorphic comparison methods with optimal complexity," in *Advances in Cryptology—ASIACRYPT 2020*, 2020, pp. 221–256.

[36] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction APIs," in *Proc. 25th USENIX Security Symp. (USENIX Security)*, 2016, pp. 601–618.

[37] D. Rho *et al.*, "Encryption-friendly LLM architecture," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2025.

[38] S. I. Mirzadeh *et al.*, "ReLU strikes back: Exploiting activation sparsity in large language models," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2024.

[39] H. Yang, S. Shen, W. Dai, L. Zhou, Z. Liu, and Y. Zhao, "Phantom: A CUDA-accelerated word-wise homomorphic encryption library," *IEEE Trans. Dependable Secure Comput.*, vol. 21, no. 5, pp. 4895–4906, 2024.

[40] Microsoft, "Microsoft SEAL, release 4.1," Microsoft Research, 2023.

[41] M. Albrecht *et al.*, "Homomorphic encryption standard," in *Protecting Privacy through Homomorphic Encryption*, 2021, pp. 31–62.

[42] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "GLUE: A multi-task benchmark and analysis platform for natural language understanding," in *Proc. EMNLP Workshop BlackboxNLP*, 2018, pp. 353–355.

[43] E. Aharoni, N. Drucker, G. Ezov, H. Shaul, and O. Soceanu, "Complex encoded tile tensors: Accelerating encrypted analytics," *IEEE Secur. & Privacy*, vol. 20, no. 5, pp. 35–43, 2022.

[44] X. Jiang, M. Kim, K. Lauter, and Y. Song, "Secure outsourced matrix computation and application to neural networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2018, pp. 1209–1222.

[45] W. Cho, G. Hanrot, T. Kim, M. Park, and D. Stehlé, "Fast and accurate homomorphic softmax evaluation," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2024, pp. 4391–4404.

# A The Second RIHP Strategy

We introduce the second RIHP strategy in this section. This strategy can be applied to two CCMMs ($\boldsymbol{QK}^\top$, $\boldsymbol{SV}$) in the attention mechanism.

Algorithm 4 provides the pseudocode of the second RIHP strategy to efficiently compute the attention score $\boldsymbol{QK}^\top$. Different from Algorithm 1, we pack pairs of feature columns into the real and imaginary components of a single complex ciphertext, such that $[\![\hat{\boldsymbol{q}}_j]\!] \leftarrow [\![\boldsymbol{q}_j]\!] \boxplus i \cdot [\![\boldsymbol{q}_{j+d'/2}]\!]$ and $[\![\hat{\boldsymbol{k}}_j]\!] \leftarrow [\![\boldsymbol{k}_j]\!] \boxminus i \cdot [\![\boldsymbol{k}_{j+d'/2}]\!]$. Leveraging the algebraic identity $\mathrm{Re}(z \cdot \bar{w}) = \mathrm{Re}(z)\mathrm{Re}(w) + \mathrm{Im}(z)\mathrm{Im}(w)$, the conjugate multiplication $[\![\hat{\boldsymbol{q}}_j]\!] \boxtimes \textsc{RotL}_r([\![\hat{\boldsymbol{k}}_j]\!])$ naturally aggregates the dot product contributions from both feature column pairs (i.e., $\boldsymbol{q}_j\boldsymbol{k}_j + \boldsymbol{q}_{j+d'/2}\boldsymbol{k}_{j+d'/2}$) into the real component, while isolating the cross-terms in the imaginary part. Crucially, this method reduces the number of key-switching operations from $(m-1)d' + m$ to $(m-1)d'/2 + 2m$.

Algorithm 5 shows the Baby-Step Giant-Step (BSGS)-optimized CCMM for $\boldsymbol{SV}$ by incorporating the second RIHP strategy, reducing the number of key-switching operations from $3\sqrt{md'} + m$ to $3\sqrt{md'}/2 + m + d'/2$.

# B Details of the AMCP

**Input Weight Construction and Aggregation.** To perform the input projection over packed ciphertexts, for each projection $\boldsymbol{O} \in \{\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}\}$ and head-group start $h \in \{0, k, \ldots, H-k\}$, we compute the $j$-th packed output column, $j \in \{0, \ldots, d'-1\}$, as follows.

$$[\![\dot{\boldsymbol{o}}_j^{(h)}]\!] = \sum\nolimits_{g=0}^{G-1} \sum\nolimits_{r=0}^{k-1} [\![\boldsymbol{C}_g^{(r)}]\!] \boxtimes \boldsymbol{w}_{O,g,j}^{(h,r)}. \tag{32}$$

Algorithm 6 constructs the packed weight vectors $\boldsymbol{w}_{O,g,j}^{(h,r)}$ from weights $w_{i,j}^{(\nu)} = (\boldsymbol{W}_O^{(\nu)})_{i,j}$, where $\nu \in \{h, \ldots, h+k-1\}$. The algorithm is invoked separately for each projection and head group. After aggregation, the input projection yields $[\![\dot{\boldsymbol{O}}^{(h)}]\!] = [[\![\dot{\boldsymbol{o}}_0^{(h)}]\!], \ldots, [\![\dot{\boldsymbol{o}}_{d'-1}^{(h)}]\!]]$, where each ciphertext $[\![\dot{\boldsymbol{o}}_j^{(h)}]\!]$ packs the $j$-th feature columns of heads $h, \ldots, h+k-1$. Algorithm 6 operates on plaintext weights and can be executed offline, so its weight construction incurs no online computational overhead.

