USENIX

THE ADVANCED COMPUTING
SYSTEMS ASSOCIATION

Breaking the Layer Barrier: Remodeling Private Transformer Inference with Hybrid CKKS and MPC

Tianshi Xu, *Peking University;* Wen-jie Lu, *TikTok;* Jiangrui Yu, Yi Chen, Chenqi Lin, Runsheng Wang, and Meng Li, *Peking University*

https://www.usenix.org/conference/usenixsecurity25/presentation/xu-tianshi

This paper is included in the Proceedings of the 34th USENIX Security Symposium.

August 13–15, 2025 • Seattle, WA, USA

978-1-939133-52-6

Open access to the Proceedings of the 34th USENIX Security Symposium is sponsored by USENIX.

Breaking the Layer Barrier: Remodeling Private Transformer Inference with Hybrid CKKS and MPC

Tianshi Xu[†] *Peking University*
Wen-jie Lu[†] *TikTok*
Jiangrui Yu[†] *Peking University*
Yi Chen *Peking University*
Chenqi Lin *Peking University*
Runsheng Wang *Peking University*
Meng Li[*] *Peking University*

Abstract

This paper presents an efficient framework for private Transformer inference that combines Homomorphic Encryption (HE) and Secure Multi-party Computation (MPC) to protect data privacy. Existing methods often leverage HE for linear layers (e.g., matrix multiplications) and MPC for non-linear layers (e.g., Softmax activation functions), but the conversion between HE and MPC introduces significant communication costs. The proposed framework, dubbed BLB, overcomes this by breaking down layers into fine-grained operators and further fusing adjacent linear operators, reducing the need for HE/MPC conversions. To manage the increased ciphertext bit width from the fused linear operators, BLB proposes the first secure conversion protocol between CKKS and MPC and enables CKKS-based computation of the fused operators. Additionally, BLB proposes an efficient matrix multiplication protocol for fused computation in Transformers. Extensive evaluations on BERT-base, BERT-large, and GPT2-base show that BLB achieves a 21× reduction in communication overhead compared to BOLT (S&P'24) and a 2× reduction compared to Bumblebee (NDSS'25), along with latency reductions of 13× and 1.8×, respectively, when leveraging GPU acceleration.

1 Introduction

Transformer architectures, e.g., BERT [34], GPT [10], and LLAMA [15], have achieved state-of-the-art (SOTA) performance in extensive real-world applications, including person re-identification [49], medical diagnosis [51], and voice assistant [5]. However, as these applications involve processing sensitive personal data, privacy concerns have emerged as a critical issue when deploying the Transformer models.

Private inference seeks to protect users' data privacy while ensuring users learn nothing about the model parameters except for the final results [12, 29, 33]. Recent advancements have proposed cryptographic frameworks based on Secure Multi-party Computation (MPC) for private Transformer inference [11, 22, 27, 28, 41, 42, 45, 55]. Among them, SOTA frameworks such as BOLT [45] and Bumblebee [41] employ a hybrid protocol combining Homomorphic Encryption (HE) and MPC. As shown in Figure 1 (a), in the hybrid protocol, HE is applied to compute linear layers (e.g., matrix multiplications), and MPC is used for evaluating nonlinear layers (e.g., Softmax, GeLU, etc). Compared to other alternatives [20, 56], the hybrid protocol effectively combines the strengths of both cryptographic primitives: HE reduces communication cost for linear layers, while MPC ensures high computation accuracy for nonlinear layers [29, 41, 45]. Therefore, in this paper, **we focus on the hybrid HE/MPC protocol.**

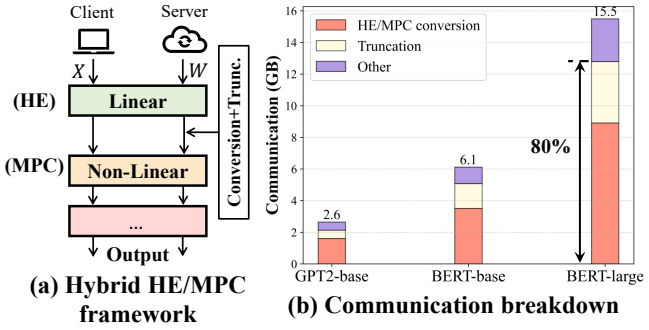

Figure 1: (a) Illustration of hybrid HE/MPC-based private inference; (b) Communication breakdown for GPT2-base, BERT-base, and BERT-large models based on the Bumblebee protocol [41].

However, existing hybrid HE/MPC protocols still face significant communication overhead when applied to Transformer models. For example, BOLT [45] requires 59.61 GB of communication to perform a single inference on a BERT-base model. Bumblebee [41] introduces HE into nonlinear layer computation to reduce communication, but still generates 15.5 GB communication for a BERT-large model. This high communication overhead presents a critical bottleneck, limiting their applicability to larger models. We profile the

[*]These authors contributed equally to this work.
[†]Corresponding author: meng.li@pku.edu.cn
Our artifact's DOI is 10.5281/zenodo.15590214.

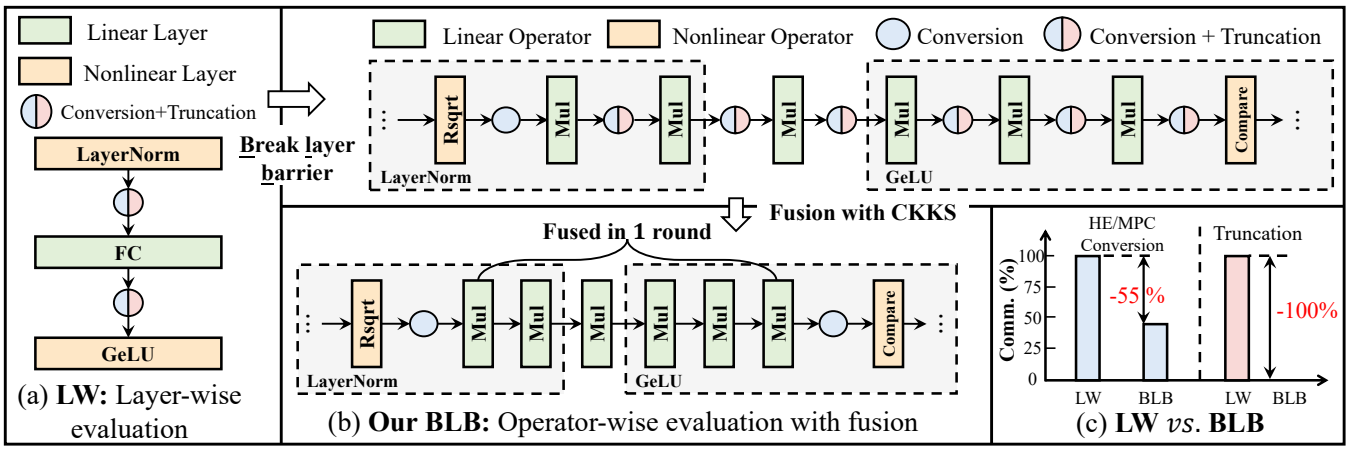

Figure 2: (a) Previous layer-wise (LW) paradigm; (b) Our proposed BLB paradigm; (c) BLB reduces communication costs by 55% for HE/MPC conversion and eliminates all of the truncation communication, compared to [41, 45].

communication breakdown of Bumblebee in Figure 1 (b) and identify two primary factors contributing to the substantial communication costs:

❶ **Excessive Truncations.** Hybrid HE/MPC protocols often rely on fixed-point arithmetic. A real number x is represented as an integer $\lfloor x \cdot 2^s \rfloor$ with a *scale* s, where $\lfloor \rfloor$ denotes round down. Multiplication of two fixed-point numbers x and y produces $\lfloor xy \cdot 2^{2s} \rfloor$ with scale $2s$. To restore the scale to s, the result must be right-shifted (truncated) by s bits through an MPC-based truncation protocol [47]. As truncations are needed after each linear multiplication and require communication between the user and server, the overall communication cost becomes excessive.

❷ **Frequent Conversions Between HE and MPC.** As truncations and nonlinear layers are evaluated using MPC protocols while linear layers are computed with HE, conversions between HE and MPC are frequently required. The HE/MPC conversion involves transferring HE ciphertexts and invoking MPC protocols for bit width, scale, and field adjustment [45], leading to high communication overhead. As in Figure 1 (b), truncations and HE/MPC conversions together account for over 80% of Bumblebee's total communication.

To address these problems, previous hybrid frameworks have attempted to fuse adjacent linear layers to reduce the communication overhead [4, 21, 45, 57]. **Here, linear layer fusion refers to using HE to evaluate consecutive linear layers within a single communication round,** thereby eliminating the conversion and truncation protocols required between the adjacent linear layers. For example, in convolutional neural networks (CNNs), PrivCirNet [21] fuses the convolution and batch normalization layers as well as the two following convolution layers in MobileNetV2. For Transformers, BOLT [45] fuses two consecutive matrix multiplications (MatMuls) in the self-attention layer. However, these methods often show insufficient communication reduction due to the following three limitations:

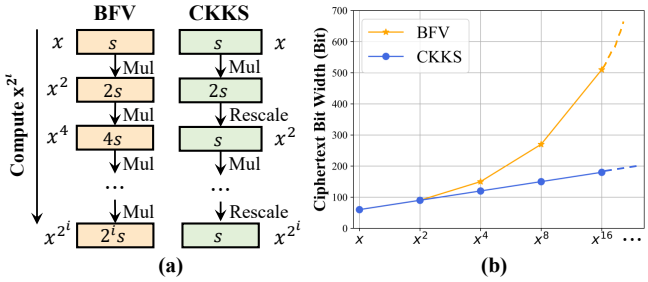

Figure 3: (a) CKKS can maintain the operand scale at s with rescale, while BFV needs exponential scale growth when computing x^{2^i}; (b) The ciphertext bit width change for the BFV and CKKS schemes during computation.

❶ **Limited Flexibility.** As shown in Figure 2 (a), current frameworks follow a layer-wise (LW) evaluation paradigm, where they design protocols for each layer and evaluate a model layer by layer. This coarse-grained fusion across neighboring layers limits the fusion patterns and their applicability to Transformer models. For instance, only two consecutive MatMuls in a self-attention layer can be fused in one Transformer block [45], resulting in limited communication reduction.

❷ **High Ciphertext Bit Width.** SOTA hybrid HE/MPC protocols often use the Brakerski-Fan-Vercauteren (BFV) HE scheme for linear layers [27, 29, 41, 45]. However, the BFV scheme suffers from a significant expansion in operand scale when computing fused multiplications. As shown in Figure 3, when repeatedly squaring an input x with an initial scale of s, i.e., computing x^{2^i}, the BFV scheme exhibits an exponential growth in scale, as the scale doubles after each square computation. Consequently, the bit widths of plaintexts and ciphertexts have to increase proportionally to maintain computation correctness [7]. The high ciphertext bit width increases both the computation and communication costs,

limiting the benefits of linear layer fusion.

❸ **Suboptimal MatMul Protocol.** As Transformer models process high-dimensional tensors while HE computes over one-dimensional vectors, the mapping from tensors to vectors, denoted as packing, directly impacts the HE computation complexity. Linear layer fusion imposes more restrictions on the packing algorithm as adjusting the packing of intermediate ciphertexts in fused linear layers involves complex homomorphic operations and incurs high computation complexity [45]. In contrast, without layer fusion, it is possible to optimize the packing for each linear layer separately. As a result, BOLT [45] requires approximately 20× more computationally-intensive HE rotations for fused MatMul compared to the unfused protocol [41]. While PowerFormer [46] proposes further packing optimization, it still incurs high costs for non-square matrices.

1.1 Technical Details

To overcome the limitations of existing solutions, we propose the Breaking the Layer Barrier framework, dubbed BLB, which features three key techniques designed to resolve the above three limitations:

❶ **Breaking the Layer Barrier.** Nonlinear layers in Transformer models, e.g., GeLU, Layer Normalization (LayerNorm), Softmax, etc, are very complex and incur high communication costs when directly evaluating with MPC protocols [27, 47]. SOTA hybrid HE/MPC frameworks [41, 45] propose piecewise linear approximations to improve communication efficiency[1]. As a result, each nonlinear layer involves computing a series of linear operators, e.g., element-wise multiplications, providing new fusion opportunities. This insight motivates us to break the layer barrier and refine fusion granularity to neighboring operators. For example, as in Figure 2 (b), the last two operators of the LayerNorm layer, the subsequent fully connected (FC) layer, and the first three operators in the following GeLU layer are all linear operators. Once fused, all HE/MPC conversions and truncations between adjacent linear operators can be eliminated, significantly reducing communication costs.

However, realizing such fusion is challenging due to the diverse packing algorithms employed for different linear operators [45, 46, 56]. For example, MatMuls and element-wise multiplications often have different preferences for packing algorithms. Ensuring the computation correctness and efficiency of fused linear operators remains an open question. We propose the FineGrainFusion paradigm to tackle the challenge systematically. We classify the operators into different categories so that operators of the same category can share a common packing algorithm. This enables us to define fusion patterns directly for different categories and design optimized packing algorithms for each fusion pattern. Through FineGrainFusion, neighboring linear operators following the defined fusion patterns can be correctly fused, significantly reducing the communication overhead.

❷ **Hybrid CKKS and MPC Framework with Secure Conversion.** For fused linear operators, we argue that the Cheon-Kim-Kim-Song (CKKS) HE scheme is a superior choice as it can mitigate the high ciphertext bit width limitation. This is because CKKS supports an important **rescale** operation that enables to reduce the operand scale back to s. As illustrated in Figure 3 (a), the rescale operation can restore the operand scale from $2s$ to s after multiplication, enabling a controlled, linear increase of ciphertext bit width (Figure 3 (b)). However, the existing CKKS and MPC conversion protocol proposed in MP2ML [14] has severe security problems and may leak the information of computation results. Specifically, MP2ML directly adapts the BFV-MPC conversion protocol to CKKS and MPC but neglects that the CKKS encoding relies on a fast Fourier transform (FFT), which produces a narrowly distributed output. As a result, the sampled noise in MP2ML cannot effectively obscure the computation results, leading to high privacy risks. We carefully analyze the CKKS mechanics and develop the first secure and efficient conversion protocol for CKKS and MPC. Through the fine-grained operator fusion and CKKS-based HE computation, we can reduce the communication costs for HE/MPC conversion by 55% and eliminate all truncations, compared to BOLT and Bumblebee [41, 45], as shown in Figure 2 (c).

