

THEMIS: A Co-Designed System for Encrypted Transformer Inference

Shihao Li¹, Wenhao Wang¹, Lu Li¹, Xiaomei Tang¹,
Jian Liu¹, Rongmao Chen¹, Cheng Hong², and Guangfu Sun¹

¹National University of Defense Technology, ²Ant Group

Abstract

Secure Transformer inference can be made non-interactive under fully homomorphic encryption (FHE), but the computation is extremely expensive. Existing FHE-based systems reduce that cost by optimizing individual stages in isolation, each within its own component rather than against the CKKS level budget they all share. A local gain therefore may not translate into an end-to-end speedup.

We present THEMIS, a co-designed system for encrypted Transformer inference organized around two design principles, one bounding the depth each kind of stage may spend and one fixing where on the modulus chain it runs. THEMIS generalizes coefficient-encoded plaintext-ciphertext matrix multiplication (PCMM) to slot-encoded ciphertexts without format conversion, retaining the same accelerated coefficient-side backend and amortizing its nearly fixed cost through multi-batch packing. Two attention ciphertext-ciphertext matrix multiplication (CCMM) kernels combine the idle imaginary lane of each slot with baby-step giant-step accumulation and hoisting, so the attention multiplications issue far fewer key switches. For softmax, THEMIS conditions the first denominator with a calibrated row-wise offset and reserves high precision for the final normalization alone, which cuts the depth the reciprocal path consumes while keeping the precision of the returned probabilities. For a single stage, THEMIS accelerates PCMM and the attention CCMMs by $21.66\times$ and up to $7.39\times$ over MOAI (ICLR'26), and by up to $23.02\times$ and $16.49\times$ over Euston (S&P'26), while THEMIS's softmax runs $2.57\times$ faster than THOR (CCS'25) at higher precision. For the complete pipeline, THEMIS serves BERT-base on a CPU in 400.06 amortized seconds per input, with accuracy on GLUE tasks comparable to plaintext. Of that latency, the stages around bootstrapping account for only 10.83%, which is the lowest share among all works compared here.

1 Introduction

Cloud-hosted inference over Transformer models [38] is now a routine service, but it hands prompts, documents, and task-

specific queries to the server, and those inputs may carry medical, financial, legal, or enterprise-confidential material that the party running the model was never meant to see. Secure Transformer inference instead asks the server to evaluate the model while learning nothing about the input beyond what follows from the final output. Two lines of work provide that guarantee, and they differ in how much the client must stay involved. Interactive protocols split the computation between client and server with secure multi-party computation (MPC), which keeps local cryptographic work light but holds the client online and pays communication on every inference [15, 17, 22, 26, 27, 29, 30, 33, 40]. Fully homomorphic encryption (FHE) needs only a single round, in which the user encrypts once and receives one encrypted output. FHE therefore keeps communication cost low while placing a heavy computational burden on the server, and a growing body of work is aimed at reducing it.

Encrypted Transformer inference is most commonly built on RNS-CKKS [9], the residue-number-system variant of CKKS [10], which we write as CKKS throughout. It works approximately over packed vectors, which suits the dense linear algebra a Transformer layer is made of. That layer splits into two kinds of stage, the matrix stages that carry the projections and the two attention multiplications, and the nonlinear stages that carry softmax, LayerNorm, and GELU; a kernel is the algorithm that executes one stage. Non-interactive systems have already turned this structure into working pipelines. NEXUS [41] served a full Transformer under FHE alone and showed that the route is feasible at all, and THOR [32], Euston [13], and MOAI [42] have since sharpened individual components of it. Latency nonetheless remains measured in hundreds of seconds per input.

Motivation: Despite that progress, the focus has remained on optimizing each component independently of the others, and four problems remain open. First, every stage draws on one shared budget of multiplicative levels, and what a stage costs depends on where in that budget it runs, since a ciphertext higher up carries more moduli and every operation on it costs proportionally more. That budget is nonetheless

accounted for inside each component rather than across the pipeline, so a gain won inside one component can be spent again at its boundary, and where a stage runs on the chain has never been treated as a design question, with prior pipelines leaving matrix multiplication high on the chain [42]. Bootstrapping (BTS) makes the cost of that accounting visible. More than a fifth of end-to-end latency is spent outside BTS, on exactly the stages a schedule can move, whereas the per-call cost of BTS itself is fixed by the parameter set and treated as a black box everywhere. What a design can still move is therefore how many calls a layer needs and what the stages between them cost, and both are decided by the schedule, not inside any one component. Second, the two matrix multiplications are each fast only under conditions the pipeline does not meet. The fast route for PCMM leaves the CKKS operator layer and computes on the ciphertext coefficients, but it expects coefficient-encoded inputs one sequence at a time, so a slot-encoded pipeline pays format conversions that cancel what the fast core saves, and since one input is far shorter than the CKKS slot count, most of the ciphertext is left empty. In the two attention CCMMs, rotations and their key switches dominate, yet the imaginary half of each slot sits idle and identical key-switch work repeats across the rotations of one source. Third, softmax forces a choice between cost and coverage. Keeping high precision at every normalization stage preserves accuracy but consumes many levels and forces extra BTS calls [32], while a fixed offset buys cheaper reciprocals [13, 41, 42] at the price of a constant that does not move with the row it conditions, so whether the first denominator lands where a low-degree reciprocal is accurate is settled by the input rather than by the design. Fourth, there is no common ground on which to compare any of this, since reported kernel results come from different libraries on different hardware, some no longer under active development. Room therefore remains for a system designed around the schedule rather than any one of its stages, and able to serve an existing trained model without retraining or distillation.

1.1 Contributions

We present THEMIS, a co-designed system for encrypted Transformer inference that solves those four problems together. Fixing the schedule first is what makes the kernels hard, since a fast plaintext-ciphertext multiplication is easy once it may change the encoding and a cheap softmax is easy once it may lose precision. Satisfying that same schedule is what makes their gains compose, since every kernel then returns its output at the level the next stage expects and no gain is spent at the boundary.

Our contributions are summarized as follows:

- **Level-budget co-design.** We take the shared level budget as the primary design constraint and derive two design principles from it, one bounding the depth each kind of

stage may spend, which governs how a kernel is built, and one fixing where on the modulus chain a stage runs, which governs where that kernel is placed. Because they change where a kernel executes rather than what it computes, the schedule contributes a gain separate from any kernel’s own speedup. THEMIS then needs three bootstrapping calls per encoder layer where NEXUS, Euston, and MOAI each need four, and its matrix and nonlinear stages together take 10.83% of end-to-end latency, the smallest share among the systems compared here.

- **Slot-encoded PCMM without format conversion.** We generalize PCMM in coefficient-encoded format to a slot-encoded form that accepts multi-batch ciphertexts and still dispatches to the same accelerated backend, without introducing an S2C or a C2S call. It accelerates PCMM by $21.66\times$ over MOAI and by up to $23.02\times$, $63.32\times$, and $98.22\times$ over Euston, THOR, and Powerformer.
- **Two attention CCMM kernels.** We propose two faster attention kernels, COLPAIR and DIAGPAIR, which pair the two CKKS lanes along the index each multiplication sums over, so that baby-step giant-step (BSGS) accumulation and hoisting reuse rotations the kernel already issues instead of repeating them. For $QK^\top (S \cdot V)$, THEMIS reaches $7.39\times$ ($2.18\times$) over MOAI and up to $16.49\times$ ($13.21\times$), $84.87\times$ ($22.65\times$), and $166.40\times$ ($99.01\times$) over Euston, THOR, and Powerformer.
- **Denominator-conditioned softmax.** We propose a faster softmax that also improves the precision of the returned probabilities. It preconditions the first denominator with a row-wise offset obtained by a homomorphic-friendly computation, with two scalars calibrated offline, which narrows the interval the early reciprocals must cover so that low-degree polynomials suffice for them. High precision is then reserved for the final normalization alone, where a low-degree polynomial supplies the initial estimate that aSOR [31] refines, so the accuracy that decides the returned probabilities is bought once rather than at every stage. Against THOR, inverse depth falls from 43 to 21, softmax-path BTS stages from 3 to 1, softmax latency from 389.38 to 151.59 seconds, and reported precision rises from 8.84 to 13.25 bits.
- **End-to-end system and unified benchmark.** We implement THEMIS in OpenFHE [1] and serve BERT-base end to end at 400.06 amortized seconds per input, at GLUE accuracy comparable to plaintext. We further reimplement the PCMM, CCMM, and softmax kernels of Powerformer, THOR, Euston, and MOAI in the same environment under matched conditions, giving later work a common baseline that is not read across libraries or hardware.

1.2 Related work

Secure Transformer inference. Interactive designs split the computation between client and server. Two-party, multi-party, function-secret-sharing, and hybrid protocols that combine homomorphic encryption (HE) with MPC keep local cryptographic work light; Iron [17], MPCFormer [26], BOLT [33], SecFormer [30], Nimbus [27], BumbleBee [29], and SHAFT [22] instantiate these protocol classes, SIGMA [15] extends function secret sharing to GPT inference, and BLB [40] lowers the CKKS–MPC conversion cost. All of them, however, keep the client online and pay in communication for every inference.

Non-interactive systems keep one complete CKKS pipeline on the server. NEXUS [41] first showed that a full Transformer can be served this way with no interaction at all. THOR [32] builds a complete BERT pipeline around a softmax that holds high precision at every normalization stage. Euston [13] targets user-friendly serving and removes the rotations from its projection layers. MOAI [42] reorganizes the pipeline into optimized modules, adopts an interleaved multi-batch layout, and brings BSGS to the attention multiplications. STIP [39] and Powerformer [34] instead change the model, each replacing softmax with a function the encrypted pipeline evaluates more cheaply.

Matrix multiplication. Matrix multiplication under CKKS follows three lines. Diagonal and rotation-based methods encode linear maps in slot diagonals and cut rotations through BSGS and successor schedules, yet rotations and their key switches stay dominant [16, 18, 20, 21, 23, 36]. The Liu-Zhang reduction instead maps PCMM to plaintext modular matrix products [28], which later work develops into exact coefficient-side kernels [2, 3]; this line assumes coefficient-encoded inputs with dimensions near the ring degree. Large-matrix CCMM constructions carry the same requirement, reaching their speedup only when the operand dimensions approach the ring degree [3, 35]. Transformer serving does not meet it, since the sequence length stays far below the ring degree of a deployed parameter set.

Softmax approximation. For the nonlinearities, Cho et al. [11] stabilize softmax over wide input intervals with a normalize-and-square iteration. THOR [32] builds on that route with a square-and-normalize schedule and evaluates each denominator with a high-precision reciprocal, accelerated by the adaptive successive over-relaxation (aSOR) iteration of Moon et al., which converges quickly only when the initial estimate is already close [31]. Powerformer [34] instead replaces softmax through distillation and retraining.

2 Preliminaries

In this section, we first specify the Transformer computation targeted by THEMIS and fix the main notation. We then

summarize the CKKS primitives, packing layout, and basic linear algebra subprograms (BLAS)-style PCMM background needed later. Notation is collected in Table 6 in Appendix A.

2.1 Transformer Architecture

Let $A \in \mathbb{R}^{L \times d}$ denote the input activation matrix, where L is the effective sequence length, d is the hidden dimension, and H and $d_h = d/H$ are the number of heads and the per-head dimension. For one attention head, the query, key, and value matrices are obtained by linear projection:

$$Q = AW_Q, \quad K = AW_K, \quad V = AW_V, \quad (1)$$

and the attention output is

$$S = \text{Softmax}\left(\frac{QK^\top}{\sqrt{d_h}}\right), \quad O = SV, \quad (2)$$

followed by an output projection $O_{\text{out}} = OW_O$, a residual, and LayerNorm. The feed-forward network (FFN) applies two linear projections separated by GELU activation:

$$\text{FFN}(x) = \text{GELU}(xW_{F1} + b_1)W_{F2} + b_2. \quad (3)$$

A single Transformer layer thus contains six PCMM stages ($W_Q, W_K, W_V, W_O, W_{F1}, W_{F2}$) and two CCMMs, namely $QK^\top$ and $S \cdot V$.