**Output Weight Construction and Aggregation.** Let $\boldsymbol{Y} \in \mathbb{R}^{tm \times d}$ denote the output projection summed over all $H$ heads. We use $G = d/k$ ciphertexts and stride $\varphi = S/m$. For each group starting at head $h \in \{0, k, \ldots, H-k\}$, ciphertext $[\![\dot{\boldsymbol{att}}_i^{(h)}]\!]$, $i \in \{0, \ldots, d'-1\}$, contains the $i$-th attention-output columns of heads $h, \ldots, h+k-1$. Figure 4 (upper right) shows their order within each stride, with $t$ batch slots per head. Algorithm 2 cyclically shifts these blocks, preserving sequence and batch indices and yielding the aligned states $\{[\![\dot{\boldsymbol{att}}_i^{(h,r)}]\!]\}_{r=0}^{k-1}$, where $\dot{\boldsymbol{att}}_i^{(h,0)} = \dot{\boldsymbol{att}}_i^{(h)}$.

The target output ciphertext $[\![\dot{\boldsymbol{y}}_g]\!]$, $g \in \{0, \ldots, G-1\}$, interleaves columns $gk, \ldots, gk+k-1$ of $\boldsymbol{Y}$, as shown by the two vectors at the right of Figure 4. Within the stride for sequence position $s$, block $a$ holds the $t$ batch values of output column $gk+a$. Thus, batch $\beta$ occupies slot $p = s\varphi + at + \beta$, where $s \in \{0, \ldots, m-1\}$, $a \in \{0, \ldots, k-1\}$, and $\beta \in \{0, \ldots, t-1\}$. To compute this output slot, aligned attention state $r$ supplies a value from head $h + ((a+r) \bmod k)$. Algorithm 7 matches it to the weight for output column $gk+a$:

$$\boldsymbol{w}_{0,i,g}^{(h,r)}[p] = w_{i,gk+a}^{(h+((a+r) \bmod k))}, \tag{33}$$

where $w_{i,j}^{(\nu)}$ is entry $(i,j)$ of the output-projection matrix $\boldsymbol{W}_0^{(\nu)}$ for head $\nu \in \{0, \ldots, H-1\}$, with $j \in \{0, \ldots, d-1\}$. Algorithm 7 copies each weight to batch slots $at$ through $(a+1)t-1$

**Algorithm 4** The Second RIHP Strategy

**Input:** Query $[\![\boldsymbol{Q}]\!] = \{[\![\boldsymbol{q}_j]\!]\}_{j=0}^{d'-1}$, key $[\![\boldsymbol{K}]\!] = \{[\![\boldsymbol{k}_j]\!]\}_{j=0}^{d'-1}$.
**Output:** Encrypted diagonals of $\boldsymbol{QK}^\top$: $\{[\![\boldsymbol{P}_r]\!]\}_{r=0}^{m-1}$.
**for** $j = 0, \ldots, d'/2 - 1$ **do**
    $[\![\hat{\boldsymbol{q}}_j]\!] \leftarrow [\![\boldsymbol{q}_j]\!] \boxplus i \cdot [\![\boldsymbol{q}_{j+d'/2}]\!]$
    $[\![\hat{\boldsymbol{k}}_j]\!] \leftarrow [\![\boldsymbol{k}_j]\!] \boxminus i \cdot [\![\boldsymbol{k}_{j+d'/2}]\!]$
**end for**
**for** $r = 0, \ldots, m - 1$ **do**
    $[\![\boldsymbol{V}_r]\!] \leftarrow \sum_{j=0}^{d'/2-1} [\![\hat{\boldsymbol{q}}_j]\!] \boxtimes \textsc{RotL}_r([\![\hat{\boldsymbol{k}}_j]\!])$
    $[\![\boldsymbol{P}_r]\!] \leftarrow ([\![\boldsymbol{V}_r]\!] \boxplus \textsc{Conj}([\![\boldsymbol{V}_r]\!])) \boxtimes 0.5$
**end for**
**return** $\{[\![\boldsymbol{P}_r]\!]\}_{r=0}^{m-1}$