❸ **Rotation-Efficient Fused MatMul Protocol.** Linear operator fusion poses extra constraints on the packing of intermediate ciphertexts. For fused MatMuls, it results in more homomorphic operations, e.g., HE rotations [45, 46]. To reduce the computation costs, we design a rotation-efficient ciphertext-ciphertext MatMul protocol. Besides, we observe that the multi-head attention computation in Transformers naturally leads to batched MatMuls, which enables packing multiple batches into a ciphertext to reduce HE rotations. We also apply the baby-step-giant-step (BSGS) optimization [32, 45] to further reduce HE rotations. Compared to BOLT [45] and PowerFormer [46], our protocol achieves a 29× and 8× reduction in HE rotations, respectively.

1.2 Contributions

Our contributions can be summarized as follows:

- ❶ We observe that truncations and HE/MPC conversions account for major communication overhead and propose a novel framework, dubbed BLB, that breaks the layer barrier to enable fine-grained linear operator fusion for communication-efficient private Transformer inference.
- ❷ BLB proposes the first secure CKKS and MPC conversion protocol to enable CKKS-based computation of fused linear operators and reduce the ciphertext bit width increase.

[1] A detailed review of SOTA protocols for nonlinear layers is provided in Appendix B.

Table 1: Notations used in the paper.

Notations	Meanings
$\mathbb{C},\mathbb{R},\mathbb{Z}$	the sets of complex, real and integral numbers
$\lceil\cdot\rceil,\lfloor\cdot\rfloor,\lfloor\cdot\rceil$	ceiling, flooring, and rounding operations
$\mathbb{Z}_q$	$\mathbb{Z}_q = \mathbb{Z} \cap [-\lceil q/2\rceil, \lceil q/2\rceil]$
$[n]$	$\{0,..,n-1\}$ for a non-negative integer n
N	polynomials degree
$\mathbb{A}_{N,q}$	the set of integer polynomials $\mathbb{A}_N = \mathbb{Z}_q[x]/(x^N+1)$
$\odot$	element-wise multiplication
q	ciphertext modulus, q is a prime

❸ We propose a rotation-efficient MatMul protocol for fused computations, reducing $8 \sim 29\times$ HE rotations compared with prior-arts [45, 46].

❹ We evaluate BLB on various Transformer models, including BERT-base, BERT-large, and GPT2-base. With extensive experiments, we demonstrate $21\times$ and $2\times$ communication reduction compared to BOLT [45] and Bumblebee [41], respectively. With GPU-based HE computation, BLB achieves $13\times$ and $1.8\times$ latency reduction compared to BOLT [45] and Bumblebee [41], respectively.

1.3 Comparison with FHE Frameworks

Another important variant of the private inference framework is based on Fully Homomorphic Encryption (FHE) [16, 32, 35, 38, 44, 46, 56]. As FHE-based frameworks evaluate the entire neural network (NN) on encrypted data in a non-interactive manner, their communication costs are usually lower compared to the hybrid HE/MPC frameworks. However, FHE-based frameworks still struggle with accurately evaluating complex nonlinear layers, e.g., Softmax, GeLU, etc, in Transformer models [16, 56]. For example, using the open-source FHE framework NEXUS [56], we find the nonlinear layer approximation exhibits a maximum relative error of 297%, which significantly impacts the overall inference accuracy and requires extensive fine-tuning. In contrast, hybrid HE/MPC frameworks [41, 45] often exhibit much better inference accuracy at the cost of higher communication costs. Therefore, in this paper, we focus on the hybrid HE/MPC protocol and propose BLB to improve its communication efficiency.

2 Preliminaries

2.1 Notations

Table 1 summarizes the notations used in this paper. We represent vectors with lower-case letters (e.g., x) and matrices with upper-case letters (e.g., X). We use lower-case letters with a "hat" symbol (e.g., $\hat{x}$) to represent a polynomial. We use $\mathbf{1}\{\mathcal{P}\}$ to denote the indicator function, which is 1 when $\mathcal{P}$ is true and 0 otherwise.

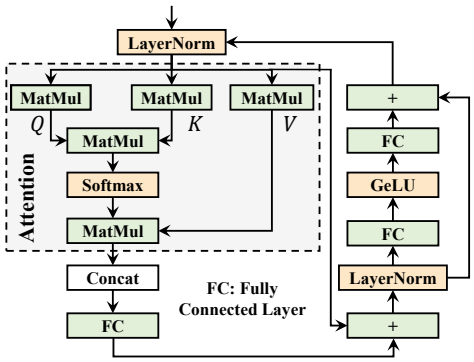

Figure 4: A Transformer block in BERT [34].

2.2 Transformer Model

Figure 4 shows the structure and workflow of a classical Transformer block, which is used in BERT [34] models. The multi-head attention is the key component in the Transformer block. Concretely, an input $X \in \mathbb{R}^{m\times d}$ is multiplied with three weight matrices to produce a query matrix $Q = XW_Q$, a key matrix $K = XW_K$, and a value matrix $V = XW_V$, where $W_Q, W_K, W_V \in \mathbb{R}^{d\times d}$, and m, d are the sequence length and the hidden dimension, respectively. Q, K, V are then divided into H heads: $Q_h, K_h, V_h \in \mathbb{R}^{m\times \frac{d}{H}}$, where $h \in [H]$, and H is the number of heads. Then, the multi-head attention is computed by:

$$\text{Att}_h = \text{Softmax}(\frac{Q_h K_h^T}{\sqrt{d/H}})V_h \quad (1)$$

The H resulting matrices Att_h are concatenated and projected by W_O, i.e., $\text{Concat}(\text{Att}_h)W_O$.

2.3 Threat Model

BLB works in a general private inference scenario that involves two parties, i.e., server and client. The server holds the proprietary NN model, and the client owns private input [19, 27, 29, 33, 41, 45]. BLB enables the client to obtain the inference results while keeping the server's model weights and the client's input private. Consistent with previous works [12, 29, 33, 41, 45, 47], BLB assumes the NN architecture is known to both sides and adopts an *honest-but-curious* security model in which both parties follow the specification of the protocol but also try to learn more than allowed. More security discussion is provided in Appendix A.

2.4 Cryptographic Primitives

CKKS HE Scheme. We now present the key implementation characteristics of CKKS. Full details can be found in [6, 7].

- **CKKS Plaintext and Ciphertext.** In CKKS, a plaintext m is represented as a vector of $N/2$ fixed-point numbers. The scale parameter s determines the width of the fractional part of each element. After encryption, each ciphertext $\mathsf{ct} \in \mathbb{A}^2_{N,q}$ is a pair of polynomials. Each polynomial has N integer coefficients modulo a large prime q.
- **Rescaling.** A rescale operation reduces the ciphertext modulus q by a factor d and the scale by an additive factor factor $\log_2 d$. Multiplying two ciphertexts with scale s produces a result with scale $2s$. By rescaling with $\log_2 d \approx s$ after multiplication, the scale is approximately reset to s.
- **Homomorphic Addition ($\boxplus$) and Multiplication ($\boxtimes$).** Given two ciphertexts or a ciphertext and a plaintext, the operations $\boxplus$ and $\boxtimes$ produce encryptions of their element-wise sum and product, respectively.
- **Rotation.** The left-rotation $\mathrm{Rot}^t_l(m)$ left-rotates the vector m by t slots. The right-rotation $\mathrm{Rot}^t_r(m)$ right-rotates the vector m by t slots. If not specified, the default is left rotation.

Additive Secret Share. We use a 2-out-of-2 additive secret share to keep the input data private throughout inference. We denote two parties by P_0 and P_1, where P_0 is the client and P_1 is the server. We use $[\![x]\!]^M$ to denote an additive share of x over $\mathbb{Z}_M$. We write $[\![x]\!]^M = ([\![x]\!]^M_0, [\![x]\!]^M_1)$ where P_0 holds $[\![x]\!]^M_0$ and P_1 holds $[\![x]\!]^M_1$, such that $[\![x]\!]^M_0 + [\![x]\!]^M_1 = x \bmod M$. In particular, we use $[\![x]\!]^B$ to represent bool sharing where x is a 1-bit value. Moreover, we designate the shares over $M = 2^l$ as ring shares and those as prime shares when $M = q$ is a prime. Besides, the MPC protocols used in this paper are based on Oblivious Transfer [41, 45, 47].

3 BLB Framework Overview

Figure 5 provides an overview of the BLB framework, which integrates MPC and CKKS HE schemes to evaluate linear and nonlinear operators, respectively. BLB features three key components: firstly, BLB proposes FineGrainFusion paradigm to systematically explore the fusion patterns of diverse linear operators in the Transformer models, as explained in § 4; secondly, to accommodate the packing requirements of fused MatMuls, efficient MatMul protocols are proposed in § 5 to reduce the required HE rotations drastically; thirdly, to overcome the significant increase of ciphertext bit widths after fusion, § 6 proposes the first secure conversion protocol between CKKS and MPC, enabling secure yet efficient transitions between linear and nonlinear operators.

4 FineGrainFusion Paradigm

As diverse linear operators exist in Transformer models, we propose the FineGrainFusion paradigm in this section to analyze the fusion patterns systematically. We begin by classifying operators into different categories and then propose a formal analysis of fusion patterns for linear operators of different categories. We also optimize the HE packing algorithms based on the categorization.

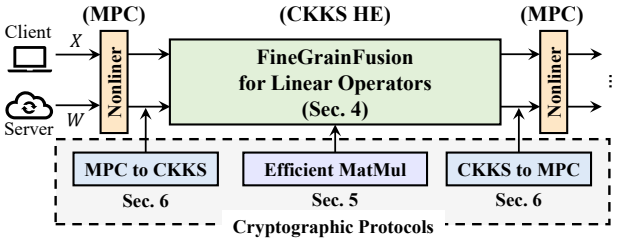

Figure 5: Overview of BLB.

Table 2: Definition of operators. ct-ct means the two inputs are both ciphertexts, and ct-pt means one input is ciphertext and the other is plaintext.

Type	Name	Description
Linear operators		
Identity	$\mathrm{ewadd}_{\mathrm{cc}}, \mathrm{ewadd}_{\mathrm{cp}}$	Element-wise addition of ct-ct, ct-pt
	$\mathrm{ewmul}_{\mathrm{cc}}, \mathrm{ewmul}_{\mathrm{cp}}$	Element-wise multiplication of ct-ct, ct-pt
Expansion	$\mathrm{sadd}_{\mathrm{cc}}, \mathrm{sadd}_{\mathrm{cp}}$	Scalar addition of ct-ct, ct-pt
	$\mathrm{smul}_{\mathrm{cc}}, \mathrm{smul}_{\mathrm{cp}}$	Scalar multiplication of ct-ct, ct-pt
Reduction	sum	Sum in a specified dimension of the input
Transformation	$\mathrm{matmul}_{\mathrm{cc}}, \mathrm{matmul}_{\mathrm{cp}}$	MatMul of ct-ct, ct-pt
Nonlinear operators		
cmp		$[\![\mathbf{1}\{x < y\}]\!]^B \leftarrow \mathrm{cmp}([\![x]\!]^M, [\![y]\!]^M)$
mux		$[\![b \cdot x]\!]^M \leftarrow \mathrm{mux}([\![b]\!]^B, [\![x]\!]^M)$
rec, rsqrt		Reciprocal, Reciprocal Sqrt [47]

4.1 Operator Categorization

As shown in Table 2, we classify the operators in a Transformer model into linear and nonlinear. Nonlinear operators mainly include comparison, multiplexer, reciprocal, and reciprocal square root. All nonlinear operators are evaluated with MPC protocols in the hybrid HE/MPC framework following [12, 47].

Linear operators in a Transformer model include element-wise addition and multiplication, scalar addition and multiplication, summation, and MatMuls between ciphertexts (denoted as ct-ct) or between ciphertexts and plaintexts (denoted as ct-pt). Note that it is often expensive to change the packing of ciphertexts as it involves costly homomorphic operations, e.g., multiplications and rotations [45]. In contrast, changing the packing of plaintexts incurs negligible overhead. Scalar addition and multiplication are implemented via broadcasting, where the scalar is expanded to match the dimensions of the target matrix, followed by element-wise operations.

To facilitate fusion analysis, we further classify the linear operators into four high-level abstract categories, including Identity, Expansion, Reduction, and Transformation, based

Table 3: Complexity comparison of packing algorithms for the summation operator followed by the scalar operator.

	#Rot	#Mul	Mul. Depth
NEXUS [56]	$3L + L\log_2 D$	$2L+1$	3
BLB (w/o fusion)	$2\log_2 D$	1	1
BLB (w/ fusion)	$\log_2 D$	0	0

on the dimension transformation between the operator's input and output. Table 2 shows the details of the operator classification. Consider the inputs and intermediate outputs in a Transformer model as 2D tensors of dimensions (L, D), where L denotes the *spatial* dimension along which computations are conducted in parallel and D denotes the *reduce* dimension along which aggregation is conducted. Then, the four categories can be defined as follows.

- Identity: The input and output dimensions are identical, with element-wise operators that do not alter the packing.
- Expansion: The dimensions change from $(L, 1)$ to (L, D), where D is expanded, or broadcasted from 1 to D. Scalar addition and multiplication are examples of this type.
- Reduction: The dimensions change from (L, D) to $(L, 1)$, where D is aggregated to 1, as in summation.
- Transformation: The dimensions change from (L, D) to (L, D'), with the reduce dimension D transformed to D', as in MatMul.

Operators in the Identity, Expansion, and Reduction categories can share a common packing method. However, for the Transformation type, $\text{matmul}_{\text{cp}}$ and $\text{matmul}_{\text{cc}}$ require separate packing algorithms and should be analyzed individually.

4.2 FineGrainFusion Patterns

With operator categorization, we develop fusion patterns for various type combinations. To ensure both correctness and efficiency for each fusion pattern, we propose two additional requirements: ❶ **Consistent packing for each operator.** To simplify the packing algorithms after fusion, the input and output of all linear operators should have the same packing structure. ❷ **Tailored packing algorithms for each fusion pattern.** For adjacent operators eligible for fusion, the packing algorithm must be tailored based on the specific combination of operator types.

4.2.1 Consistent Packing for Single Operator

Each operator must adhere to a consistent *spatial-first* packing rule for input and output. In this context, the *spatial-first* packing means the elements along the spatial dimension are placed consecutively before moving to the next row in the reduce dimension. Identity operators naturally satisfy this rule. For Transformation operators, both $\text{matmul}_{\text{cp}}$ and $\text{matmul}_{\text{cc}}$ meet the rule following the protocols proposed in BOLT [45] and Jiang et al. [31].