2.2 The CKKS scheme

CKKS is a fully homomorphic encryption scheme for approximate arithmetic over vectors of complex numbers. A plaintext message $\mathbf{m} \in \mathbb{C}^{N/2}$ is encoded into a polynomial in $R = \mathbb{Z}[X]/(X^N + 1)$, where N is the ring dimension. We call this packed representation the *slot-encode format*. The *coefficient-encode format* instead places data directly in polynomial coefficients. After encoding, the polynomial is encrypted into a ciphertext $\text{ct} = (b, a) \in R_q^2$ under a secret key $s \in R$, such that $b + a \cdot s \approx m \cdot \Delta + e$ for a scaling factor Δ and a small noise term e .

Deployments carry each ciphertext coefficient in RNS, as residues modulo a chain of machine-word-sized primes, so every homomorphic operation runs as word-sized arithmetic on one residue at a time. The number of active moduli, one more than the level, therefore fixes how many such passes an operation costs, and a ciphertext high on the chain carries more residues and runs proportionally slower than the same ciphertext near the bottom. Grafting [7] treats this modulus structure in detail. Even a single tower carries degree- N arithmetic, so the chain has no broad flat region at its bottom.

Primitives. We write $\boxplus$ and $\boxtimes$ for element-wise addition and multiplication, applied independently to each slot. A multiplication followed by rescaling consumes one *multiplicative level*, and ciphertext–ciphertext multiplication additionally

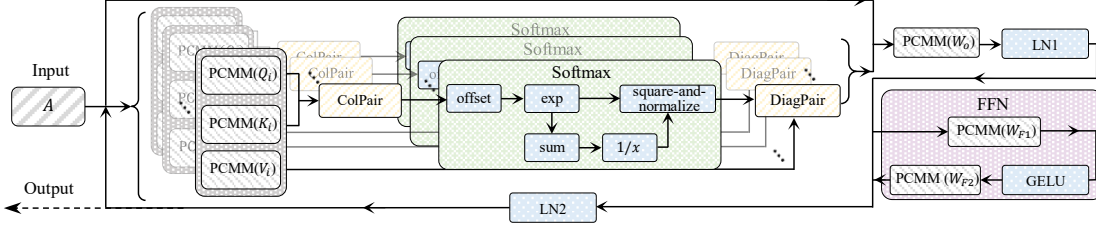


Figure 1: Overview of the THEMIS workflow. One encoder layer runs from the input activation through the Q , K , and V projections, the COLPAIR attention multiplication, the denominator-conditioned softmax, the DIAGPAIR attention multiplication, the output projection, and the feed-forward block, with LN1 and LN2 in their standard positions.

requires key-switching, while plaintext–ciphertext multiplication does not. Slot permutations and conjugation are ring automorphisms followed by key-switching. A rotation $\text{Rot}(\text{ct}, k)$ cyclically shifts all $N/2$ slots by k positions, negative k giving a right shift, and conjugation $\text{Conj}(\text{ct})$ maps each slot m_j to $\bar{m}_j$ at a cost comparable to a ciphertext–ciphertext multiplication. Hoisting evaluates several rotations of one ciphertext from a single shared key-switch precomputation, which makes each rotation cheaper but pays only when that ciphertext is rotated more than once [16]. We write $\text{HOISTROT}(\text{ct}, \mathcal{K})$ for the hoisted evaluation of $\{\text{Rot}(\text{ct}, k)\}_{k \in \mathcal{K}}$ as one operation, and an unqualified Rot denotes a single ordinary rotation. *CoeffsToSlots* (C2S) and *SlotsToCoeffs* (S2C) move between the two encoding formats and reappear when we relate coefficient-encode BLAS computation to slot-encode execution. When the multiplicative budget is exhausted, BTS restores it [8].

2.3 Interleaved packing

Interleaved slot packing [42] processes multiple input samples in parallel. For a batch of B samples with sequence length L , the slot at position $\text{row} \cdot B + b$ stores the entry of row row and batch index b . A rotation by $\delta \cdot B$ positions then shifts all batch lanes by δ rows without cross-batch interference. Figure 3(a) shows a toy example.

2.4 PCMM in Coefficient-Encode Format

A CKKS ciphertext is a pair of ring elements, so a batch of ciphertexts hands the server two integer coefficient matrices. Applying a public weight matrix is a linear map on those coefficients, and Liu and Zhang first observed that the plaintext–ciphertext multiplication can therefore be evaluated outside the homomorphic operator layer, as ordinary modular integer matrix arithmetic over server-visible data [28]. We call this the ciphertext-to-plaintext matrix-multiplication reduction (C2PMM). Throughout, coefficient-encode and slot-encode name where the data is encoded, whereas coefficient-side names arithmetic carried out on a ciphertext’s polynomial coefficients; the two are independent, and Section 4 exploits that independence. Bae et al. turn it into an exact bit-chunked al-

gorithm and connect it to dense linear-algebra backends [2, 3]. That line targets coefficient-encoded ciphertexts at or above the ring degree, and smaller dimensions first pass through modulus decomposition and packing [4, 6]. The reduction and its chunked core are therefore prior art. Encrypted Transformer serving instead runs slot-encoded operators at $L \ll N$, a setting this line does not address.

3 THEMIS Architecture

This section presents the THEMIS pipeline and the schedule it is built around. Section 3.1 states the threat model, Section 3.2 walks the encrypted state through one layer, and Section 3.3 derives the two design principles from a measured level-cost model.

3.1 Threat Model and Security

THEMIS operates in a two-party semi-honest (honest-but-curious) setting. A *client* holds private input data and the secret key, encodes the input into ciphertexts, and sends them to the *server*, which holds the Transformer model weights in plaintext. The server runs the inference homomorphically on the encrypted input and returns the encrypted result. The client decrypts the result locally. Neither party deviates from the protocol, but each may attempt to learn additional information from the messages it receives.

Under this model, the server never observes the plaintext input or any intermediate activation and only sees ciphertexts, which are indistinguishable from random under the ring learning-with-errors (RLWE) assumption underlying CKKS. The client learns only the final model output and does not receive intermediate activations or model weights. The input-privacy guarantee therefore reduces to the semantic security of CKKS, while model confidentiality is limited to what can be inferred from that final output alone.

Only the client decrypts, and no intermediate ciphertext or decrypted result reaches the server or a third party. The protocol therefore never returns a decrypted approximate value to anyone but the client, which is what the attack of Li et al. [24] requires, so its stated precondition is absent here,

though we do not formally analyze that attack surface. Standard noise-flooding countermeasures for released approximate values [14, 25] are compatible with this interface but are not part of the protocol considered here.

3.2 Block-Level Execution Pipeline

Figure 1 shows the end-to-end THEMIS workflow and how the encrypted state moves from input to output. The PCMM, CCMM, and softmax kernels are developed later.

THEMIS follows the column-packed convention used in encrypted Transformer systems such as Euston [13] and combines it with the interleaved multi-batch layout of Section 2.3. The Q , K , and V projections are column-packed. The QK^T stage produces attention scores in diagonal-packed form, which softmax preserves. $S \cdot V$ returns the attention output to a column-packed layout, and the output projection and feed-forward stages remain column-packed. Unless stated otherwise, the PCMM and C2PMM descriptions assume this column-packed interleaved layout.

THEMIS preserves the slot-encode format throughout, since the CCMM and nonlinear stages consume it directly and a conversion at every boundary would cost more than the kernels on either side save. Operator boundaries therefore schedule work rather than convert packing layouts. Section 4 shows how PCMM and CCMM preserve this representation; Section 5 places the intervening normalization in the same per-layer schedule.

BTS is placed at operator boundaries when the next stage needs more levels than the current ciphertexts carry; the softmax path additionally applies BTS to its auxiliary denominator ciphertext (Section 5).

3.3 Level Cost Model and Placement

THEMIS characterizes an encrypted Transformer block along two orthogonal axes. Along the cost axis, CKKS operation cost grows with the number of active RNS limbs. Figure 2 shows both views. Figure 2(a) gives the measured latency of a rotation, a relinearization, and a hoisted rotation across the active chain, which rises by roughly 8–14 $\times$ from Level 1 to Level 17, while Figure 2(b) separates chain length from per-limb overhead. Above Level 12, full rotation and relinearization become super-linear relative to the Level 1 limb baseline (1.38 $\times$ and 1.52 $\times$), whereas the hoisting core stays sublinear (0.91 $\times$). Accordingly, the margin exposed by hoisting grows from about 1.29 $\times$ near the chain bottom to about 2.0 $\times$ near its top.

Along the stage axis, a Transformer block alternates strictly between the matrix and nonlinear stages, and the same alternation of projection, multiplication, normalization, and activation recurs across current Transformer variants, so the principles below are stated for the two kinds rather than for one architecture. The two kinds sit differently on the chain. A

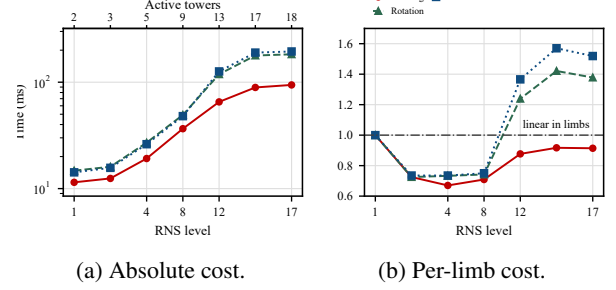


Figure 2: RNS-level cost model. The top axis gives the active tower count, one more than the level. (a) Absolute primitive latency. (b) Per-limb cost, normalized to Level 1; the dip at two to three limbs is fixed overhead at low parallelism.

matrix stage issues many homomorphic operations but occupies a narrow band of levels, whereas a nonlinear evaluator issues fewer operations of much greater depth and occupies a wide one. What decides the cost of moving a stage is that band width rather than the operation count.

These two axes yield the design principles that every kernel in this paper is built to satisfy.

Design Principle 1 (depth). Every matrix stage consumes exactly one level, and every nonlinear evaluator compresses its depth as far as precision allows.

Design Principle 2 (position). Matrix stages run as low on the chain as the level budget allows, and, since the two kinds alternate, the nonlinear evaluators occupy the levels above them.

Principle 1 follows from the budget itself, because a group of stages fits between two BTS calls only when its level consumption fits the budget one call restores, so depth spent by either kind is depth the other cannot use. Principle 2 follows from the asymmetry between the two kinds of stage. A stage pays a cost summed over the towers its band spans, and the two kinds also face displacements of different size in the first place. A matrix stage that loses the bottom must clear the whole span of the nonlinear evaluator beside it, whereas a nonlinear evaluator that loses the bottom rises by the single level that matrix stage occupies. Principle 1 is therefore what makes Principle 2 affordable, since a matrix stage confined to one level is the narrowest band on the chain and the one whose displacement the schedule can most easily pay for.

A standalone microbenchmark prices this ordering before any kernel is assembled. The same attention CCMM kernel and the same coefficient-side PCMM cost 5.96 $\times$ and 7.07 $\times$ more near the top of the chain than at its bottom, while a degree-119 Chebyshev polynomial displaced upward by a single level costs 11.5% more. Figure 2 predicts that direction, and the measured points trace it. Rotation and relinearization grow by 7.4 $\times$ and 8.1 $\times$ from level 2 to level 12 and reach 11.1 $\times$ and 12.1 $\times$ at level 16. What the schedule uses is this trend across the chain rather than a per-level table. Prior sched-

Table 1: Per-input online costs for one complete BERT-base Q projection in the unified benchmark.