**Algorithm 5** Attention Output with the Second RIHP

**Input:** Activation output $[\![\boldsymbol{S}]\!] = \{[\![\boldsymbol{s}_k]\!]\}_{k=0}^{m-1}$, value matrix $[\![\boldsymbol{V}]\!] = \{[\![\boldsymbol{v}_k]\!]\}_{k=0}^{d'-1}$.
**Output:** Encrypted columns of $\boldsymbol{SV}$: $\{[\![\boldsymbol{att}_j]\!]\}_{j=0}^{d'-1}$.
$b \leftarrow \lceil \sqrt{m} \rceil$, $g \leftarrow \lceil m/b \rceil$
**for** $k = 0, \ldots, d'/2 - 1$ **do**
    $[\![\hat{\boldsymbol{v}}_k]\!] \leftarrow [\![\boldsymbol{v}_k]\!] \boxplus i \cdot [\![\boldsymbol{v}_{k+d'/2}]\!]$
**end for**
**for** $k = 0, \ldots, d'/2 - 1$ **do**
    **for** $r = 0, \ldots, b - 1$ **do**
        $\boldsymbol{\beta}_r \leftarrow \textsc{RotL}_r([\![\hat{\boldsymbol{v}}_k]\!])$
    **end for**
    $[\![\hat{\boldsymbol{att}}_k]\!] \leftarrow \sum_{\alpha=0}^{g-1} \textsc{RotL}_{\alpha b}\big(\sum_{r=0}^{b-1} \textsc{RotL}_{m-\alpha b}([\![\boldsymbol{s}_{\alpha b+r}]\!]) \boxtimes \boldsymbol{\beta}_r\big)$
    $[\![\boldsymbol{att}_k]\!] \leftarrow \big([\![\hat{\boldsymbol{att}}_k]\!] \boxplus \textsc{Conj}([\![\hat{\boldsymbol{att}}_k]\!])\big) \boxtimes 0.5$
    $[\![\boldsymbol{att}_{k+d'/2}]\!] \leftarrow \big([\![\hat{\boldsymbol{att}}_k]\!] \boxminus \textsc{Conj}([\![\hat{\boldsymbol{att}}_k]\!])\big) \boxtimes (-0.5i)$
**end for**
**return** $\{[\![\boldsymbol{att}_j]\!]\}_{j=0}^{d'-1}$

(inclusive), using the length-$t$ all-ones vector $\mathbf{1}_t$. Each plaintext weight vector is precomputed offline by repeating the zero-padded stride $m$ times. We evaluate

$$[\![\dot{\boldsymbol{y}}_g]\!] = \sum_{h \in \{0,k,\ldots,H-k\}} \sum_{i=0}^{d'-1} \sum_{r=0}^{k-1} [\![\dot{\boldsymbol{att}}_i^{(h,r)}]\!] \boxtimes \boldsymbol{w}_{0,i,g}^{(h,r)}. \tag{34}$$

For fixed $a$, varying $r$ covers each head in the group once. Thus, Eq. (34) yields $\dot{\boldsymbol{y}}_g[p] = (\boldsymbol{Y}^{(\beta)})_{s,gk+a}$, where $\boldsymbol{Y}^{(\beta)}$ is the projected output for batch $\beta$. The $G$ output ciphertexts therefore match the interleaved input layout.

Algorithm 8 reuses the aligned states of each attention ciphertext across all output groups, avoiding repeated rotations. The matching layouts allow the residual addition $[\![\dot{\boldsymbol{y}}_g]\!] \boxplus [\![\boldsymbol{C}_g]\!]$ directly.

**LN and FFN on Compact Outputs.** For LN, MOAI and Euston store columns in separate ciphertexts, allowing direct addition across features. However, for $k > 1$, AMCP packs columns into different slots of one ciphertext, requiring $\mathcal{R}_r$ rotations to compute each row's mean and variance. For any interleaved ciphertext $[\![\boldsymbol{C}]\!]$, define $\mathcal{R}_r([\![\boldsymbol{C}]\!])$ as the cyclic shift implemented by Algorithm 2. For the first LN inputs $\{[\![\boldsymbol{U}_g]\!] = [\![\dot{\boldsymbol{y}}_g]\!] \boxplus [\![\boldsymbol{C}_g]\!]\}_{g=0}^{G-1}$ after residual addition, compute

**Algorithm 6** Input Weight Packing Algorithm

**Input:** Batch $t$, sequence $m$, packing $k$, total slots $S$, projection $\boldsymbol{O} \in \{\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}\}$, head-group start $h \in \{0, k, \ldots, H-k\}$, weights $w_{i,j}^{(\nu)} = (\boldsymbol{W}_O^{(\nu)})_{i,j}$, where $i \in \{0, \ldots, d-1\}$, $j \in \{0, \ldots, d'-1\}$, and $\nu \in \{h, \ldots, h+k-1\}$.
**Output:** $\{\boldsymbol{w}_{O,g,j}^{(h,r)}\}_{g \in \{0,\ldots,G-1\}, j \in \{0,\ldots,d'-1\}, r \in \{0,\ldots,k-1\}}$.
$\varphi \leftarrow S/m$, $G \leftarrow d/k$
**for** $g = 0, 1, \ldots, G-1$ **do**
  **for** $j = 0, 1, \ldots, d'-1$ **do**
    $\boldsymbol{w} \leftarrow [\underbrace{w_{gk,j}^{(h)}, \ldots}_{t}, \underbrace{w_{gk+1,j}^{(h+1)}, \ldots}_{t}, \ldots,$
      $\underbrace{w_{gk+k-1,j}^{(h+k-1)}, \ldots}_{t}, \underbrace{0, \ldots, 0}_{\varphi - kt}]$
    $\boldsymbol{w}_{O,g,j}^{(h,0)} \leftarrow [\underbrace{\boldsymbol{w}, \boldsymbol{w}, \ldots, \boldsymbol{w}}_{m}]$
    **for** $r = 1, 2, \ldots, k-1$ **do**
      $\hat{\boldsymbol{w}} \leftarrow [\underbrace{w_{gk+(r \bmod k),j}^{(h)}, \ldots}_{t}, \underbrace{w_{gk+((r+1) \bmod k),j}^{(h+1)}, \ldots}_{t},$
        $\ldots, \underbrace{w_{gk+((r+k-1) \bmod k),j}^{(h+k-1)}, \ldots}_{t}, \underbrace{0, \ldots, 0}_{\varphi - kt}]$
      $\boldsymbol{w}_{O,g,j}^{(h,r)} \leftarrow [\underbrace{\hat{\boldsymbol{w}}, \hat{\boldsymbol{w}}, \ldots, \hat{\boldsymbol{w}}}_{m}]$
    **end for**
  **end for**
**end for**
**return** $\{\boldsymbol{w}_{O,g,j}^{(h,r)}\}_{g \in \{0,\ldots,G-1\}, j \in \{0,\ldots,d'-1\}, r \in \{0,\ldots,k-1\}}$