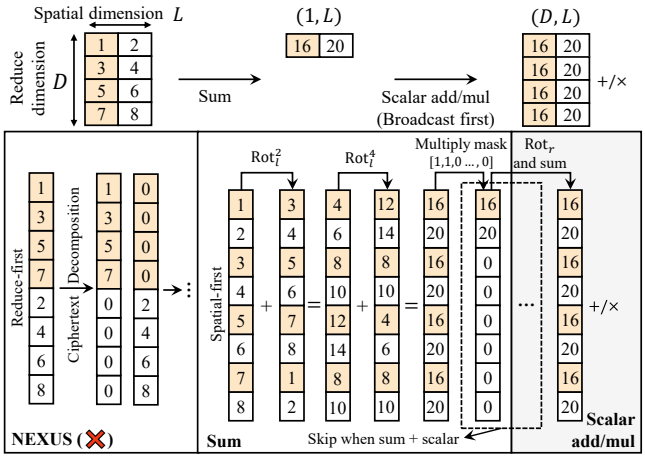

Figure 6: A toy example comparing NEXUS's packing algorithm with ours for summation and scalar operators.

For Expansion and Reduction operators, existing packing algorithms in NEXUS [56] modify the packing and increase the number of ciphertexts, leading to considerably more complex packing algorithms. The primary issue lies in NEXUS's adoption of *reduce-first* packing, which proves unsuitable for these operators. We propose a simple yet efficient packing algorithm for summation and scalar addition/multiplication, as illustrated in Figure 6. These operators are implemented through a rotate-and-sum process defined as follows:

$$\text{sum}(m) = [\underbrace{1,\ldots,1}_{L},0,\ldots,0] \odot m_l^{\log_2 D}$$

$$\text{broadcast}(m) = m_r^{\log_2 D}$$

$$m_l^0 = m, m_l^i = m_l^{i-1} + \text{Rot}_l^{2^{i-1}L}(m_l^{i-1}) \text{ for } i = 1,\ldots,\log_2 D$$

$$m_r^0 = m, m_r^i = m_r^{i-1} + \text{Rot}_r^{2^{i-1}L}(m_r^{i-1}) \text{ for } i = 1,\ldots,\log_2 D$$

We perform a broadcast for scalar addition and multiplication and then conduct element-wise operations. When a summation operator precedes a scalar operator, the masking and the broadcast steps can be omitted, as shown in Figure 6. Table 3 compares the theoretical complexity with NEXUS [56]. Our packing algorithms need only $\log_2 D$ rotations while preserving a consistent packing for input and output.

4.2.2 Tailored Packing Algorithms for Fusion Patterns

After establishing packing algorithms for individual operators, we analyze the fusion patterns of adjacent operator pairs. For two candidate operators, we can: ❶ infer the type of the resulting fused operator and ❷ determine whether the fusion

Table 4: Fusion pattern analysis. The first column and the first row show the mapping types of the first and second operators. The colored cells show the mapping type of the operator after fusion. Green cells indicate legal fusions that do not require new packing algorithms. Orange cells represent legal fusions that necessitate new packing algorithms. Red cells denote fusion patterns that do not occur in Transformer models.

First op \ Second op	Identity	Expansion	Reduction	Transformation
Identity	Identity	Expansion	Reduction	Transformation
Expansion	Expansion	×	Identity	Expansion
Reduction	Reduction	Identity	×	×
Transformation	Transformation	×	Reduction	Transformation

imposes new constraints on packing, ensuring compatibility and efficiency of this fusion.

Table 4 provides the detailed fusion patterns. The first column and the first row represent the types of the first and the second operators to be fused, and the colored cells indicate the resulting operator type. The fusion patterns are categorized into green, orange, and red. Below, we elaborate on the green and orange fusion patterns in Table 4.

- The identity operator can be freely fused with any other operator type. Expansion and Reduction operators can be fused with each other using the packing algorithms in Figure 6. Additionally, the Expansion operator can be fused with the Transformation operator by applying the respective packing algorithms for each.
- Transformation with others. The Transformation operator can be fused with the subsequent Reduction operator, representing the combination of MatMul and summation. When fused with Transformation operators, it represents the fusion of two consecutive MatMuls. For $\text{matmul}_{\text{cp}} + \text{matmul}_{\text{cp}}$, fusion is straightforward. As for $\text{matmul}_{\text{cp}} + \text{matmul}_{\text{cc}}$ and $\text{matmul}_{\text{cc}} + \text{matmul}_{\text{cp}}$, although fusion is feasible following the packing algorithms proposed in [45, 46], the associated computational overhead is substantial. To alleviate the computational cost, we developed an efficient MatMul protocol for the fused computations, which will be detailed in § 5.

With the FineGrainFusion paradigm, NN layers are broken into basic operators, creating an operator-level computation graph. Fusion patterns are matched to merge compatible linear operators, reducing communication overhead while ensuring correctness.

5 Efficient Protocol for Fused MatMuls

In this section, we introduce BLB's protocol for fused MatMuls. The multi-head attention computation in a Transformer model involves consecutive MatMuls with fusion opportunities, i.e., $Q_h K_h^T$ (where Q_h, K_h are produced by $Q = XW_Q$ and $K = XW_K$) and $\text{Concat}(\text{Att}_h)W_O$ (where $\text{Att}_h = \text{Softmax}(\cdot) \times V_h$), as shown in Equation 1. As Q_h and K_h are ciphertexts generated through the $\text{matmul}_{\text{cp}}$ protocol, $Q_h K_h^T$ requires the $\text{matmul}_{\text{cc}}$ protocol. Similarly, $\text{Concat}(\text{Att}_h)W_O$ and Att_h are computed through the $\text{matmul}_{\text{cp}}$ and $\text{matmul}_{\text{cc}}$, respectively.

Fusing these operators is not straightforward, because upon fusion, the packing of intermediate ciphertexts, e.g., Q_h, K_h, Att_h, etc, is limited by the previous MatMul protocol while modifying the packing requires costly homomorphic operations. As both inputs of $\text{matmul}_{\text{cc}}$ are ciphertexts, the $\text{matmul}_{\text{cc}}$ protocol often becomes the bottleneck due to the packing constraints. For example, BOLT [45] only optimizes $\text{matmul}_{\text{cp}}$ and overlooks $\text{matmul}_{\text{cc}}$, leading to 20× more rotations. Powerformer [46] optimizes $\text{matmul}_{\text{cc}}$ but still suffers from high cost for non-square matrices. We propose a rotation-efficient protocol for $\text{matmul}_{\text{cc}}$ with further multi-head and BSGS optimizations, which reduces rotations by $8 \sim 29\times$ compared to BOLT and Powerformer.

5.1 Building Blocks

Rotation-Efficient ct-ct **MatMul.** We begin with a ct-ct MatMul, $C = A \times B$ where $A \in \mathbb{R}^{L \times D}$ and $B \in \mathbb{R}^{D \times L}$. Matrix A is in a spatial-first packing, and B is in a reduce-first packing. Let $A\{i, j\}$ denote the element in the i-th row and j-th column of A. $A\{i,:\}$ denotes the i-th row, and $A\{:, j\}$ denotes the j-th column of A. Figure 7 shows an example where $L = 4, D = 2$. Our rotation-efficient protocol is based on the following two observations:

❶ **Element-wise multiplication of $A\{:,k\}$ and $B\{k,:\}$ gives the k-th partial sum of the main diagonal of C.** For example in Figure 7 (b), multiplying $A\{:,0\}$ with $B\{0,:\}$ produces the 0-th partial sum of C's main diagonal, denoted as $C[0](0)$.

❷ **Rotating $B\{k,:\}$ left by t steps, the multiplication produces the k-th partial sum of the t-th diagonal of C, denoted as $C[t](k)$.** After rotating $B\{k,:\}$ by t steps, the i-th element of $B\{k,:\}$, denoted $B\{k,i\}$, becomes $B\{k,i+t\}$ in the original matrix. Multiplying $A\{i,k\}$ by $B\{k,i+t\}$ produces the k-th partial sum of $C\{i,i+t\}$, corresponding to the i-th element in the t-th diagonal of C. As illustrated in Figure 7(c), rotating $B\{1,:\}$ by 1 step and multiply $A\{:,1\}$ with it, we will obtain $C[1](1)$.

With these two observations, we can perform MatMuls by multiplying $A\{:,k\}$ with rotated $B\{k,:\}$ to obtain partial sums of all diagonals and sum all partial sums to get the final result. Figure 7(d) illustrates our protocol which involves three steps: ❶ **Preprocessing.** This step rotates $B\{t,:\}, \forall t \in [D]$ by t steps. After preprocessing, B remains row-packed, but the elements within each row are rotated. This enables the multiplication with A to produce different diagonals in a single ciphertext, eliminating the need for accumulating elements within the same ciphertext. In contrast, BOLT [45] does not incorporate this preprocessing, causing partial sums

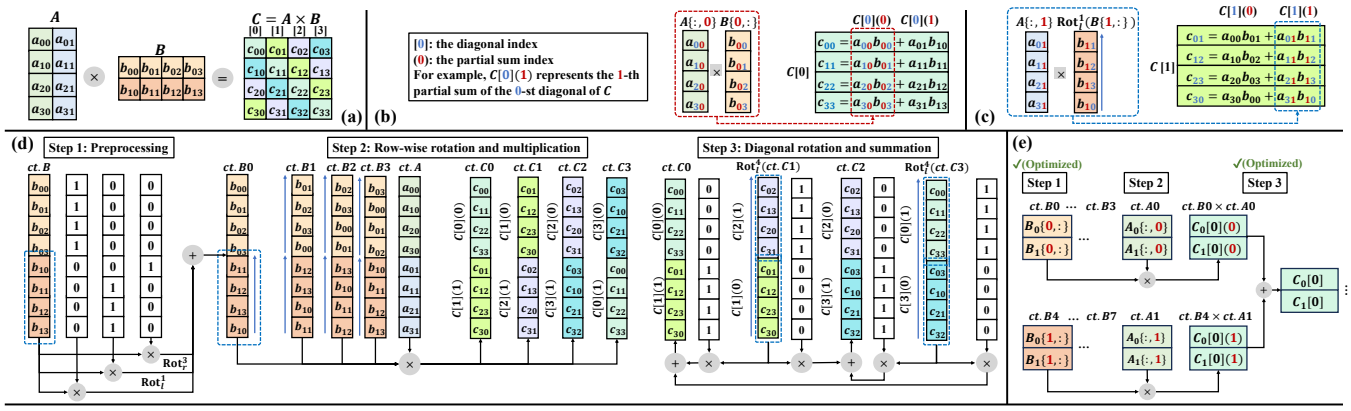

Figure 7: A toy example for matmul$_{cc}$: (a) Multiplying matrices A and B in plaintext; (b) Observation 1: Element-wise multiplication of A's columns and B's rows generates the partial sums of C's diagonals. We use [i] to denote the index of the diagonal and (j) to denote the index of the partial sum; (c) Observation 2: Rotating $B\{k,:\}$ left by t steps, the multiplication produces the k-th partial sum of the t-th diagonal of C; (d) Illustration of BLB's protocol, where Rot$_r$ and Rot$_l$ denote right-rotate and left-rotate, respectively; (e) Multi-head optimization.

from the same diagonal to be packed into a single ciphertext. This requires $\log L$ rotations per ciphertext to accumulate the partial sums, significantly increasing the number of rotations.

Additionally, direct row rotation is not natively supported but can be achieved through masking followed by rotation, named *inner rotation*. As shown in Figure 7 (d), to rotate $B\{1,:\}$ one step to the left, we apply three masks to extract $[b_{00},b_{01},b_{02},b_{03},0,\ldots]$, $[\ldots,0,b_{11},b_{12},b_{13}]$, and $[\ldots,b_{10},\ldots]$. We then rotate $[\ldots,0,b_{11},b_{12},b_{13}]$ left by one step and $[\ldots,b_{10},\ldots]$ right by three steps. Adding the three results gives the final rotated output. Step 1 requires D inner rotations, each involving two rotations and ct-pt multiplications. ❷ **Row-wise rotation and multiplication.** We apply inner rotations within each row of B by t steps ($\forall t \in [L]$), generating L ciphertexts. These ciphertexts are multiplied by A to obtain the partial sums for all diagonals. This step requires $L-1$ inner rotations and ct-ct multiplications. ❸ **Diagonal rotation and summation.** After step 2, we obtain L ciphertexts, each containing the partial sums of different diagonals. We then rotate across diagonals to align their positions, extract the partial sums from each diagonal, and sum them to obtain the final result. This step requires L rotations and $2L$ ct-pt multiplications.

Multi-Head Optimization. In multi-head attention, there are H heads that can be computed in parallel as H separate MatMuls. Formally, we need to compute H parallel products $C_h = A_h \times B_h$, where $A_h \in \mathbb{R}^{L\times D}$ and $B_h \in \mathbb{R}^{D\times L}$ for $h \in [H]$. We observe that **different heads are independent**, allowing us to pack multiple heads into a single ciphertext. Specifically, instead of packing the entire matrix A_h into a ciphertext, we pack D/H columns of all A_h matrices into one ciphertext. This results in a ciphertext containing a tensor of dimensions $(L,H,D/H)$, as shown in Figure 7 (e) for $H=2$. This multi-head packing (MHP) reduces the rotations in steps 1 and 3 of our MatMul protocol. In step 1, each row of B needs an inner rotation. With MHP, we can rotate H heads in parallel, reducing the row count from D to D/H. As shown in Figure 7 (e), step 1 can be skipped since D/H is exactly 1. Moreover, after step 2, the partial sums of the same diagonal are aligned at the same position. This enables us to sum these ciphertext results first in Step 3, then apply the rotation. In contrast, without MHP, each ciphertext would need to be rotated individually.

To achieve MHP in matmul$_{cc}$, the packing of neighboring matmul$_{cp}$ needs to change. Specifically, for matmul$_{cp}$ + matmul$_{cc}$, we need to reorder the columns of the weight matrix in plaintext in matmul$_{cp}$ to match the MHP format of matmul$_{cc}$'s inputs. For matmul$_{cc}$ + matmul$_{cp}$, we need to reorder the rows of the weight matrix in plaintext to align with the MHP format of matmul$_{cc}$'s output. Note that such reordering does not affect the computational complexity of matmul$_{cp}$.

BSGS Optimization. The BSGS optimization splits rotations into baby-step and giant-step phases and is widely used to reduce the number of rotations in matrix multiplications [21, 45]. In our Fused MatMuls protocol, the rotations in steps 1 and 2 can be further reduced using BSGS with a detailed explanation provided in Appendix C.1. The multiplication depth of our protocol is 4, the same as Powerformer [46]. Although BOLT has a depth of 2, its overall latency is much higher due to excessive rotations, as shown in § 7.

