BOOTSTRAP LESS, INFER FASTER: REDESIGNING TRANSFORMERS FOR FULLY HOMOMORPHIC INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Fully homomorphic encryption (FHE) enables non-interactive inference on encrypted inputs, but evaluating deep Transformers with CKKS remains costly due to repeated and expensive bootstrapping. Beyond optimizing packing and bootstrap placement within an existing computation, BLADE introduces three new, complementary FHE-specific techniques for attention, FFN, and LayerNorm that jointly reshape Transformer architecture and encrypted dataflow to reduce bootstrap inputs. First, ciphertext-aligned cyclic sparse attention maps each selected relative offset to one complete cyclic score diagonal, so unselected score ciphertexts are never computed or bootstrapped. Second, low-rank feed-forward networks bootstrap compact intermediates before expansion. Third, LayerNorm bootstraps only the packed variance ciphertext while retaining the centered values for subsequent normalization. The three techniques share an execution schedule that preserves sufficient multiplicative depth for retained and refreshed branches. We further develop a numerically robust, mean-centered homomorphic Softmax with normalization before and after squaring. On a complete 12-layer BERT-Base encoder, BLADE reduces the number of ciphertexts submitted to bootstrapping from 37,008 in MOAI to 17,448, a 2.12× reduction, and achieves 2.16× CPU and 2.00× GPU encoder speedups over MOAI. The adapted models score 1.46 metric points below BERT-Base on average across five GLUE tasks with exact nonlinearities. For the SST-2 checkpoint used in the runtime evaluation, CKKS execution matches the 90.02% accuracy of the same adapted checkpoint evaluated in plaintext with exact nonlinearities.

1 INTRODUCTION

Transformer models are widely used for language understanding, generation, and document processing (Vaswani et al., 2017; Devlin et al., 2019). Cloud deployment raises privacy concerns when users submit confidential data. Fully homomorphic encryption (FHE) enables model evaluation directly on encrypted inputs, supporting non-interactive inference: the client sends an encrypted input and receives an encrypted prediction without intermediate interaction or revealing either value to the server. However, FHE's high computational overhead limits practical Transformer inference.

Among FHE schemes, CKKS supports approximate arithmetic and single-instruction multiple-data (SIMD) computation over packed values, making it well suited to encrypted Transformer inference (Cheon et al., 2017). CKKS supports only a limited multiplicative depth before bootstrapping is needed to continue the computation. Deep Transformer evaluation therefore requires repeated bootstrapping, and each bootstrap is substantially more expensive than basic homomorphic operations (Cheon et al., 2018). An intermediate tensor may span multiple ciphertexts. The total bootstrapping cost therefore depends on both the number of bootstrapping locations in the computation and the number of ciphertexts refreshed at each location. Consequently, the representation entering each bootstrapping is an important determinant of execution cost.

Recent work has improved the efficiency of non-interactive Transformer inference. NEXUS uses ciphertext compression and slot folding (Zhang et al., 2025a), THOR improves matrix multiplication and nonlinear evaluation for end-to-end BERT-Base inference (Moon et al., 2025), and Encrypt-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

edLLM evaluates GPT-2 using GPU FHE (De Castro et al., 2025). MOAI avoids format conversions with consistent column and diagonal packing and uses interleaved batching and rotation-free Softmax and LayerNorm (Zhang et al., 2026). Euston uses singular value decomposition (SVD) in homomorphic matrix multiplication to reduce ciphertext size and rotations (Gao et al., 2026). Other work adapts the Transformer architecture to simplify encrypted inference (Chen et al., 2022; Zhang et al., 2025b). Polynomial Transformer replaces non-polynomial operators with polynomials (Zimerman et al., 2024). Encryption-Friendly LLM Architecture combines LoRA fine-tuning with Gaussian-kernel attention (Rho et al., 2025), while Powerformer replaces Softmax and LayerNorm with power and linear functions, respectively (Park et al., 2025). At the compilation level, prior work optimizes SIMD vectorization (Dathathri et al., 2019; Cowan et al., 2021; Viand et al., 2023) and CKKS scale management (Dathathri et al., 2020; Lee et al., 2022; 2023).

Despite these advances, bootstrapping remains a major cost in deep FHE evaluation. Fhelipe automates bootstrapping placement alongside packing (Krastev et al., 2024); DaCapo uses estimated execution time (Cheon et al., 2024), while Orion combines placement, packing, and scale management (Ebel et al., 2025). NEXUS and THOR place bootstrapping at narrower intermediates already present in the computation (Zhang et al., 2025a; Moon et al., 2025). The savings from bootstrapping placement depend on the intermediate representations already available. Factorizing an FFN projection, for example, creates a compact intermediate that can be bootstrapped before expansion. This motivates a complementary question: *can the Transformer computation be redesigned so that fewer ciphertexts need to enter bootstrapping in the first place?*

Realizing this opportunity requires coordinating model structure, ciphertext packing, and the execution schedule. Sparsity must eliminate whole ciphertexts under the chosen packing, while compact intermediates and retained branches need sufficient levels for subsequent operations. Together, these choices must reduce the total bootstrapping workload without unacceptable loss of model quality.

Contributions. We address this question with **BLADE** (**B**ootstrapping **L**ess through **A**rchitecture and **D**ataflow redesign for **E**ncrypted inference). BLADE introduces new FHE-specific attention, FFN, and LayerNorm designs that jointly restructure Transformer architecture and encrypted dataflow to create compact bootstrapping inputs and selectively refresh intermediate values.

- **Reducing bootstrapping inputs through joint architecture and dataflow redesign.** For self-attention, we introduce ciphertext-aligned cyclic sparse attention, exploiting the correspondence between fixed relative offsets and cyclic score diagonals. Each selected offset maps to one complete diagonal in one ciphertext; unselected score ciphertexts are never computed or bootstrapped. For the FFN, low-rank factorization creates a rank-r_l intermediate absent from the original dense projection; bootstrapping it before expansion reduces the number of ciphertexts refreshed. For LayerNorm, we separate retained centered-value ciphertexts from the variance-dependent nonlinear branch and bootstrap only the single packed variance ciphertext. The three designs share a bootstrapping schedule satisfying the depth requirements of retained ciphertexts and subsequent components.
- **Encrypted encoder evaluation and task utility.** For numerically robust encrypted attention, our homomorphic Softmax subtracts each encrypted row's mean, removing row-dependent score shifts that MOAI's layer-specific public offset does not track. It then normalizes before and after squaring, adapting normalize-and-square (Cho et al., 2024) without computing an encrypted maximum. We evaluate BLADE on a complete 12-layer BERT-Base encoder. BLADE reduces ciphertext inputs to bootstrapping from 37,008 to 17,448, a $2.12\times$ reduction over MOAI, and achieves $2.16\times$ CPU and $2.00\times$ GPU encoder speedups. With exact nonlinearities, the adapted models score 1.46 metric points below BERT-Base on average across five GLUE tasks. For the SST-2 checkpoint used in runtime measurements, CKKS execution matches the 90.02% accuracy of the same adapted checkpoint evaluated in plaintext with exact nonlinearities.

2 PRELIMINARIES

We introduce notation and background for encrypted Transformer inference. For a positive integer n, let $[n] = \{0, \ldots, n-1\}$. We use uppercase letters for matrices and bold lowercase letters for vectors. For $X \in \mathbb{R}^{n\times d}$, $X[i,j]$ denotes its (i,j)-th entry, $\boldsymbol{x}_j \in \mathbb{R}^n$ denotes its j-th column, and $X^\intercal$ denotes its transpose.

For $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^n$, let $\boldsymbol{u} \odot \boldsymbol{v} = [u_i v_i]_{i\in[n]}$ and $\boldsymbol{u} + \boldsymbol{v} = [u_i + v_i]_{i\in[n]}$ denote element-wise multiplication and addition. We use $\mathrm{Rot}_\rho(\boldsymbol{u})$ to denote a cyclic left rotation by ρ positions, with indices taken modulo n, and negative offsets correspond to right rotations. For $X \in \mathbb{R}^{n\times n}$, its cyclic diagonal at offset δ is denoted as $\mathrm{Diag}_\delta(X) = [X[i, (i+\delta) \bmod n]]_{i\in[n]} \in \mathbb{R}^n$.

2.1 TRANSFORMER

We focus on Transformer encoders (Vaswani et al., 2017), which consist of L layers, each containing multi-head self-attention and a fully connected feed-forward network (FFN). In the BERT architecture considered here, each sublayer has a residual connection followed by LayerNorm (Ba et al., 2016). For layer input $X \in \mathbb{R}^{n\times d}$, n is the sequence length and d the hidden dimension. The encoder uses h heads of dimension d_h and an FFN of hidden dimension d_f. BERT-Base uses $L = 12$, $d = 768$, $h = 12$, $d_h = 64$, and $d_f = 3{,}072$, with GELU (Hendrycks & Gimpel, 2016) as the FFN activation (Devlin et al., 2019). Appendix A gives the standard attention, FFN, and LayerNorm definitions.

2.2 FULLY HOMOMORPHIC ENCRYPTION

Fully homomorphic encryption (FHE) enables computation on encrypted data. We use CKKS (Cheon et al., 2017), which supports approximate arithmetic in up to $N/2$ complex SIMD slots for polynomial ring degree N. We pack real-valued Transformer data in these slots, writing $[\![\boldsymbol{a}]\!]$ for a ciphertext encrypting vector $\boldsymbol{a}$ and $\boldsymbol{p}$ for a public encoded plaintext.

The additions $[\![\boldsymbol{a}]\!] + [\![\boldsymbol{b}]\!]$ and $[\![\boldsymbol{a}]\!] + \boldsymbol{p}$ are evaluated slot-wise, as are the products $[\![\boldsymbol{a}]\!] \odot [\![\boldsymbol{b}]\!]$ and $[\![\boldsymbol{a}]\!] \odot \boldsymbol{p}$. $\mathrm{Rot}_\rho([\![\boldsymbol{a}]\!])$ cyclically shifts the slots by the public offset ρ. Nonlinear computations use polynomial approximations (Zhang et al., 2025a; Moon et al., 2025). For a public polynomial P, $\mathsf{PolyEval}(P, [\![\boldsymbol{a}]\!])$ denotes its homomorphic evaluation.

These computations are constrained by CKKS levels, which track the remaining rescaling budget and thus limit multiplicative depth. Rescaling after multiplication, when needed, reduces the scale and consumes one level, whereas rotations consume no levels. When the remaining levels are insufficient, bootstrapping, written $[\![\boldsymbol{a}]\!]' \leftarrow \mathsf{Boot}([\![\boldsymbol{a}]\!])$, restores usable levels for further encrypted computation while preserving packed values up to approximation error. For a fixed configuration with maximum level $L_{\max}$, the bootstrap circuit consumes L_{boot} levels, leaving $L_{\max} - L_{\mathrm{boot}}$ usable levels. Bootstrapping is substantially more expensive than additions, multiplications, and rotations. In our implementation, each submitted ciphertext requires a separate bootstrapping, so we measure bootstrapping workload by the total number of ciphertexts submitted.

We use column packing for feature matrices and diagonal packing for attention-score matrices. Under column packing, $\mathrm{Enc}_{\mathrm{col}}(X)$ represents $X \in \mathbb{R}^{n\times d}$ using d ciphertexts, one per column. Under diagonal packing, $\mathrm{Enc}_{\mathrm{diag}}(S)$ represents $S \in \mathbb{R}^{n\times n}$ using n ciphertexts, one per cyclic diagonal, containing entries $S[t, (t+\delta) \bmod n]$ for a fixed offset δ.

Each ciphertext interleaves $B = N/(2n)$ inputs, with the row-t entry of input b stored at slot $Bt + b$. Thus, $\mathrm{Rot}_{B\delta}$ shifts token positions cyclically by δ within each input without mixing inputs. Column-to-diagonal multiplication directly generates each score diagonal from column-packed query and key matrices Q and K (Pang et al., 2024; Zhang et al., 2025a; 2026). Computing only m selected diagonals therefore produces m score ciphertexts. The formal definitions of packing are given in Appendix B.1.

2.3 THREAT MODEL

In our non-interactive setting, a client encrypts its private input and sends the ciphertexts and evaluation keys to a server. The server evaluates a public Transformer encoder entirely under FHE, including nonlinear functions and bootstrapping, without further client interaction. The model parameters, packing, and execution schedule are public and fixed before inference; the encrypted operation sequence does not depend on the private input. We assume a semi-honest server that follows this computation but may try to learn the client's input from the ciphertexts. Input confidentiality relies on the security of CKKS under its underlying RLWE assumption.

3 BLADE DESIGN

An encrypted Transformer representation may span multiple ciphertexts, each requiring a separate bootstrapping when refreshed. BLADE jointly redesigns Transformer architecture and encrypted dataflow to reduce the total number of bootstrapping ciphertexts during a complete encoder evaluation.

Let $\mathcal{B}$ denote the set of bootstrapping points used in a given encrypted encoder evaluation, and let $\mathcal{T}_i$ denote the set of ciphertexts submitted to bootstrapping at point i. We define the *bootstrapping workload* C_{boot} as the total number of ciphertexts submitted to bootstrapping over that evaluation:

$$C_{\mathrm{boot}} = \sum_{i \in \mathcal{B}} |\mathcal{T}_i| \,. \tag{1}$$

The attention, FFN, and LayerNorm designs jointly target this bootstrapping workload. Attention eliminates unselected score ciphertexts through sparsity aligned with diagonal packing; low-rank FFNs create compact intermediates for bootstrapping before expansion; and LayerNorm only performs bootstrapping on the packed variance ciphertext while retaining the centered-value ciphertexts for final normalization. A shared execution schedule coordinates these mechanisms to satisfy multiplicative-depth requirements and reduce C_{boot} across the complete encoder.

3.1 ATTENTION: SPARSE REPRESENTATION BEFORE BOOTSTRAPPING

Self-attention is costly under FHE because the dense score matrix contains n^2 query–key interactions. These interactions can be represented by n cyclic score diagonals, each stored in one ciphertext under diagonal packing. To reduce bootstrapping inputs under this packing, sparsity must eliminate entire score ciphertexts. We therefore align the attention pattern with cyclic diagonals, computing and bootstrapping only selected diagonals.