**Algorithm 7** Output Weight Packing Algorithm

**Input:** Weights $\{\boldsymbol{W}_0^{(\nu)} \in \mathbb{R}^{d' \times d}\}_{\nu=0}^{H-1}$; batch size $t$, sequence length $m$, packing factor $k$, total slots $S$.
**Output:** Plaintext vectors $\{\boldsymbol{w}_{0,i,g}^{(h,r)}\}$ for $h \in \{0, k, \ldots, H-k\}$, $i \in \{0, \ldots, d'-1\}$, $g \in \{0, \ldots, G-1\}$, $r \in \{0, \ldots, k-1\}$.
$\varphi \leftarrow S/m$, $G \leftarrow d/k$
**for** $h = 0, k, \ldots, H-k$ **do**
  **for** $i = 0, 1, \ldots, d'-1$ **do**
    **for** $g = 0, 1, \ldots, G-1$ **do**
      **for** $r = 0, 1, \ldots, k-1$ **do**
        $\boldsymbol{w} \leftarrow \boldsymbol{0}_\varphi$
        **for** $a = 0, 1, \ldots, k-1$ **do**
          $\nu \leftarrow h + ((a+r) \bmod k)$
          $\boldsymbol{w}[at : (a+1)t - 1] \leftarrow w_{i,gk+a}^{(\nu)} \mathbf{1}_t$
        **end for**
        $\boldsymbol{w}_{0,i,g}^{(h,r)} \leftarrow [\underbrace{\boldsymbol{w}, \boldsymbol{w}, \ldots, \boldsymbol{w}}_{m}]$
      **end for**
    **end for**
  **end for**
**end for**
**return** $\{\boldsymbol{w}_{0,i,g}^{(h,r)}\}$

$$\begin{aligned}\llbracket \boldsymbol{\mu} \rrbracket &= \frac{1}{d} \boxtimes \sum_{r=0}^{k-1} \mathcal{R}_r\Big(\sum_{g=0}^{G-1} \llbracket \boldsymbol{U}_g \rrbracket\Big), \\ \llbracket \boldsymbol{q} \rrbracket &= \sum_{r=0}^{k-1} \mathcal{R}_r\Big(\sum_{g=0}^{G-1} \llbracket \boldsymbol{Z}_g \rrbracket^2\Big),\end{aligned} \tag{35}$$

where $\llbracket \boldsymbol{Z}_g \rrbracket = \llbracket \boldsymbol{U}_g \rrbracket \boxminus \llbracket \boldsymbol{\mu} \rrbracket$. Apply Section 5.2 slotwise with $n = d$ and $\boldsymbol{v} = \sqrt{\lambda \boldsymbol{q}}$, using each feature's LN scale $\gamma$ and bias $\beta$ in its corresponding slots. For depth, under AMCP, mask multiplications in Algorithm 2 increase LN1 and LN2 depth by one and two levels, reaching four and ten levels, respectively. However, AMCP's gains in parallel processing efficiency outweigh this depth overhead.

For FFN, the cyclic shift $\mathcal{R}_r$ aligns packed input columns with their plaintext weights, allowing both linear layers to retain the compact layout. For an FFN weight matrix $\boldsymbol{W}_\ell \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$, $\ell \in \{1, 2\}$, reuse the slot index $p = s\varphi + at + \beta$ and precompute plaintext weights

$$\boldsymbol{w}_{\ell,g,j}^{(r)}[p] = (\boldsymbol{W}_\ell)_{gk+((a+r) \bmod k),\, jk+a}. \tag{36}$$

For compact FFN inputs $\llbracket \boldsymbol{T}_g \rrbracket$, evaluate

$$\llbracket \boldsymbol{V}_j \rrbracket = \sum_{g=0}^{d_{\text{in}}/k-1} \sum_{r=0}^{k-1} \mathcal{R}_r(\llbracket \boldsymbol{T}_g \rrbracket) \boxtimes \boldsymbol{w}_{\ell,g,j}^{(r)}, \tag{37}$$

where $j = 0, \ldots, d_{\text{out}}/k - 1$. Reuse each input's rotations across output groups. Apply $\boldsymbol{W}_1$ before ReLU and $\boldsymbol{W}_2$ afterwards, as in Figure 1. For depth, the masks in Algorithm 2 are fused into the FFN weights offline, so each PCMM still consumes only one multiplicative level.