5.2 Efficient Fused Computations in Multi-Head Attention

We explain how our protocol is applied to fused computations in multi-head attention. For $Q_h K_h^T, \forall h \in [H]$, we first reorder the columns of W_Q and W_K, then apply BOLT's protocol to compute Q_h and K_h^T. Q_h and K_h are both in spatial-first

Table 5: Theoretical complexity comparison of $\text{matmul}_{\text{cc}}$ protocol across prior works. "CMult." refers to the ct-ct multiplications. m, d, H are defined in § 2.2. The data is based on the dimensions in BERT-large, with $s = 16384$ indicating the length of a plaintext vector.

$\text{matmul}_{\text{cc}}$		BOLT [45]	Powerformer [46]	BLB
$Q_h K_h^T$ $h \in [H]$	# Rot	$O(\frac{md}{s}(m+\log_2 m))$ 18432	$O(\frac{md}{s}(\frac{5}{2}m+4\sqrt{m}))$ 5856	$O(\frac{md}{s}(\frac{\sqrt{d}}{H}+\sqrt{m})+m)$ 640
	# CMult.	$O(\frac{m^2 d}{s})$ 2048	$O(\frac{m^2 d}{s})$ 1024	$O(\frac{m^2 d}{s})$ 1024
Softmax $\times V_h$ $h \in [H]$	# Rot	$O(\frac{m^2 H}{s}(m+\log_2 m))$ 28560	$O(\frac{m^2 H}{s}(\frac{5}{2}m+4\sqrt{m}))$ 6508	$O(\frac{m^2 H}{s}(\sqrt{\frac{m}{H}}+\sqrt{m})+m)$ 1056
	# CMult.	$O(\frac{m^2 d}{s})$ 1024	$O(\frac{m^3 H}{s})$ 2048	$O(\frac{m^3 H}{s})$ 2048

packing, thus K_h^T is in reduce-first packing. These are then multiplied using our ct-ct MatMul protocol, resulting in $Q_h K_h^T$ in diagonal packing. Since the next operator is nonlinear, we convert $Q_h K_h^T$ to secret-share form and repack it.

For $\text{Concat}(\text{Softmax}(\cdot) \times \text{V}_h) W_O$, $\text{Softmax}(\cdot)$ is already in secret-share form, so we can directly repack it to the multi-head spatial-first packing. We then convert V_h to secret-share form and repack it to the multi-head reduce-first packing. Our $\text{matmul}_{\text{cc}}$ protocol requires the input dimension and the output dimension to be the same, so we pad with zeros if $\text{Softmax}(\cdot) \times V_h$ does not meet this condition. After $\text{matmul}_{\text{cc}}$, we get Att_h in multi-head diagonal packing. To fuse with the next $\text{matmul}_{\text{cp}}$, we reorder the rows of W_O and apply a modified $\text{matmul}_{\text{cp}}$ protocol to get the result in a spatial-first packing. The details are in Appendix C.2, and the complexity of $\text{matmul}_{\text{cp}}$ remains the same as in BOLT.

Theoretical complexity. Table 5 shows the complexity comparison of BLB with prior-art methods in the number of rotations and ct-ct multiplications. Our technique saves about 29× and 8× rotations compared to BOLT and Powerformer.

6 Secure Conversion between CKKS and MPC

To mitigate the growth in ciphertext bit width after fusion, BLB utilizes the CKKS HE for evaluating linear operators. However, we identify a critical security flaw in the previously proposed CKKS-to-MPC conversion protocol [14]. This section analyzes the security issues in the existing protocol [14] and presents our secure conversion protocol.

6.1 Problem Statement

To understand the security issue of [14], we first illustrate the detailed mechanisms of the CKKS scheme.

6.1.1 CKKS Approximate HE and Message Encoding

The message space of the CKKS scheme is $\mathbb{A}_{N,q}$, a polynomial ring of coefficient modulus q. Let $\hat{x}, \hat{y} \in \mathbb{A}_{N,q}$ being

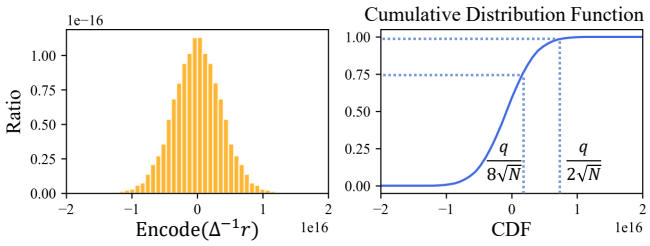

Figure 8: This example demonstrates the distribution of the coefficients of $\text{Encode}(\Delta^{-1} r) \in \mathbb{A}_{N,q}$ for uniform random vector $r \in \mathbb{Z}_q^{N/2}$ with $N = 8192$ and $q \approx 2^{60}$. From this example, we can see that the output distribution is highly concentrated within the range $[-q/(8\sqrt{N}), q/(8\sqrt{N})]$.

two messages. The CKKS scheme supports approximate homomorphic addition $\text{Dec}(\text{Enc}(\hat{x}) \boxplus \text{Enc}(\hat{y})) \approx \hat{x} + \hat{y} \in \mathbb{A}_{N,q}$ and approximate homomorphic multiplication $\text{Dec}(\text{Enc}(\hat{x}) \boxtimes \text{Enc}(\hat{y})) \approx \hat{x} \cdot \hat{y} \in \mathbb{A}_{N,q}$.

One of the most important features of the CKKS scheme is supporting computation on ciphertexts of real values. Particularly, the CKKS scheme utilizes a function $\pi : \mathbb{A}_{N,q} \mapsto \mathbb{R}^{N/2}$ to achieve this feature. The CKKS scheme selects a positive real value Δ called scaling factor and defines the encoding and decoding functions as follows[2].

$$\begin{aligned} \text{Encode}(m \in \mathbb{R}^{N/2}) &= \lfloor \Delta \cdot \pi^{-1}(m) \rceil \in \mathbb{A}_{N,q}, \\ \text{Decode}(\hat{m} \in \mathbb{A}_{N,q}) &= \pi(\lfloor \Delta^{-1} \cdot \hat{m} \rceil) \in \mathbb{R}^{N/2}. \end{aligned} \tag{2}$$

The rounding $\lfloor \cdot \rceil$ is applied to each polynomial coefficient. Indeed, the mapping π^{-1} is realized via a fast Fourier transform (FFT).

6.1.2 From CKKS to MPC

The conversion from CKKS to MPC can be described as an interactive two-party protocol that takes $\text{Enc}(\text{Encode}(m))$ a CKKS ciphertext of $m \in \mathbb{R}^{N/2}$ as the input, and generates two integer vectors $u_0, u_1 \in \mathbb{Z}_q^{N/2}$ such that $\lfloor \Delta \cdot m \rceil \approx u_0 + u_1 \bmod q$. The approximation here is due to the approximate property of the CKKS scheme. In other words, the integer vectors u_0, u_1 can be viewed as the additive secret shares of the real vector m over $\mathbb{Z}_q$ with the fixed-point scale $s \approx \log_2 \Delta$. Both q and Δ are parameters of the CKKS scheme.

6.2 The Pitfalls of Previous Conversion

MP2ML [14] proposed a CKKS-to-MPC conversion protocol, which has been utilized by many other works such as [2, 53]. Their protocol involves the following steps:

- Server samples uniformly random vectors $r \in \mathbb{Z}_q^{N/2}$ and encodes it as $\hat{r} = \text{Encode}(\Delta^{-1} \cdot r) \in \mathbb{A}_{N,q}$.

[2] Indeed, the CKKS encoding supports a complex space $\mathbb{C}^{N/2}$. We omit the imaginary part and consider $\mathbb{R}^{N/2}$ to simplify the presentation.

Functionality $\mathcal{F}_{C2M}$

Receive a CKKS encryption $\mathrm{Enc}(\hat{a}) \in \mathbb{A}_{N,q}^2$ from P_1 and receive the decryption key from P_0, this functionality

1. Samples $\hat{r} \in \mathbb{A}_{N,q}$ uniformly at random.
2. Decrypt $\mathrm{Enc}(\hat{a})$ with decryption key and gets $\hat{a}$.
3. Gives $\lceil \mathrm{Decode}(\hat{a}+\hat{r} \bmod 2^l) \rfloor \bmod 2^l \in \mathbb{Z}_{2^l}^{N/2}$ to P_0.
4. Gives $\lceil \mathrm{Decode}(-\hat{r} \bmod 2^l) \rfloor \bmod 2^l \in \mathbb{Z}_{2^l}^{N/2}$ to P_1.

Figure 9: CKKS-to-MPC Conversion Functionality

- Server then masks the ciphertext through homomorphic addition: $C = \mathrm{Enc}(\mathrm{Encode}(m)) \boxplus \hat{r}$.
- The masked ciphertext C is then sent to the client, who obtains his share $u_0 = \lfloor \Delta \cdot \mathrm{Decode}(\mathrm{Dec}(C)) \rceil \bmod q$ after decryption. The server's share is defined as $u_1 = -r \bmod q$.

However, although the vector r is sampled uniformly at random, the distribution of the encoding $\hat{r}$ is narrowly concentrated around zero due to the properties of FFT, as demonstrated in Figure 8. Particularly, the values of $\hat{r}$ majorly range over $[-q/(8\sqrt{N}), q/(8\sqrt{N})]$. Since a relatively large dimension N is commonly used for the CKKS scheme, such as $N \geq 2^{13}$, this bell-shaped distribution fails to mask the high-end bits of the message m.

In private transformer inference, the server starts with a CKKS ciphertext of $\mathrm{Encode}(x \odot w)$ where x is the client's input and w is the server's model weight. The server aims to convert the encryption of $\mathrm{Encode}(x \odot w)$ to the additive secret share form by adding the mask $\mathrm{Encode}(x \odot w) + \hat{r}$. However, the masking polynomial $\hat{r}$ distributes narrowly around zero while the distribution expectation of $\mathrm{Encode}(x \odot w)$ depends on the message $x \odot w$. Thus, the zero-centered and bell-shaped masking $\hat{r}$ may fail to mask the messages when the expectation of $x \odot w$ is far from zero. More importantly, when the product magnitude $\|\Delta \cdot x \odot w\|_\infty$ is relatively large, such as $> q/8$, some high-end bits of w will be leaked to the client. All these above lead to the conclusion that the narrowly distributed random masking used by MP2ML [14] is insecure.

Beyond security concerns, the previous conversion suffers from limited efficiency since it only supports MPC protocols that operate on $\mathbb{Z}_q$. In contrast, MPC protocols over $\mathbb{Z}_{2^l}$ are significantly more communication-efficient [12, 47]. Thus, a modulus conversion protocol between $\mathbb{Z}_q$ and $\mathbb{Z}_{2^l}$ is essential to enable lightweight MPC protocols.

6.3 Secure and Efficient Conversion Protocols

Secure Masking. To solve the security issue, we suggest sampling the masking polynomial $\hat{r}$ directly from $\mathbb{A}_{N,q}$ uniformly at random rather than computing it via the CKKS encoding function, as described in Theorem 1. The proof is available in Appendix A.

Algorithm 1: Secure CKKS to MPC conversion

Input: P_1 holds $\mathrm{Enc}(\mathrm{Encode}(x)) \in \mathbb{A}_{N,q}^2$.
Output: $\forall b \in \{0,1\}$, P_b gets $[\![x]\!]_b^{2^l}$

1 P_1 samples $\hat{r} \in \mathbb{A}_{N,q}$ uniformly at random and computes $\mathrm{Enc}(\mathrm{Encode}(x)) \boxplus \hat{r}$. Then P_1 sends the masked ciphertexts to P_0 and sets $[\![\mathtt{tmp}]\!]_1^q = -\hat{r}$.
2 P_0 decrypts to get $[\![\mathtt{tmp}]\!]_0^q = \mathrm{Encode}(x) + \hat{r}$.
3 P_0 and P_1 invoke $[\![\mathtt{tmp}]\!]^{2^l} = \text{Field-to-Ring}([\![\mathtt{tmp}]\!]^q)$.
4 $\forall b \in \{0,1\}$, P_b locally evaluates the CKKS decoding and learn $[\![x]\!]_b^{2^l} = \lceil \mathrm{Decode}([\![\mathtt{tmp}]\!]_b^{2^l}) \rfloor$.

Algorithm 2: Secure MPC to CKKS conversion

Input: $\forall b \in \{0,1\}$, P_b holds $[\![x]\!]_b^{2^l}$.
Output: P_1 gets $\mathrm{Enc}(\mathrm{Encode}(x)) \in \mathbb{A}_{N,q}^2$.

1 $\forall b \in \{0,1\}$, P_b locally evaluates the CKKS encoding and learn $[\![\mathtt{tmp}]\!]_b^{2^l} = \mathrm{Encode}([\![x]\!]_b^{2^l})$.
2 P_0 and P_1 invoke $[\![\mathtt{tmp}]\!]^q = \text{ring-to-field}([\![\mathtt{tmp}]\!]^{2^l})$.
3 P_0 encrypts $[\![\mathtt{tmp}]\!]_0^q$ as $\mathrm{Enc}([\![\mathtt{tmp}]\!]_0^q)$ and sends it to P_1.
4 P_1 conduct homomorphic addition to get $\mathrm{Enc}(\mathrm{Encode}(x)) = \mathrm{Enc}([\![\mathtt{tmp}]\!]_0^q) \boxplus [\![\mathtt{tmp}]\!]_1^q$.

Theorem 1. *The CKKS-to-MPC conversion in Algorithm 1 securely realizes the $\mathcal{F}_{C2M}$ functionality in Figure 9 against honest-but-curious adversary.*

In the context of private transformer inference, the client first obtains a masked polynomial $\hat{d} = \mathrm{Encode}(x \odot w) + \hat{e} + \hat{r} \bmod q$ for a noisy term $\hat{e}$ after the decryption. This masked polynomial $\hat{d}$ distributes uniformly over $\mathbb{A}_{N,q}$ since $\hat{r} \in \mathbb{A}_{N,q}$ is a uniform random polynomial. No information about w (i.e., server's input) and $\hat{e}$ is revealed to the client.

Efficient and Accurate Conversion Protocol. After secure masking, we obtain the shares of $\mathrm{Encode}(x \odot w)$ over $\mathbb{Z}_q$. To enable lightweight MPC protocols over $\mathbb{Z}_{2^l}$, we utilize the field-to-ring and ring-to-field protocols from [45, 47] for switching between $\mathbb{Z}_q$ and $\mathbb{Z}_{2^l}$, as detailed in Appendix C.3.