We adopt a fixed-offset cyclic sparse attention pattern. For one attention head, let $\mathcal{D}$ denote the public set of selected relative offsets, with $|\mathcal{D}| = m \ll n$. Each relative offset δ identifies a cyclic diagonal of the score matrix $S = \frac{QK^{\mathsf{T}}}{\sqrt{d_h}} \in \mathbb{R}^{n \times n}$, i.e., the entries $S[t, (t+\delta) \bmod n]$ with $t = 0, \ldots, n-1$. Selecting an offset therefore selects a complete cyclic score diagonal stored in one ciphertext; excluding it eliminates that ciphertext and its bootstrap.

Cyclic indexing matches CKKS slot rotations without boundary-dependent masks; padded positions also participate in attention (Appendix B.2). Appendix C compares the boundary conventions on SST-2.

Consider one attention head with $Q, K, V \in \mathbb{R}^{n \times d_h}$, and let $\boldsymbol{q}_j, \boldsymbol{k}_j, \boldsymbol{v}_j \in \mathbb{R}^n$ denote their column vectors. We write $\mathrm{Enc}_{\mathrm{col}}(Q) = \{[\![\boldsymbol{q}_j]\!]\}_{j=0}^{d_h-1}$ and $\mathrm{Enc}_{\mathrm{col}}(K) = \{[\![\boldsymbol{k}_j]\!]\}_{j=0}^{d_h-1}$. With B inputs interleaved in each ciphertext, column-to-diagonal multiplication directly computes each selected score diagonal $\boldsymbol{s}_\delta \in \mathbb{R}^n$:

$$[\![\boldsymbol{s}_\delta]\!] = \frac{1}{\sqrt{d_h}} \sum_{j=0}^{d_h-1} [\![\boldsymbol{q}_j]\!] \odot \mathrm{Rot}_{B\delta}([\![\boldsymbol{k}_j]\!]) \,. \tag{2}$$

The selected score diagonals are therefore represented as $\mathrm{Enc}_{\mathrm{diag}}(S) = \{[\![\boldsymbol{s}_\delta]\!]\}_{\delta \in \mathcal{D}}$. Subsequently, we only perform bootstrapping on these selected score ciphertexts before Softmax:

$$[\![\boldsymbol{s}_\delta]\!] \leftarrow \mathsf{Boot}([\![\boldsymbol{s}_\delta]\!]) \,, \qquad \delta \in \mathcal{D}. \tag{3}$$

The remaining $n - m$ diagonals of the score matrix S are never computed. Relative to refreshing all dense score diagonals at this point, this reduces the score contribution to C_{boot} from hn to hm per encoder layer across h heads.

The bootstrapped scores enter our mean-centered homomorphic Softmax circuit, which normalizes exponential values before and after squaring for numerical robustness (Appendix B.5). The resulting probabilities weight the value matrix V to produce the attention output. Algorithm 2 in Appendix B.2 gives the complete encrypted dataflow.

3.2 LOW-RANK FFN: COMPRESS BEFORE BOOTSTRAPPING

A standard FFN first projects the hidden representation from dimension d to a larger intermediate dimension d_f, applies a nonlinear activation, and projects it back to d. Under column packing, the input and expanded intermediate occupy d and d_f ciphertexts, respectively. Low-rank factorization creates a compact intermediate absent from the original dense projection, enabling bootstrapping before expansion on fewer ciphertexts than either representation.

For encoder layer l, we parameterize the first FFN weight matrix in the adapted model as $W_1 = A_1 B_1$, where $A_1 \in \mathbb{R}^{d \times r_l}$, $B_1 \in \mathbb{R}^{r_l \times d_f}$, and $r_l < d$ is the layer-specific rank. For an input $X \in \mathbb{R}^{n \times d}$, the first factor produces $Z = XA_1 \in \mathbb{R}^{n \times r_l}$, which occupies only r_l column ciphertexts. We bootstrap these ciphertexts before applying B_1 to expand the representation to dimension d_f:

$$d \xrightarrow{A_1} r_l \xrightarrow{\text{Boot}} r_l \xrightarrow{B_1} d_f. \tag{4}$$

This preserves the FFN's input and output dimensions while bootstrapping the newly created compact intermediate.

Given the column-packed input $\mathrm{Enc}_{\mathrm{col}}(X) = \{[\![\boldsymbol{x}_i]\!]\}_{i=0}^{d-1}$, ciphertext–plaintext multiplication with A_1 (Algorithm 1) produces $\mathrm{Enc}_{\mathrm{col}}(Z) = \{[\![\boldsymbol{z}_j]\!]\}_{j=0}^{r_l-1}$. We then bootstrap each projected ciphertext:

$$[\![\boldsymbol{z}_j]\!] \leftarrow \mathsf{Boot}([\![\boldsymbol{z}_j]\!])\,. \tag{5}$$

This point contributes r_l ciphertext inputs to C_{boot} per layer, rather than d when refreshing the FFN input. The refreshed columns are expanded to $U = ZB_1 + \boldsymbol{b}_1 \in \mathbb{R}^{n \times d_f}$, with public bias $\boldsymbol{b}_1 \in \mathbb{R}^{d_f}$ broadcast across rows. The resulting $\mathrm{Enc}_{\mathrm{col}}(U) = \{[\![\boldsymbol{u}_k]\!]\}_{k=0}^{d_f-1}$ contains d_f ciphertexts, but no additional bootstrapping is performed during expansion. Refreshing the expanded intermediate instead would require d_f ciphertext inputs.

After GELU is evaluated slot-wise on $\mathrm{Enc}_{\mathrm{col}}(U)$, the second public linear map uses $W_2 = A_2 B_2 \in \mathbb{R}^{d_f \times d}$, with $A_2 \in \mathbb{R}^{d_f \times r_l}$, $B_2 \in \mathbb{R}^{r_l \times d}$, and bias $\boldsymbol{b}_2 \in \mathbb{R}^d$. It produces $Y = \mathrm{GELU}(U)A_2B_2 + \boldsymbol{b}_2 \in \mathbb{R}^{n \times d}$ in d column ciphertexts. No bootstrapping is performed between these two factors. Algorithm 3 in Appendix B.3 gives the complete encrypted FFN dataflow.

3.3 LAYERNORM REFORMULATION

LayerNorm provides another opportunity to reduce the number of ciphertexts submitted to bootstrapping. Although its output has d hidden dimensions per token, the inverse-square-root factor depends only on the token's variance and is shared across all d dimensions. We split the dataflow into two branches: one retains centered-value ciphertexts without bootstrapping; the other refreshes the single packed variance ciphertext and evaluates the inverse square root. The branches merge for normalization, requiring sufficient levels on the retained branch for the final operations.

Consider a LayerNorm input $X \in \mathbb{R}^{n \times d}$ under column packing: $\mathrm{Enc}_{\mathrm{col}}(X) = \{[\![\boldsymbol{x}_i]\!]\}_{i=0}^{d-1}$. We first compute the mean ciphertext $[\![\boldsymbol{\mu}]\!] = \frac{1}{d}\sum_{i=0}^{d-1}[\![\boldsymbol{x}_i]\!]$ and center the columns $[\![\boldsymbol{z}_i]\!] = [\![\boldsymbol{x}_i]\!] - [\![\boldsymbol{\mu}]\!]$ for $i = 0, \ldots, d-1$. The resulting d centered-value ciphertexts are retained without bootstrapping. For the second branch, we compute the variance ciphertext $[\![\boldsymbol{v}]\!] = \frac{1}{d}\sum_{i=0}^{d-1}[\![\boldsymbol{z}_i]\!] \odot [\![\boldsymbol{z}_i]\!]$. Each slot holds one token's variance, packing the variances of all interleaved inputs into one ciphertext. We bootstrap this single variance ciphertext:

$$[\![\boldsymbol{v}]\!] \leftarrow \mathsf{Boot}([\![\boldsymbol{v}]\!])\,. \tag{6}$$

We then add the public stabilization constant $\epsilon > 0$ to every slot before evaluating the inverse square root. Let $P_{\mathrm{isr},l,k}$ denote the composite inverse-square-root polynomial defined by the seed and Newton iterations for LayerNorm invocation k ($k \in \{1, 2\}$) in layer l (Appendix B.4). We evaluate it on the bootstrapped variance ciphertext: $[\![\boldsymbol{r}]\!] = \mathsf{PolyEval}(P_{\mathrm{isr},l,k}, [\![\boldsymbol{v}]\!])$, where $\boldsymbol{r} \in \mathbb{R}^n$ contains the approximate inverse standard deviations. This per-token factor is shared across hidden dimensions. With public scale and bias $\boldsymbol{\gamma}, \boldsymbol{\beta} \in \mathbb{R}^d$, we merge the branches to compute each output column:

$$[\![\boldsymbol{y}_i]\!] = ([\![\boldsymbol{z}_i]\!] \odot [\![\boldsymbol{r}]\!]) \odot \gamma_i + \beta_i. \tag{7}$$

Here γ_i and β_i are the i-th entries of $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$, applied to every slot. The output $Y \in \mathbb{R}^{n \times d}$ remains column-packed, with $\mathrm{Enc}_{\mathrm{col}}(Y) = \{[\![\boldsymbol{y}_i]\!]\}_{i=0}^{d-1}$. Algorithm 4 in Appendix B.4 gives the complete encrypted LayerNorm dataflow.

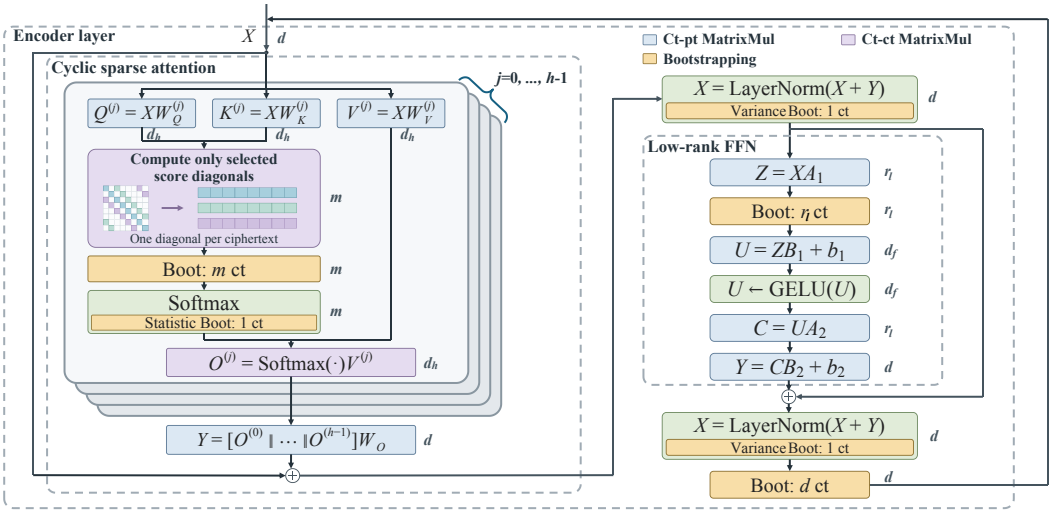


Figure 1: BLADE jointly restructures Transformer architecture and encrypted dataflow to reduce the number of ciphertexts entering bootstrapping. Sparse attention computes only m selected score ciphertexts per head, the low-rank FFN bootstraps the r_l ciphertexts in its compact intermediate before expansion, and LayerNorm bootstraps only one packed variance ciphertext. Yellow blocks mark bootstrapping points, and the numbers beside them denote the corresponding ciphertext counts.

Each LayerNorm invocation contributes one ciphertext input to C_{boot}, compared with d when refreshing a full column-packed representation: 1 rather than 768 for BERT-Base. The two invocations contribute 2 rather than $2d$ inputs per encoder layer; the separate layer-output bootstrap remains part of the overall schedule.

3.4 Bootstrapping Ciphertext Counts

We aggregate ciphertext counts over each system's complete encoder schedule. Figure 1 shows the BLADE dataflow and bootstrapping points. In the l-th encoder layer, BLADE bootstraps hm selected score ciphertexts, h ciphertexts containing Softmax statistics, and r_l ciphertexts in the low-rank FFN. It also bootstraps one variance ciphertext for each of the two LayerNorm invocations and d ciphertexts representing the layer output. Summing over the L encoder layers gives

$$C_{\text{boot}}^{\text{BLADE}} = L(hm + h + 2 + d) + R, \tag{8}$$

where $R = \sum_{l=0}^{L-1} r_l$ is the sum of the FFN ranks.

Under the MOAI schedule, each encoder layer bootstraps one ciphertext per attention head for Softmax statistics and four column-packed representations, each containing d ciphertexts (Zhang et al., 2026). Thus, $C_{\text{boot}}^{\text{MOAI}} = L(h + 4d)$. Both systems bootstrap one Softmax statistic ciphertext per head to provide sufficient levels for subsequent reciprocal computation, although they refresh different intermediate sums. Unlike the local comparisons above, these totals account for the different bootstrap locations, including BLADE's score bootstraps. For these schedules, BLADE submits fewer ciphertexts exactly when $hm + R/L + 2 < 3d$. For the evaluated 12-layer encoder, these expressions give $C_{\text{boot}}^{\text{MOAI}} = 37{,}008$ and $C_{\text{boot}}^{\text{BLADE}} = 17{,}448$, corresponding to a $2.12\times$ reduction.

4 Evaluation

We evaluate BERT-Base and its BLADE adaptations at sequence length 128. We report 12-layer encoder runtime on CPU and GPU compared with MOAI, along with component ablations, five-task utility, and encrypted SST-2 accuracy. Appendix B.5 describes the homomorphic Softmax.