Scheme	CAdd	PMult	Rot.	Level	Backend
Powerformer [34]	1,535	1,536	85	1	×
THOR [32]	1,556	1,580	40	2	×
Euston [13]	2,301	2,304	0	1	×
MOAI [42]	2,301	2,304	0	1	×
THEMIS	0	0	0	1	✓

CAdd, PMult, and Rot. denote ciphertext additions, plaintext-ciphertext multiplications, and rotations. Counts are per input at $B = 256$ ($B = 1$ for Powerformer and THOR). Level counts serial online multiplications, not RNS levels. The three zeros for THEMIS record that its PCMM issues no CKKS operator calls at all. The arithmetic runs in the dense linear-algebra backend, and the only ciphertext-side step is one coefficient-side rescale.

ules place the same stages in that upper range. MOAI evaluates its projections at $15 \rightarrow 14$ and $QK^\top$ at $14 \rightarrow 13$ [42], and THOR aligns each attention input at level 21 and advances five further levels before the ciphertext-ciphertext multiplication begins [32]. The two numberings run in opposite directions and are not comparable, but both place the attention matrix work well above the bottom, where the same primitives cost several times what they cost at levels 1 to 4.

The next two sections instantiate the model’s depth-light, operation-dense and depth-heavy, operation-sparse sides.

4 Matrix Multiplication

Following the placement model in Section 3.3, this section instantiates its matrix side. The layer’s six PCMM stages and two attention CCMM stages map to three slot-encoded kernels.

4.1 Coefficient-Side Execution

A PCMM assembled from CKKS operators pays for rotations, plaintext-ciphertext multiplications, and homomorphic accumulations. Read one layer lower, on the integer coefficients that make up the ciphertext, the same product is an ordinary dense integer matrix multiplication, the operation a general matrix-multiply (GEMM) routine performs. THEMIS reaches that product from slot-encoded $L \ll N$ Transformer matrices directly, without the modulus decomposition and repacking that smaller dimensions would otherwise need [4, 6], by handling the two ciphertext-component coefficient matrices to a fast dense linear-algebra backend, for which we use BLAS. The only ciphertext-side step in each PCMM is one coefficient-side rescale.

Two properties of C2PMM keep it out of Transformer serving. First, it is efficient only when the operand dimensions approach the ring degree and it carries no notion of batching, so a single sequence with $L \ll N$ leaves most of the ring representation empty while the backend still processes all of it. This is a slot-utilization problem rather than a backend-utilization

one. Second, it expects coefficient-encode inputs, whereas the attention and nonlinear stages that follow consume slot-encode inputs, so applying it directly would surround every PCMM with an S2C and a C2S call whose conversion cost would erode the fast numerical core’s benefit.

Slot-encoded C2PMM (SC2PMM) is the execution path THEMIS contributes for this setting. It reads the two component coefficient matrices straight out of slot-encoded ciphertexts, fills the slots through interleaved multi-batch packing, drives the exact chunked core once per component, and returns slot-encoded ciphertexts without issuing a conversion. Proposition 1 states its correctness, and the conjugate branch it removes does not follow from FFT and IFFT complementarity alone. Table 1 gives the resulting online cost.

Multi-batch slot filling. A PCMM layer applies a public weight $\mathbf{W} \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$ to an activation matrix of shape $L \times d_{\text{in}}$. THEMIS packs $B = \lfloor N/(2L) \rfloor$ samples into one ciphertext using the interleaved layout of MOAI [42]. The slot at position $\text{row} \cdot B + b$ stores row row of batch b , so the combined activation carries $L \cdot B \approx N/2$ useful values.

When the packed ciphertext feeds C2PMM, the backend matrix shape is fixed by the ring dimension and weight dimensions; changing B changes the populated slots, not the kernel shape. Appendix D reports a scan over B in which the online batch time stays near 44.5 seconds, so multi-batch packing amortizes one nearly fixed batch cost over more inputs; the required weight replication is a plaintext memory permutation performed once per model, and interleaving matches the later CCMM kernels. The resulting cost model is an approximately constant batch time and a per-input cost that falls as $1/B$, as evaluated in Section 6.

Slot-encode execution. S2C and C2S act from the left on the ring index of the coefficient representation, whereas right-multiplication by $\mathbf{W}$ acts on its column index; the two actions occupy disjoint axes. Consequently, the conceptual S2C–C2PMM–C2S envelope commutes and cancels, so the runtime issues neither an S2C nor a C2S call. At the slot level, this is the complementary FFT/IFFT structure established for $d \geq N$ by Bae et al. [3], whose Section 6.3 gives the slot-encoding compatibility.

Proposition 1 (Slot-encode coefficient execution). *Let M be held column-wise in slot-encoded ciphertexts with component matrices $\mathbf{A}, \mathbf{B}$. Then $(\mathbf{A}\mathbf{W}, \mathbf{B}\mathbf{W})$ is a ciphertext collection whose slots encode $M\mathbf{W}$.*

Proof. S2C/C2S lift to $2N \times 2N$ left actions, while $\mathbf{W}$ acts from the right and commutes with them. The first block row of the lifted inverse gives

$$N^{-1}(\hat{U}_0^\top \hat{U}_0 + \hat{U}_0^\top \hat{U}_0) = I, \quad N^{-1}(\hat{U}_0^\top \hat{U}_1 + \hat{U}_0^\top \hat{U}_1) = \mathbf{0}.$$

The first identity preserves $(\mathbf{A}\mathbf{W}, \mathbf{B}\mathbf{W})$ and the second removes its conjugate branch; Appendix E derives both identities and the resulting slot content. $\square$

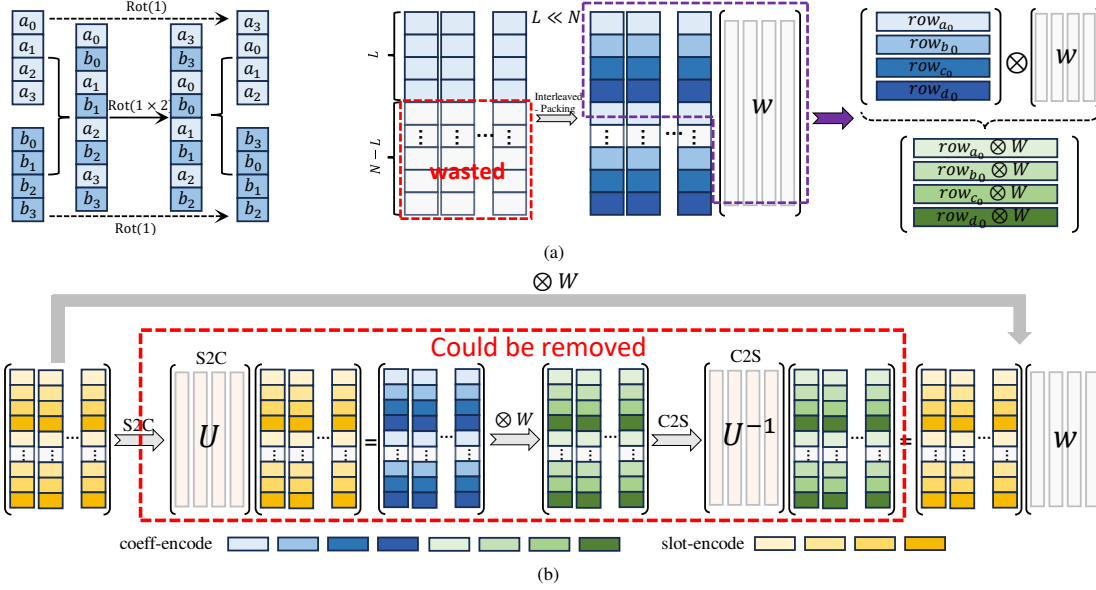


Figure 3: Interleaved packing and coefficient-side execution. Panel (a) gives the row-batch layout and the resulting slot filling, panel (b) the S2C–C2PMM–C2S envelope that cancels in THEMIS.

Algorithm 1 C2PMM

Input: Coefficient matrix $\mathbf{A} \in \mathbb{Z}_q^{m \times \kappa}$ extracted from κ ciphertext polynomials whose inputs are in slot-encode format, weight matrix $\mathbf{W} \in \mathbb{Z}_q^{\kappa \times n}$, inner dimension κ , and chunk width b

Output: $\mathbf{C} = \mathbf{A}\mathbf{W} \bmod q$

```

1:  $n_{\text{chk}} \leftarrow \lceil \lceil \log_2 q \rceil / b \rceil$ 
2: for  $j = 0$  to  $n_{\text{chk}} - 1$  do
3:    $\mathbf{W}_j \leftarrow \lfloor \mathbf{W} / 2^{jb} \rfloor \bmod 2^b$ 
4: end for
5: for  $i = 0$  to  $n_{\text{chk}} - 1$  do
6:    $\mathbf{A}_i \leftarrow \lfloor \mathbf{A} / 2^{ib} \rfloor \bmod 2^b$ 
7: end for
8:  $\mathbf{C} \leftarrow \mathbf{0}$ 
9: for  $i = 0$  to  $n_{\text{chk}} - 1$  do
10:  for  $j = 0$  to  $n_{\text{chk}} - 1$  do
11:     $\mathbf{P}_{ij} \leftarrow \text{dgemm}(\mathbf{A}_i, \mathbf{W}_j)$ 
12:     $\mathbf{C} \leftarrow \mathbf{C} + \mathbf{P}_{ij} \cdot 2^{(i+j)b} \bmod q$ 
13:  end for
14: end for
15: return  $\mathbf{C}$ 

```

Exact modular reduction. The coefficient-side backend computes $\mathbf{C} = \mathbf{A}\mathbf{W} \bmod q$ exactly as modular integer arithmetic. Algorithm 1 uses n_{chk}^2 backend BLAS calls with the weight chunks precomputed once per model, and Appendix F gives the decomposition and recomposition.

4.2 Ciphertext-Ciphertext Matrix Multiplication

THEMIS introduces two attention kernels, COLPAIR and DIAGPAIR. The two multiplications $\mathbf{Q}\mathbf{K}^\top$ and $\mathbf{S}\cdot\mathbf{V}$ are CCMMs, so their encrypted operands are accumulated through ciphertext multiplication, rotation, relinearization, and rescaling. From Euston [13], THEMIS keeps only the rotation-and-accumulation outer structure, and it adapts the interleaved multi-batch layout of MOAI [42] to both kernels, preserving the PCMM layout without repair masks or an auxiliary transfer protocol. Merging two real ciphertexts into one complex ciphertext is standard [5, 19, 37] and is no contribution by itself. What prior kernels leave unexamined is which axis the two lanes are bound to, and that choice is what decides whether the rotations the binding produces can be reused at all. THEMIS binds the lanes to the contraction axis of each multiplication, a column contraction for $\mathbf{Q}\mathbf{K}^\top$ and a diagonal contraction for $\mathbf{S}\cdot\mathbf{V}$. Figure 4 shows the two bindings.

The plaintext-reduction line of Bae et al. [3] shows that a CCMM can likewise be turned into plaintext matrix products, combined with ciphertext transposes, and handed to the same backend; it is efficient when the dimensions approach the ring degree N [3, 35]. Here $L, d_h \ll N$, and reconnecting that layout to the interleaved Transformer pipeline would require additional permutation. We therefore retain rotation-based CCMM and specialize its reuse axis separately for the two attention multiplications.