Subsequently, both parties perform local decoding in the fixed-point representation over $\mathbb{Z}_{2^l}$ to get the share of $x \odot w$. The decoding involves multiplying by a public FFT matrix. We use the $O(N \log_2 N)$ FFT algorithm to compute the matrix multiplication, which demands $O(\log_2 N)$ multiplicative depths. To avoid extra communications due to fixed-point truncations, we extend the shares to a larger ring, e.g., $\mathbb{Z}^{l+40}$, and conduct local truncations with an error rate below 2^{-40}. Further details are provided in Appendix C.4.

Putting It Together. With these improvements, we present the full CKKS-to-MPC protocol in Algorithm 1. The reverse

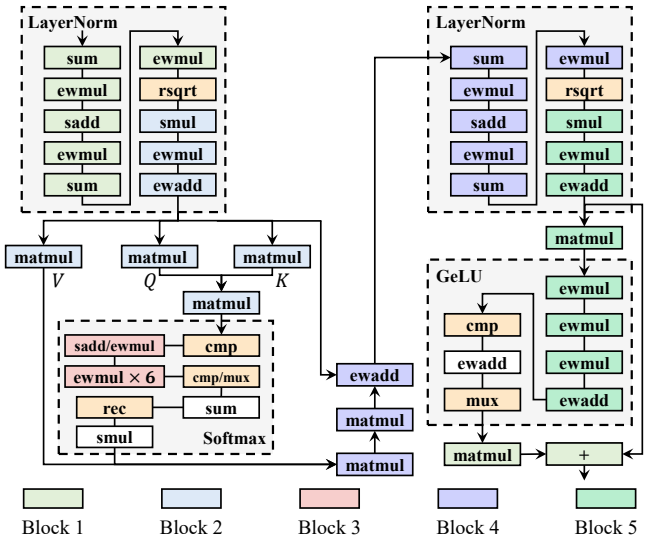

Figure 10: Five blocks of fused linear operators in a Transformer layer using BLB. All the operators shown in the figure are defined in Table 2.

conversion, MPC-to-CKKS, follows a similar process outlined in Algorithm 2. In the MPC-to-CKKS protocol, both parties first perform local encoding for a shared vector $[\![x]\!]^{2^\ell}$, then switch the secret share modulus from 2^ℓ to q using ring-to-field. Finally, the client encrypts its share and sends the ciphertext to the server, which can reconstruct the CKKS encryption of Encode(x) via one homomorphic addition.

7 Evaluation

7.1 Experimental Setup

Implementation. We implement BLB based on the SEAL [50] library and the Ezpc library [3, 37] in C++ . The GPU acceleration was developed based on PhantomFHE [52]. We set the security parameter $\lambda = 128$. We use a bit width of $l = 43$ and scale $s = 13$ for nonlinear operators. We apply BLB to a Transformer block, and after the fine-grained fusion, there are in total 5 blocks of fused linear operators, marked with different colors in Figure 10. To determine the HE parameters (i.e., ciphertext bit width and scale) for each block, we apply the EVA CKKS compiler [8] and require the Mean Square Error (MSE) of CKKS-based computation to be $< 10^{-11}$ compared to the plaintext counterparts. The detailed parameters are shown in Table 6.

Oblivious Transfer (OT) Primitives. Following [29, 41, 45], our nonlinear protocols are implemented with OT. There are two main types of OT primitives, i.e., the IKNP-style OT [36] used in BOLT [45] and the VOLE-style OT [13] used in Bumblebee [41]. For a fair comparison, we evaluate BLB with both OT primitives. Note that OT primitives do not affect the protocol construction of linear and nonlinear operators.

Table 6: CKKS HE parameters for different blocks.

Block	Mul. Depth	Degree N	Ciphertext bit width*	Scale	MSE
1	4	16384	{60,60,60,60,60}	38	7.2×10^{-13}
2	7	32768	{60,35,60,60,60,60,60,60}	44	3.4×10^{-12}
3	7	32768	{60,40,40,40,40,40,40,40,60}	40	5.2×10^{-13}
4	7	32768	{60,60,60,60,60,60,60,60}	43	2.2×10^{-13}
5	6	32768	{60,32,60,60,60,60,60,60}	46	6.9×10^{-14}

* The ciphertext bit width is in the Residue Number System (RNS) representation.

Table 7: The number of HE rotations and latency comparison for $Q_h K_h^T$ and Softmax $\times V_h$ with previous methods.

Model	Operator	Method	# Rot	Latency (s)
BERT-large 128 input tokens	$Q_h K_h^T$ $h \in [H]$	BOLT [45]	14302	24.0
		Powerformer [46]	5856	8.0
		BLB	**640**	**1.7**
	Softmax $\times V_h$ $h \in [H]$	BOLT [45]	14332	25.0
		Powerformer [46]	6508	9.4
		BLB	**1056**	**2.9**
GPT2-base 64 input tokens	$Q_h K_h^T$ $h \in [H]$	BOLT [45]	5356	5.0
		Powerformer [46]	2304	2.5
		BLB	**488**	**1.4**
	Softmax $\times V_h$ $h \in [H]$	BOLT [45]	5374	5.1
		Powerformer [46]	2496	2.6
		BLB	**488**	**1.4**

Protocols for Nonlinear Layers. We use Bumblebee's protocol for LayerNorm and Softmax, and BOLT's protocol for GeLU. These nonlinear layers are further decomposed into operators and then processed using BLB. Appendix B provides a detailed review of these protocols.

Test Environment. All experiments are conducted on machines with a 2.7 GHz Intel Xeon(R) Platinum 8558P CPU and 64 threads. We run the GPU experiments on an NVIDIA A100 GPU. Wireless network conditions are simulated using Linux's traffic control command: for the Local Area Network (LAN), we use 1 Gbps bandwidth and a 0.3ms roundtrip time. For the Wide Area Network (WAN), we simulate three settings: WAN_1 (400Mbps, 4ms), WAN_2 (100Mbps, 4ms) and WAN_3 (100Mbps, 80ms).

Models and Datasets. Following [41, 45], we evaluate BLB on BERT-base [34], BERT-large, and GPT2-base [10] across four datasets from the GLUE benchmarks [54]: MRPC, RTE, SST2, and QNLI. We use the pretrained models from huggingface [30] **without fine-tuning**.

Baselines. We compare BLB with prior-art private Transformer inference frameworks, including hybrid HE/MPC frameworks Iron [27], BOLT [45], and Bumblebee [41]; 3PC frameworks SIGMA [26] and MPCFormer [39]; and the FHE-based framework NEXUS [56].

7.2 Microbenchmarks

MatMul Protocol Comparison. In Table 7, we benchmark BLB on the $Q_h K_h^T$ and Softmax $\times V_h$ from BERT-large and GPT2-base on CPU. Notably, BLB reduces the number of rotations substantially, resulting in 6.9× and 3.1× lower

Table 8: Accuracy comparison between plaintext and BLB inference.

	Bert-base		Bert-large		GPT2-base	
	Plaintext	**BLB**	**Plaintext**	**BLB**	**Plaintext**	**BLB**
MRPC	88.8	88.5	89.2	89.1	82.7	82.5
RTE	69.3	69.7	70.4	70.0	66.1	66.4
SST2	93.2	93.0	92.5	92.4	89.7	89.9
QNLI	91.3	91.2	91.9	91.7	88.9	88.9

latency than BOLT and Powerformer, respectively.

7.3 End-to-End Evaluation

Accuracy Comparison. Table 8 shows the private inference accuracy of BLB. We can see that BLB achieves comparable levels of accuracy with plaintext inference. It should be emphasized that the main errors come from the piecewise linear approximations for nonlinear layers [41], and we **do not perform any fine-tuning** on the models. On certain datasets, approximation slightly improves accuracy. We attribute this to its implicit regularization effect, which helps alleviate overfitting by introducing controlled noise. Similar observations have been reported in Figure 1 in PUMA [9] and Table 2 in BOLT [45].

Communication Breakdown Analysis. In Figure 11, we show the communication breakdown of BLB and compare it with BOLT and Bumblebee. As can be observed, **1)** BLB reduces communication by approximately 4× and 2× across three models compared to BOLT and Bumblebee, respectively. **2)** The improvement stems from the elimination of truncations and a reduction in HE/MPC conversions. **3)** BOLT relies on OT for MatMul operators in nonlinear layers, which introduces extra "OT message" communication. In contrast, Bumblebee and BLB use HE for all linear operators, removing the need for "OT message" communication. Besides, the "other nonlinear" parts including cmp, rec and rsqrt operators.

Latency Breakdown Analysis. Figure 12 and Figure 13 present the latency breakdown of BLB, BOLT, and Bumblebee across four network settings using the BERT-base model on both CPU and GPU. Notably, BLB uses the same OT primitive as both BOLT and Bumblebee. The key observations are as follows: **1)** Compared to BOLT, BLB (IKNP) achieves a $3.3 \sim 4.0\times$ latency reduction on CPU and a $3.6 \sim 4.9\times$ reduction on GPU. **2)** Compared to Bumblebee, there exists a clear trade-off between communication and computation. After fusion, the HE *level* increases, which leads to higher computational cost and consequently increased latency under LAN compared to Bumblebee. Nevertheless, our design adheres to the optimization principle of **"accuracy > communication > computation"**. That is, while ensuring accuracy, communication overhead is minimized even at the expense of increased computation. This prioritization is justified by the fact that computation can be more readily accelerated and parallelized through hardware and system-level improvements [18, 23, 52]. As shown in Figure 13, on GPU, BLB achieves a $1.5 \sim 1.8\times$ latency reduction compared to Bumblebee.

Table 9: End-to-end comparison with existing private Transformer inference frameworks. Iron and NEXUS results are taken from their papers due to a lack of end-to-end implementations. For other baselines, including SIGMA [26], MPCFormer [39], BOLT [45] and Bumblebee [41], we re-run their codes on the same machine as BLB. For SIGMA, we report the latency of both the offline key transmission and online inference, whereas the original SIGMA paper only reports the latency of the online phase. We use VOLE-OT for BLB.

Model	Framework	Latency (min)			Comm. (GB)
		LAN	WAN_2	WAN_3	
GPT2-base 64 input tokens	SIGMA [26]*	7.7	25.1	38.5	28.7
	MPCFormer [39]*	2.1	7.2	10.5	7.3
	BOLT [45]	9.7	34.0	53.6	34.8
	Bumblebee [41]	**2.9**	**5.2**	12.3	2.5
	BLB	4.0	6.8	**10.2**	**1.5**
	BOLT (GPU) [45]	7.6	31.8	51.6	34.8
	Bumblebee (GPU) [41]	2.6	4.9	12.0	2.5
	BLB (GPU)	**2.0**	**3.9**	**8.1**	**1.5**
BERT-base 128 input tokens	NEXUS [56]	457	458	470	0.2
	Iron [27]	66.7	/	/	76.5
	SIGMA [26]*	7.8	30.4	48.6	34.4
	MPCFormer [39]*	5.5	18.0	27.5	12.1
	BOLT [45]	22.1	85.0	130.0	63.6
	Bumblebee [41]	**5.0**	12.3	22.1	5.8
	BLB	6.1	**10.5**	**17.2**	**3.0**
	NEXUS (GPU) [56]	19.9	20.8	32.5	0.2
	BOLT (GPU) [45]	15.5	77.9	122.7	63.6
	Bumblebee (GPU) [41]	4.3	11.6	21.3	5.8
	BLB (GPU)	**2.5**	**6.6**	**13.2**	**3.0**
BERT-large 128 input tokens	Iron [27]	92.0	/	/	220.0
	SIGMA [26]*	20.5	102.5	155.5	92.8
	MPCFormer [39]*	7.7	34.1	52.0	32.6
	BOLT [45]	57.6	222.0	274.4	158.9
	Bumblebee [41]	**10.6**	32.4	51.3	15.2
	BLB	15.1	**29.0**	**37.7**	**7.8**
	BOLT (GPU) [45]	43.8	208.2	260.6	158.9
	Bumblebee (GPU) [41]	9.7	30.8	49.2	15.2
	BLB (GPU)	**6.6**	**16.2**	**24.9**	**7.8**

* Frameworks that require three parties.

Comparison with Existing Frameworks. We present the end-to-end comparisons of BLB with prior-art private inference frameworks across three Transformer models in Table 9. In summary, BLB achieves state-of-the-art communication and latency performance, with superior GPU efficiency. Specifically, BLB reduces communication by $1.7 \sim 29\times$ and latency by $1.4 \sim 13\times$ on GPU. Although NEXUS [56] achieves lower communication, it remains less efficient than BLB in terms of latency due to its reliance on heavy bootstrapping.

7.4 Ablation Study

Comparison with BFV. In this study, we assess the performance of the CKKS scheme relative to the BFV scheme. The results are summarized in Table 10 where block idx refers to

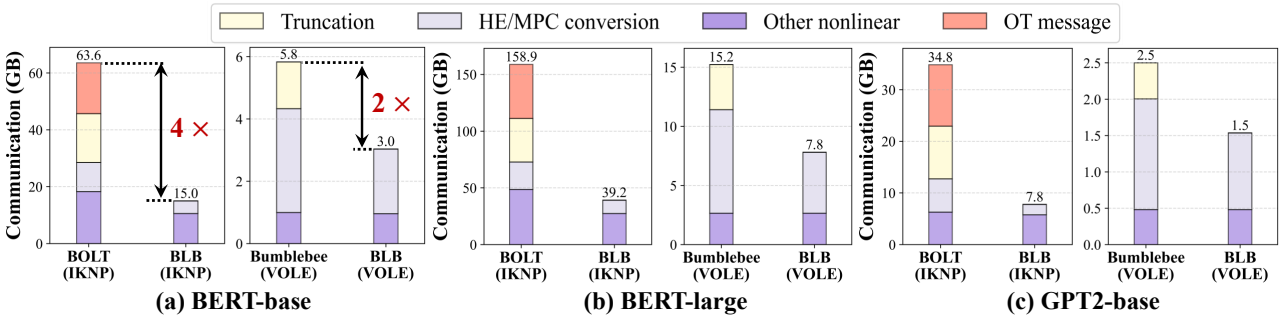

Figure 11: Communication comparison between BLB, BOLT and Bumblebee. The ones using the same OT primitive are grouped.