4.1 Experimental Setup

The CPU implementation uses Microsoft SEAL 4.1.2 (SEAL) and Intel HEXL on an Intel Xeon Platinum 8488C with 32 worker threads. The GPU implementation uses PhantomFHE (Yang et al., 2024) on one NVIDIA RTX PRO 6000 Blackwell Max-Q accelerator. Both BLADE and MOAI use CKKS with polynomial ring degree $N = 2^{16}$, 2^{15} complex slots, a 1,743-bit coefficient modulus, and scale 2^{46} for 128-bit security (Zhang et al., 2026).

We use the BERT-Base dimensions in Section 2.1 with sequence length $n = 128$, and interleave $B = 256$ inputs in each ciphertext using the packing in Appendix B.1. For BLADE, each head uses $m = 24$ public cyclic relative offsets, fixed before encrypted inference. Offset identities may vary across tasks, layers, and heads; the fixed count m keeps the number of score ciphertexts per head unchanged. The FFN rank is $r_l = 256$ in layers 0–2 and 9–11, and $r_l = 512$ in layers 3–8. Factor initialization and model training are described in Appendix D. Appendix F details BLADE's per-layer depth, packing, and bootstrapping schedule.

On each platform, BLADE and MOAI use the same BERT-Base encoder dimensions, sequence length, batch size, CKKS parameters, bootstrapping implementation, hardware, and timed interval. MOAI timing retains its released nonlinear approximation routines; BLADE uses layer-specific profiles. For GELU, MOAI uses a degree-24 power polynomial, whereas BLADE uses Chebyshev polynomials of degrees 28–120. Thus, the runtime comparison uses each system's encrypted-execution configuration. We report server-side execution time for all 12 encrypted encoder layers, including the final layer-output bootstrap, amortized over 256 inputs. For BLADE, the corresponding batch wall times are approximately 39.7 hours on CPU and 6.51 hours on GPU. The timed interval excludes client-side cryptographic operations, data transfer, setup, the pooler, and the classifier. The pooler and classifier are evaluated homomorphically for accuracy but excluded from the encoder-only runtime comparison with MOAI.

4.2 Encoder Runtime

Table 1(a)–(b) reports complete-encoder runtime by operation under the setup in Section 4.1. The BLADE measurements use the adapted SST-2 checkpoint evaluated for encrypted accuracy below, while MOAI retains the standard BERT-Base encoder architecture. BLADE achieves $2.16\times$ and $2.00\times$ speedups over MOAI on CPU and GPU, respectively. Although $Q/K/V$ projection dimensions are unchanged, runtime also depends on CKKS levels; modulus switching to the levels in Table 10 contributes to BLADE's lower projection time. For BLADE, the FFN expansion row combines the runtimes of A_1 and B_1, and the FFN output row combines those of A_2 and B_2. Bootstrapping time is reported separately. Table 1(c) lists the ciphertext counts and runtime at each bootstrapping point, including those within Softmax and LayerNorm.

Bootstrapping is the largest source of acceleration, accounting for approximately 68.5% and 61.2% of encoder time saved on CPU and GPU. Its runtime decreases by $2.13\times$ and $2.18\times$, respectively, close to the $2.12\times$ reduction in ciphertexts submitted to bootstrapping. Other operations contribute the remaining 203.38 and 35.62 seconds saved per input on CPU and GPU, respectively. Bootstrapping still accounts for 69.8% and 52.1% of BLADE encoder runtime on CPU and GPU, respectively, with the layer-output bootstrap its largest remaining cost.

4.3 Component Ablations

We evaluate representation and dataflow choices using four GPU ablations on two representative layers (Table 2). Layer 0 with $r_l = 256$ and layer 3 with $r_l = 512$ cover the two FFN ranks used across the encoder. Each variant uses the same packing, CKKS parameters, and layer-specific nonlinear approximations as its BLADE baseline, with a valid CKKS level schedule through the next bootstrapping point. Slowdown is relative to the baseline for both the affected operations and the complete encoder layer.

Attention. D1 computes all 128 score diagonals per head and bootstraps them at the same point as BLADE. Only the baseline's 24 selected diagonals then enter Softmax and value aggregation. D1 therefore isolates the overhead of computing and refreshing unused diagonals, rather than comparing dense and sparse model quality. This overhead produces a $5.41\times$ operation slowdown and $1.60/1.46\times$ complete-layer slowdowns under this schedule.

Table 1: Encoder runtime and bootstrapping workload for BLADE and MOAI across 12 layers. Times are seconds per input, amortized over 256 inputs; ciphertext counts cover one complete batched encoder evaluation. In (c), each row reports bootstrap inputs and CPU/GPU time at that point. Totals use unrounded measurements.

(a) CPU Runtime

Operation	BLADE	MOAI
$Q/K/V$ projections	9.82	88.35
$QK^\intercal$	2.58	78.40
Softmax	8.05	7.33
Value aggregation	3.48	2.66
Output projection (W_O)	5.12	3.08
LayerNorm 1	0.49	0.88
FFN expansion	57.18	158.79
GELU	59.82	5.90
FFN output projection	21.38	25.34
LayerNorm 2	0.44	0.99
Total w/o bootstrapping	168.36	371.73
Bootstrapping	389.49	831.05
Total	**557.84**	**1,202.78**

(b) GPU Runtime

Operation	BLADE	MOAI
$Q/K/V$ projections	8.49	13.35
$QK^\intercal$	0.27	8.21
Softmax	0.41	0.37
Value aggregation	0.35	0.28
Output projection (W_O)	4.22	7.06
LayerNorm 1	0.04	0.05
FFN expansion	11.09	16.69
GELU	8.14	1.09
FFN output projection	10.84	32.37
LayerNorm 2	0.04	0.05
Total w/o bootstrapping	43.89	79.52
Bootstrapping	47.71	103.99
Total	**91.61**	**183.51**

(c) Bootstrapping Workload and Runtime

Bootstrapping Point	Ciphertext Counts		CPU Runtime (s)		GPU Runtime (s)	
	BLADE	MOAI	BLADE	MOAI	BLADE	MOAI
Score diagonals	3,456	0	68.05	—	9.280	—
Softmax statistics	144	144	42.58	42.64	0.387	0.383
Attention output	0	9,216	—	186.82	—	25.870
LN1 variance	12	0	3.43	—	0.032	—
LN1 output	0	9,216	—	193.33	—	25.930
Low-rank FFN	4,608	0	89.87	—	12.450	—
FFN output	0	9,216	—	202.42	—	25.880
LN2 variance	12	0	3.86	—	0.034	—
Layer output	9,216	9,216	181.69	205.84	25.530	25.930
Total	**17,448**	**37,008**	**389.49**	**831.05**	**47.713**	**103.993**

Table 2: GPU component ablations. Paired slowdowns compare with BLADE at $r_l = 256$ and 512. Ciphertext counts are per layer for D1/E1/E2 and per LayerNorm invocation for F1.

Variant	Change	Slowdown factor	
		Operation	Encoder layer
D1	Dense scores (1,536 ciphertexts)	$5.41/5.41$	$1.60/1.46$
E1	FFN bootstrap (768 ciphertexts)	$3.01/1.50$	$1.24/1.09$
E2	FFN bootstrap (3,072 ciphertexts)	$11.98/5.93$	$2.32/1.92$
F1	LayerNorm bootstrap (768 ciphertexts)	$374.1/368.5$	$1.79/1.57$

FFN. E1 and E2 retain the low-rank FFN but bootstrap its 768-dimensional input or 3,072-dimensional expanded intermediate, respectively, instead of its rank-r_l intermediate. Bootstrapping time increases nearly in proportion to ciphertext count at both ranks. These comparisons measure the benefit of refreshing the compact intermediate while retaining the same low-rank FFN; complete-layer slowdowns show that the benefit persists beyond the affected operations.

LayerNorm. F1 adds bootstrapping of all 768 centered-value ciphertexts to the variance bootstrap in each LayerNorm, without changing the hidden dimension. It increases complete-layer runtime by $1.79/1.57\times$ for $r_l = 256/512$.

4.4 Task Utility and Encrypted Accuracy

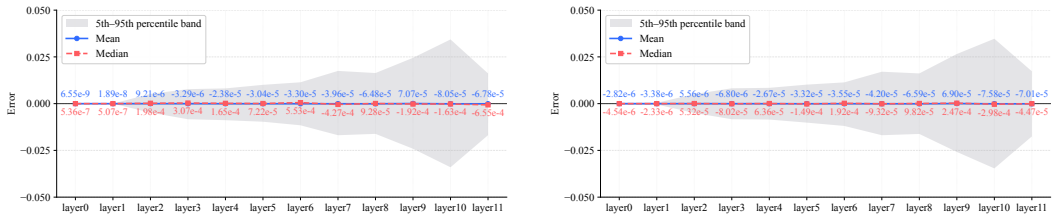
To assess the adapted models' task utility, we evaluate five task-specific checkpoints with exact non-linearities. Their scores average 1.46 metric points below the reported BERT-Base baselines across

five GLUE tasks (Wang et al., 2018) (Table 3). The five adapted models share the encoder dimensions, FFN rank schedule, and 24-offset budget per head in Section 4.1. Task-specific adaptation settings are given in Appendix D.

Table 3: Development-set utility of the adapted models evaluated with exact nonlinearities. Accuracy is reported as a percentage; STS-B and CoLA correlations are multiplied by 100. Δ denotes BLADE minus BERT-Base in metric points, computed from unrounded scores.

Task	Metric	BERT-Base	BLADE	Δ
SST-2	Accuracy	92.43	90.48	-1.95
QQP	Accuracy	90.91	89.54	-1.36
STS-B	Pearson/Spearman mean	87.86	85.09	-2.77
CoLA	MCC	53.39	54.72	$+1.33$
RTE	Accuracy	73.29	70.76	-2.53

For encrypted evaluation, we use the same SST-2 checkpoint as the BLADE runtime measurements in Table 1, distinct from the five task-specific checkpoints above. The runtime and encrypted-accuracy evaluations use identical attention offsets, nonlinear approximation profiles, and bootstrapping schedules on both CPU and GPU. Figure 2 compares layer output errors from plaintext polynomial evaluation and CKKS execution relative to this exact reference. Across the 872 evaluated SST-2 examples, CKKS execution on CPU and GPU preserves every exact-nonlinearity prediction (90.02% accuracy).



(a) Plaintext polynomial evaluation **(b)** CKKS execution

Figure 2: Layer-output errors of plaintext polynomial and CKKS evaluations relative to exact nonlinearities on the same SST-2 checkpoint. Lines show mean and median signed errors; bands show 5th–95th percentiles across validation samples, tokens, and channels.

Plaintext polynomial evaluation keeps the 5th–95th percentile of signed layer-output errors within $[-0.05, 0.05]$ at every layer. CKKS yields a similar aggregate error profile relative to the exact-nonlinearity reference.

Softmax robustness. Because layerwise errors combine nonlinear approximation effects, we isolate Softmax by comparing our mean-centered circuit with MOAI's in plaintext, holding the checkpoint and other operations fixed. Both circuits share a 1,000-example SST-2 fitting set and five separately sampled 1,000-example evaluation subsets. Our circuit reduces accuracy loss relative to exact Softmax from 5.08 to 0.30 pp and worst-layer P99 row error, averaged over subsets, from 9.34×10^{-1} to 6.28×10^{-4}. Its OOD-row rate (rows with non-finite values or inputs outside the Softmax approximation intervals) is higher, but its non-finite-row rate is lower. Appendix E gives the metric definitions and full results.

5 Conclusion

BLADE jointly redesigns Transformer architecture and encrypted dataflow to reduce bootstrapping across a complete 12-layer BERT-Base encoder. Relative to MOAI, it cuts bootstrap inputs from 37,008 to 17,448 and achieves $2.16\times$ CPU and $2.00\times$ GPU encoder speedups. Adapted models average 1.46 metric points below BERT-Base across five GLUE tasks with exact nonlinearities; CKKS matches the runtime SST-2 checkpoint's 90.02% exact-nonlinearity accuracy. A mean-centered homomorphic Softmax supports numerically robust encrypted attention.

AI USE STATEMENT

In this work, we used generative AI tools to assist with manuscript drafting and editing, literature retrieval and synthesis, and feedback on methodology and the interpretation of experimental results. The authors reviewed the AI-assisted material included in the paper and checked the cited sources. The authors take responsibility for the final content of this work.

ETHICS STATEMENT

This work aims to improve the efficiency of privacy-preserving Transformer inference. Our evaluation uses publicly available benchmark datasets. In the proposed setting, the server evaluates encrypted inputs without access to their plaintext values, and input confidentiality relies on the security assumptions of CKKS under the stated semi-honest threat model.

REPRODUCIBILITY STATEMENT

The component designs and bootstrap accounting are described in Sections 3 and 3.4. Section 4.1 and Appendix F report the experimental setup and per-layer execution schedule. Appendices B.2, B.3, and B.4 provide the encrypted component dataflows, and Appendix B.5 describes the homomorphic Softmax circuit. The model adaptation and training configurations for the five-task utility study are reported in Appendix D.

REFERENCES

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Tianyu Chen, Hangbo Bao, Shaohan Huang, Li Dong, Binxing Jiao, Daxin Jiang, Haoyi Zhou, Jianxin Li, and Furu Wei. THE-X: Privacy-preserving transformer inference with homomorphic encryption. In *Findings of the Association for Computational Linguistics: ACL 2022*, pp. 3510–3520. Association for Computational Linguistics, 2022. doi: 10.18653/v1/2022.findings-acl.277.

Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic Encryption for Arithmetic of Approximate Numbers. In *Advances in Cryptology – ASIACRYPT 2017*, pp. 409–437, 2017.

Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. Bootstrapping for Approximate Homomorphic Encryption. In *Advances in Cryptology – EUROCRYPT 2018*, pp. 360–384, 2018.

Seonyoung Cheon, Yongwoo Lee, Dongkwan Kim, Ju Min Lee, Sunchul Jung, Taekyung Kim, Dongyoon Lee, and Hanjun Kim. DaCapo: Automatic Bootstrapping Management for Efficient Fully Homomorphic Encryption. In *33rd USENIX Security Symposium (USENIX Security 24)*, pp. 6993–7010, 2024.