Shared BSGS and hoisting structure. After either pairing, each output is a sum over rotations by δB . Writing $\delta = gB_\star + b$ lets baby rotations be reused across giant steps, while giant ro-

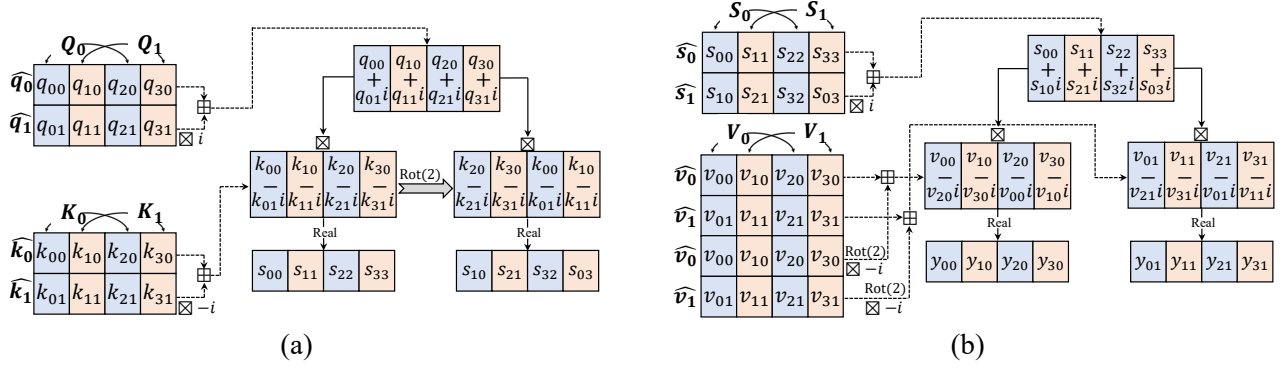


Figure 4: Pairing patterns for the two attention CCMMs. COLPAIR binds the two CKKS lanes to paired columns in $QK^\top$, while DIAGPAIR binds them to paired diagonals in $S \cdot V$.

Table 2: Reported key-switching-operation counts for attention CCMMs under each scheme’s effective input-batch convention. Parameters: BERT-base with $N = 2^{16}$, $L = 128$, $d = 768$, $d_h = a = 64$, $H = 12$, $B = 256$, $(B_Q, G_Q) = (16, 8)$, $(B_S, G_S) = (r, s) = (8, 8)$, $(g_M, b_M) = (12, 11)$, $c = 16$, and $(r_\sigma, s_\sigma) = (13, 5)$.

Scheme	$\{Q^{(h)}(K^{(h)})^\top\}_{h=1}^H$		$\{S^{(h)}V^{(h)}\}_{h=1}^H$	
	Key-switch equation	Number	Key-switch equation	Number
Powerformer [34]	$\approx (17 + 14\sqrt{2})\sqrt{a} + 3a$, $a = d_h$	486	$\lceil H/8 \rceil (R_\sigma + 2) + \frac{H}{4} (R_\tau + R_\theta + r) + (\frac{H}{2} - 1)$	351
THOR [32]	$\frac{L}{2c} (\frac{L}{2} + 7) + \frac{L}{8} (\log_2 c + 3) + \frac{L}{2}$	460	$\frac{L(L+6)}{4c}$	268
Euston [13]	$(L-1)d_h H / (N/(4L))$	762	$(L-1)d_h H / (N/(4L))$	762
MOAI [42]	$(L-1)d_h H / (N/(2L))$	381	$[(L - g_M) + d_h((g_M - 1) + (b_M - 1) + b_M d_h)]H/B$	101.44
THEMIS	$\lceil \frac{d_h}{2} (B_Q + G_Q - 2) + 2L - B_Q \rceil H/B$	44.25	$d_h(B_S + 2G_S) + \frac{L}{2} - B_S$	74.63

For the Powerformer $S \cdot V$ equation, $R_\sigma = 1 + 2(s_\sigma - 1) + (r_\sigma - 1)$, $R_\tau = (s - 1) + (r - 1)$, and $R_\theta = 1 + (s - 1) + (d_h - 1) + (r - 1)$.

Powerformer and THOR use no input-batch divisor, so their effective batch is $B = 1$. Euston divides by $N/(4L)$, which gives $B = 128$ under the stated parameters. MOAI’s $QK^\top$ entry divides by $N/(2L) = 256$; its $S \cdot V$ entry and both THEMIS entries use the per-head count multiplied by H/B , with $B = 256$; (g_M, b_M) , c , and (r_σ, s_σ) are scheme-specific parameters from the cited papers.

tations align the partial sums. Every baby step rotates the same source ciphertext, so each baby set is a single HOISTROT call. Algorithms 2 and 3 draw at least eight rotations from each call, well past the two at which the shared precomputation breaks even. RELIN denotes the post-accumulation relinearization, and $\text{Re}_2(z) = z + \bar{z} = 2\text{Re}(z)$ extracts the real result with a known factor of two. BSGS and hoisting change the execution schedule, not the algebraic product.

The COLPAIR kernel ($QK^\top$). Coefficient-side execution produces Q and K in natural-order interleaved packing, where a row rotation by δB preserves each batch residue. COLPAIR binds the real and imaginary lanes to the two halves of the column contraction, so the real part of one paired product recovers both column contributions. Its direct effect is small. A real-only schedule needs Ld_h rotations, generic complex merging needs $Ld_h/2 + d_h$, and COLPAIR needs $Ld_h/2$, saving only the one pre-rotation per column. Its value is structural. Generic merging can decompose a BSGS index of $L/2$ and costs $d_h(B + G)$ rotations with $BG = L/2$, whereas COLPAIR decomposes the full index L and costs $\frac{d_h}{2}(B_Q + G_Q)$ with $B_Q G_Q = L$. Neither expression depends on the sequence length or the batch size chosen for deployment. At $L = 128$ and $d_h = 64$ the selected splits give 1024 and 768 rotations,

a $1.33 \times$ reduction. Appendix G gives the algebra and count derivation; Algorithm 2 gives the executable schedule.

The DIAGPAIR kernel ($S \cdot V$). Here S is diagonal-packed, V is column-packed, and the contraction runs over the diagonal index shared by all output columns. DIAGPAIR binds the two complex lanes to the first and second halves of that index, while a half-sequence rotation merges the matching V entries. It changes the direct rotation count from Ld_h to $(\frac{L}{2} + 1)d_h$ before BSGS and hoisting, a reduction that approaches two as L grows and reaches $1.97 \times$ at $L = 128$ and $d_h = 64$. This is not a second instance of COLPAIR. COLPAIR widens the decomposable BSGS index, whereas DIAGPAIR directly halves the contraction axis. MOAI [42] already applies BSGS to $S \cdot V$, so the predicted difference is DIAGPAIR’s $1.97 \times$ reduction plus hoisting, consistent with the measured $2.18 \times$ gain. Appendix G gives the algebra; Algorithm 3 gives the executable schedule.

Both kernels return $\text{Re}_2(z) = 2\text{Re}(z)$. For $QK^\top$, the missing factor $1/2$ is folded into the subsequent $1/\sqrt{d_h}$ plaintext constant without another multiplicative level; the $S \cdot V$ factor is analogously absorbed by its next linear stage.

Performance comparison. Table 2 compares the amortized key-switching cost of the two attention CCMMs against

Algorithm 2 COLPAIR for $QK^\top$

Input: Column ciphertexts $\{Q_j\}_{j=0}^{d_h-1}$ and $\{K_j\}_{j=0}^{d_h-1}$, batch size B , BSGS factorization $L = B_Q \cdot G_Q$
Output: Diagonal ciphertexts $\{D_\delta\}_{\delta=0}^{L-1}$ encoding the diagonals of $QK^\top$

- 1: **for** $j = 0$ to $d_h/2 - 1$ **do**
- 2: $Q_j^{\text{pair}} \leftarrow Q_j + iQ_{j+d_h/2}, K_j^{\text{pair}} \leftarrow K_j - iK_{j+d_h/2}$
- 3: **end for**
- 4: $D_\delta \leftarrow 0$ for all $\delta \in \{0, \dots, L-1\}$
- 5: **for** $j = 0$ to $d_h/2 - 1$ **do**
- 6: $\{T_b\}_{b=0}^{B_Q-1} \leftarrow \text{HOISTROT}(K_j^{\text{pair}}, \{bB\}_{b=0}^{B_Q-1})$
- 7: $\{\tilde{Q}_{j,g}^{\text{pair}}\}_{g=0}^{G_Q-1} \leftarrow \text{HOISTROT}(Q_j^{\text{pair}}, \{-gB_QB\}_{g=0}^{G_Q-1})$
- 8: **for** $g = 0$ to $G_Q - 1$ **do**
- 9: **for** $b = 0$ to $B_Q - 1$ **do**
- 10: $D_{gB_Q+b} \leftarrow D_{gB_Q+b} + \tilde{Q}_{j,g}^{\text{pair}} \boxtimes T_b$
- 11: **end for**
- 12: **end for**
- 13: **end for**
- 14: **for** $g = 0$ to $G_Q - 1$ **do**
- 15: **for** $b = 0$ to $B_Q - 1$ **do**
- 16: $\delta \leftarrow gB_Q + b$
- 17: $D_\delta \leftarrow \text{Re}_2(\text{Rot}(\text{RELIN}(D_\delta), gB_QB))$
- 18: **end for**
- 19: **end for**
- 20: **return** $\{D_\delta\}_{\delta=0}^{L-1}$

Algorithm 3 DIAGPAIR for $S \cdot V$

Input: Diagonal ciphertexts $\{S_\delta\}_{\delta=0}^{L-1}$, column ciphertexts $\{V_j\}_{j=0}^{d_h-1}$, batch size B , BSGS factorization $L/2 = B_S \cdot G_S$
Output: Column ciphertexts $\{O_j\}_{j=0}^{d_h-1}$ encoding the columns of $S \cdot V$

- 1: **for** $g = 0$ to $G_S - 1$, $b = 0$ to $B_S - 1$ **do**
- 2: $\delta \leftarrow gB_S + b$
- 3: $\hat{S}_{g,b} \leftarrow \text{Rot}(S_\delta + iS_{\delta+L/2}, -gB_SB)$
- 4: **end for**
- 5: **for** $j = 0$ to $d_h - 1$ **do**
- 6: $V_j^{\text{merge}} \leftarrow V_j - i \cdot \text{Rot}(V_j, \frac{L}{2}B)$
- 7: $\{T_b\}_{b=0}^{B_S-1} \leftarrow \text{HOISTROT}(V_j^{\text{merge}}, \{bB\}_{b=0}^{B_S-1})$
- 8: $O_j \leftarrow 0$
- 9: **for** $g = 0$ to $G_S - 1$ **do**
- 10: $U_g \leftarrow \sum_{b=0}^{B_S-1} \hat{S}_{g,b} \boxtimes T_b$
- 11: $O_j \leftarrow O_j + \text{Rot}(U_g, gB_SB)$
- 12: **end for**
- 13: $O_j \leftarrow \text{Re}_2(O_j)$
- 14: **end for**
- 15: **return** $\{O_j\}_{j=0}^{d_h-1}$

prior encrypted Transformer systems. The MOAI comparison uses the same $B = 256$, so its gap is algorithmic, whereas

the Powerformer, THOR, and Euston comparisons use different effective batches and therefore combine algorithmic and packing-batch effects. In THEMIS, hoisting is enabled on all inner rotations. Wall-clock measurements are deferred to Section 6.

With PCMM executed directly in slot-encode format and the two attention multiplications handled by rotation-based kernels in the same domain, the PCMM to CCMM boundary inside a Transformer layer carries no representation transition.

5 Denominator-Conditioned Softmax

This section presents the denominator-conditioned softmax. A calibrated row-wise offset preconditions the first denominator, low-cost reciprocals serve intermediate normalizations, and a high-precision approximation produces the final output. The method keeps the square-and-normalize backbone for precision while removing its repeated high-precision depth cost, under the tight level budget that Section 3.3 assigns to depth-heavy stages.