**Algorithm 8** Compact Output Projection and Aggregation

**Input:** Ciphertexts $\{\llbracket \dot{\boldsymbol{att}}_i^{(h)} \rrbracket\}$, where $h \in \{0, k, \ldots, H-k\}$ and $i \in \{0, \ldots, d'-1\}$, weights $\{\boldsymbol{w}_{0,i,g}^{(h,r)}\}$, head count $H$, per-head dimension $d'$, model dimension $d = Hd'$, batch size $t$, sequence length $m$, packing factor $k$, total slots $S$.
**Output:** Compact output $\llbracket \dot{\boldsymbol{Y}} \rrbracket = [\llbracket \dot{\boldsymbol{y}}_0 \rrbracket, \ldots, \llbracket \dot{\boldsymbol{y}}_{G-1} \rrbracket]$.
$G \leftarrow d/k$
**for** $g = 0, 1, \ldots, G-1$ **do**
  $\llbracket \dot{\boldsymbol{y}}_g \rrbracket \leftarrow 0$
**end for**
**for** $h = 0, k, \ldots, H-k$ **do**
  **for** $i = 0, 1, \ldots, d'-1$ **do**
    Apply Algorithm 2 to $\llbracket \dot{\boldsymbol{att}}_i^{(h)} \rrbracket$ using $(t, k, W = kt, S, m)$.
    Denote its outputs by $\{\llbracket \dot{\boldsymbol{att}}_i^{(h,r)} \rrbracket\}_{r=0}^{k-1}$.
    **for** $r = 0, 1, \ldots, k-1$ **do**
      **for** $g = 0, 1, \ldots, G-1$ **do**
        $\llbracket \dot{\boldsymbol{y}}_g \rrbracket \leftarrow \llbracket \dot{\boldsymbol{y}}_g \rrbracket \boxplus (\llbracket \dot{\boldsymbol{att}}_i^{(h,r)} \rrbracket \boxtimes \boldsymbol{w}_{0,i,g}^{(h,r)})$
      **end for**
    **end for**
  **end for**
**end for**
**return** $[\llbracket \dot{\boldsymbol{y}}_0 \rrbracket, \ldots, \llbracket \dot{\boldsymbol{y}}_{G-1} \rrbracket]$

# C Integrated Workflows

**Integrated Workflow of RIHP and DHP.** As illustrated in Figure 7, the architecture effectively integrates the RIHP and DHP strategies to maximize computational density. The

workflow is orchestrated in a "Split-Compute-Merge" paradigm. For clarity, the workflow is illustrated using Softmax as the representative attention function.

*Phase 1: DHP-based Input Projection.* The process begins with the input $\boldsymbol{X}$ encoded in the standard column-packed format. We apply DHP to the input projection. By utilizing pre-coded complex weight matrices $\tilde{\boldsymbol{W}}_Q^{(h)}, \tilde{\boldsymbol{W}}_K^{(h)}, \tilde{\boldsymbol{W}}_V^{(h)}$, we simultaneously compute the projections for two adjacent heads (head $h$ and $h+1$) within a single PCMM operation.

The hybrid ciphertexts $\tilde{\boldsymbol{Q}}^{(h)}, \tilde{\boldsymbol{K}}^{(h)}, \tilde{\boldsymbol{V}}^{(h)}$ encode the data for head $h$ in the real component and head $h+1$ in the imaginary component.

*Phase 2: Unpacking and Layout Transformation.* Proceeding to the middle phase, we perform the conjugate unpacking to separate the complex ciphertexts back into real-valued ciphertexts for each head. Notably, as indicated by the $\hat{\boldsymbol{K}}^{(h)}$ node, the key matrix is transformed into the RIHP format. This prepares the key matrix for the rotation-optimized attention calculation in the subsequent step.

*Phase 3: RIHP-based CCMMs.* First, we perform the CCMM between the query matrices $\boldsymbol{Q}^{(h)}$ and the hybrid key matrices $\hat{\boldsymbol{K}}^{(h)}$. For $r \in [0, m/2-1]$, we compute the hybrid diagonal ciphertext $\boldsymbol{C}^{(h)} = \{\boldsymbol{C}_r^{(h)}\}$. Crucially, due to the RIHP, $\boldsymbol{C}_r^{(h)}$ simultaneously encodes the $r$-th diagonal of $\boldsymbol{P}^{(h)}$ in its real part and the $(r+m/2)$-th diagonal in its imaginary part. Second, to recover the standard diagonal packing format, we apply the conjugate unpacking to each $\boldsymbol{C}_r^{(h)}$. The recovered ciphertexts constitute the score matrix $\boldsymbol{P}^{(h)}$, which is then processed by the Softmax to obtain $\boldsymbol{S}^{(h)}$. Subsequently, we execute the CCMM between $\boldsymbol{S}^{(h)}$ and the hybrid value matrices $\hat{\boldsymbol{V}}^{(h)}$ to produce the attention output $\hat{\boldsymbol{Att}}^{(h)}$. The conjugate unpacking then retrieves the attention output $\boldsymbol{Att}^{(h)}$.

*Phase 4: DHP-based Output Projection.* In the final output projection phase, we re-introduce DHP to aggregate the results. The independent outputs are repacked into a complex ciphertext $\tilde{\boldsymbol{Att}}^{(h)}$. We then perform the output projection using the pre-coded conjugate weight matrix $\tilde{\boldsymbol{W}}_0^{(h)}$. The property of this complex multiplication naturally accumulates the contributions from both heads into the real component. Finally, the aggregated result $\boldsymbol{Y}$ is extracted.