Wonhee Cho, Guillaume Hanrot, Taeseong Kim, Minje Park, and Damien Stehlé. Fast and Accurate Homomorphic Softmax Evaluation. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, CCS '24, pp. 4391–4404, 2024.

Meghan Cowan, Deeksha Dangwal, Armin Alaghi, Caroline Trippel, Vincent T. Lee, and Brandon Reagen. Porcupine: a synthesizing compiler for vectorized homomorphic encryption. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, PLDI 2021, pp. 375–389, 2021.

Roshan Dathathri, Olli Saarikivi, Hao Chen, Kim Laine, Kristin Lauter, Saeed Maleki, Madanlal Musuvathi, and Todd Mytkowicz. CHET: an optimizing compiler for fully-homomorphic neural-network inferencing. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019, pp. 142–156, 2019.

Roshan Dathathri, Blagovesta Kostova, Olli Saarikivi, Wei Dai, Kim Laine, and Madan Musuvathi. 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*, PLDI 2020, pp. 546–561, 2020.

Leo De Castro, Daniel Escudero, Adya Agrawal, Antigoni Polychroniadou, and Manuela Veloso. EncryptedLLM: Privacy-Preserving Large Language Model Inference via GPU-Accelerated Fully Homomorphic Encryption. In *Proceedings of the 42nd International Conference on Machine Learning*, Proceedings of Machine Learning Research, pp. 12677–12688, 2025.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pp. 4171–4186, 2019.

Austin Ebel, Karthik Garimella, and Brandon Reagen. Orion: A Fully Homomorphic Encryption Framework for Deep Learning. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '25, pp. 734–749, 2025.

Xinwen Gao, Shaojing Fu, Lin Liu, Zhuotao Liu, Yuchuan Luo, and Yongjun Wang. Euston: Efficient and user-friendly secure transformer inference with non-interactivity. In *2026 IEEE Symposium on Security and Privacy (SP)*, pp. 1443–1462. IEEE, 2026.

Shai Halevi and Victor Shoup. Algorithms in HElib. In *Advances in Cryptology – CRYPTO 2014*, pp. 554–571. Springer Berlin Heidelberg, 2014.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (GELUs). *arXiv preprint arXiv:1606.08415*, 2016.

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*, CCS '18, pp. 1209–1222, 2018.

Aleksandar Krastev, Nikola Samardzic, Simon Langowski, Srinivas Devadas, and Daniel Sanchez. A tensor compiler with automatic data packing for simple and efficient fully homomorphic encryption. *Proc. ACM Program. Lang.*, (PLDI), 2024.

Yongwoo Lee, Seonyeong Heo, Seonyoung Cheon, Shinnung Jeong, Changsu Kim, Eunkyung Kim, Dongyoon Lee, and Hanjun Kim. HECATE: Performance-aware scale optimization for homomorphic encryption compiler. In *2022 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pp. 193–204. IEEE, 2022.

Yongwoo Lee, Seonyoung Cheon, Dongkwan Kim, Dongyoon Lee, and Hanjun Kim. ELASM: Error-Latency-Aware Scale Management for Fully Homomorphic Encryption. In *32nd USENIX Security Symposium (USENIX Security 23)*, pp. 4697–4714, 2023.

Jungho Moon, Dongwoo Yoo, Xiaoqian Jiang, and Miran Kim. THOR: Secure Transformer Inference with Homomorphic Encryption. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, CCS '25, pp. 3765–3779, 2025.

Qi Pang, Jinhao Zhu, Helen Möllering, Wenting Zheng, and Thomas Schneider. BOLT: Privacy-Preserving, Accurate and Efficient Inference for Transformers. In *2024 IEEE Symposium on Security and Privacy (SP)*, pp. 4753–4771, 2024.

Dongjin Park, Eunsang Lee, and Joon-Woo Lee. Powerformer: Efficient and high-accuracy privacy-preserving language model with homomorphic encryption. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, pp. 11090–11111, 2025.

Donghwan Rho, Taeseong Kim, Minje Park, Jung Woo Kim, Hyunsik Chae, Ernest Ryu, and Jung Hee Cheon. Encryption-Friendly LLM Architecture. In *International Conference on Learning Representations*, pp. 41638–41664, 2025.

SEAL. Microsoft SEAL (release 4.1). https://github.com/Microsoft/SEAL, 2023. Microsoft Research, Redmond, WA.

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. Patient Knowledge Distillation for BERT Model Compression. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 4323–4332, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems*, 2017.

Alexander Viand, Patrick Jattke, Miro Haller, and Anwar Hithnawi. HECO: Fully Homomorphic Encryption Compiler. In *32nd USENIX Security Symposium (USENIX Security 23)*, pp. 4715–4732, 2023.

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. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pp. 353–355, 2018.

Hao Yang, Shiyu Shen, Wangchen Dai, Lu Zhou, Zhe Liu, and Yunlei Zhao. Phantom: A CUDA-accelerated word-wise homomorphic encryption library. *IEEE Transactions on Dependable and Secure Computing*, 21(5):4895–4906, 2024. doi: 10.1109/TDSC.2024.3363900.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big Bird: Transformers for Longer Sequences. In *Advances in Neural Information Processing Systems*, pp. 17283–17297, 2020.

Jiawen Zhang, Xinpeng Yang, Lipeng He, Kejia Chen, Wen-jie Lu, Yinghao Wang, Xiaoyang Hou, Jian Liu, Kui Ren, and Xiaohu Yang. Secure Transformer Inference Made Non-interactive. In *32nd Annual Network and Distributed System Security Symposium*, 2025a.

Linru Zhang, Xiangning Wang, Jun Jie Sim, Zhicong Huang, Jiahao Zhong, Huaxiong Wang, Pu Duan, and Kwok Yan Lam. MOAI: Module-optimizing architecture for non-interactive secure transformer inference. In *International Conference on Learning Representations*, pp. 27927–27953, 2026.

Yancheng Zhang, Jiaqi Xue, Mengxin Zheng, Mimi Xie, Mingzhe Zhang, Lei Jiang, and Qian Lou. CipherPrune: Efficient and scalable private transformer inference. In *International Conference on Learning Representations*, pp. 59578–59596, 2025b. ISBN 979-8-3313-2085-0.

Itamar Zimerman, Moran Baruch, Nir Drucker, Gilad Ezov, Omri Soceanu, and Lior Wolf. Converting transformers to polynomial form for secure inference over homomorphic encryption. In *Proceedings of the 41st International Conference on Machine Learning*, Proceedings of Machine Learning Research, pp. 62803–62814, 2024.

A Transformer Background

This appendix summarizes the standard Transformer operations using the notation introduced in Section 2.1.

Attention. The input X is projected into a query matrix $Q = XW_Q$, a key matrix $K = XW_K$, and a value matrix $V = XW_V$, where $W_Q, W_K, W_V \in \mathbb{R}^{d \times d_h}$ are learned weight matrices and $Q, K, V \in \mathbb{R}^{n \times d_h}$. Scaled dot-product attention is

$$\mathrm{Attention}(Q, K, V) = \mathrm{Softmax}\left(\frac{QK^\mathsf{T}}{\sqrt{d_h}}\right) V. \tag{9}$$

Dense attention computes n^2 query–key scores; sparse attention computes only selected scores, as in sliding-window attention (Zaheer et al., 2020). In multi-head attention, the h outputs $\mathrm{Attention}(Q_j, K_j, V_j)$, $j \in [h]$, are computed in parallel and concatenated.

FFN. The fully connected feed-forward network applies the same two linear transformations to every row of X:

$$\mathrm{FFN}(X) = \mathrm{GELU}(XW_1 + \boldsymbol{b}_1)\, W_2 + \boldsymbol{b}_2. \tag{10}$$

Here $W_1 \in \mathbb{R}^{d \times d_f}$, $W_2 \in \mathbb{R}^{d_f \times d}$, $\boldsymbol{b}_1 \in \mathbb{R}^{d_f}$, and $\boldsymbol{b}_2 \in \mathbb{R}^d$; each bias is added to every row. The output is added to the residual input, and the sum is normalized by LayerNorm.

LayerNorm. For a token vector $\boldsymbol{x} \in \mathbb{R}^d$, let $\mu = \frac{1}{d}\sum_{i=0}^{d-1} x_i$ and $v = \frac{1}{d}\sum_{i=0}^{d-1}(x_i - \mu)^2 + \epsilon$. With learned $\boldsymbol{\gamma}, \boldsymbol{\beta} \in \mathbb{R}^d$ and a stabilization constant $\epsilon > 0$, the output for $i = 0, \ldots, d-1$ is

$$y_i = \gamma_i \frac{x_i - \mu}{\sqrt{v}} + \beta_i. \tag{11}$$

B Encrypted Computation Details

B.1 Ciphertext Packing Details

This subsection gives the full definitions of the column and diagonal packings (Halevi & Shoup, 2014), together with the interleaved batching and column-to-diagonal multiplication semantics.

Definition B.1 (Column Packing) An $n \times d$ matrix $X \in \mathbb{R}^{n \times d}$ is represented by d ciphertexts under column packing

$$\mathrm{Enc}_{\mathrm{col}}(X) = \{[\![\boldsymbol{x}_0]\!], \ldots, [\![\boldsymbol{x}_{d-1}]\!]\}, \tag{12}$$

where $[\![\boldsymbol{x}_j]\!]$ encrypts the j-th column of X, for $0 \le j < d$.

Definition B.2 (Diagonal Packing) An $n \times n$ square matrix $X \in \mathbb{R}^{n \times n}$ is represented by n ciphertexts under diagonal packing

$$\mathrm{Enc}_{\mathrm{diag}}(X) = \left\{[\![\mathrm{Diag}_0(X)]\!], \ldots, [\![\mathrm{Diag}_{n-1}(X)]\!]\right\}, \tag{13}$$

where each ciphertext $[\![\mathrm{Diag}_j(X)]\!]$ represents all entries of X associated with one relative offset. Signed offsets are interpreted modulo n. For a subset of offsets $\mathcal{D} \subseteq [n]$, only the corresponding diagonal ciphertexts are materialized; we denote this restricted representation by $\mathrm{Enc}^{\mathcal{D}}_{\mathrm{diag}}(X)$. When $\mathcal{D}$ is fixed, we write $\mathrm{Enc}_{\mathrm{diag}}(X)$ for $\mathrm{Enc}^{\mathcal{D}}_{\mathrm{diag}}(X)$.

We use both packing methods with interleaved batching (Jiang et al., 2018). With $B = \frac{N}{2n}$ inputs per ciphertext, row t of input b in a fixed column or diagonal occupies $\mathrm{slot}(t, b) = Bt + b$, where $0 \le t < n$ and $0 \le b < B$. In the MOAI configuration used by both systems, each column ciphertext packs one hidden dimension across $B = 256$ requests of $n = 128$ token positions, occupying all $Bn = 2^{15} = N/2$ slots for $N = 2^{16}$.

For column-packed $X, Y \in \mathbb{R}^{n \times d}$, each cyclic diagonal of XY^{T} can be generated independently via column-to-diagonal multiplication (Pang et al., 2024; Zhang et al., 2025a; 2026). For offset δ, we rotate the columns of Y by $B\delta$ slots, multiply them slot-wise with the corresponding columns of X, and sum across columns. This directly produces the required diagonal without first materializing the product in another packing.

For a column-packed encrypted matrix $M \in \mathbb{R}^{n \times a}$ and a public plaintext matrix $W \in \mathbb{R}^{a \times b}$, Algorithm 1 computes the column-packed representation of $R = MW \in \mathbb{R}^{n \times b}$.

Algorithm 1: $\mathrm{Mult}_{\mathrm{ct,pt}}()$: Ciphertext–Plaintext Multiplication with a Column-Packed Matrix

Input: $\mathrm{Enc}_{\mathrm{col}}(M) = \{[\![\boldsymbol{m}_i]\!]\}_{i=0}^{a-1}$, where $M \in \mathbb{R}^{n\times a}$; $W \in \mathbb{R}^{a\times b}$: public plaintext weight matrix, where $w_{i,j}$ denotes the (i,j)-th entry of W.

Output: $\mathrm{Enc}_{\mathrm{col}}(R) = \{[\![\boldsymbol{r}_j]\!]\}_{j=0}^{b-1}$, where $R = MW \in \mathbb{R}^{n\times b}$.

for $j \leftarrow 0$ **to** $b-1$ **do**
  $[\![\boldsymbol{r}_j]\!] \leftarrow [\![\boldsymbol{m}_0]\!] \odot w_{0,j}$;
  for $i \leftarrow 1$ **to** $a-1$ **do**
    $[\![\boldsymbol{r}_j]\!] \leftarrow [\![\boldsymbol{r}_j]\!] + [\![\boldsymbol{m}_i]\!] \odot w_{i,j}$;
  end for
end for
return $\{[\![\boldsymbol{r}_j]\!]\}_{j=0}^{b-1}$.

Each output ciphertext is therefore a linear combination of the input column ciphertexts with public scalar coefficients. Since the same coefficient is applied to every slot of a column ciphertext, $\mathrm{Mult}_{\mathrm{ct,pt}}$ requires slot-wise ciphertext–plaintext multiplications and ciphertext additions and does not require rotations.

B.2 FHE-Friendly Cyclic Sparse Attention

Algorithm 2 gives the complete encrypted dataflow for Section 3.1. It computes and bootstraps only the score ciphertexts indexed by $\mathcal{D}$, applies Softmax in diagonal packing, and returns the attention output in column packing. The packing and rotation semantics follow Appendix B.1.

Padded input positions are initially zero-filled. No padding masks are applied within the encoder: selected offsets pointing to padded positions participate in Softmax normalization and value aggregation. Plaintext training, exact-nonlinearity evaluation, plaintext polynomial evaluation, and CKKS execution use the same padding convention.

Algorithm 2: FHE-Friendly Cyclic Sparse Attention

Input: $\mathrm{Enc}_{\mathrm{col}}(Q) = \{[\![\boldsymbol{q}_j]\!]\}_{j=0}^{d_h-1}$, $\mathrm{Enc}_{\mathrm{col}}(K) = \{[\![\boldsymbol{k}_j]\!]\}_{j=0}^{d_h-1}$, and $\mathrm{Enc}_{\mathrm{col}}(V) = \{[\![\boldsymbol{v}_j]\!]\}_{j=0}^{d_h-1}$: column-packed ciphertexts encoding $Q, K, V \in \mathbb{R}^{n\times d_h}$ for one attention head.