5.1 Offset as Denominator Preconditioning

The first denominator is the main numerical bottleneck because it fixes the operating range seen by the reciprocal path that follows. A THOR-style power-of-two scaling [32] can shrink the absolute interval, but it does not improve the empirical interval ratio that governs reciprocal approximation difficulty.

Key observation. Only the final normalization determines the returned probabilities. Error in an intermediate reciprocal changes only the row-wise scale, which the next normalization cancels; high precision can therefore be concentrated in the final reciprocal. THEMIS couples offset preconditioning, which narrows the first-denominator interval ratio, with precision staging, which retains high precision only for the final normalization.

Softmax shift invariance gives a more effective conditioning handle. A row-wise offset can move the first denominator toward a favorable operating range without changing the plaintext softmax value. THEMIS therefore uses this invariance to precondition the first denominator before the square-and-normalize iterations begin:

$$\text{Softmax}(x) = \text{Softmax}(x - a(x)). \quad (4)$$

This property acts as a denominator preconditioner rather than as an encrypted maximum estimator. Let $T_\ell = 2^{K_\ell}$ be the initial scaling factor used before square-and-normalize in layer ℓ ; it is an implementation scale, not a change to the target softmax. The first normalized denominator has the form

$$D_0(a) = C \cdot \text{mean}_i \exp\left(\frac{x_i - a(x)}{T_\ell}\right), \quad (5)$$

where C is a plaintext scaling constant. The row-wise offset changes this denominator through a row-dependent scale.

The ideal offset would place the first denominator at a target value d_* , which takes the scaled log-partition value of the row shifted by $\log C - \log d_*$. Evaluating that quantity online would be as hard as computing the softmax denominator itself, so THEMIS uses a homomorphic-friendly proxy

$$a_\ell(x) = \mu(x) + \beta_\ell \text{Var}(x) + c_\ell, \quad (6)$$

where $\mu(x)$ and $\text{Var}(x)$ are row-wise statistics and the approximation parameters β_ℓ, c_ℓ are calibrated offline on a held-out, task-specific calibration split. This offset leaves the plaintext softmax value unchanged because it is a uniform row shift.

Algorithm 5 summarizes the offline calibration. For each valid attention row, THEMIS records

$$\begin{aligned} \text{base}(x) &= \log \left(\frac{1}{L} \sum_i \exp \left(\frac{x_i - \mu(x)}{T_\ell} \right) \right), \\ v(x) &= \text{Var}(x). \end{aligned} \quad (7)$$

The β_ℓ search minimizes the empirical width of $\text{base}(x) - \beta v(x)/T_\ell$, and then c_ℓ aligns the lower endpoint of the scaled first denominator to d_* , a numerical conditioning target rather than a softmax semantics parameter.

5.2 Precision-Staged Square-and-Normalize

After conditioning the first denominator, the remaining question is how to execute the full square-and-normalize schedule efficiently. The issue is no longer the first reciprocal interval itself, but whether every subsequent denominator stage must still receive the same high-precision inverse treatment. Following Cho et al. [11], we view the denominator path as an auxiliary thread that carries only one ciphertext, while the main thread carries the full softmax row. The observation in Section 5.1 also distinguishes the roles of these stages. Only the last denominator stage directly affects the returned softmax output.

The reciprocal kernel used by Algorithm 4 is stage-dependent:

$$\hat{r}_t(D_t) = \begin{cases} P_{t,\ell}^{\text{lo}}(D_t), & 0 \leq t < K_\ell, \\ \hat{r}_\ell^{\text{hi}}(D_t), & t = K_\ell, \end{cases} \quad (8)$$

where $P_{t,\ell}^{\text{lo}}$ is a low-degree Chebyshev relative reciprocal and $\hat{r}_\ell^{\text{hi}}$ is the high-precision Chebyshev+aSOR final reciprocal [31]. The denominator D_0 is therefore also a non-final low-precision reciprocal stage whenever $K_\ell > 0$.

We therefore keep the THOR-style square-and-normalize backbone but replace uniform inverse handling with a precision-staged schedule. For a positive vector y , define

$$\mathcal{N}(y)_i = \frac{y_i}{\text{mean}_j y_j}, \quad \mathcal{S}(y) = \mathcal{N}(y^2). \quad (9)$$

Algorithm 4 Denominator-Conditioned Softmax.

Input: Encrypted row x , layer ℓ , offset (β_ℓ, c_ℓ) , public scale C , iteration count K_ℓ

Output: Mean-normalized softmax row $p^{(K_\ell)}$

```

1:  $a_\ell(x) \leftarrow \mu(x) + \beta_\ell \text{Var}(x) + c_\ell$ 
2:  $e_i^{(0)} \leftarrow C \exp((x_i - a_\ell(x))/2^{K_\ell})$ 
3:  $D_0 \leftarrow \text{mean}_i e_i^{(0)}$ 
4:  $p_i^{(0)} \leftarrow e_i^{(0)} \hat{r}_0(D_0)$ 
5: for  $t = 1$  to  $K_\ell$  do
6:    $D_t \leftarrow \text{mean}_i (p_i^{(t-1)})^2$ 
7:    $p_i^{(t)} \leftarrow (p_i^{(t-1)})^2 \hat{r}_t(D_t)$ 
8: end for
9: return  $p^{(K_\ell)}$ 

```

If $y_i = \exp(x_i/2^{K_\ell})$ for layer-wise iteration count K_ℓ , then repeated application of $\mathcal{S}$ gives

$$\mathcal{S}^{K_\ell}(\mathcal{N}(y))_i = \frac{\exp(x_i)}{\text{mean}_j \exp(x_j)}. \quad (10)$$

The exponential polynomial only sees the scaled interval, while each normalization keeps the intermediate vector in a bounded positive range.

For numerical convenience, we express row reductions as means. The mean-normalized softmax differs from the conventional sum-normalized softmax by the public factor L , which is absorbed into the following $\mathcal{S} \cdot V$ multiplication as a plaintext scale.

Algorithm 4 gives the resulting online schedule, with C the same public scale used in the offset calibration and $\hat{r}_t$ the stage-dependent kernel of (8). The schedule makes the precision split explicit. Requiring every denominator stage to approximate $1/D_t$ with high precision and to apply BTS to its auxiliary ciphertext is unnecessary and expensive, so THEMIS uses a mixed-precision schedule and reserves BTS for the final denominator stage. Appendix L.3 gives the complete correctness argument.

The final reciprocal must approximate $1/D_t$ accurately because it produces the returned probabilities. Non-final stages need good denominator conditioning instead. Writing $\alpha(D_t) = D_t \hat{r}_t(D_t)$, such a stage returns $\alpha(D_t)$ times the exactly normalized row, and the next normalization cancels $\alpha(D_t)^2$ after the following square. The scale is not free. If $\alpha(D_t)$ varies widely over the empirical denominator range, the next denominator interval widens and the square step amplifies that widening quadratically, so the best low-cost non-final kernel is still reciprocal-shaped, keeping $\alpha(D_t)$ narrow even without high absolute precision. THEMIS therefore uses low-degree Chebyshev polynomials for non-final stages and calibrates them by the relative error $|D_t P_t(D_t) - 1|$ rather than by absolute reciprocal error (Appendix L.2).

The final reciprocal is where the accuracy of the returned probabilities is bought, and how it is bought decides the

cost. Adaptive successive over-relaxation (aSOR) converges quickly, but only from an initial estimate already close to $1/D_i$, so after a fixed number of refinement rounds the precision a schedule can promise is set by where the iteration starts rather than by the iteration alone. THEMIS therefore supplies that starting point instead of leaving it to the iteration. The offset conditioning of Section 5.1 and the relative-error calibration of the non-final stages keep the final denominator inside an interval narrow enough for a low-degree Chebyshev polynomial to serve as the initial estimate, and aSOR refines from there. Reserving aSOR for this one stage is what makes it affordable, since each refinement round consumes two computational levels and a non-final stage would pay much of that convergence cost for a reciprocal whose error the next normalization removes anyway. The final normalization therefore uses a Chebyshev initializer followed by only a small number of aSOR refinement rounds [31]. Layer-specific denominator intervals and iteration schedules are deferred to the softmax evaluation in Section 6 and Appendix I.

LayerNorm and GELU need neither offset conditioning nor square-and-normalize, and Appendix H gives their approximation intervals.

6 Evaluation

We evaluate the kernel, softmax, accuracy, and end-to-end claims against state-of-the-art non-interactive encrypted Transformer systems; earlier systems are situated through the comparisons reported by those works.

6.1 Experimental setup

THEMIS uses OpenFHE [1] on an Intel Xeon Platinum 8473C CPU with 52 cores and 104 threads. Latency is amortized per input at batch size 256 for BERT-base with $L=128$ unless noted. We evaluate accuracy on RTE, SST-2, and QNLI, and Appendix C gives the kernel parameters.

6.2 Micro-benchmarks

PCMM and CCMM performance. To compare on equal terms, all PCMM and CCMM kernels used by the systems in this comparison are implemented in OpenFHE, run in the same environment, and measured after producing outputs at the bottom level expected by the next stage (Section 3.3). Figure 5 compares THEMIS with Powerformer, THOR, Euston, and MOAI. NEXUS [41] is not reimplemented in this kernel comparison, since Euston [13] already reports it against this class of matrix kernels. It returns in Section 6.4, where what is compared is a within-system ratio rather than a kernel time. Relative to the fastest prior baseline MOAI, THEMIS speeds up PCMM by 21.66 $\times$, $QK^\top$ by 7.39 $\times$, and $S \cdot V$ by 2.18 $\times$.

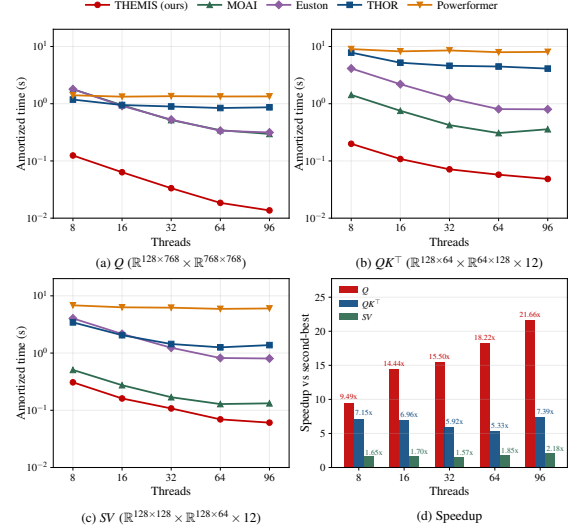


Figure 5: PCMM and CCMM performance, one OpenFHE environment, all outputs at the expected next-stage level.

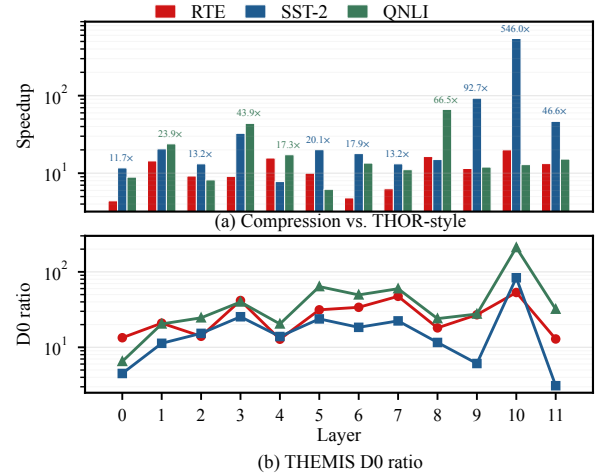


Figure 6: First-reciprocal denominator intervals in softmax. Panel (a) compares the THOR baseline with THEMIS offset conditioning, panel (b) the resulting D_0 ratio, upper endpoint over lower endpoint.