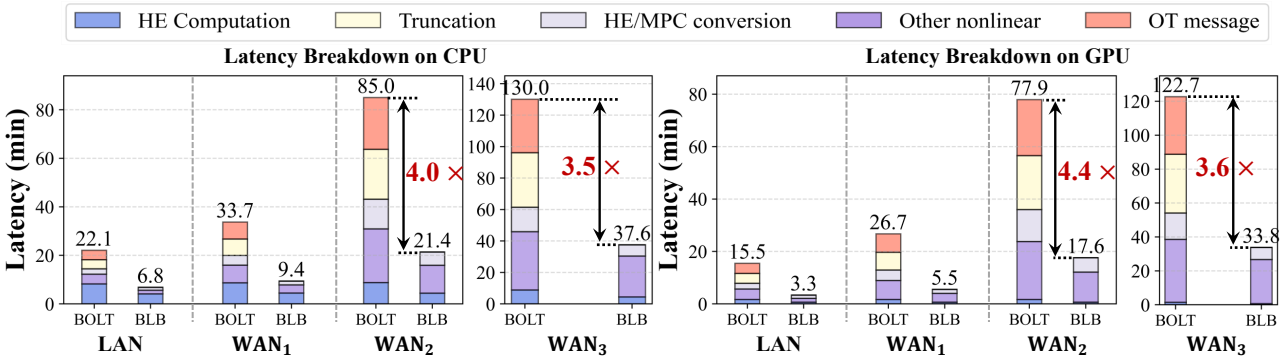

Figure 12: Latency comparison between BLB and BOLT on BERT-base model on CPU and GPU. The bandwidths and round-trip times of four network conditions are: LAN:{ 1Gbps, 0.3ms}, WAN_1:{400Mbps, 4ms}, WAN_2:{100Mbps, 4ms}, WAN_3:{100Mbps, 80ms}.

Table 10: Comparison of ciphertext bit width, degree, communication, and latency between BFV and CKKS.

Block idx	Ciphertext Bit Width (bits)		Degree		Comm. (MB)		Latency (s)	
	BFV	CKKS	BFV	CKKS	BFV	CKKS	BFV	CKKS
1	414	300	16384	16384	55	31	10	6
2	675	455	32768	32768	62	23	12	7
3	1400	400	65536*	32768	/	46	/	2
4	636	420	32768	32768	81	40	14	8
5	902	452	65536*	32768	/	26	/	5

* Degrees larger than 32768 are not supported in the SEAL library.

Table 11: Ablation study on different components.

Model	Framework	Comm. (GB)	Latency (min)		
			LAN	WAN_2	WAN_3
GPT2-base 64 input tokens	baseline (BFV+MPC)	2.5	2.6	4.9	12.0
	+FineGrainFusion/ CKKS-MPC Conversion	1.5	2.8	4.7	9.0
	+Fused MatMul Protocol	1.5	2.0	3.9	8.1
BERT-base 128 input tokens	baseline (BFV+MPC)	5.8	4.3	11.6	21.3
	+FineGrainFusion/ CKKS-MPC Conversion	3.0	4.1	8.2	14.8
	+Fused MatMul Protocol	3.0	2.5	6.6	13.2
BERT-large 128 input tokens	baseline (BFV+MPC)	15.2	9.7	30.8	49.2
	+FineGrainFusion/ CKKS-MPC Conversion	7.8	10.7	20.4	29.1
	+Fused MatMul Protocol	7.8	6.6	16.2	24.9

the five blocks of fused linear operators in a Transformer layer, as shown in Figure 10. The key observations are as follows: **1)** The BFV scheme requires a larger ciphertext bit width due to its greater scale. This necessitates a higher polynomial degree to achieve the desired security level λ, a requirement that is not supported by certain homomorphic encryption libraries, such as SEAL [50]. **2)** The communication overhead of BFV is approximately twice that of CKKS, primarily because of its larger plaintext bit width, which increases the ciphertext size. In contrast, CKKS mitigates this issue by controlling the scale expansion, thereby reducing the communication cost. **3)** Given its larger ciphertext bit width, BFV incurs significantly higher computational costs compared to CKKS, resulting in substantially increased latency. These observations highlight the advantages of the CKKS scheme, particularly in the efficient evaluation of consecutive linear operators.

Effectiveness of Different Components. We present ablation studies across three models in Table 11, using Bumblebee's protocol with BFV and MPC as a strong baseline without fusion. Applying FineGrainFusion and our hybrid CKKS-MPC protocol achieves around a 2× reduction in communication, with some increase in HE computation. This trade-off becomes more favorable under poor network conditions. Under

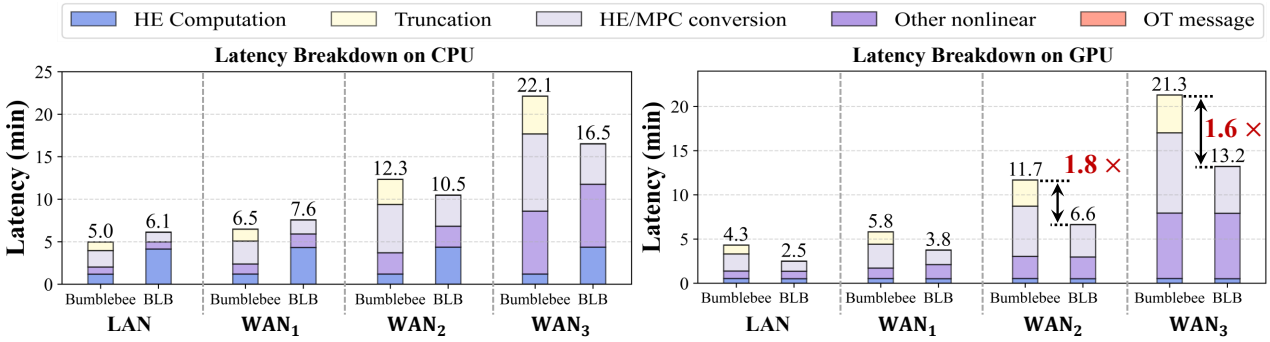

Figure 13: Latency comparison between BLB and Bumblebee on BERT-base model on CPU and GPU. The bandwidths and round-trip times of four network conditions are: LAN:{ 1Gbps, 0.3ms}, WAN_1:{400Mbps, 4ms}, WAN_2:{100Mbps, 4ms}, WAN_3:{100Mbps, 80ms}.

WAN_3, FineGrainFusion reduces latency by $1.3 \sim 1.8\times$. Incorporating the fused MatMul protocol further lowers HE computation, leading to a $1.4 \sim 1.6\times$ reduction in end-to-end latency under LAN.

Comparison with NEXUS under Batched Inputs. We provide this experiment in Appendix D. It should be emphasized that batching is challenging in private inference because data from different users **cannot** be packed into the same ciphertext, as each user's data is encrypted under a different secret key. As a result, batching is only applicable to multiple queries from the same user, which significantly limits its use.

8 Related works

Private Transformer Inference. With the proliferation of ChatGPT, significant efforts have been made to enable private Transformer inference [1, 4, 9, 11, 21, 26–28, 39, 41, 42, 44–46, 56]. Iron [27] is the first 2PC-based framework that optimizes the packing for $\text{matmul}_{\text{cp}}$ and nonlinear layer protocols. Building on this, BOLT [45] and Bumblebee [41] further optimize the packing strategies and propose accurate piecewise linear approximations for nonlinear layers without fine-tuning. Other works [11, 28, 42] further improve 2PC protocols for nonlinear operators. FHE-based frameworks have also been explored for private Transformer inference. NEXUS [56] is the first FHE-based framework that features novel HE-based computation of linear and nonlinear Transformer operators. However, NEXUS requires high-order polynomial approximation for nonlinear layers and thus suffers from high computational costs. Powerformer [46] improves upon NEXUS by incorporating the $\text{matmul}_{\text{cc}}$ protocol from [31] but still suffers from high cost for non-square MatMuls in Transformers. THOR [44] proposes a new diagonal packing algorithm to further reduce the HE rotations but at the cost of higher multiplicative depth of both $\text{matmul}_{\text{cp}}$ and $\text{matmul}_{\text{cc}}$. Additionally, protocols such as PUMA [9], MPCFormer [39], PrivFormer [1], and SIGMA [26] have also studied three-party computation schemes for private Transformer inference.

Layer Fusion Optimization. Layer fusion has been proposed in several works to reduce the cost of private inference. PrivCirNet [21] fuses consecutive convolution layers in MobileNetV2 [48]. Zhang et al. [57] jointly optimizes the evaluation of the convolution layer and ReLU layer in CNNs, reducing the number of HE rotations and multiplications. For Transformer, BOLT [45] fuses two consecutive MatMuls in the attention layers. All these approaches are based on layer-wise evaluation, which limits their performance improvements.

9 Conclusion

We propose a hybrid HE/MPC framework, dubbed BLB, to remodel private Transformer inference. BLB breaks the layer barrier in Transformers to allow fine-grained fusion of linear operators, drastically reducing the communication induced by HE/MPC conversion. BLB employs CKKS-based computation of fused linear operators to overcome the ciphertext bit width increase, enabled by our first secure CKKS/MPC conversion protocol. BLB further designs efficient packing algorithms for fused matrix multiplications with multi-head and BSGS optimizations. Our experiments demonstrate up to $21\times$ communication reduction compared to existing methods, leading to up to $13\times$ latency reduction.

10 Ethical Considerations

This work focuses on improving the efficiency of privacy-preserving inference frameworks. All datasets and models used are publicly available; no proprietary, sensitive, or personal data is involved. We followed strict ethical standards to ensure no privacy violations occurred during experimentation or could result from our proposed methods.

Our contributions aim to strengthen secure computation without introducing new risks to data security or user privacy. By enhancing the practicality and scalability of privacy-

preserving machine learning, this work encourages responsible adoption of privacy-centric technologies.

The ethical impact has been carefully evaluated. Since the methods do not involve sensitive data and are designed to bolster privacy and security in machine learning applications. We believe this work aligns with principles of privacy, security, and responsible innovation.

11 Open Science

To promote availability, all datasets and pre-trained models used in this work are publicly accessible through their sources, as referenced in the paper. Upon acceptance, we will release the experimental logs and testing code in a public repository to enable immediate availability of our results. Further organization and release of the full source code, including implementation details, will follow in subsequent updates. The repository will be accompanied by documentation describing the experimental setup, including hardware configurations, software dependencies, and parameter settings. By adhering to open science principles, we aim to support validation efforts and foster further research in privacy-preserving inference.

References

[1] Yoshimasa Akimoto, Kazuto Fukuchi, Youhei Akimoto, and Jun Sakuma. Privformer: Privacy-preserving transformer with mpc. In *2023 IEEE 8th EuroS&P*, pages 392–410. IEEE, 2023.

[2] Shashank Balla and Farinaz Koushanfar. Heliks: He linear algebra kernels for secure inference. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 2306–2320, 2023.

[3] Nishanth Chandran and Divya Gupta et al. Ezpc: Programmable, efficient, and scalable secure two-party computation for machine learning. *Cryptology ePrint Archive*, 2017.

[4] Yuntian Chen and Zhanyong Tang et al. Accelerating private large transformers inference through fine-grained collaborative computation. *arXiv preprint arXiv:2412.16537*, 2024.

[5] Peng Cheng and Utz Roedig. Personal voice assistant security and privacy—a survey. *Proceedings of the IEEE*, 110(4):476–507, 2022.

[6] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. A full rns variant of approximate homomorphic encryption. In *Selected Areas in Cryptography–SAC 2018: 25th International Conference*, pages 347–368. Springer, 2019.

[7] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security*, pages 409–437. Springer, 2017.

[8] Roshan Dathathri and Blagovesta Kostova et al. Eva: An encrypted vector arithmetic language and compiler for efficient homomorphic computation. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2020.

[9] Ye Dong and Wen jie Lu et al. Puma: Secure inference of llama-7b in five minutes. *arXiv preprint arXiv:2307.12533*, 2023.

[10] Alec Radford et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[11] Chenkai Zeng et al. Securegpt: A framework for multi-party privacy-preserving transformer inference in gpt. *IEEE Transactions on Information Forensics and Security*, 2024.

[12] Deevashwer Rathee et al. Cryptflow2: Practical 2-party secure inference. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 325–342, 2020.

[13] Elette Boyle et al. Efficient two-round ot extension and silent non-interactive secure computation. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*.

[14] Fabian Boemer et al. Mp2ml: A mixed-protocol machine learning framework for private inference. In *Proceedings of the 15th International Conference on Availability, Reliability and Security*, pages 1–10, 2020.

[15] Hugo Touvron et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[16] Jingwei Chen et al. Secure transformer-based neural network inference for protein sequence classification. *Cryptology ePrint Archive*, 2024.

[17] Manuel B Santos et al. Curl: Private llms through wavelet-encoded look-up tables. *Conference on Applied Machine Learning for Information Security*, 2024.

[18] Nikola Samardzic et al. Craterlake: a hardware accelerator for efficient unbounded computation on encrypted data. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022.

[19] Pratyush Mishra et al. Delphi: A cryptographic inference service for neural networks, Jan 2020.

[20] Siam Umar Hussain et al. Coinn: Crypto/ml codesign for oblivious inference via neural networks. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 3266–3281, 2021.

[21] Tianshi Xu et al. Privcirnet: Efficient private inference via block circulant transformation. *Neural Information Processing Systems (NeurIPS)*, 2024.

[22] Yongqin Wang et al. Characterization of mpc-based private inference for transformer-based models. In *2022 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 187–197.

[23] Samardzic Nikola et el. F1: A fast and programmable accelerator for fully homomorphic encryption. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 238–252, 2021.

[24] Oded Goldreich. Secure multi-party computation. *Manuscript. Preliminary version*, 78(110):1–108, 1998.

[25] Oded Goldreich. *The Foundations of Cryptography - Volume 2: Basic Applications*. Cambridge University Press, 2004.

[26] Kanav Gupta, Neha Jawalkar, Ananta Mukherjee, Nishanth Chandran, Divya Gupta, Ashish Panwar, and Rahul Sharma. Sigma: Secure gpt inference with function secret sharing. *Cryptology ePrint Archive*, 2023.

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

[28] Hai Huang and Yongjian Wang. Secbert: Privacy-preserving pre-training based neural network inference system. *Neural Networks*, 172:106135, 2024.

[29] Zhicong Huang, Wen-jie Lu, Cheng Hong, and Jiansheng Ding. Cheetah: Lean and fast secure {Two-Party} deep neural network inference. In *USENIX Security Symposium 2022*, pages 809–826, 2022.

[30] Huggingface. https://huggingface.co/.

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

[32] Jae Hyung Ju, Jaiyoung Park, Jongmin Kim, Donghwan Kim, and Jung Ho Ahn. Neujeans: Private neural network inference with joint optimization of convolution and bootstrapping. *The ACM Conference on Computer and Communications Security (CCS)*, 2024.

[33] Chiraag Juvekar, Vinod Vaikuntanathan, and AnanthaP. Chandrakasan. GAZELLE: A low latency framework for secure neural network inference, Jan 2018.