Output: $\mathrm{Enc}_{\mathrm{col}}(O) = \{[\![\boldsymbol{o}_j]\!]\}_{j=0}^{d_h-1}$: column-packed ciphertexts encoding the attention output $O \in \mathbb{R}^{n\times d_h}$.

Public parameters: $\mathcal{D}$: selected cyclic relative-offset set with $|\mathcal{D}| = m$; B: number of inputs interleaved in each ciphertext.

for $\delta \in \mathcal{D}$ **do**
  $[\![\boldsymbol{s}_\delta]\!] \leftarrow [\![\boldsymbol{q}_0]\!] \odot \mathrm{Rot}_{B\delta}([\![\boldsymbol{k}_0]\!])$;
  for $j \leftarrow 1$ **to** d_h-1 **do**
    $[\![\boldsymbol{s}_\delta]\!] \leftarrow [\![\boldsymbol{s}_\delta]\!] + [\![\boldsymbol{q}_j]\!] \odot \mathrm{Rot}_{B\delta}([\![\boldsymbol{k}_j]\!])$;
  end for
  $[\![\boldsymbol{s}_\delta]\!] \leftarrow [\![\boldsymbol{s}_\delta]\!] \odot \frac{1}{\sqrt{d_h}}$;
end for
for $\delta \in \mathcal{D}$ **do**
  $[\![\boldsymbol{s}_\delta]\!] \leftarrow \mathsf{Boot}([\![\boldsymbol{s}_\delta]\!])$;
end for
$\{[\![\boldsymbol{p}_\delta]\!]\}_{\delta\in\mathcal{D}} \leftarrow \mathrm{Softmax}(\{[\![\boldsymbol{s}_\delta]\!]\}_{\delta\in\mathcal{D}})$;
for $j \leftarrow 0$ **to** d_h-1 **do**
  $[\![\boldsymbol{o}_j]\!] \leftarrow [\![\boldsymbol{0}]\!]$;
  for $\delta \in \mathcal{D}$ **do**
    $[\![\boldsymbol{o}_j]\!] \leftarrow [\![\boldsymbol{o}_j]\!] + [\![\boldsymbol{p}_\delta]\!] \odot \mathrm{Rot}_{B\delta}([\![\boldsymbol{v}_j]\!])$;
  end for
end for
return $\{[\![\boldsymbol{o}_j]\!]\}_{j=0}^{d_h-1}$.

B.3 LOW-RANK FFN

Algorithm 3 gives the complete encrypted dataflow for Section 3.2. It first forms and bootstraps the rank-r_l intermediate before expansion, and then carries out the remaining factorized FFN computation in column packing.

Algorithm 3: Low-Rank FFN

Input: $\mathrm{Enc}_{\mathrm{col}}(X)$: column-packed ciphertexts encoding $X \in \mathbb{R}^{n\times d}$.
Output: $\mathrm{Enc}_{\mathrm{col}}(Y)$: column-packed ciphertexts encoding $Y \in \mathbb{R}^{n\times d}$.
Public parameters: $A_1 \in \mathbb{R}^{d\times r_l}$ and $B_1 \in \mathbb{R}^{r_l\times d_f}$: low-rank factors of $W_1 = A_1B_1$;
$A_2 \in \mathbb{R}^{d_f\times r_l}$ and $B_2 \in \mathbb{R}^{r_l\times d}$: low-rank factors of $W_2 = A_2B_2$;
$\boldsymbol{b}_1 \in \mathbb{R}^{d_f}$ and $\boldsymbol{b}_2 \in \mathbb{R}^{d}$: FFN bias vectors.

$\mathrm{Enc}_{\mathrm{col}}(Z) \leftarrow \mathrm{Mult}_{\mathrm{ct,pt}}(\mathrm{Enc}_{\mathrm{col}}(X), A_1)$;
for $[\![\boldsymbol{z}]\!] \in \mathrm{Enc}_{\mathrm{col}}(Z)$ **do**
  $[\![\boldsymbol{z}]\!] \leftarrow \mathsf{Boot}([\![\boldsymbol{z}]\!])$;
end for
$\mathrm{Enc}_{\mathrm{col}}(U) \leftarrow \mathrm{Mult}_{\mathrm{ct,pt}}(\mathrm{Enc}_{\mathrm{col}}(Z), B_1)$;
$\mathrm{Enc}_{\mathrm{col}}(U) \leftarrow \mathrm{Add}(\mathrm{Enc}_{\mathrm{col}}(U), \boldsymbol{b}_1)$;
for $[\![\boldsymbol{u}]\!] \in \mathrm{Enc}_{\mathrm{col}}(U)$ **do**
  $[\![\boldsymbol{u}]\!] \leftarrow \mathrm{GELU}([\![\boldsymbol{u}]\!])$;
end for
$\mathrm{Enc}_{\mathrm{col}}(C) \leftarrow \mathrm{Mult}_{\mathrm{ct,pt}}(\mathrm{Enc}_{\mathrm{col}}(U), A_2)$;
$\mathrm{Enc}_{\mathrm{col}}(Y) \leftarrow \mathrm{Mult}_{\mathrm{ct,pt}}(\mathrm{Enc}_{\mathrm{col}}(C), B_2)$;
$\mathrm{Enc}_{\mathrm{col}}(Y) \leftarrow \mathrm{Add}(\mathrm{Enc}_{\mathrm{col}}(Y), \boldsymbol{b}_2)$;
return $\mathrm{Enc}_{\mathrm{col}}(Y)$.

B.4 LAYERNORM COMPUTATION

We detail the ciphertext operations for the LayerNorm reformulation described in Section 3.3. The packing semantics follow Appendix B.1. Under interleaved column packing, each ciphertext $[\![\boldsymbol{x}_i]\!]$ contains one hidden-dimension column for all B inputs, and all operations below are performed slot-wise. Only the variance ciphertext $[\![\boldsymbol{v}]\!]$ is bootstrapped. The d centered-value ciphertexts $\{[\![\boldsymbol{z}_i]\!]\}_{i=0}^{d-1}$ are retained and reused for final normalization.

Algorithm 4 expresses the mathematical dataflow using unscaled centered values. The implementation uses $\widetilde{\boldsymbol{z}}_i = d\boldsymbol{x}_i - \sum_j \boldsymbol{x}_j = d\boldsymbol{z}_i$, compensating with $\frac{\gamma_i}{d}$ in the normalization branch. The variance branch computes $\frac{1}{d^3}\sum_i \widetilde{\boldsymbol{z}}_i \odot \widetilde{\boldsymbol{z}}_i$ and adds ϵ after bootstrapping, before inverse-square-root evaluation. The inverse-square-root approximation is evaluated using a Chebyshev seed followed by Newton refinement, with settings specific to each layer and LayerNorm invocation. Appendix F details the approximation settings, rescaling, and level and scale alignment before the branches merge.

B.5 MEAN-CENTERED SOFTMAX COMPUTATION

Polynomial Softmax evaluation is sensitive to the range of encrypted attention scores. Plaintext implementations commonly subtract the row maximum for numerical stability, but homomorphic maximum evaluation is costly (Cho et al., 2024). MOAI uses a layer-specific public shift (Zhang et al., 2026), which does not adapt to each row's common score offset. We subtract the encrypted row mean to remove that offset without changing exact Softmax probabilities or requiring encrypted comparisons. We then halve the centered scores to narrow the exponential approximation interval; squaring after the first normalization restores the original score differences. Finally, we subtract a public shift $\tau \in \mathbb{R}$ to make the exponential polynomial's input intervals more balanced around zero overall. The polynomial $P_{\exp}$ approximates $\exp(x+\tau)$ on these shifted inputs, so the shift changes its evaluation coordinate without changing the intended exponential values.

We use the interleaved diagonal packing described in Appendix B.1. $[\![\boldsymbol{x}]\!]$ denotes a ciphertext encrypting vector $\boldsymbol{x}$. Here, $\mathsf{PolyEval}(P, [\![\boldsymbol{x}]\!])$ evaluates the public polynomial P on each encrypted entry of $\boldsymbol{x}$, and $\odot$ denotes slot-wise multiplication. Public scalars are encoded with the same value in each occupied slot.

Algorithm 4: Encrypted LayerNorm Computation

Input: $\mathrm{Enc}_{\mathrm{col}}(X) = \{[\![\boldsymbol{x}_i]\!]\}_{i=0}^{d-1}$, where $X = [\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{d-1}] \in \mathbb{R}^{n\times d}$.
Output: $\mathrm{Enc}_{\mathrm{col}}(Y) = \{[\![\boldsymbol{y}_i]\!]\}_{i=0}^{d-1}$, where $Y = [\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{d-1}] \in \mathbb{R}^{n\times d}$.
Public parameters: ϵ: LayerNorm stabilization constant; $\boldsymbol{\gamma}, \boldsymbol{\beta} \in \mathbb{R}^d$: LayerNorm scale and bias vectors; $P_{\mathrm{seed},l,k}$ and $N_{l,k}$: Chebyshev seed polynomial and Newton iteration count for invocation $k \in \{1, 2\}$ in layer l.

$[\![\boldsymbol{\mu}]\!] \leftarrow [\![\boldsymbol{x}_0]\!]$;
for $i \leftarrow 1$ **to** $d-1$ **do**
  $[\![\boldsymbol{\mu}]\!] \leftarrow [\![\boldsymbol{\mu}]\!] + [\![\boldsymbol{x}_i]\!]$;
end for
$[\![\boldsymbol{\mu}]\!] \leftarrow [\![\boldsymbol{\mu}]\!] \odot \frac{1}{d}$;
for $i \leftarrow 0$ **to** $d-1$ **do**
  $[\![\boldsymbol{z}_i]\!] \leftarrow [\![\boldsymbol{x}_i]\!] - [\![\boldsymbol{\mu}]\!]$;
end for
$[\![\boldsymbol{v}]\!] \leftarrow [\![\boldsymbol{z}_0]\!] \odot [\![\boldsymbol{z}_0]\!]$;
for $i \leftarrow 1$ **to** $d-1$ **do**
  $[\![\boldsymbol{v}]\!] \leftarrow [\![\boldsymbol{v}]\!] + [\![\boldsymbol{z}_i]\!] \odot [\![\boldsymbol{z}_i]\!]$;
end for
$[\![\boldsymbol{v}]\!] \leftarrow [\![\boldsymbol{v}]\!] \odot \frac{1}{d}$;
$[\![\boldsymbol{v}]\!] \leftarrow \mathsf{Boot}([\![\boldsymbol{v}]\!])$;
$[\![\boldsymbol{v}]\!] \leftarrow [\![\boldsymbol{v}]\!] + \epsilon$;
$[\![\boldsymbol{r}]\!] \leftarrow \mathsf{PolyEval}(P_{\mathrm{seed},l,k}, [\![\boldsymbol{v}]\!])$;
for $t \leftarrow 1$ **to** $N_{l,k}$ **do**
  $[\![\boldsymbol{r}]\!] \leftarrow \frac{1}{2} \odot [\![\boldsymbol{r}]\!] \odot (3 - [\![\boldsymbol{v}]\!] \odot [\![\boldsymbol{r}]\!] \odot [\![\boldsymbol{r}]\!])$;
end for
for $i \leftarrow 0$ **to** $d-1$ **do**
  $[\![\boldsymbol{y}_i]\!] \leftarrow [\![\boldsymbol{z}_i]\!] \odot [\![\boldsymbol{r}]\!]$;
  $[\![\boldsymbol{y}_i]\!] \leftarrow [\![\boldsymbol{y}_i]\!] \odot \gamma_i$;
  $[\![\boldsymbol{y}_i]\!] \leftarrow [\![\boldsymbol{y}_i]\!] + \beta_i$;
end for
return $\{[\![\boldsymbol{y}_i]\!]\}_{i=0}^{d-1}$.

For one attention head, the circuit takes the m selected score ciphertexts $\mathrm{Enc}_{\mathrm{diag}}(S) = \{[\![\boldsymbol{s}_\delta]\!]\}_{\delta\in\mathcal{D}}$ after score bootstrapping in Section 3.1. Here $S \in \mathbb{R}^{n\times n}$ is the score matrix for one input; each ciphertext interleaves the corresponding cyclic diagonals $\boldsymbol{s}_\delta \in \mathbb{R}^n$ of B inputs. All vectors below have n entries per input. For each $\delta \in \mathcal{D}$, we compute

$$[\![\bar{\boldsymbol{s}}]\!] = \frac{1}{m}\sum_{\delta\in\mathcal{D}} [\![\boldsymbol{s}_\delta]\!], \qquad [\![\boldsymbol{u}_\delta]\!] = ([\![\boldsymbol{s}_\delta]\!] - [\![\bar{\boldsymbol{s}}]\!]) \odot \frac{1}{2} - \tau. \tag{14}$$

Corresponding query positions occupy the same slots across ciphertexts, so centering requires only additions and multiplications by public scalars, without rotations or encrypted comparisons.

Squaring the normalized exponential values compensates for the factor $\frac{1}{2}$ by doubling the exponent. Normalizing only after squaring would require approximating the reciprocal of an unnormalized sum of squares, whose magnitude can vary widely across rows even after mean centering. We therefore also normalize before squaring, following the normalize-and-square strategy (Cho et al., 2024), to control the intermediate magnitudes and the input range of the final reciprocal approximation.