THEMIS reaches amortized runtimes of 0.013 seconds for one PCMM query projection, 0.048 seconds for 12-head $QK^\top$, and 0.061 seconds for 12-head $S \cdot V$. Against Euston, THOR, and Powerformer, the maximum speedups are 23.02 $\times$, 63.32 $\times$, and 98.22 $\times$ for PCMM, 16.49 $\times$, 84.87 $\times$, and 166.40 $\times$ for $QK^\top$, and 13.21 $\times$, 22.65 $\times$, and 99.01 $\times$ for $S \cdot V$, respectively.

Sections 4.1 and 4.2 give the mechanisms behind this ordering; Tables 1 and 2 give their operation boundaries. Euston and MOAI overlap in Figure 5(a) because their isomorphic online paths share one benchmark implementation and differ

Table 3: Softmax cost comparison.

	#Iter.	#BTS	Inv. depth	Total (s)	Amort. (s)	Prec. (bits)
THOR [32]	2	3	43	389.38	1.52	8.84
THEMIS ^a	2	1	21	151.59	0.59	13.25

^a Inv. depth denotes reciprocal-polynomial approximation depth, not RNS-level consumption, and cannot be subtracted from available levels. #BTS counts softmax-path denominator calls rather than the operator-boundary calls in Table 4; applying it only to the smaller denominator follows [11].

only in batch-slot order.

The three speedups differ by an order of magnitude because the three kernels change different things. PCMM leaves the CKKS operator layer altogether, so Table 1 records no ciphertext additions, no plaintext-ciphertext multiplications, and no rotations for THEMIS, and the arithmetic runs in a dense linear-algebra backend instead. $QK^\top$ stays rotation-based, and its gain comes from the wider index COLPAIR makes decomposable, which replaces a $d_h(B + G)$ rotation schedule with a $\frac{d_h}{2}(B_Q + G_Q)$ one. $S \cdot V$ gains least because MOAI already applies BSGS to it, so what remains for DI-AGPAIR is the $1.97\times$ halving of the contraction axis and the narrow margin hoisting adds at the bottom of the chain.

Appendix G.1 decomposes the two attention kernels cumulatively over their unpaired baselines. Merging the complex lanes accounts for $1.98\times$ and $1.89\times$, the BSGS schedule for the largest remaining step at $4.67\times$ and $3.82\times$, and hoisting for a further 0.8% and 5.9%, which is small by construction because these kernels run near the bottom of the chain where the hoisting advantage is narrowest.

Softmax. The softmax micro-benchmark evaluates latency and numerical accuracy of the denominator-conditioned schedule. The cost comparison in Table 3 is measured on RTE.

Figure 6 isolates the role of the row-wise offset before the first reciprocal. Without such an offset, the first denominator inherits a wide empirical interval, and the denominator ratio is what makes reciprocal approximation difficult.

Across the three datasets, Figure 6(a) shows first-denominator interval-ratio reductions of $11.3\times$ on RTE, $69.8\times$ on SST-2, and $19.9\times$ on QNLI. A single evaluated layer-dataset point reaches $546\times$, the maximum over all points; we therefore do not use this extreme as representative. Figure 6(b) shows conditioned ratios mostly below 100, which lets THEMIS use very low-degree Chebyshev reciprocal polynomials, from 5 to 13 in our deployment. Those low degrees are what the depth and BTS reductions reported next rest on, since the reciprocal path is where the softmax spends its depth.

Table 3 reports the full precision-staged schedule. For a fair comparison, both rows use the THEMIS packing layout and start at the same bottom-level point with 16 available computational levels, and the THOR row evaluates the softmax computation of THOR [32] under that same layout, so only the computation flow differs between it and the THEMIS

denominator-conditioned schedule. Neither row counts the exponential polynomial, while the THEMIS row still counts the row-wise offset to balance that omission. Reported time is the per-input amortized latency of one softmax layer, obtained by dividing measured layer runtime by 256. With the same square-and-normalize iteration count, THEMIS reduces BTS from 3 to 1 and inverse depth from 43 to 21.

These changes reduce amortized softmax latency by $2.57\times$, from 1.52 to 0.59 seconds, while raising reported precision to 13.25 bits. Table 3 attributes the reduction to reserving the high-precision reciprocal for the final normalization that determines the returned probabilities. THOR folds the constant aSOR scale into the CKKS scale, an implementation-level fusion that is difficult in OpenFHE; the comparison therefore does not count it as one fewer level. This orthogonal optimization would also apply to THEMIS. Appendix L.4 gives the layer-specific deployment. Table 13 reports calibrated square-and-normalize iterations, and Algorithm 5 produces β_ℓ and c_ℓ from the calibration split.

6.3 Accuracy under encrypted execution

Appendix B reports the staged accuracy decomposition under the same trained model and task configuration.

Table 8 reports task accuracy for the approximation pipeline we evaluate, not exact preservation of the original Transformer function. On RTE, SST-2, and QNLI, the encrypted pipeline reports 71.48%, 92.32%, and 91.36% accuracy, versus plaintext baselines of 72.56%, 92.43%, and 91.54%. The final encrypted column remains close to the last unencrypted approximation column, showing that CKKS execution tracks the chosen approximation pipeline. The remaining baseline gap splits along the same columns. On RTE, approximating GELU costs nothing, approximating Layer-Norm costs 0.36 points, approximating softmax costs nothing further, and the last 0.72 points appear only under encrypted execution, so polynomial approximation and BTS precision account for the two parts in that order, with residual CKKS error inside the second. Among systems serving the original trained model, THEMIS reports the highest encrypted accuracy on RTE and QNLI; on SST-2, it trails Euston and MOAI, whose plaintext baselines are also higher.

6.4 End-to-end cost

Table 4 reports the full encrypted pipeline. It gives a total amortized latency of 400.06 seconds per input, or about 6.7 minutes, and the level consumption of the evaluated per-layer schedule. Different layers can require minor schedule adjustments, but the table captures the layout used by most layers in our deployment.

Table 4 summarizes how the three technical components interact in the final system. The six PCMM stages and two attention CCMM stages total 9.72 seconds per input, or 2.4%,

Table 4: End-to-end latency summed over the 12 encoder layers and amortized over $B = 256$ inputs.

Component	Operation	Depth	Time(s)
Attention	Q, K (PCMM)	$3 \rightarrow 2$	0.49
	V (PCMM) ^①	$4 \rightarrow 3$	0.41
	QK^T (CCMM)	$2 \rightarrow 1$	0.58
	BTS	$1 \rightarrow 17$	181.40
	Softmax	$17 \rightarrow 3$	14.99
	$S \cdot V$ (CCMM)	$3 \rightarrow 2$	0.72
	W_O (PCMM)	$2 \rightarrow 1$	0.27
LayerNorm	BTS	$1 \rightarrow 17$	90.70
	LN1	$17 \rightarrow 9$	0.99
FFN	W_{F1} (PCMM) ^②	$9 \rightarrow 9$	6.43
	GELU	$9 \rightarrow 2$	16.67
	W_{F2} (PCMM)	$2 \rightarrow 1$	0.82
LayerNorm	BTS	$1 \rightarrow 13$	84.65
	LN2	$13 \rightarrow 4$	0.94
Total			400.06

^① Q and K drop one level before their projection, while V keeps that level for $S \cdot V$.

^② Merged into LN1, consuming no level.

Table 5: Share of end-to-end latency spent outside bootstrapping, and scheduled BTS calls per encoder layer.

System	BTS calls/layer	Non-BTS time
NEXUS [41]	4	37.69%
MOAI [42]	4	33.04%
Euston [13]	4	20.99%
THEMIS	3	10.83%

Non-BTS time is the sum of everything a layer spends outside the BTS calls between operators, given as a share of each system’s own reported end-to-end total.

whereas the three explicitly scheduled BTS entries total 356.75 seconds, or 89.2%. The latter share covers only those explicit entries. Softmax-internal denominator BTS calls remain in the Softmax row, so it undercounts total BTS time. Among the remaining nonlinear stages, softmax and GELU take 14.99 and 16.67 seconds, respectively. The one matrix stage that does not sit near the bottom is W_{F1} , which runs at the level LN1 leaves because bringing it down would cost an additional BTS call.

A BTS call has a fixed cost that no schedule can reduce, so what a co-design can move is how many calls a layer needs and how much the stages between them cost. Table 5 reports both. THEMIS issues three bootstrapping calls per encoder layer where NEXUS, Euston, and MOAI each issue four, and its matrix and nonlinear stages together take 43.31 seconds, or 10.83% of end-to-end latency, against 20.99% to 37.69% for those systems.

The table covers the non-interactive systems that, like THEMIS, use multi-batch packing and report CPU measurements. Powerformer and THOR are excluded because their reported schedules are single-input and run on a different hardware class, on which the balance between kernel time

and bootstrapping time does not carry over. Within that scope the quantity compared is a within-system ratio, so it survives the library and hardware differences that make absolute latencies incomparable, although it is not fully invariant, since kernels and bootstrapping do not scale identically with core count. The call count records operator-boundary calls on the main activation path, and the single auxiliary denominator call inside THEMIS softmax is excluded and left in the non-BTS total, so the THEMIS share is an upper bound. The ratio also does not rank designs by itself, since issuing more bootstrapping calls would lower it. Read together with the call count it says something narrower, that what remains in THEMIS is bootstrapping rather than the kernels around it, and that the level budget the two principles impose is what removed the fourth call.

7 Conclusion and Discussion

In this paper, we present THEMIS, an encrypted Transformer inference system organized around one level-budget co-design. Two design principles fix how much depth each kind of stage spends and where on the modulus chain it runs, and every kernel is built to satisfy them. Over MOAI, THEMIS accelerates PCMM by $21.66\times$ and the attention CCMMs by up to $7.39\times$, its softmax runs $2.57\times$ faster than THOR at higher precision, and it serves BERT-base at an amortized 400.06 seconds per input on the original trained model. Each kernel is faster on its own, and because all of them meet the same schedule, those gains survive into the complete pipeline, where the stages around bootstrapping account for 10.83% of end-to-end latency.

The largest remaining target follows from the design itself. THEMIS treats BTS as a black box with a fixed per-call cost and moves only how many calls a layer needs, so once the fourth call is gone, BTS is what remains, at 89.2% of end-to-end latency. Lowering the cost of a single call rather than their number is the natural next step, and both BTS and the dense integer product beneath PCMM map well onto GPUs.

Ethical Considerations

This work studies privacy-preserving inference and does not involve human-subject experiments, collection of private user data, or vulnerability disclosure. The evaluation uses public benchmark datasets and reports aggregate task accuracy and system performance.