[34] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of naacL-HLT*, volume 1, page 2, 2019.

[35] Dongwoo Kim and Cyril Guyot. Optimized privacy-preserving cnn inference with fully homomorphic encryption. *IEEE Transactions on Information Forensics and Security*, 18:2175–2187, 2023.

[36] Vladimir Kolesnikov and Ranjit Kumaresan. Improved ot extension for transferring short secrets. In *Advances in Cryptology–CRYPTO 2013: 33rd Annual Cryptology Conference, 2013.*, pages 54–70, 2013.

[37] Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow: Secure tensorflow inference. In *2020 IEEE Symposium on Security and Privacy (SP)*, 2020.

[38] Eunsang Lee, Joon-Woo Lee, Junghyun Lee, Young-Sik Kim, Yongjune Kim, Jong-Seon No, and Woosuk Choi. Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions. In *International Conference on Machine Learning*, pages 12403–12422. PMLR, 2022.

[39] Dacheng Li, Rulin Shao, Hongyi Wang, Han Guo, Eric P Xing, and Hao Zhang. Mpcformer: fast, performant and private transformer inference with mpc. *arXiv preprint arXiv:2211.01452*, 2022.

[40] Yun Li, Yufei Duan, Zhicong Huang, Cheng Hong, Chao Zhang, and Yifan Song. Efficient 3pc for binary circuits with application to maliciously-secure dnn inference. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 5377–5394, 2023.

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

[42] Jinglong Luo and Yehong Zhang et al. Secformer: Fast and accurate privacy-preserving inference for transformer models via smpc. In *Findings of the Association for Computational Linguistics ACL*, 2024.

[43] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE symposium on security and privacy (SP)*, pages 19–38. IEEE, 2017.

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

[45] Q. Pang, J. Zhu, H. Möllering, W. Zheng, and T. Schneider. Bolt: Privacy-preserving, accurate and efficient inference for transformers. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 133–133, Los Alamitos, CA, USA, may 2024. IEEE Computer Society.

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

[47] Deevashwer Rathee, Mayank Rathee, Rahul Kranti Kiran Goli, Divya Gupta, Rahul Sharma, Nishanth Chandran, and Aseem Rastogi. Sirnn: A math library for secure rnn inference. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1003–1020. IEEE, 2021.

[48] Mark Sandler and Andrew Howard et al. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[49] Prodip Kumar Sarker, Qingjie Zhao, and Md Kamal Uddin. Transformer-based person re-identification: a comprehensive review. *IEEE Transactions on Intelligent Vehicles*, 2024.

[50] Microsoft SEAL (release 3.6). https://github.com/Microsoft/SEAL, November 2020. Microsoft Research, Redmond, WA.

[51] Fahad Shamshad and Salman Khan et al. Transformers in medical imaging: A survey. *Medical Image Analysis*, 88:102802, 2023.

[52] Shiyu Shen and Hao Yang et al. Leveraging gpu in homomorphic encryption: Framework design and analysis of bfv variants. *IEEE Transactions on Computers*, 73(12):2817–2829, 2024.

[53] Sarabjeet Singh, Shreyas Singh, Sumanth Gudaparthi, Xiong Fan, and Rajeev Balasubramonian. Hyena: Balancing packing, reuse, and rotations for encrypted inference. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 107–107. IEEE Computer Society, 2024.

[54] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding, 2019.

[55] Wenxuan Zeng, Meng Li, Wenjie Xiong, Wenjie Lu, Jin Tan, Runsheng Wang, and Ru Huang. Mpcvit: Searching for mpc-friendly vision transformer with heterogeneous attention. *arXiv preprint arXiv:2211.13955*, 2022.

[56] Jiawen Zhang and Xinpeng Yang et al. Secure transformer inference made non-interactive. Network and Distributed System Security (NDSS), 2025.

[57] Qiao Zhang, Tao Xiang, Chunsheng Xin, and Hongyi Wu. From individual computation to allied optimization: Remodeling privacy-preserving neural inference with function input tuning. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 101–101.

A Security Discussion

Security of BLB In our threat model, we assume that both the client and the server behave *honestly-but-curiously*. The methods introduced in § 4 and § 5 are built upon well-established cryptographic foundations, including CKKS and MPC protocols, which have been thoroughly validated as secure [7, 24]. Additionally, the security of the proposed CKKS and MPC conversion protocol is verified below.

Proof of Theorem 1

Proof. Our proof follows the simulation-based proof [25]. Denote $O_0, O_1 \leftarrow \mathcal{F}_{C2M}(\text{Enc}(\text{Encode}(x))$ as the outputs of the functionality. Let $\texttt{view}_b, \texttt{out}_b$ be the view and output of the party P_b in the execution of Algorithm 1. We construct simulators $\mathcal{S}_0$ and $\mathcal{S}_1$ such that:

$$\{\mathcal{S}_0(\text{Key}, O_0), O_0, O_1\} \equiv^c \{\texttt{view}_0, \texttt{out}_0, \texttt{out}_1\}$$
$$\{\mathcal{S}_1(\text{Enc}(\text{Encode}(x), O_1), O_0, O_1\} \equiv^c \{\texttt{view}_1, \texttt{out}_0, \texttt{out}_1\}$$

where $\equiv^c$ denotes computationally indistinguishable.

The view of P_0 in the execution of the real protocol includes: 1) the secret key, 2) the masked CKKS ciphertext, and 3) its view in the Field-to-Ring sub-protocol [47]. $\mathcal{S}_0$ simulates these messages by:

❶ S_0 inputs with random generated key and O_0 where O_0 is the decoded result of a uniform plaintext polynomial. Then, S_0 encodes O_0 to the uniform plaintext polynomial and then encrypts it with Key to get a CKKS ciphertext which is indistinguishable from the 2nd message in $\texttt{view}_0$.

❷ Calling the underlying simulator from [47] .

For the output part, recall that the functionality $\mathcal{F}_{C2M}$ outputs O_0, O_1, an additive share of a uniformly masked CKKS decoded plaintext. Thus, O_0, O_1 are both decoded results of the uniformly distributed polynomials over $\mathbb{A}_{N,2^l}$. These are exactly how Algorithm 1 computes $\texttt{output}_0 = \text{Decode}([\![\texttt{tmp}]\!]_0^{2^l}), \texttt{output}_1 = \text{Decode}([\![\texttt{tmp}]\!]_1^{2^l})$. Therefore, by the semantic security of CKKS and the uniformity of the additive share construction, the joint distribution of $\{\mathcal{S}_0(\text{Key}, O_0), O_0, O_1\}$ is indistinguishable from $\{\texttt{view}_0, \texttt{output}_0, \texttt{output}_1\}$.

Algorithm 3: Secure LayerNorm, $\Pi_{\text{LayerNorm}}$

Input: P_0 & P_1 hold $[\![X]\!]$ where $X \in \mathbb{Z}^{L\times D}$. P_1 holds weights $\gamma, \beta \in \mathbb{Z}^D$.
Output: P_0 & P_1 get $[\![\text{LayerNorm}(X)]\!]$

P_0 and P_1 invoke $[\![X_{sum}]\!] = \text{sum}_j([\![X]\!]_{i,j})$.
P_0 and P_1 invoke $[\![\mu]\!] = \text{ewmul}_{\text{cp}}([\![X_{sum}]\!], \frac{1}{D})$.
P_0 and P_1 invoke $[\![X_\mu]\!] = \text{sadd}_{\text{cc}}([\![X]\!], -[\![\mu]\!])$.
P_0 and P_1 invoke $[\![X_\mu^2]\!] = \text{ewmul}_{\text{cc}}([\![X_\mu]\!], [\![X_\mu]\!])$.
P_0 and P_1 invoke $[\![X_{tmp}]\!] = \text{sum}_j([\![X_\mu^2]\!]_{i,j})$.
P_0 and P_1 invoke $[\![\sigma^2]\!] = \text{ewmul}_{\text{cp}}([\![X_{tmp}]\!], \frac{1}{D})$.
P_0 and P_1 invoke $[\![\frac{1}{\sigma}]\!] = \text{rsqrt}([\![\sigma^2]\!])$.
P_0 and P_1 invoke $[\![\frac{X_\mu}{\sigma}]\!] = \text{smul}_{\text{cc}}([\![X_\mu]\!], [\![\frac{1}{\sigma}]\!])$.
P_0 and P_1 invoke $[\![\frac{X_\mu}{\sigma} \cdot \gamma]\!] = \text{ewmul}_{\text{cp}}([\![\frac{X_\mu}{\sigma}]\!], \gamma)$.
P_0 and P_1 invoke $[\![\text{LayerNorm}(X)]\!] = \text{ewadd}_{\text{cp}}([\![\frac{X_\mu}{\sigma} \cdot \gamma]\!], \beta)$.

The view of P_1 in the execution of the real protocol includes 1) Its input $\text{Enc}(\text{Encode}(x))$, 2) $\hat{r}$, a randomly sampled polynomial, 3) its view in the Field-to-Ring sub-protocol. $\mathcal{S}_1$ simulates these messages by:

❶ S_1 inputs with a ciphertext $\text{Enc}(\text{Encode}(x))$ and O_1. The ciphertext $\text{Enc}(\text{Encode}(x))$ is computationally indistinguishable from the 1st message in $\texttt{view}_1$.

❷ Sampling polynomial from $\mathbb{A}_{N,q}$ uniformly at random. The distribution of this message is identical to the 2nd message in $\texttt{view}_1$.

❸ Calling the underlying simulator from [47].

For the output part, O_0, O_1 are both decoding results of uniform random polynomials, which is exactly how Algorithm 1 computes $\texttt{output}_0 = \text{Decode}([\![\texttt{tmp}]\!]_0^{2^l}), \texttt{output}_1 = \text{Decode}([\![\texttt{tmp}]\!]_1^{2^l})$. Thus, the joint distribution of $\{\mathcal{S}_1(\text{Enc}(\text{Encode}(x), O_1), O_0, O_1\}$ is indistinguishable from $\{\texttt{view}_1, \texttt{output}_0, \texttt{output}_1\}$. □

B Protocols for Nonlinear Layers

The nonlinear layers in Transformer models mainly include LayerNorm, Softmax, and GeLU [10, 34]. For LayerNorm and Softmax, we leverage Bumblebee's protocols [41], while for GeLU, we employ BOLT's protocol [45].

LayerNorm. Given an input matrix $X \in \mathbb{R}^{L\times D}$, where L is the sequence length and D is the hidden dimension, the LayerNorm operation is defined as: $\text{LayerNorm}(X)_{i,j} = \gamma_j \cdot \frac{X_{i,j}-\mu_i}{\sigma_i} + \beta_j$, where $\mu_i = \frac{1}{D}\sum_{j=1}^D X_{i,j}$ is the mean and $\sigma_i = \sqrt{\frac{1}{D}\sum_{j=1}^D (X_{i,j}-\mu_i)^2}$ is the variance, γ_j and β_j are learnable parameters.

GeLU. Given an input element $x \in \mathbb{R}$, prior-art works approximate GeLU function using piecewise functions. We adopt the approximation proposed in BOLT [45]:

Algorithm 4: Secure GeLU, Π_{GeLU}

Input: P_0 & P_1 hold $[\![x]\!]$.
Output: P_0 & P_1 get $[\![\text{GeLU}(x)]\!]$

P_0 and P_1 invoke $[\![x^2]\!] = \text{ewmul}_{\text{cc}}([\![x]\!], [\![x]\!])$.
P_0 and P_1 invoke $[\![x^3]\!] = \text{ewmul}_{\text{cc}}([\![x^2]\!], [\![x]\!])$ and $[\![x^4]\!] = \text{ewmul}_{\text{cc}}([\![x^2]\!], [\![x^2]\!])$.
P_0 and P_1 evaluate $[\![ax^4]\!], [\![bx^3]\!], [\![cx^2]\!], [\![(0.5+d)x]\!]$ and $[\![(0.5-d)x]\!]$ through ewmul_{cp}.
P_0 and P_1 evaluate two polynomials $[\![F_0(x)]\!] = [\![ax^4 - bx^3 + cx^2 + (0.5-d)x + e]\!]$ and $[\![F_1(x)]\!] = [\![ax^4 + bx^3 + cx^2 + (0.5+d)x + e]\!]$ through ewadd_{cc} and ewadd_{cp}.
P_0 and P_1 invoke $[\![b_0]\!]^B = \text{cmp}([\![x]\!], -2.7), [\![b_1]\!]^B = \text{cmp}([\![x]\!], 0)$ and $[\![b_2]\!]^B = \text{cmp}(2.7, [\![x]\!])$.
$\forall b \in \{0,1\}, P_b$ locally compute $[\![z_0]\!]^B = [\![b_0]\!]^B \oplus [\![b_1]\!]^B, [\![z_1]\!]^B = [\![b_1]\!]^B \oplus [\![b_2]\!]^B \oplus b$ and $[\![z_2]\!]^B = [\![b_2]\!]^B$. Note $z_0 = 1\{-2.7 < x \le 0\}$, $z_1 = 1\{0 < x \le 2.7\}$ and $z_2 = 1\{2.7 < x\}$
P_0 and P_1 invoke $[\![\text{GeLU}(x)]\!] = \text{mux}([\![F_0(x)]\!], [\![z_0]\!]^B) + \text{mux}([\![F_1(x)]\!], [\![z_1]\!]^B) + \text{mux}(x, [\![z_2]\!]^B)$.

Algorithm 5: Secure Softmax, Π_{Softmax}

Input: P_0 & P_1 hold $[\![X]\!]$ where $X \in \mathbb{Z}^{L\times L}$.
Output: P_0 & P_1 get $[\![\text{Softmax}(X)]\!]$

P_0 and P_1 evaluate $[\![\bar{X}]\!]$ where $\bar{X}_i = \max_j([\![x]\!]_{i,j})$ through L times cmp.
P_0 and P_1 invoke $[\![X_{tmp1}]\!] = \text{sadd}_{\text{cc}}([\![x]\!], -[\![\bar{X}]\!])$.
P_0 and P_1 invoke $[\![\frac{X_{tmp1}}{2^6}]\!] = \text{ewmul}_{\text{cp}}([\![X_{tmp1}]\!], \frac{1}{2^6})$.
P_0 and P_1 invoke $[\![\exp(X_{tmp1})]\!] = [\![(1 + \frac{X_{tmp1}}{2^6})^{2^6}]\!]$ through one ewadd_{cp} and six times ewmul_{cc}.
P_0 and P_1 invoke $[\![b]\!]^B = \text{cmp}(T_{\exp}, [\![x]\!])$.
P_0 and P_1 invoke $[\![X_{\exp}]\!] = \text{mux}([\![\exp(X_{tmp1})]\!], [\![b]\!]^B)$.
P_0 and P_1 invoke $[\![X_{tmp2}]\!] = \text{sum}_j([\![X_{\exp}]\!]_{i,j})$.
P_0 and P_1 invoke $[\![\frac{1}{X_{tmp2}}]\!] = \text{rec}([\![\frac{1}{X_{tmp2}}]\!])$.
P_0 and P_1 invoke $[\![\text{Softmax}(X)]\!] = \text{smul}_{\text{cc}}([\![X_{\exp}]\!], [\![\frac{1}{X_{tmp2}}]\!])$.

$$\text{ApproxGELU}(x) = \begin{cases} x & \text{if } x > 2.7, \\ a|x|^4 + b|x|^3 + c|x|^2 + d|x| + e + 0.5x & \text{if } |x| \le 2.7, \\ 0 & \text{if } x < -2.7. \end{cases}$$

The concrete parameters can be found in BOLT [45].