With the public polynomial $P_{\exp}$ approximating $\exp(x+\tau)$ on the shifted input, we compute $[\![\boldsymbol{a}_\delta]\!] = \mathsf{PolyEval}(P_{\exp}, [\![\boldsymbol{u}_\delta]\!])$ and $[\![\boldsymbol{a}_{\mathrm{sum}}]\!] = \sum_{\delta\in\mathcal{D}} [\![\boldsymbol{a}_\delta]\!]$. Before the first reciprocal polynomial, we scale the sum and apply a public affine transformation:

$$[\![\boldsymbol{a}_{\mathrm{scaled}}]\!] = [\![\boldsymbol{a}_{\mathrm{sum}}]\!] \odot \frac{1}{d_{0,\mathrm{scale}}}, \qquad [\![\boldsymbol{v}_0]\!] = [\![\boldsymbol{a}_{\mathrm{scaled}}]\!] \odot \alpha_0 + \beta_0.$$

Here $d_{0,\mathrm{scale}}$, α_0, and β_0 are public; layer indices are suppressed. The R_0 intervals in Table 7 describe a_{scaled} before the affine map. The degree-15 normalized-domain power polynomial $P_{\mathrm{rec},0}$

takes $[\![\boldsymbol{v}_0]\!]$ as input. Its coefficients include the factor $\frac{1}{d_{0,\mathrm{scale}}}$, so its target as a function of v is

$$P_{\mathrm{rec},0}(v) \approx \frac{\alpha_0}{d_{0,\mathrm{scale}}(v-\beta_0)}.$$

Thus $[\![\boldsymbol{r}_0]\!] = \mathsf{PolyEval}(P_{\mathrm{rec},0}, [\![\boldsymbol{v}_0]\!])$ approximates the reciprocal of the original sum and is used to normalize each selected diagonal before slot-wise squaring:

$$[\![\boldsymbol{q}_\delta]\!] = \left([\![\boldsymbol{a}_\delta]\!] \odot [\![\boldsymbol{r}_0]\!]\right)^2. \tag{15}$$

We retain these m ciphertexts for the final normalization. With exact exponential and reciprocal evaluations, the values before squaring lie in $[0,1]$ and sum to one at each query position.

To normalize the squared values, we form $[\![\boldsymbol{q}_{\mathrm{sum}}]\!] = 4\sum_{\delta\in\mathcal{D}}[\![\boldsymbol{q}_\delta]\!]$. The preceding normalization bounds its entries in $[\frac{4}{m}, 4]$ under exact evaluation. Only this sum ciphertext is bootstrapped:

$$[\![\boldsymbol{q}_{\mathrm{sum}}]\!] \leftarrow \mathsf{Boot}([\![\boldsymbol{q}_{\mathrm{sum}}]\!]). \tag{16}$$

Let $d_{1,\mathrm{scale}}$, α_1, and β_1 be the public parameters of the second reciprocal stage. After bootstrapping, we form

$$[\![\boldsymbol{v}_1]\!] = [\![\boldsymbol{q}_{\mathrm{sum}}]\!] \odot \frac{\alpha_1}{d_{1,\mathrm{scale}}} + \beta_1, \qquad [\![\boldsymbol{r}_1]\!] = \mathsf{PolyEval}(P_{\mathrm{rec},1}, [\![\boldsymbol{v}_1]\!]) \odot \frac{4}{d_{1,\mathrm{scale}}}.$$

The multiplication by $\frac{\alpha_1}{d_{1,\mathrm{scale}}}$ and the final output scaling are each followed by rescaling. The degree-43 Chebyshev-U polynomial acts on v_1 and approximates $P_{\mathrm{rec},1}(v) \approx \frac{\alpha_1}{v-\beta_1}$. Thus $[\![\boldsymbol{r}_1]\!]$ approximates $\frac{4}{q_{\mathrm{sum}}}$, compensating for the factor 4 in the sum. Appendix F specifies the polynomial degrees, layer-specific approximation intervals, and public shift τ; we suppress layer indices here for readability. We reuse the retained ciphertexts to compute, for each $\delta \in \mathcal{D}$,

$$[\![\boldsymbol{p}_\delta]\!] = [\![\boldsymbol{q}_\delta]\!] \odot [\![\boldsymbol{r}_1]\!]. \tag{17}$$

This final multiplication leaves the output at level 6 and scale Δ^2, where $\Delta = 2^{46}$; rescaling is deferred until after the value-aggregation products have been accumulated, as detailed in Appendix F. The output $\mathrm{Enc}_{\mathrm{diag}}(P) = \{[\![\boldsymbol{p}_\delta]\!]\}_{\delta\in\mathcal{D}}$ represents the selected diagonals of the sparse probability matrix $P \in \mathbb{R}^{n\times n}$. With exact function evaluations, final normalization cancels factors common to all selected scores in a row and recovers Softmax over the selected offsets. The circuit preserves interleaved diagonal packing and contributes h ciphertext inputs to C_{boot} per encoder layer. Algorithm 5 gives the complete encrypted dataflow.

Exact-function identity. We follow Algorithm 5 for one input and one attention head. Let n be the sequence length and $t \in \{0,\dots,n-1\}$ a query position. For any vector $\boldsymbol{x}$ in the algorithm, $x(t)$ denotes its entry for this input at query position t. In particular, for $\delta \in \mathcal{D}$, the input score is $s_\delta(t) = S[t, (t+\delta) \bmod n]$, where S is the scaled attention score matrix. The derivation uses exact exponential and reciprocal functions and exact arithmetic; bootstrapping is therefore treated as preserving the represented values.

The first accumulation and multiplication by $\frac{1}{m}$ in the algorithm give the row mean $\bar{s}(t) = \frac{1}{m}\sum_{\delta'\in\mathcal{D}} s_{\delta'}(t)$. This is the entry at query position t of $\bar{\boldsymbol{s}}$ in Equation (14). The mean is taken over the m selected offsets with the input, attention head, and query position fixed. Subtracting this mean, halving, and applying the public shift give $u_\delta(t) = \frac{s_\delta(t)-\bar{s}(t)}{2} - \tau$. The exact target of the exponential polynomial then gives $a_\delta(t) = \exp\big(u_\delta(t)+\tau\big) = \exp\big(\frac{s_\delta(t)-\bar{s}(t)}{2}\big)$.

The algorithm sums these positive exponential values to obtain $a_{\mathrm{sum}}(t) = \sum_{\delta'\in\mathcal{D}} a_{\delta'}(t)$, then forms $a_{\mathrm{scaled}}(t) = \frac{a_{\mathrm{sum}}(t)}{d_{0,\mathrm{scale}}}$ and $v_0(t) = \alpha_0 a_{\mathrm{scaled}}(t) + \beta_0$. Under exact reciprocal evaluation,

$$r_0(t) = \frac{\alpha_0}{d_{0,\mathrm{scale}}(v_0(t)-\beta_0)} = \frac{1}{a_{\mathrm{sum}}(t)}.$$

Multiplying by $r_0(t)$ normalizes the values to sum to one at query position t. The algorithm then squares each normalized value. In the derivation below, $q_\delta(t) = (r_0(t)a_\delta(t))^2$ denotes this squared

Algorithm 5: Encrypted Mean-Centered Softmax

Input: $\mathrm{Enc}_{\mathrm{diag}}(S) = \{[\![\boldsymbol{s}_\delta]\!]\}_{\delta\in\mathcal{D}}$, containing the m selected score diagonals indexed by $\mathcal{D}$.
Output: $\mathrm{Enc}_{\mathrm{diag}}(P) = \{[\![\boldsymbol{p}_\delta]\!]\}_{\delta\in\mathcal{D}}$, containing the corresponding Softmax probability diagonals indexed by $\mathcal{D}$.
Public parameters: $\mathcal{D}$: selected relative-offset set with $|\mathcal{D}| = m$; τ: public shift; $d_{0,\mathrm{scale}}$, α_0, and β_0: public first-reciprocal scaling and affine parameters; $d_{1,\mathrm{scale}}$, α_1, and β_1: public second-reciprocal scaling and affine parameters; $P_{\exp}$: polynomial approximation of $\exp(x+\tau)$ on the shifted input; $P_{\mathrm{rec},0}$: normalized-domain reciprocal polynomial with $\frac{1}{d_{0,\mathrm{scale}}}$ folded into its coefficients; $P_{\mathrm{rec},1}$: degree-43 Chebyshev-U polynomial approximating $\frac{\alpha_1}{v-\beta_1}$ on input v.

$[\![\bar{\boldsymbol{s}}]\!] \leftarrow [\![\boldsymbol{0}]\!]$;
for $\delta \in \mathcal{D}$ **do**
  $[\![\bar{\boldsymbol{s}}]\!] \leftarrow [\![\bar{\boldsymbol{s}}]\!] + [\![\boldsymbol{s}_\delta]\!]$;
end for
$[\![\bar{\boldsymbol{s}}]\!] \leftarrow [\![\bar{\boldsymbol{s}}]\!] \odot \frac{1}{m}$;
for $\delta \in \mathcal{D}$ **do**
  $[\![\boldsymbol{u}_\delta]\!] \leftarrow [\![\boldsymbol{s}_\delta]\!] - [\![\bar{\boldsymbol{s}}]\!]$;
  $[\![\boldsymbol{u}_\delta]\!] \leftarrow [\![\boldsymbol{u}_\delta]\!] \odot \frac{1}{2}$;
  $[\![\boldsymbol{u}_\delta]\!] \leftarrow [\![\boldsymbol{u}_\delta]\!] - \tau$;
  $[\![\boldsymbol{a}_\delta]\!] \leftarrow \mathsf{PolyEval}(P_{\exp}, [\![\boldsymbol{u}_\delta]\!])$;
end for
$[\![\boldsymbol{a}_{\mathrm{sum}}]\!] \leftarrow [\![\boldsymbol{0}]\!]$;
for $\delta \in \mathcal{D}$ **do**
  $[\![\boldsymbol{a}_{\mathrm{sum}}]\!] \leftarrow [\![\boldsymbol{a}_{\mathrm{sum}}]\!] + [\![\boldsymbol{a}_\delta]\!]$;
end for
$[\![\boldsymbol{a}_{\mathrm{scaled}}]\!] \leftarrow [\![\boldsymbol{a}_{\mathrm{sum}}]\!] \odot \frac{1}{d_{0,\mathrm{scale}}}$;
$[\![\boldsymbol{v}_0]\!] \leftarrow [\![\boldsymbol{a}_{\mathrm{scaled}}]\!] \odot \alpha_0 + \beta_0$;
$[\![\boldsymbol{r}_0]\!] \leftarrow \mathsf{PolyEval}(P_{\mathrm{rec},0}, [\![\boldsymbol{v}_0]\!])$;
for $\delta \in \mathcal{D}$ **do**
  $[\![\boldsymbol{q}_\delta]\!] \leftarrow [\![\boldsymbol{a}_\delta]\!] \odot [\![\boldsymbol{r}_0]\!]$;
  $[\![\boldsymbol{q}_\delta]\!] \leftarrow [\![\boldsymbol{q}_\delta]\!] \odot [\![\boldsymbol{q}_\delta]\!]$;
end for
$[\![\boldsymbol{q}_{\mathrm{sum}}]\!] \leftarrow [\![\boldsymbol{0}]\!]$;
for $\delta \in \mathcal{D}$ **do**
  $[\![\boldsymbol{q}_{\mathrm{sum}}]\!] \leftarrow [\![\boldsymbol{q}_{\mathrm{sum}}]\!] + [\![\boldsymbol{q}_\delta]\!]$;
end for
$[\![\boldsymbol{q}_{\mathrm{sum}}]\!] \leftarrow [\![\boldsymbol{q}_{\mathrm{sum}}]\!] \odot 4$;
$[\![\boldsymbol{q}_{\mathrm{sum}}]\!] \leftarrow \mathsf{Boot}([\![\boldsymbol{q}_{\mathrm{sum}}]\!])$;
$[\![\boldsymbol{v}_1]\!] \leftarrow [\![\boldsymbol{q}_{\mathrm{sum}}]\!] \odot \frac{\alpha_1}{d_{1,\mathrm{scale}}} + \beta_1$;
$[\![\boldsymbol{r}_1]\!] \leftarrow \mathsf{PolyEval}(P_{\mathrm{rec},1}, [\![\boldsymbol{v}_1]\!])$;
$[\![\boldsymbol{r}_1]\!] \leftarrow [\![\boldsymbol{r}_1]\!] \odot \frac{4}{d_{1,\mathrm{scale}}}$;
for $\delta \in \mathcal{D}$ **do**
  $[\![\boldsymbol{p}_\delta]\!] \leftarrow [\![\boldsymbol{q}_\delta]\!] \odot [\![\boldsymbol{r}_1]\!]$;
end for
return $\{[\![\boldsymbol{p}_\delta]\!]\}_{\delta\in\mathcal{D}}$.

value; equivalently, $q_\delta(t) = \left(\frac{a_\delta(t)}{a_{\mathrm{sum}}(t)}\right)^2$. Substituting the expressions for $a_\delta(t)$ and $u_\delta(t)$ gives

$$\begin{aligned} q_\delta(t) &= r_0(t)^2 \exp\big(2(u_\delta(t)+\tau)\big) \\ &= r_0(t)^2 \exp\big(-\bar{s}(t)\big)\exp\big(s_\delta(t)\big) \\ &= \kappa_t \exp\big(s_\delta(t)\big), \end{aligned} \tag{18}$$

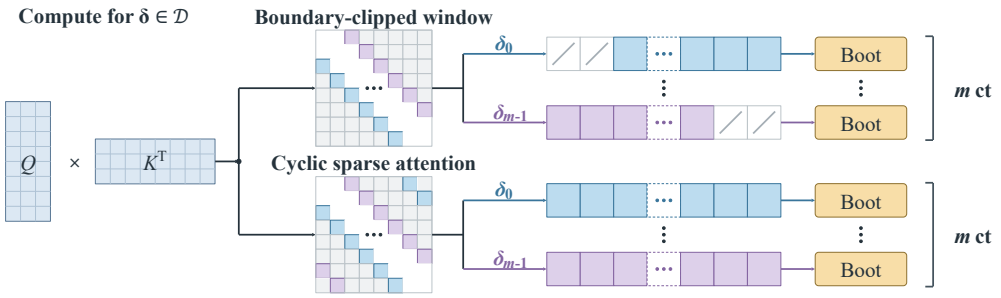


Figure 3: Each selected attention offset maps to one ciphertext under boundary-clipped or cyclic indexing. Cyclic indexing keeps the diagonal complete across query positions. Slashes mark excluded boundary positions, while ellipses indicate omitted positions or offsets.

where $\kappa_t = r_0(t)^2 \exp\big(-\bar{s}(t)\big) > 0$. Squaring doubles the exponent, compensating for the earlier factor $\frac{1}{2}$. Since $r_0(t)$ and $\bar{s}(t)$ are shared by all selected offsets at the fixed query position, κ_t is independent of δ.