**Integrated Workflow of RIHP, DHP, and AMCP.** As illustrated in Figure 8, the architecture effectively integrates the RIHP, DHP, and AMCP. For ease of exposition, we consider the case ($k = 2$) and assume a block-wise packing format, which in practice corresponds to interleaved packing.

For input projection, DHP merges adjacent-head weights, followed by the permutation of the weight matrix via AMCP. After projection, double-head conjugate unpacking and rotation yield $\hat{\boldsymbol{K}}^{(h)}$ and $\hat{\boldsymbol{V}}^{(h)}$. Two CCMMs followed by conjugate unpacking produce $\boldsymbol{P}^{(h)}$ and $\boldsymbol{Att}^{(h)}$. Output projection applies DHP, followed by unpacking and head aggregation into ciphertexts containing $k$ feature columns. This layout supports direct residual addition and subsequent LN and FFN.

# D Non-linear Algorithms

Algorithm 9 provides the computations for the nonlinear layers, including the GK, LN, and ReLU.

The homomorphic exponential approximation [45] (Algorithm 10) adopts the scaling decomposition, where the input is scaled down by a factor of $2^\tau$ to ensure it falls within the bounded interval $(-5, 0)$. In this stable domain, the term $e^{x/2^\tau}$ is estimated using a degree-seven polynomial, and the final result is recovered via $\tau$ successive squaring operations.

The homomorphic square root (Algorithm 11) is implemented using a second-order Taylor expansion of the function $\sqrt{1+(x-1)}$ as Euston. Within the input range $(0, 2)$, this method