References

- [1] Ahmad Al Badawi, Jack Bates, Flávio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, Zeyu Liu, Daniele Micciancio, Ian Quah, Yuri

- Polyakov, R. V. Saraswathy, Kurt Rohloff, Jonathan Saylor, Dmitriy Saponitsky, Matthew Triplett, Vinod Vaikuntanathan, and Vincent Zucca. OpenFHE: Open-source fully homomorphic encryption library. In *WAHC@CCS 2022*, pages 53–63, New York, NY, USA, 2022. ACM. doi:10.1145/3560827.3563379.
- [2] Youngjin Bae, Jung Hee Cheon, Guillaume Hanrot, Jai Hyun Park, and Damien Stehlé. Plaintext-ciphertext matrix multiplication and FHE bootstrapping: Fast and fused. In *Advances in Cryptology – CRYPTO 2024*, volume 14922 of *Lecture Notes in Computer Science*, pages 387–421. Springer, 2024. doi:10.1007/978-3-031-68382-4_12.
- [3] Youngjin Bae, Jung Hee Cheon, Guillaume Hanrot, Jai Hyun Park, and Damien Stehlé. Fast homomorphic linear algebra with BLAS. *Journal of Cryptology*, 39(3):25, 2026. doi:10.1007/s00145-026-09580-x.
- [4] Youngjin Bae, Jung Hee Cheon, Jaehyung Kim, Jai Hyun Park, and Damien Stehlé. HERMES: Efficient ring packing using MLWE ciphertexts and application to transciphering. In *Advances in Cryptology – CRYPTO 2023*, pages 37–69. Springer, 2023. doi:10.1007/978-3-031-38551-3_2.
- [5] Fabian Boemer, Anamaria Costache, Rosario Cammarota, and Casimir Wierzynski. nGraph-HE2: A high-throughput framework for neural network inference on encrypted data. In *Proceedings of the 2019 Workshop on Encrypted Computing and Applied Homomorphic Cryptography*, pages 45–56. ACM, 2019. doi:10.1145/3338469.3358944.
- [6] Christina Boura, Nicolas Gama, Mariya Georgieva, and Dimitar Jetchev. CHIMERA: Combining ring-LWE-based fully homomorphic encryption schemes. *Journal of Mathematical Cryptology*, 14(1):316–338, 2020. doi:10.1515/jmc-2019-0026.
- [7] Jung Hee Cheon, Hyeongmin Choe, Minsik Kang, Jaehyung Kim, Seonghak Kim, Johannes Mono, and Taeyeon Noh. Grafting: Decoupled scale factors and modulus in RNS-CKKS. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, pages 1098–1112. ACM, 2025. doi:10.1145/3719027.3765083.
- [8] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. Bootstrapping for approximate homomorphic encryption. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part I*, Lecture Notes in Computer Science, pages 360–384. Springer, 2018. doi:10.1007/978-3-319-78381-9_14.
- [9] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. A full RNS variant of approximate homomorphic encryption. In Carlos Cid and Michael J. Jacobson Jr., editors, *Selected Areas in Cryptography - SAC 2018 - 25th International Conference, Calgary, AB, Canada, August 15-17, 2018, Revised Selected Papers*, Lecture Notes in Computer Science, pages 347–368. Springer, 2018. doi:10.1007/978-3-030-10970-7_16.
- [10] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, Lecture Notes in Computer Science, pages 409–437. Springer, 2017. doi:10.1007/978-3-319-70694-8_15.
- [11] Wonhee Cho, Guillaume Hanrot, Taeseong Kim, Minje Park, and Damien Stehlé. Fast and accurate homomorphic softmax evaluation. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024*, pages 4391–4404. ACM, 2024. doi:10.1145/3658644.3670369.
- [12] Benjamin R. Curtis and Rachel Player. On the feasibility and impact of standardising sparse-secret LWE parameter sets for homomorphic encryption. In *Proceedings of the 7th ACM Workshop on Encrypted Computing and Applied Homomorphic Cryptography*, pages 1–10, 2019. doi:10.1145/3338469.3358940.
- [13] 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)*, pages 1443–1462. IEEE, 2026. doi:10.1109/SP63933.2026.00048.
- [14] Qian Guo, Denis Nabokov, Elias Suvanto, and Thomas Johansson. Key recovery attacks on approximate homomorphic encryption with non-worst-case noise flooding countermeasures. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 7447–7461. USENIX Association, 2024. URL: <https://www.usenix.org/conference/usenixsecurity24/presentation/guo-qian>.

- [15] Kanav Gupta, Neha Jawalkar, Ananta Mukherjee, Nishanth Chandran, Divya Gupta, Ashish Panwar, and Rahul Sharma. SIGMA: secure GPT inference with function secret sharing. *Proc. Priv. Enhancing Technol.*, 2024(4):61–79, 2024. URL: <https://doi.org/10.56553/popets-2024-0107>, doi:10.56553/POPET S-2024-0107.
- [16] Shai Halevi and Victor Shoup. Algorithms in HELib. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, Lecture Notes in Computer Science, pages 554–571. Springer, 2014. doi:10.1007/978-3-662-44371-2_31.
- [17] Meng Hao, Hongwei Li, Hanxiao Chen, Pengzhi Xing, Guowen Xu, and Tianwei Zhang. Iron: Private inference on transformers. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems, NeurIPS 2022*, 2022. URL: http://papers.nips.cc/paper_files/paper/2022/hash/64e2449d74f84e5b1a5c96ba7b3d308e-Abstract-Conference.html, doi:10.52202/068431-1143.
- [18] Zhicong Huang, Cheng Hong, Chenkai Weng, Wen-jie Lu, and Hunter Qu. More efficient secure matrix multiplication for unbalanced recommender systems. *IEEE Transactions on Dependable and Secure Computing*, 20(1):551–562, 2023. doi:10.1109/TDSC.2021.3139318.
- [19] Francesco Intoci, Sinem Sav, Apostolos Pyrgelis, Jean-Philippe Bossuat, Juan Ramón Troncoso-Pastoriza, and Jean-Pierre Hubaux. slytHERin: An agile framework for encrypted deep neural network inference. In *Applied Cryptography and Network Security Workshops*, volume 13907 of *Lecture Notes in Computer Science*, pages 359–377. Springer, 2023. doi:10.1007/978-3-031-41181-6_20.
- [20] Jaehee Jang, Younho Lee, Andrey Kim, Byunggook Na, Donggeon Yhee, Byoungnan Lee, Jung Hee Cheon, and Sungroh Yoon. Privacy-preserving deep sequential model with matrix homomorphic encryption. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, pages 377–391. ACM, 2022. doi:10.1145/3488932.3523253.
- [21] Xiaoqian Jiang, Miran Kim, Kristin E. Lauter, and Yongsoo Song. Secure outsourced matrix computation and application to neural networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1209–1222. ACM, 2018. doi:10.1145/3243734.3243837.
- [22] Andes Y. L. Kei and Sherman S. M. Chow. SHAFT: secure, handy, accurate and fast transformer inference. In *32nd Annual Network and Distributed System Security Symposium, NDSS 2025*. The Internet Society, 2025. URL: <https://www.ndss-symposium.org/ndss-paper/shaft-secure-handy-accurate-and-fast-transformer-inference/>, doi:10.14722/ndss.2025.242287.
- [23] Joon-Woo Lee, Eunsang Lee, Young-Sik Kim, and Jong-Seon No. Rotation key reduction for client-server systems of deep neural network on fully homomorphic encryption. In *Advances in Cryptology – ASIACRYPT 2023*, volume 14443 of *Lecture Notes in Computer Science*, pages 36–68. Springer, 2023. doi:10.1007/978-981-99-8736-8_2.
- [24] Baiyu Li and Daniele Micciancio. On the security of homomorphic encryption on approximate numbers. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 648–677. Springer, 2021. doi:10.1007/978-3-030-77870-5_23.
- [25] Baiyu Li, Daniele Micciancio, Mark Schultz, and Jessica Sorrell. Securing approximate homomorphic encryption using differential privacy. In *Advances in Cryptology – CRYPTO 2022*, volume 13507 of *Lecture Notes in Computer Science*, pages 560–589. Springer, 2022. doi:10.1007/978-3-031-15802-5_20.
- [26] Dacheng Li, Rulin Shao, Hongyi Wang, Han Guo, Eric P. Xing, and Hao Zhang. MPCFormer: fast, performant and private transformer inference with MPC. *CoRR*, abs/2211.01452, 2022. URL: <https://doi.org/10.48550/arXiv.2211.01452>, arXiv:2211.01452, doi:10.48550/ARXIV.2211.01452.
- [27] Zhengyi Li, Kang Yang, Jin Tan, Wen-jie Lu, Haoqi Wu, Xiao Wang, Yu Yu, Derun Zhao, Yancheng Zheng, Minyi Guo, and Jingwen Leng. Nimbus: Secure and efficient two-party inference for transformers. In *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems, NeurIPS 2024*, 2024. URL: http://papers.nips.cc/paper_files/paper/2024/hash/264a9b3ce46abdf572dcfe0401141989-Abstract-Conference.html, doi:10.52202/079017-0680.
- [28] Jing Liu and L. F. Zhang. Privacy-preserving and publicly verifiable matrix multiplication. *IEEE Transactions on Services Computing*, 16(3):2059–2071, 2023. doi:10.1109/TSC.2022.3215499.
- [29] Wen-jie Lu, Zhicong Huang, Zhen Gu, Jingyu Li, Jian Liu, Cheng Hong, Kui Ren, Tao Wei, and Wenguang

- Chen. BumbleBee: Secure two-party inference framework for large transformers. In *32nd Annual Network and Distributed System Security Symposium, NDSS 2025*. The Internet Society, 2025. doi:10.14722/ndss.2025.230057.
- [30] Jinglong Luo, Yehong Zhang, Zhuo Zhang, Jiaqi Zhang, Xin Mu, Hui Wang, Yue Yu, and Zenglin Xu. SecFormer: Fast and accurate privacy-preserving inference for transformer models via SMPC. In *Findings of the Association for Computational Linguistics, ACL 2024*, pages 13333–13348. Association for Computational Linguistics, 2024. URL: <https://doi.org/10.18653/v1/2024.findings-acl.790>, doi:10.18653/v1/2024.FINDINGS-ACL.790.
- [31] Jungho Moon, Zhanibek Omarov, Donghoon Yoo, Yongdae An, and Heewon Chung. Adaptive successive over-relaxation method for a faster iterative approximation of homomorphic operations. *IACR Cryptol. ePrint Arch.*, 2024:1366, 2024. URL: <https://eprint.iacr.org/2024/1366>.
- [32] 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*, pages 3765–3779, New York, NY, USA, 2025. ACM. doi:10.1145/3719027.3765150.
- [33] Qi Pang, Jinhao Zhu, Helen Möllering, Wenting Zheng, and Thomas Schneider. BOLT: privacy-preserving, accurate and efficient inference for transformers. In *IEEE Symposium on Security and Privacy, SP 2024*, pages 4753–4771. IEEE, 2024. doi:10.1109/SP54263.2024.00130.
- [34] 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, ACL 2025*, pages 11090–11111. Association for Computational Linguistics, 2025. URL: <https://aclanthology.org/2025.acl-long.543/>, doi:10.18653/v1/2025.acl-long.543.
- [35] Jai Hyun Park. Ciphertext-ciphertext matrix multiplication: Fast for large matrices. In *Advances in Cryptology – EUROCRYPT 2025*, volume 15608 of *Lecture Notes in Computer Science*, pages 153–180. Springer, 2025. doi:10.1007/978-3-031-91101-9_6.
- [36] Panagiotis Rizomiliotis and Aikaterini Triakosia. On matrix multiplication with homomorphic encryption. In *Proceedings of the 2022 Workshop on Encrypted Computing and Applied Homomorphic Cryptography*, pages 53–61. ACM, 2022. doi:10.1145/3560810.3564267.
- [37] Sinem Sav, Jean-Philippe Bossuat, Juan Ramón Troncoso-Pastoriza, Manfred Claassen, and Jean-Pierre Hubaux. Privacy-preserving federated neural network learning for disease-associated cell classification. *Patterns*, 3(5):100487, 2022. doi:10.1016/j.patter.2022.100487.
- [38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, pages 5998–6008. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.
- [39] Zihao Wang, Rongmao Chen, Xinwen Gao, Yi Wang, Lin Liu, Zixin Lan, Zhaoyu Wang, Shaojing Fu, Qiong Wang, and Xinyi Huang. STIP: Efficient and secure non-interactive transformer inference via compact packing. *Cryptology ePrint Archive, Paper 2026/174*, 2026. URL: <https://eprint.iacr.org/2026/174>.
- [40] Tianshi Xu, Wen-jie Lu, Jiangrui Yu, Yi Chen, Chenqi Lin, Runsheng Wang, and Meng Li. Breaking the layer barrier: Remodeling private transformer inference with hybrid CKKS and MPC. In *34th USENIX Security Symposium, USENIX Security 2025*, pages 2653–2672. USENIX Association, 2025. URL: <https://www.usenix.org/conference/usenixsecurity25/presentation/xu-tianshi>.
- [41] 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 *NDSS*, 2025. doi:10.14722/ndss.2025.230868.
- [42] 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 *The Fourteenth International Conference on Learning Representations (ICLR)*, 2026.