To normalize the squared values, the algorithm forms $q_{\mathrm{sum}}(t) = 4\sum_{\delta\in\mathcal{D}} q_\delta(t)$ and bootstraps the ciphertext containing this sum. Under exact arithmetic, bootstrapping preserves its value. The second reciprocal stage maps the sum to $v_1(t) = \alpha_1 \frac{q_{\mathrm{sum}}(t)}{d_{1,\mathrm{scale}}} + \beta_1$. Its reciprocal evaluation and output scaling give

$$r_1(t) = \frac{4\alpha_1}{d_{1,\mathrm{scale}}(v_1(t) - \beta_1)} = \frac{4}{q_{\mathrm{sum}}(t)} = \frac{1}{\sum_{\delta'\in\mathcal{D}} q_{\delta'}(t)}. \tag{19}$$

The final multiplication reuses the retained squared values to compute $p_\delta(t) = q_\delta(t) r_1(t)$. Substituting Equations (19) and (18) and canceling the common positive factor κ_t gives

$$p_\delta(t) = \frac{q_\delta(t)}{\sum_{\delta'\in\mathcal{D}} q_{\delta'}(t)} = \frac{\kappa_t \exp\big(s_\delta(t)\big)}{\kappa_t \sum_{\delta'\in\mathcal{D}} \exp\big(s_{\delta'}(t)\big)} = \frac{\exp\big(s_\delta(t)\big)}{\sum_{\delta'\in\mathcal{D}} \exp\big(s_{\delta'}(t)\big)}. \tag{20}$$

Thus, the exact computation recovers the Softmax probabilities over the selected offsets. Replacing the exponential and reciprocal functions with P_{exp}, $P_{\mathrm{rec},0}$, and $P_{\mathrm{rec},1}$ introduces polynomial approximation error. Homomorphic evaluation additionally incurs CKKS arithmetic error, including error introduced by bootstrapping.

C Cyclic Boundary-Semantics Ablation

We evaluate whether the cyclic boundary semantics used by FHE-friendly cyclic sparse attention affect task utility relative to conventional boundary-clipped sliding-window attention. The comparison uses SST-2 and five paired training seeds. For each seed, the two variants use the same training data, selected 24-offset set, layer-wise FFN rank schedule, training budget, and model-selection procedure; they differ only in their treatment of sequence boundaries.

In the cyclic variant, query position t attends to $(t+\delta) \bmod n$ for every selected offset $\delta \in \mathcal{D}$. In the boundary-clipped variant, a pair is retained only when $0 \le t+\delta < n$, so the wrap-around entries introduced by cyclic rotations do not belong to the model's attention pattern. Figure 3 illustrates these boundary rules and their ciphertext representations.

Table 4: SST-2 development accuracy with exact nonlinearities under cyclic and boundary-clipped sliding-window attention. Differences are cyclic minus boundary-clipped in percentage points; the last row reports the mean and sample standard deviation across paired seeds.

Seed	Cyclic	Boundary-clipped	Difference
42	89.4495	89.3349	$+0.1147$
43	88.9908	89.3349	-0.3440
44	88.8761	88.7615	$+0.1147$
45	90.1376	89.2202	$+0.9174$
46	89.3349	89.1055	$+0.2294$
Mean $\pm$ SD	89.3578 ± 0.4959	89.1514 ± 0.2378	$+0.2064 \pm 0.4544$

As shown in Table 4, the cyclic variant scores higher in four of the five paired runs and achieves a mean accuracy of 89.36%, compared with 89.15% for the boundary-clipped variant. The paired mean difference is $+0.21$ percentage points. We therefore interpret the two boundary conventions as providing comparable task utility with exact nonlinearities in this experiment. The cyclic construction additionally matches the semantics of CKKS rotations, so every selected offset corresponds to a complete score diagonal without boundary-specific masking.

D Model Adaptation and Configuration

This section describes the architecture, initialization, and adaptation settings for the five task-specific models in Table 3 and the separate SST-2 checkpoint used for runtime and encrypted-accuracy evaluation.

D.1 Shared Architecture and Initialization

All five models use a 12-layer BERT-Base encoder with hidden size 768, 12 attention heads of dimension 64, 768-dimensional $Q/K/V$ projections, FFN width 3,072, and maximum input sequence length of 128. Each attention head uses 24 cyclic relative offsets under the mapping $(q+\delta) \bmod 128$.

All five tasks use the same layer-wise FFN rank schedule: $r_l = 256$ in encoder layers 0–2 and 9–11, and $r_l = 512$ in layers 3–8. Both FFN matrices use the same r_l within a layer. The first FFN projection is factorized as $768 \to r_l \to 3{,}072$, and the second as $3{,}072 \to r_l \to 768$. The factors are initialized from rank-r_l truncated SVDs of the corresponding dense FFN matrices, while compatible embeddings, attention parameters, LayerNorm parameters, the pooler, and task heads are copied from the source checkpoint.

For SST-2, the source is the project checkpoint `baseline_sst2.pt`. For QQP, STS-B, CoLA, and RTE, the sources are the Hugging Face checkpoints named `textattack/bert-base-uncased-QQP`, `textattack/bert-base-uncased-STS-B`, `textattack/bert-base-uncased-CoLA`, and `textattack/bert-base-uncased-RTE`, respectively. Each source checkpoint also yields its BERT-Base score in Table 3.

D.2 Attention-Offset Configurations

The SST-2 checkpoint used for both runtime measurements and encrypted-accuracy evaluation uses the fixed 24-offset set $\mathcal{D} = \{-11, -10, \ldots, 12\}$ across all heads and layers. It achieves 90.02% accuracy with exact nonlinearities, matched by CKKS execution on CPU and GPU. For the cross-task utility study, each task configuration uses a fixed offset table of shape $12 \times 12 \times 24$ (layers $\times$ heads $\times$ offsets), containing one 24-offset set for every layer–head pair. SST-2 and RTE use task-specific tables, while QQP, STS-B, and CoLA use the same QQP-derived table. All configurations therefore materialize 24 score diagonals per head, despite differing offset identities.

The SST-2 task-specific offset table is constructed using teacher attention weights from the first 4,096 SST-2 training examples. For each layer and head, we first retain candidate 24-offset sets with the smallest coverage depth, defined as the smallest number of steps required for all 128 positions to become reachable from position 0 using offsets from that set. Among these sets, we select the

one capturing the most teacher attention mass, using locality to break ties. QQP and RTE use fixed coverage-derived tables in their reported training runs, while STS-B and CoLA perform no additional task-specific offset selection.

D.3 Fine-Tuning Configuration

Following prior distillation-based Transformer adaptation (Sun et al., 2019; Park et al., 2025), we fine-tune the five task-specific models with supervised and distillation objectives. The five task-specific models use AdamW-based optimization. Table 5 summarizes the task-specific learning rates, training batch sizes, training budgets, and seeds. For CoLA, the listed seed corresponds to the run with the highest development MCC among three completed runs.

Table 5: Task-specific adaptation settings.

Setting	SST-2	QQP	STS-B	CoLA	RTE
Learning rate(s)	2×10^{-5}	1×10^{-6}	P: $5 \times 10^{-7}/3 \times 10^{-6}$ R: $4 \times 10^{-7}/2 \times 10^{-6}$	$3 \times 10^{-6}/10^{-5}$	$2 \times 10^{-7}/10^{-6}/2 \times 10^{-6}$
Batch size	32	16	16	32	8
Training budget	2 epochs	3,500 steps	1,800 P + 300 R steps	10 epochs	200 steps
Seed(s)	20260903	20260903	20260884	20260831	20261911

For STS-B, P and R denote the pseudo and restore stages, respectively. These stages correspond to approximately 1.072 passes over the pseudo-pair data and 0.835 passes over the official training set, respectively; we therefore report their optimizer steps separately. The two learning rates in each stage apply to the encoder and head/pooler. The training batch size is 16 and the evaluation batch size is 32. For CoLA, the two learning rates apply to the student backbone and task head. For RTE, the three learning rates apply to the encoder, pooler, and task head.

Training objectives. SST-2 combines cross-entropy, logit distillation, and hidden-state MSE with weights 1.0, 1.0, and 0.05, respectively, using temperature 2. QQP uses staged cross-entropy, logit distillation, hidden-state, and symmetry losses with temperature 2. STS-B uses teacher-pair and self-distillation together with distillation from an additional DeBERTa teacher. CoLA combines hard-label, DeBERTa-teacher, and BERT-teacher losses with weights 0.35, 0.45, and 0.20. RTE uses staged cross-entropy, logit distillation, and hidden-state losses with temperature 2.

Result reporting. SST-2 reports the checkpoint with the highest development accuracy from its fixed two-epoch run. QQP reports the fixed stochastic weight averaging (SWA) checkpoint at step 3,500. STS-B reports the completed run selected by the mean of development Pearson and Spearman correlations. CoLA reports the best of three completed seeds by development MCC. RTE reports the fixed SWA checkpoint from the best of four prespecified training routes on the development set. No GLUE test split is used. Table 3 compares the resulting models under their respective training procedures; it does not isolate the effect of architectural changes under matched training.

Runtime and encrypted-accuracy checkpoint. The adapted SST-2 checkpoint (90.02% accuracy with exact nonlinearities) used for runtime measurements and encrypted-accuracy evaluation uses the same initialization procedure and development-accuracy selection criterion as the task-specific SST-2 model, with the fixed attention offsets described in Appendix D.2. Its fine-tuning configuration uses a learning rate of 4×10^{-8}, batch size 16, 350 steps, and random seed 20260706.

With the encoder frozen, a final L-BFGS refit optimizes only the classifier weights and bias on the 67,349 SST-2 training examples, using weighted cross-entropy and L2 regularization (10^{-4}). Training examples misclassified by the starting model or in the lowest 20% of absolute margins receive weight 8. The final checkpoint is selected by development accuracy after refitting.

E PLAINTEXT SOFTMAX ROBUSTNESS

To evaluate Softmax separately, we compare the mean-centered circuit in Appendix B.5 with the MOAI Softmax circuit in plaintext, keeping the checkpoint and all other operations unchanged. This comparison isolates the Softmax circuits on the same adapted model; it does not measure MOAI's dense-model task accuracy. Each Softmax row contains 24 scores for one input, one attention head, and one query position. Both circuits use the same 1,000-example SST-2 fitting set and the same five separately sampled 1,000-example evaluation subsets from the remaining training examples. Table 6 reports means over these subsets.

Rates are computed over non-padding query rows across all 12 layers. This exclusion applies only to metric aggregation; padded positions remain part of the attention computation. Non-finite rows contain a NaN or infinite intermediate or output. The out-of-domain (OOD) rate counts these rows and any row with an exponential or reciprocal input outside its evaluation interval. Row error is the maximum absolute error across the 24 probabilities in a row, relative to exact Softmax. Before computing this maximum, NaN or infinite probability errors are replaced by $+\infty$; all non-padding query rows remain in the percentile population. For each subset, P99 row error is the maximum of the 12 per-layer 99th percentiles of these row errors. The reported value is then averaged over the five subsets.

Loss is the SST-2 accuracy loss relative to exact Softmax, in percentage points (pp). Approximate Softmax outputs propagate through the remaining model without sanitization or exact-Softmax fallback. Samples with finite final logits use the usual argmax prediction; any sample with a non-finite final logit is counted as incorrect and retained in the accuracy denominator.

Table 6: Plaintext comparison of the MOAI and mean-centered Softmax circuits, reporting means over five SST-2 evaluation subsets. Both circuits use the same 1,000-example fitting set.

Softmax circuit	OOD (%)	Non-finite (%)	P99 error	Loss (pp)
MOAI	4.44×10^{-4}	4.44×10^{-4}	9.34×10^{-1}	5.08
BLADE	6.89×10^{-4}	1.24×10^{-4}	6.28×10^{-4}	0.30

The mean-centered circuit reduces accuracy loss from 5.08 to 0.30 percentage points, P99 row error by about $1{,}487\times$, and the non-finite rate by $3.58\times$. Despite its higher OOD rate, it achieves lower P99 row error and accuracy loss under this evaluation protocol; non-finite rows are reduced but not eliminated.

F APPROXIMATION SETTINGS AND EXECUTION SCHEDULE

F.1 NONLINEAR APPROXIMATION SETTINGS

The nonlinear approximation profile used for the CPU and GPU runtime measurements and the encrypted-accuracy evaluation of the SST-2 checkpoint is summarized in Tables 7, 8, and 9. Following common reporting practice in FHE Transformer evaluation (Pang et al., 2024; Moon et al., 2025; Park et al., 2025; Zhang et al., 2025a; 2026), we specify the nonlinear approximation profile as part of the fixed encrypted-execution configuration and report its polynomial degrees and operating intervals. Layers are indexed by $l = 0, \ldots, 11$. In Softmax, $P_{\exp}$ is a degree-15 power polynomial, $P_{\mathrm{rec},0}$ is a degree-15 normalized-domain power polynomial, and $P_{\mathrm{rec},1}$ is a degree-43 Chebyshev-U polynomial. These polynomial degrees and the public shift $\tau = 0.7677989$ are shared across layers, while the approximation intervals are layer-specific. The E intervals in Table 7 apply to u_δ, while the R_0 and R_1 intervals apply to a_{scaled} and $\frac{q_{\mathrm{sum}}}{d_{1,\mathrm{scale}}}$, respectively, before their affine maps.