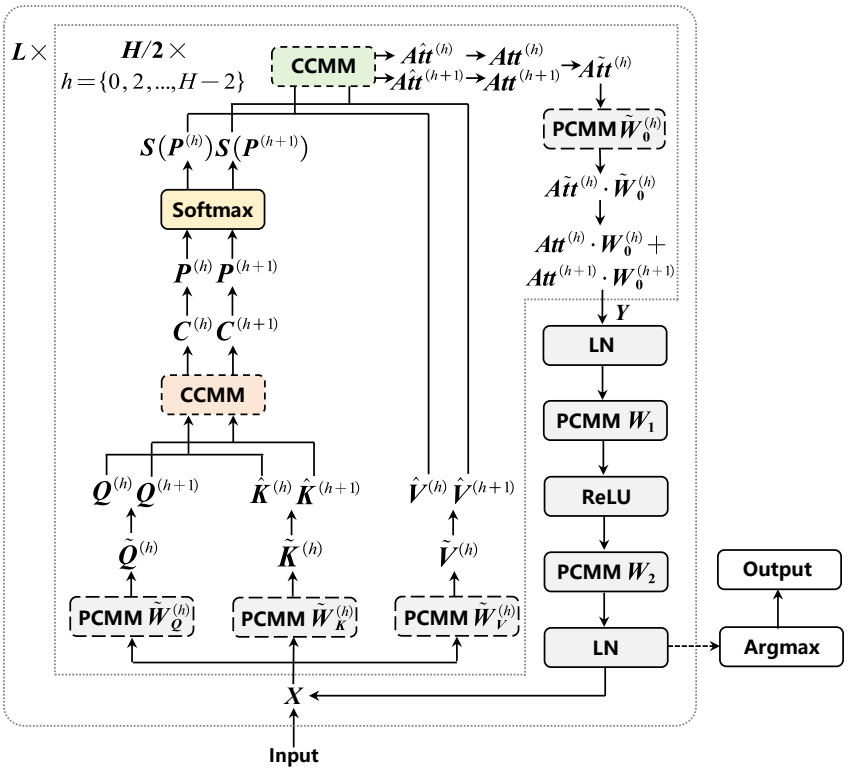



**Figure 7:** The integrated workflow of RIHP and DHP.

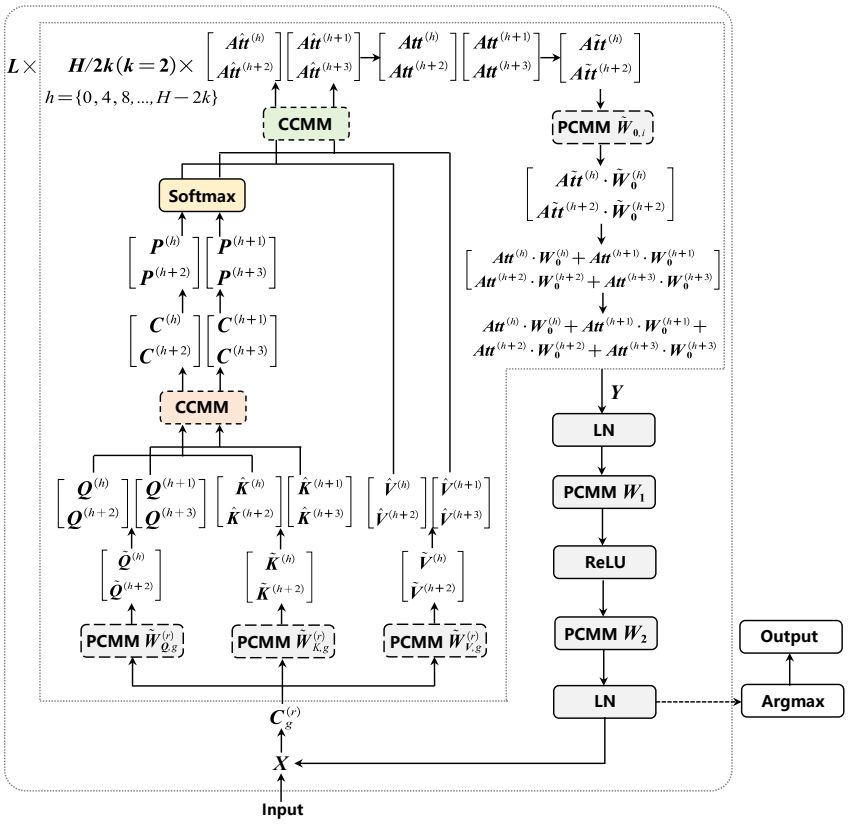



**Figure 8:** The integrated workflow of RIHP, DHP, and AMCP.

**Algorithm 9** Nonlinear Functions in STIP

**Input:** Column or diagonal-packed input $[\![\boldsymbol{X}]\!] = \{[\![\boldsymbol{x}_i]\!]\}_{i=1}^{n}$.
**Output:** Column or diagonal-packed output $[\![\boldsymbol{Y}]\!] = \{[\![\boldsymbol{y}_i]\!]\}_{i=1}^{n}$.
**Function: Gaussian Kernel**
**for** $i = 1, 2, \ldots, n$ **do**
    $[\![\boldsymbol{y}_i]\!] \leftarrow \textsc{Exp}\left(-\frac{1}{2\sqrt{d'}} \boxtimes [\![\boldsymbol{x}_i]\!]\right)$
**end for**
**Return** $[\![\boldsymbol{Y}]\!] = [[\![\boldsymbol{y}_1]\!], [\![\boldsymbol{y}_2]\!], \ldots, [\![\boldsymbol{y}_n]\!]]$

**Function: Inverse-Free LN**
**for** $i = 1, 2, \ldots, n$ **do**
    **for** $j = 1, 2, \ldots, n$ **do**
        $[\![\boldsymbol{a}_j]\!] \leftarrow [\![\boldsymbol{x}_i]\!] \boxminus [\![\boldsymbol{x}_j]\!]$
    **end for**
    $[\![\boldsymbol{z}_i]\!] \leftarrow \frac{1}{n} \boxtimes \sum_{j=1}^{n} [\![\boldsymbol{a}_j]\!]$
**end for**
$[\![\boldsymbol{v}]\!] \leftarrow \textsc{Sqrt}(\lambda \boxtimes \sum_{i=1}^{n} ([\![\boldsymbol{z}_i]\!] \boxtimes [\![\boldsymbol{z}_i]\!]))$
**for** $i = 1, 2, \ldots, n$ **do**
    $[\![\boldsymbol{y}_i]\!] \leftarrow (\gamma\sqrt{n\lambda} \boxtimes [\![\boldsymbol{z}_i]\!]) \boxplus (\beta \boxtimes [\![\boldsymbol{v}]\!])$
**end for**
**Return** $[\![\boldsymbol{Y}]\!] = [[\![\boldsymbol{y}_1]\!], [\![\boldsymbol{y}_2]\!], \ldots, [\![\boldsymbol{y}_n]\!]], [\![\boldsymbol{v}]\!]$

**Function: ReLU**
**for** $i = 1, 2, \ldots, n$ **do**
    $[\![\boldsymbol{s}_i]\!] \leftarrow \textsc{Sgn}([\![\boldsymbol{x}_i]\!])$
    $[\![\boldsymbol{y}_i]\!] \leftarrow [\![\boldsymbol{x}_i]\!] \boxtimes ([\![\boldsymbol{s}_i]\!] \boxplus 1) \boxtimes 0.5$
**end for**
**Return** $[\![\boldsymbol{Y}]\!] = [[\![\boldsymbol{y}_1]\!], [\![\boldsymbol{y}_2]\!], \ldots, [\![\boldsymbol{y}_n]\!]]$

**Algorithm 10** Homomorphic Exponential Approximation

**Input:** Ciphertext $[\![\boldsymbol{x}]\!]$, polynomial $f(x) = \sum_{i=0}^{7} a_i x^i$, positive integer $\tau \leqslant 3$.
**Output:** $[\![\boldsymbol{y}]\!] \approx \textsc{Exp}([\![\boldsymbol{x}]\!])$.
$[\![\boldsymbol{z}]\!] \leftarrow [\![\boldsymbol{x}]\!] \boxtimes \frac{1}{2^\tau}$
$[\![\boldsymbol{y}]\!] \leftarrow f([\![\boldsymbol{z}]\!])$
**for** $i = 1, 2, \ldots, \tau$ **do**
    $[\![\boldsymbol{y}]\!] \leftarrow [\![\boldsymbol{y}]\!] \boxtimes [\![\boldsymbol{y}]\!]$
**end for**
**Return** $[\![\boldsymbol{y}]\!]$