Softmax. Given an input matrix $X \in \mathbb{R}^{L\times L}$, the Softmax function is defined as: $\text{Softmax}(X)_{i,j} = \frac{e^{X_{i,j}-\bar{X}_i}}{\sum_{j\in[L]} e^{X_{i,j}-\bar{X}_i}}, i \in [L], j \in [L]$, where $\bar{X}_i = \max_{j\in[L]} X_{i,j}$ is the maximum value in the i-th row. Since the input of the exponentiation is negative, prior-art protocol computes the exponentiation using the Taylor series with a simple clipping [41]:

$$\text{negExp}(x) = \begin{cases} 0, & x < T_{\exp} \\ \left(1 + \frac{x}{2^t}\right)^{2^t}, & x \in [T_{\exp}, 0] \end{cases} \quad (3)$$

We set $l = 6$ and $T_{\exp} = -13$ to ensure accuracy.

Next, we present the operator-wise decomposition of these nonlinear layers. The LayerNorm decomposition is shown in Algorithm 3, GeLU in Algorithm 4, and Softmax in Algorithm 5. All operators used in these algorithms are defined in Table 2. Additionally, we transform sadd_{cp} and smul_{cp} to ewadd_{cp} and ewmul_{cp} respectively, as plaintext can be repacked freely without incurring additional costs. Each operator in these algorithms requires a private protocol for evaluation, and all intermediate results are in secret-shared form.

C Detailed Protocols

C.1 BSGS Optimization

BSGS algorithm reduces the number of rotations during the multiply-accumulate process and has been widely adopted [21, 32, 45, 46]. For the computation $\sum_{i=0}^{R} A \times \text{Rot}^i(B)$, instead of rotating each input individually, the BSGS algorithm divides the rotations into two steps: baby-step and giant-step, which can be formulated as: $\sum_{j=0}^{G-1} \text{Rot}^{j\cdot B}\left(\sum_{i=0}^{B-1} \text{Rot}^{-j\cdot B}(A) \times \text{Rot}^i(B)\right)$, where G and B represent the number of giant and baby steps, respectively, with $G \times B = R$. The total number of rotations is reduced from R to $B + G - 2 \approx 2\sqrt{R}$. It can be extended to compute $\sum_{i=0}^{R} A_i \times \text{Rot}^i(B)$ when A_i is a plaintext.

The BSGS optimization can be applied to step 1 and step 2 of our $\text{matmul}_{\text{cc}}$ protocol to reduce rotations. In Step 1, which involves plaintext multiplication and accumulation, BSGS can be directly applied. Step 2, involving ciphertext rotation and multiplication, does not immediately benefit from BSGS since there is no accumulation to reduce rotations during the giant step. However, we observe that the giant-step rotations can be fused with step 3. As shown in Figure 14, we postpone the giant-step rotations until step 3, where we complete the partial sum accumulation by adding one additional mask multiplication and rotation to each ciphertext generated by the giant step. This integration allows us to maintain a multiplication depth of 4 while reducing the number of rotations from $B \times G$ to $B + 2G$. Although an additional G is introduced, the complexity remains reduced from R to $2\sqrt{2R}$.

C.2 Diagonal Ciphertext-Plaintext MatMul

When using BLB's ct-ct MatMul protocol to compute $\text{Att}_h = \text{Softmax}(\cdot) \times V_h$, the output Att_h is in diagonal packing. However, the current ct-pt MatMul protocol in BOLT only supports inputs with spatial-first packing and cannot directly handle $\text{Concat}(\text{Att}_h)W_O$. Moreover, the final result must be in spatial-first packing to ease subsequent computations. Thus, we develop a new ct-pt MatMul protocol that supports diagonal-packed inputs and outputs in spatial-first packing.

For clarity, consider the example $E = C \times W$, where $C \in \mathbb{R}^{L\times D}$ is a ciphertext and $W \in \mathbb{R}^{D\times D}$ is a plaintext. Figure 15 illustrates this with $L = 4$ and $D = 2$. Notably, C is produced by a $\text{matmul}_{\text{cc}}$ in a diagonal packing. If zero padding is used in $\text{matmul}_{\text{cc}}$ to ensure the input and output dimensions are the same, the dimension of W must be adjusted accordingly. However, we find that if zero padding is exactly doubled, we can add adjacent ciphertexts of C to obtain the output in a dense diagonal packing, as shown in Figure 15 (b). Therefore, zero padding at most doubles the dimension D. Figure 15 (c) illustrates our optimized protocol. This protocol is based on the observation that the i-th column of E can be represented as a combination of diagonals in C, multiplied by the elements of the i-th column in W. With C in diagonal packing, we adjust the weight matrix W to a duplicated spatial-first packing. We then rotate C and multiply it with W to produce the output E in spatial-first packing. This step can also utilize BSGS to reduce the number of rotations. Consequently, our modified $\text{matmul}_{\text{cp}}$ protocol maintains the same computational complexity as BOLT. Additionally, the Concat is inherently handled in the ciphertext domain, as all ciphertexts are tightly packed. Therefore, no additional protocol is needed.

C.3 Ring and Field Conversion

We now describe the protocols for conversions between ring and field, as proposed in BOLT [45] and SiRNN [47].For field-to-ring conversion, we employ the signed extension protocol $\Pi_{\text{Ext}}^{2^{l_1},2^{l_2}}$ in [47], which extends shares from $\mathbb{Z}2^{l_1}$ to $\mathbb{Z}2^{l_2}$ ($l_1 < l_2$). Following BOLT [45], we adapt it to the field-to-ring setting as $\Pi_{\text{Ext}}^{q,2^{\lceil \log_2 q \rceil}}$. Then, we perform a local modulus reduction to $\mathbb{Z}_{2^l}$. We can avoid local modulus reduction errors by setting a proper l such that the output range is within $\mathbb{Z}_{2^l}$.

For ring-to-field conversion, we use Π_{Ext} to extend the shares $[\![m]\!]$ to a larger ring, e.g., from $\mathbb{Z}_{2^l}$ to $\mathbb{Z}_{2^{l+40}}$. In this case, the probability that $[\![m]\!]_0 + [\![m]\!]_1 > 2^{l+40}$ is greater than $1 - 2^{-40}$. This enables us to locally change the modulus to q with an error rate below $2^{-40} \approx 10^{-12}$:

$$[\![m]\!]_0^q = [\![m]\!]_0^{2^{l+40}} \bmod q, \quad [\![m]\!]_1^q = [\![m]\!]_1^{2^{l+40}} - 2^{l+40} \bmod q$$

$$\begin{aligned}[\![m]\!]_0^q + [\![m]\!]_1^q &= [\![m]\!]_0^{2^{l+40}} + [\![m]\!]_1^{2^{l+40}} - 2^{l+40} \bmod q \\ &= m + 2^{l+40} - 2^{l+40} \bmod q = m \bmod q\end{aligned}$$

C.4 Probabilistic Local Truncation Method

We leverage the (non-interactive) probabilistic truncation protocol introduced in SecureML [43]. The protocol takes as input a secret-shared value $[\![m]\!]^{2^l}$ and outputs $[\![m']\!]^{2^l}$, where $m' = \lfloor m/2^s \rfloor + u$, with $\Pr(u = 0) = 1 - \frac{m \bmod 2^s}{2^s}$, $\Pr(u = 1) = \frac{m \bmod 2^s}{2^s}$. Li et al. [40] point out that some probabilistic truncation protocols are not secure in the simulation-based sense. To prove the security of our truncation protocol in the simulation-based paradigm, we change our truncation functionality in Figure 16, which gives the cutoff point k to the adversary. Note that this extra element reveals only as much information

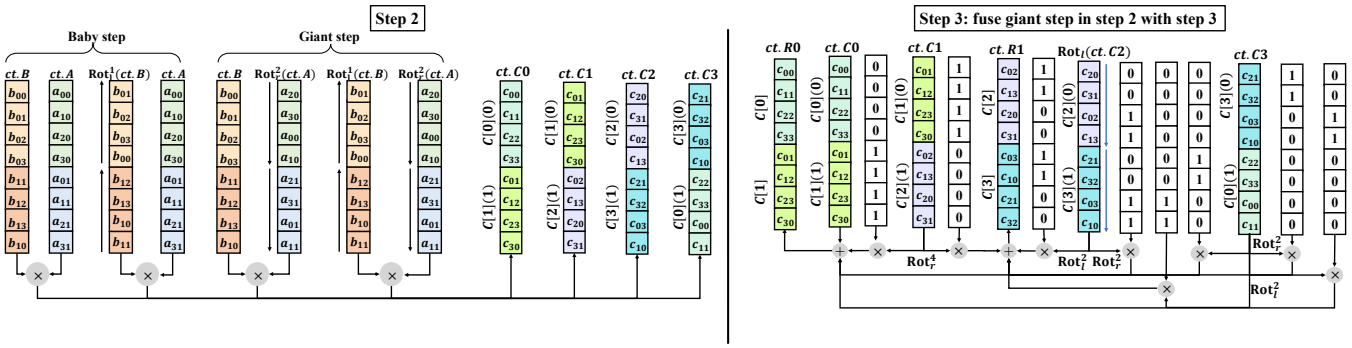

Figure 14: Applying BSGS optimization to the step 2 of our matmul$_{cc}$ protocol.

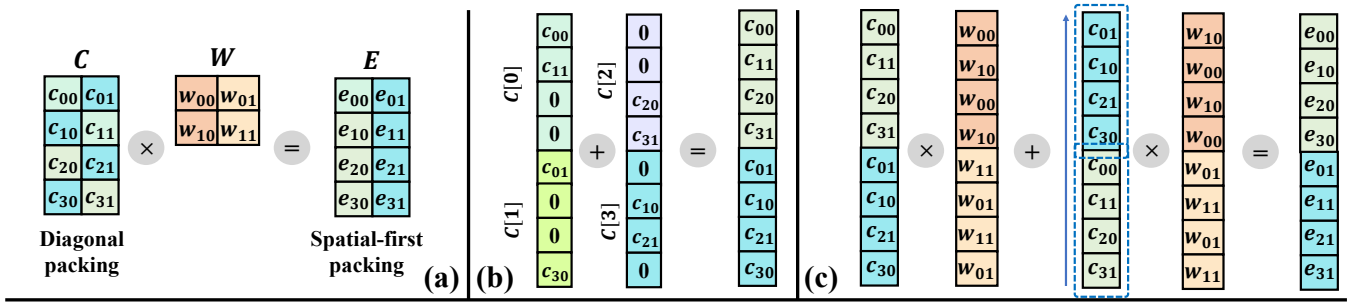

Figure 15: An example for BLB's modified ct-pt MatMul protocol for Att $\times W_O$: (a) Multiplying matrices C and W in plaintext. (b) If the output dimension of V_h in the previous step Softmax$(\cdot) \times V_h$ is half the input dimension, we pad V_h with zeros and perform the matmul$_{cc}$. Afterward, we can add adjacent ciphertext results to obtain a dense diagonal packing of the output. (c) With C in diagonal packing, we adjust the weight matrix W to a duplicated spatial-first packing. We then rotate C and multiply it with W to produce the output E in spatial-first packing.

as the definition of probabilistic truncation itself, where a rounding error $u = 1$ is introduced with probability $\frac{m \mod 2^s}{2^s}$. The detailed proof under the simulation-based paradigm can be found in Section 3 in [17].

However, the protocol [43] may fail, potentially causing a max significant bit (MSB) error, with a probability of $|m|/2^{l-1}$. To overcome this, we take an extend-then-truncate approach. We first use the [47] protocol to extend the modulus size: $[\![m]\!]^{2^{l+40}} \leftarrow \texttt{Extend}([\![m]\!]^{2^l})$. Then we perform SecureML's truncation over the larger ring using arithmetic right-shift ($\gg_a$): $[\![m']\!]_b^{2^{l+40}} = [\![m]\!]_b^{2^{l+40}} \gg_a s$. The failure probability is now smaller than 2^{-40}. With one more modulo operation, i.e., $[\![m']\!]_b^{2^l} = [\![m']\!]_b^{2^{l+40}} \mod 2^l$ for $b = 0, 1$, we obtain the truncated result over the ring $\mathbb{Z}_{2^l}$.

$$\mathcal{F}_{l,s}^{\text{TruncPr}}([\![m]\!])\,(s \le 40)$$

1. Reconstruct m from sharing $[\![m]\!]$.
2. Set $m^{(s)} = \lfloor m/2^s \rfloor$ and $m_{(s)} = (m \mod 2^s)$.
3. Pick cutoff point $s_{(s)}$ at random in $\mathbb{Z}_{2^s}$.
4. Set $m' = m^{(s)} + u$ where $u = \mathbf{1}\{m_{(s)} > s_{(s)}\}$
5. Generate a random sharing $[\![m']\!]$ of m'.
6. Output $[\![m']\!]$ and leak $s_{(s)}$ to the adversary.

Figure 16: The modified probabilistic truncation functionality

Table 12: Amortized latency comparison on BERT-base.

Framework	Amortized Latency (min)		
	LAN	WAN$_2$	WAN$_3$
BLB	2.5	6.6	13.2
BLB +batch size 32	0.79	2.1	4.1
NEXUS+batch size 32	0.63	0.64	1.0

D Additional Experiments

Comparison with NEXUS under Batched Inputs. Although BLB defaults to a batch size of 1, it also supports batched inputs. Batching enables BLB to pack more data along the spatial dimension, reduce rotations, improve GPU utilization, and amortize communication rounds, resulting in lower amortized latency. In Table 12, BLB shows a 3× speedup with batch. Compared to NEXUS, BLB performs similarly under LAN but is slower under WAN due to higher communication costs.