The R_0 and R_1 intervals determine the public affine maps used by the two reciprocal stages in Appendix B.5. Their input scales are fixed across layers: $d_{0,\mathrm{scale}} = 139.34071508787872$ and $d_{1,\mathrm{scale}} = 0.8081083327819361$. For stage $j \in \{0, 1\}$ and layer l, let $[L_{j,l}, U_{j,l}]$ denote the corresponding full-precision interval before the affine map. The map to $[-1, 1]$ uses $\alpha_{j,l} = \frac{2}{U_{j,l} - L_{j,l}}$ and $\beta_{j,l} = -\frac{U_{j,l} + L_{j,l}}{U_{j,l} - L_{j,l}}$. Table 8 reports the resulting layer-specific affine parameters, rounded

Table 7: Layer-specific Softmax approximation intervals for the SST-2 checkpoint used in runtime and encrypted-accuracy evaluations. E, R_0, and R_1 denote the exponential, first reciprocal, and second reciprocal stages.

Layer	E	R_0	R_1
0	$[-6.34648, 6.82727]$	$[0.167072, 0.423916]$	$[0.200056, 1.45290]$
1	$[-7.18187, 9.05219]$	$[0.167072, 0.806282]$	$[0.200056, 2.85923]$
2	$[-15.1537, 17.0514]$	$[0.167072, 24.0885]$	$[0.176104, 9.62198]$
3	$[-6.35573, 7.68257]$	$[0.167072, 0.451956]$	$[0.200056, 2.02810]$
4	$[-5.90734, 8.66311]$	$[0.167072, 0.629098]$	$[0.200056, 2.72039]$
5	$[-8.69663, 9.55837]$	$[0.167072, 1.07596]$	$[0.200056, 3.33209]$
6	$[-7.18986, 9.95217]$	$[0.167072, 1.44816]$	$[0.200059, 3.01927]$
7	$[-7.95243, 11.0354]$	$[0.167072, 2.51071]$	$[0.200063, 3.30254]$
8	$[-8.88886, 12.1067]$	$[0.167072, 4.84106]$	$[0.199638, 3.47814]$
9	$[-11.5836, 12.6554]$	$[0.167072, 3.52358]$	$[0.199983, 4.05068]$
10	$[-15.5832, 14.2630]$	$[0.167072, 9.70295]$	$[0.195865, 5.32616]$
11	$[-10.6360, 9.54147]$	$[0.167072, 1.31802]$	$[0.200058, 3.20049]$

Table 8: Layer-specific public affine parameters for the two Softmax reciprocal stages of the SST-2 checkpoint used in runtime and encrypted-accuracy evaluations. Values are rounded for readability.

Layer	First reciprocal		Second reciprocal	
	α_0	β_0	α_1	β_1
0	7.78685	-2.30097	1.59637	-1.31936
1	3.12886	-1.52275	0.752113	-1.15046
2	0.0836072	-1.01397	0.211733	-1.03729
3	7.02041	-2.17292	1.09407	-1.21887
4	4.32876	-1.72322	0.793545	-1.15875
5	2.20048	-1.36764	0.638563	-1.12775
6	1.56117	-1.26083	0.709418	-1.14193
7	0.853374	-1.14258	0.644645	-1.12897
8	0.427901	-1.07149	0.610034	-1.12179
9	0.595858	-1.09955	0.519386	-1.10387
10	0.209734	-1.03504	0.389841	-1.07636
11	1.73770	-1.29032	0.666570	-1.13335

for readability. In the timing comparison, MOAI's Softmax routine uses layer-specific shifts $(7.5, 9.9, 13.6, 13.3, 9.5, 8, 10.3, 9, 9, 9, 11, 7)$ for layers 0–11, respectively.

GELU and the LayerNorm seeds are evaluated directly in the Chebyshev-T basis. Each LayerNorm seed is followed by Newton refinement, without Goldschmidt iterations. Table 9 reports separate settings for the first and second LayerNorm invocations (LN1 and LN2). The LayerNorm intervals apply to $v = \frac{1}{d}\sum_i (x_i - \mu)^2 + \epsilon$, using population variance and $\epsilon = 10^{-12}$. Starting from seed degree p_0, the effective algebraic degree obeys $p_{t+1} = 3p_t + 1$ after each Newton iteration. The implementation evaluates the seed and refinement steps, rather than expanding the resulting high-degree polynomial.

F.2 Depth, Packing, and Bootstrapping Schedule

Table 10 summarizes the per-layer depth, packing, and bootstrapping schedule. In the table, output ciphertext counts refer to the packed batch, whereas data dimensions and operation counts are reported per input. Ct and pt denote ciphertext and plaintext, and Col and Diag denote the interleaved column and diagonal packings defined in Appendix B.1. Depth denotes the available CKKS multiplicative depth at operation boundaries. Transitions include modulus switching to retain only the required levels or meet our implementation's level-0 bootstrap input requirement. The schedule and scale handling apply to both CPU and GPU, with base scale $\Delta = 2^{46}$.

Table 9: Layer-specific GELU and LayerNorm approximation settings for the SST-2 checkpoint used in runtime and encrypted-accuracy evaluations. Degree is the GELU polynomial degree or LayerNorm seed degree; Newton is the refinement count, and Composite degree is the algebraic degree in the variance input v after Newton refinement.

Layer	Function	Degree	Newton	Composite degree	Interval
0	GELU	63	–	–	$[-19.8168, 12.9190]$
	LN1	13	4	1093	$[0.00140677, 0.554631]$
	LN2	3	3	94	$[0.919498, 13.4756]$
1	GELU	63	–	–	$[-17.2070, 7.75026]$
	LN1	3	2	31	$[0.186745, 0.694223]$
	LN2	3	3	94	$[0.913990, 7.46921]$
2	GELU	96	–	–	$[-26.7746, 55.7780]$
	LN1	3	2	31	$[0.189689, 0.890233]$
	LN2	5	4	445	$[0.764397, 81.4274]$
3	GELU	40	–	–	$[-21.6417, 11.6283]$
	LN1	3	2	31	$[0.188610, 0.809727]$
	LN2	5	4	445	$[0.823616, 65.7383]$
4	GELU	48	–	–	$[-20.3127, 13.9020]$
	LN1	3	2	31	$[0.156050, 0.829358]$
	LN2	5	4	445	$[0.766163, 49.7424]$
5	GELU	31	–	–	$[-15.2437, 8.24731]$
	LN1	3	3	94	$[0.139072, 0.857723]$
	LN2	3	4	283	$[0.800091, 25.7525]$
6	GELU	28	–	–	$[-14.0929, 7.43662]$
	LN1	3	2	31	$[0.189549, 0.916642]$
	LN2	3	3	94	$[0.744875, 10.5836]$
7	GELU	28	–	–	$[-15.2861, 9.85804]$
	LN1	3	2	31	$[0.164272, 1.01818]$
	LN2	3	3	94	$[0.718165, 9.89366]$
8	GELU	63	–	–	$[-22.0023, 9.54843]$
	LN1	3	3	94	$[0.132338, 1.37848]$
	LN2	3	3	94	$[0.604123, 10.2737]$
9	GELU	80	–	–	$[-43.3167, 32.5983]$
	LN1	3	3	94	$[0.121843, 2.04524]$
	LN2	11	4	931	$[0.573350, 255.777]$
10	GELU	120	–	–	$[-35.1154, 95.4793]$
	LN1	3	4	283	$[0.0529792, 1.96089]$
	LN2	15	4	1255	$[0.494798, 742.352]$
11	GELU	28	–	–	$[-12.9186, 7.00917]$
	LN1	5	4	445	$[0.0570383, 3.11271]$
	LN2	3	2	31	$[0.801256, 2.02111]$

The projections in the first two rows share the same column-packed layer input. Each encoder layer begins at level 8. The first row evaluates 12 V projections, producing 768 ciphertexts at level 7, and includes their subsequent alignment to level 6 for value aggregation. The second row first modulus-switches the input to level 3, then evaluates the Q and K projections at level 2, producing 768 ciphertexts for each. Score computation consumes one further level, followed by alignment to level 0 for bootstrapping.

The Softmax and LayerNorm rows summarize composite operations, including their internal statistic and variance bootstraps. Their output ciphertext counts therefore differ from their bootstrap input counts. Each retains one branch while refreshing a compact statistic on the other; the refreshed branch is subsequently modulus-switched down to the retained branch's level for their final multiplication.

Softmax and value aggregation. Softmax retains the squared normalized values at level 6 and scale Δ while bootstrapping their packed sum. After reciprocal evaluation, the refreshed branch is modulus-switched to level 6 at scale Δ. Their final multiplication is not immediately rescaled, leaving the Softmax output at level 6 and scale Δ^2. The value ciphertexts are aligned to level 6 at scale Δ. Value aggregation accumulates the 24 offset products at level 6 and scale Δ^3, then performs one relinearization and two rescales per output ciphertext, yielding level 4 and scale Δ. Table 10 records the first rescale in the value-aggregation row ($6 \to 5$) and the second at the start of the W_O row ($5 \to 4$). The W_O projection then consumes one further level, producing the level-3, scale-Δ input to the first LayerNorm.

LayerNorm and FFN. The implementation retains scaled centered values $\widetilde{z}_i = dx_i - \sum_j x_j = d(x_i - \mu)$ and compensates with the factor $\frac{\gamma_i}{d}$ in the normalization branch. The retained ciphertexts have scale Δ and remain at level 3 for the first LayerNorm and level 2 for the second. The variance branch bootstraps its single packed ciphertext, evaluates the inverse square root using a Chebyshev seed followed by Newton refinement, and multiplies by $\frac{\gamma_i}{d}$. This branch is aligned to the retained branch's level and scale Δ. The final ciphertext multiplication, relinearization, and rescaling yield level 2 for the first LayerNorm and level 1 for the second, both at scale Δ, before adding β_i. The retained branches are never bootstrapped, and alignment requires no modulus raising.

After the first LayerNorm, projection to the rank-r_l FFN intermediate consumes one level, followed by alignment to level 0 and bootstrapping to level 20. Before expansion, the refreshed ciphertexts are modulus-switched to D_l, the layer-specific input level for FFN expansion in Table 10: $D_l = 13$ for $l \in \{0, 1, 3, 4, 8\}$, $D_l = 12$ for $l \in \{5, 6, 7, 11\}$, and $D_l = 14$ for $l \in \{2, 9, 10\}$. The chosen D_l values accommodate the layer-specific GELU approximations listed in Table 9, leaving level 4 after GELU. The two output factors subsequently reduce the level from 4 to 2.

After the second LayerNorm, the full-width output is aligned from level 1 to level 0 and bootstrapped to level 20. The final table row shows the alignment to level 8 used between encoder layers after the layer-output bootstrap. The timed evaluation includes the output bootstrap after the final encoder layer.

For the complete encoder, the FFN ranks above sum to $R = \sum_{l=0}^{L-1} r_l = 4{,}608$. The expressions in Section 3.4 then give $C_{\text{boot}}^{\text{BLADE}} = 17{,}448$ and $C_{\text{boot}}^{\text{MOAI}} = 37{,}008$, corresponding to a $2.12\times$ reduction.

The totals above combine bootstrap points of different sizes: per layer, Softmax refreshes 12 statistic ciphertexts and each LayerNorm one variance ciphertext, versus 288 score ciphertexts and r_l FFN ciphertexts. These smaller groups expose less ciphertext-level parallelism on the 32-thread CPU, consistent with the higher per-ciphertext times in Table 1(c).

Table 10: Per-layer data dimensions, CKKS depth, packing, and output ciphertext counts for our encoder adapted from BERT-Base. D_l denotes the layer-specific CKKS level at the input to FFN expansion.

Component	Operation	Depth	Packing	(Data dimensions) × (Number of operations)	Number of ciphertexts
Attention	Ct-pt MatrixMul	$8 \to 7 \to 6$	Col → Col	$(\mathbb{R}^{128\times768} \times \mathbb{R}^{768\times64}) \times 12$	768
	Ct-pt MatrixMul	$8 \to 3 \to 2$	Col → Col	$(\mathbb{R}^{128\times768} \times \mathbb{R}^{768\times64}) \times 24$	$768 + 768$
	Ct-ct MatrixMul	$2 \to 1 \to 0$	Col → Diag	$(\mathbb{R}^{128\times64} \times \mathbb{R}^{64\times128}) \times 12$	288
	Bootstrapping	$0 \to 20$	Diag → Diag	$\mathbb{R}^{128\times24} \times 12$	288
	Softmax	$20 \to 6$	Diag → Diag	$\mathbb{R}^{128\times24} \times 12$	288
	Ct-ct MatrixMul	$6 \to 5$	Diag → Col	$(\mathbb{R}^{128\times128} \times \mathbb{R}^{128\times64}) \times 12$	768
Self Output	Ct-pt MatrixMul	$5 \to 4 \to 3$	Col → Col	$(\mathbb{R}^{128\times768} \times \mathbb{R}^{768\times768})$	768
	LayerNorm	$3 \to 2$	Col → Col	$\mathbb{R}^{128\times768}$	768
FFN	Ct-pt MatrixMul	$2 \to 1 \to 0$	Col → Col	$(\mathbb{R}^{128\times768} \times \mathbb{R}^{768\times r_l})$	r_l
	Bootstrapping	$0 \to 20 \to D_l$	Col → Col	$\mathbb{R}^{128\times r_l}$	r_l
	Ct-pt MatrixMul	$D_l \to D_l - 1$	Col → Col	$(\mathbb{R}^{128\times r_l} \times \mathbb{R}^{r_l\times3072})$	3,072
	GELU	$D_l - 1 \to 4$	Col → Col	$\mathbb{R}^{128\times3072}$	3,072
	Ct-pt MatrixMul	$4 \to 3$	Col → Col	$(\mathbb{R}^{128\times3072} \times \mathbb{R}^{3072\times r_l})$	r_l
	Ct-pt MatrixMul	$3 \to 2$	Col → Col	$(\mathbb{R}^{128\times r_l} \times \mathbb{R}^{r_l\times768})$	768
Final	LayerNorm	$2 \to 1$	Col → Col	$\mathbb{R}^{128\times768}$	768
	Bootstrapping	$1 \to 0 \to 20 \to 8$	Col → Col	$\mathbb{R}^{128\times768}$	768