**Algorithm 11** Homomorphic Square Root Approximation

**Input:** Ciphertext $[\![\boldsymbol{x}]\!]$ (Assuming $\boldsymbol{x} \in (0, 2)$).
**Output:** $[\![\boldsymbol{y}]\!] \approx \textsc{Sqrt}([\![\boldsymbol{x}]\!])$.
$[\![\boldsymbol{t}]\!] \leftarrow [\![\boldsymbol{x}]\!] \boxminus 1$
$[\![\boldsymbol{t}_1]\!] \leftarrow [\![\boldsymbol{t}]\!] \boxtimes 0.5$
$[\![\boldsymbol{t}_2]\!] \leftarrow [\![\boldsymbol{t}]\!] \boxtimes [\![\boldsymbol{t}]\!] \boxtimes 0.125$
$[\![\boldsymbol{y}]\!] \leftarrow 1 \boxplus [\![\boldsymbol{t}_1]\!] \boxminus [\![\boldsymbol{t}_2]\!]$
**Return** $[\![\boldsymbol{y}]\!]$

approximates $\sqrt{x}$ by computing a weighted summation of the powers of the residual term $(x-1)$. We reduce Euston's square-root implementation from two levels to one by rewriting its second-order Taylor polynomial as $[12-(x-3)^2]/8$. This requires one ciphertext square, with division by eight absorbed into the ciphertext scale.

# E Other LLM Benchmarks

**Table 10:** Comparison of an A100 GPU execution time (s) breakdown under LLaMA-3-8B$^\dagger$ and GPT-2-1.5B$^\ddagger$.

| | STIP | | | Euston | | |
|---|---|---|---|---|---|---|
| **Operation** | **Depth** | **Time$^\dagger$** | **Time$^\ddagger$** | **Depth** | **Time$^\dagger$** | **Time$^\ddagger$** |
| Input projection | 15→12 | 91.52 | 40.80 | 17→16 | 1201.28 | 270.24 |
| Attention score | 12→10 | 22.40 | 17.28 | 16→14 | 1312.00 | 773.76 |
| GK / Softmax | 10→6 | 3.84 | 5.76 | 14→4 | 54.08 | 63.84 |
| Attention output | 6→4 | 5.12 | 3.84 | 4→2 | 1292.48 | 754.08 |
| Output projection | 4→1 | 32.64 | 15.36 | 2→1 | 400.32 | 90.24 |
| Bootstrappings | 1→5 | 245.12 | 183.84 | 1→10 | 2685.12 | 1573.44 |
| LN1 | 5→1 | 0.224 | 0.24 | 10→1 | 85.44 | 49.92 |
| Bootstrappings | 1→11 | 325.44 | 244.32 | 1→15 | 3159.36 | 1851.36 |
| FC1 | 11→10 | 146.12 | 39.88 | 15→14 | 1404.80 | 374.88 |
| Activation layer | 10→2 | 61.12 | 40.80 | 14→2 | 1085.76 | 727.20 |
| FC2 | 2→1 | 155.59 | 43.08 | 2→1 | 1379.84 | 371.52 |
| Bootstrappings | 1→11 | 325.44 | 244.32 | 1→10 | 2685.12 | 1573.44 |
| LN2 | 11→1 | 0.64 | 0.48 | 10→1 | 85.44 | 49.92 |
| Bootstrappings | 1→15 | 376.96 | 282.72 | 1→17 | 3456.00 | 2016.48 |
| Classification | - | 7.4 | 4.4 | - | 7.4 | 4.4 |
| **Total** | - | **1799.57** | **1167.12** | - | **20294.44** | **10544.72** |

Table 10 compares the end-to-end inference times of STIP and Euston on LLaMA-3-8B and GPT-2-1.5B at batch size $t = 32$. Our implementations of both models draw on Euston and NEXUS. STIP achieves 11.3× and 9.0× speedups over Euston on LLaMA-3-8B and GPT-2-1.5B, respectively. For LLaMA-3-8B ($H = 32$), we use $k = 8$. For GPT-2-1.5B ($H = 25$), we relax the constraint $k \mid H$ and use $k = 8$: the first 24 heads form three full groups, while the remaining head is processed separately. Table 11 reports the inference accuracy of GPT-2-1.5B. STIP exhibits acceptable deviations from the plaintext baseline.

**Table 11:** Inference accuracy evaluation on GLUE benchmark.

| **Model** | **Dataset** | **Softmax + GELU** | **GK + ReLU** | |
|---|---|---|---|---|
| | | **Plaintext** | **Plaintext** | **STIP** |
| **GPT-2-1.5B** | **RTE** | 75.21% | 75.36% | 75.15% |
| | **SST-2** | 92.20% | 92.01% | 91.85% |
| | **QNLI** | 91.80% | 91.13% | 90.92% |

Since LLaMA-3-8B and GPT-2-1.5B use pre-layer normalization rather than post-layer normalization, we adapt the inverse-free LN to their residual connections. We retain standard LN before attention and use inverse-free LN before the FFN, with ReLU, scaled biases, and matched residual scales. For $\hat{\boldsymbol{x}} = v\boldsymbol{x}$ ($v > 0$), standard LN absorbs $v$ before attention. Each attention output vector $\boldsymbol{y}$ is multiplied by the same factor $v$ before residual addition: $\hat{\boldsymbol{x}} + v\boldsymbol{y} = v(\boldsymbol{x} + \boldsymbol{y})$. The inverse-free LN accepts this uniformly scaled input and records its output scale, recomputed from the current input, as $v_{\text{new}}$ for propagation through the FFN. Moreover, LLaMA-3-8B's RMSNorm likewise admits the inverse-free scaled output $\gamma \cdot \boldsymbol{x}$ without mean subtraction or bias.