A Additional Symbol Table

Table 6 collects the notation used throughout.

Table 6: Notation.

Symbol	Meaning
N	RLWE ring dimension
L	Attention row length
d, H, d_h	Hidden dimension, number of heads, and head dimension
B	Interleaved batch size
A	Layer input
$W_Q, \dots, W_{F2}$	Linear weights
Q, K, V, S, O	Transformer intermediates
$\text{Rot}(\text{ct}, k)$	Slot rotation by k positions
$\text{Conj}(\text{ct})$	Ciphertext conjugation
$\boxplus, \boxtimes$	Ciphertext addition and multiplication
ℓ, T_ℓ, K_ℓ	Layer index, public scale, and softmax iteration count
$N_{\text{rot}}, N_{\text{mul}}$	Rotation and multiplication counts
S2C, C2S	SlotsToCoeffs and CoeffsToSlots.
CAdd	Ciphertext-ciphertext additions.
PMult	Plaintext-ciphertext multiplications.
C2PMM	The original BLAS-style PCMM reduction over coefficient-encode format.
SC2PMM	Slot-encoded C2PMM, THEMIS's execution path for PCMM.
$\mathbf{C}, \mathbf{C}_{\text{out}}$	Stacked component-coefficient matrices (input/output) in the PCMM proof.
COLPAIR, DIAGPAIR	Attention CCMM kernels for $QK^\top$ and $S \cdot V$.
HOISTROT	Hoisted evaluation of several rotations of one ciphertext.
B_Q, G_Q	BSGS factors for $QK^\top$, with $B_Q \cdot G_Q = L$.
B_S, G_S	BSGS factors for $S \cdot V$, with $B_S \cdot G_S = L/2$.
μ, Var	Row mean and variance.
D_t	Softmax denominator at stage t .
$\alpha(D_t)$	Row-wise scale left by a non-final reciprocal.
Δ, s, q	CKKS scaling factor, secret key, and ciphertext modulus.
$a_\ell(x)$	Layer-wise denominator-conditioning offset.
β_ℓ, c_ℓ	Offset calibration parameters.
C, d_π	Public first-denominator scale and numerical denominator target.
$P_t, P_{t,\ell}^{\text{lo}}$	Generic and deployed non-final relative reciprocal polynomials.
$r_t, \hat{r}_t$	Exact and approximate reciprocal kernels.
$\hat{r}_t^{\text{hi}}$	Final high-precision reciprocal kernel.
ARECIP	Reciprocal stage kernel.
N_{BLAS}	Backend BLAS call count.

B Accuracy Decomposition

Table 7 adds the plaintext GELU, LayerNorm, and softmax approximations before encrypted execution for the same trained model, using the complete development sets of 277 RTE, 872 SST-2, and 5463 QNLI examples.

Table 7: Accuracy decomposition of the THEMIS pipeline.

Dataset	Baseline	G	G-LN	G-LN-S	Encrypted
RTE	72.56	72.56	72.20	72.20	71.48
SST-2	92.43	92.43	92.43	92.43	92.32
QNLI	91.54	91.54	91.56	91.51	91.36

C Kernel Instantiation Parameters

Table 9 records the concrete parameters used to instantiate the symbolic matrix-multiplication algorithms from Section 4

Table 8: End-task accuracy of encrypted Transformer systems. Each difference is against that system's own plaintext baseline.

Dataset	System	Orig.	Plaintext	Encrypted	Difference
RTE	THOR [32]	✓	72.20	71.12	−1.08
	Euston [13]	✓	70.04	70.04	0.00
	MOAI [42]	✓	65.7	65.5	−0.2
	Powerformer [34]	×	73.29	73.29	0.00
	THEMIS	✓	72.56	71.48	−1.08
SST-2	THOR [32]	✓	91.51	90.71	−0.80
	Euston [13]	✓	93.23	93.12	−0.11
	MOAI [42]	✓	93.9	93.8	−0.1
	THEMIS	✓	92.43	92.32	−0.11
QNLI	Euston [13]	✓	91.25	91.10	−0.15
	MOAI [42]	✓	90.7	90.5	−0.2
	THEMIS	✓	91.54	91.36	−0.18

All systems use BERT-base with $L = 128$; data are from THOR Table 7, Euston Table 4, MOAI Table 6, and Powerformer Table 9. The $\times$ marks Powerformer's BPMMax replacement of softmax with distillation retraining. Boldface is the best encrypted result per task among $\checkmark$ systems; baselines are independently fine-tuned.

Table 10: Online batch time across the tested batch sizes.

B	L	Online batch time (s)	Amortized per input (s)
1	128	44.3099	44.3099
64	128	44.6257	0.697276
256	128	44.6120	0.174266

in the BERT-base evaluation. The BTS configuration restores up to 16 computational levels with a 50-bit scaling factor, and each call targets the level count the following stages need. The secret is sparse ternary with Hamming weight $h = 192$, a common BTS choice [12].

Table 9: Kernel parameters used in the evaluated BERT-base setting.

Parameter	Value
Ring dimension	2^{16}
CKKS slots	32768
Scaling factor	50 bits
Secret distribution	sparse ternary ($h = 192$)
Sequence length	$L = 128$
Head dimension	$d_h = 64$
Batch size	$B = 256$
$QK^\top$ BSGS factorization	$B_Q = 16, G_Q = 8$
$S \cdot V$ BSGS factorization	$B_S = 8, G_S = 8$

D Batch Scan

Table 10 reports the online batch time measured across the tested batch sizes. The batch time is nearly independent of B , so the per-input cost falls as $1/B$.

E Slot-Encode C2PMM Execution: Full Proof

Setting. Let $R_q = \mathbb{Z}_q[x]/(x^N + 1)$ and let $\gamma(c) = (\vec{a}, \vec{b})^\top$ be the $2N$ -coefficient vector of $c = (a, b) \in R_q^2$. For a slot-side linear map T , write $T \odot c$ for its homomorphic action and define its lift by $\gamma(T \odot c) = \hat{T}\gamma(c)$. Let the columns of $M \in \mathbb{C}^{(N/2) \times \kappa}$ be encrypted by $c_1, \dots, c_\kappa$, and collect their two component vectors as $\mathbf{C} = (\mathbf{A}^\top, \mathbf{B}^\top)^\top \in \mathbb{Z}_q^{2N \times \kappa}$. The public weight is $\mathbf{W} \in \mathbb{Z}_q^{\kappa \times n}$.

Lemma 1. *For every ciphertext c , $I \odot c = c$ and $\mathbf{0} \odot c = 0$.*

Proof. The diagonal representation of I has one nonzero diagonal, encoded by the plaintext 1, so its product returns both components of c unchanged. Every zero-transform diagonal encodes zero, so its accumulated output is zero. $\square$

Lemma 2 (Coefficient lift). *Every slot-side linear map T induces a matrix $\hat{T} \in \mathbb{Z}_q^{2N \times 2N}$ satisfying $\gamma(T \odot c) = \hat{T}\gamma(c)$.*

Proof. For ciphertexts c_1, c_2 and $r \in \mathbb{Z}_q$, the homomorphic diagonal evaluation of T satisfies $T \odot (c_1 + c_2) = T \odot c_1 + T \odot c_2$ and $T \odot (rc_1) = r(T \odot c_1)$. It is therefore a linear endomorphism of the $2N$ -dimensional coefficient module, represented by $\hat{T}$. $\square$

Corollary 1. *For slot maps T_1, T_2 , their lifts satisfy $\widehat{T_1 T_2} = \hat{T}_1 \hat{T}_2$ on ciphertext coefficient vectors.*

Proof. Applying T_2 and then T_1 gives $\gamma(T_1 \odot (T_2 \odot c)) = \hat{T}_1 \hat{T}_2 \gamma(c)$. Its slots contain $(T_1 T_2)m$, so Lemma 2 identifies the same action with $\widehat{T_1 T_2}$. $\square$

Let the canonical-embedding blocks be U_0, U_1 and write

$$U = \begin{bmatrix} U_0 & U_1 \\ \overline{U}_0 & \overline{U}_1 \end{bmatrix}, \quad U^{-1} = \frac{1}{N} \begin{bmatrix} \overline{U}_0^\top & U_0^\top \\ \overline{U}_1^\top & U_1^\top \end{bmatrix}.$$

For each input ciphertext, S2C forms $t = U_0 \odot c + U_1 \odot \bar{c}$ and $\bar{t} = \overline{U}_0 \odot c + \overline{U}_1 \odot \bar{c}$. Applying Lemma 2 column-wise therefore gives

$$\mathbf{C}' = \hat{U}_0 \mathbf{C} + \hat{U}_1 \overline{\mathbf{C}}, \quad \overline{\mathbf{C}'} = \hat{\overline{U}}_0 \mathbf{C} + \hat{\overline{U}}_1 \overline{\mathbf{C}}.$$

The coefficient-side product acts on the column index, so

$$\mathbf{C}' \mathbf{W} = \hat{U}_0 (\mathbf{C} \mathbf{W}) + \hat{U}_1 (\overline{\mathbf{C}} \mathbf{W}),$$

and the analogous equality holds for $\overline{\mathbf{C}'} \mathbf{W}$. Thus right-multiplication by $\mathbf{W}$ commutes with every lifted S2C block without changing the ring-index action. C2S applies the first block row of U^{-1} . Substitution yields

$$\begin{aligned} \mathbf{C}_{\text{out}} &= N^{-1} (\hat{\overline{U}}_0^\top \hat{U}_0 + \hat{U}_0^\top \hat{\overline{U}}_0) (\mathbf{C} \mathbf{W}) \\ &\quad + N^{-1} (\hat{\overline{U}}_0^\top \hat{U}_1 + \hat{U}_0^\top \hat{\overline{U}}_1) (\overline{\mathbf{C}} \mathbf{W}). \end{aligned}$$

The first block row of $U^{-1}U = I$ gives the slot-side identities $N^{-1}(\overline{U}_0^\top U_0 + U_0^\top \overline{U}_0) = I$ and $N^{-1}(\overline{U}_0^\top U_1 + U_0^\top \overline{U}_1) = \mathbf{0}$. By Corollary 1 and Lemma 1, their coefficient lifts are exactly the two identities in Proposition 1. Hence $\mathbf{C}_{\text{out}} = \mathbf{C} \mathbf{W} = (\mathbf{A} \mathbf{W}, \mathbf{B} \mathbf{W})^\top$; its ciphertext columns encrypt the columns of $M \mathbf{W}$ in slots, completing the proof.

F Exact Coefficient-Side Matrix Multiplication

The axis separation used in the main text has a direct implementation consequence. The j -th column of $\mathbf{A}$ and $\mathbf{B}$ is the coefficient vector of a ciphertext in slot-encode format, and the numerical kernel needs only these coefficient vectors and the public weight matrix. The same bit-chunked BLAS routine therefore accepts inputs originating in either coefficient-encode or slot-encode format; the encoding label remains ciphertext-level bookkeeping and does not enter the backend.

With column-packed interleaved inputs, the runtime path applies the public weight to the two coefficient matrices induced by the ciphertext components and performs modular integer additions and multiplications inside C2PMM. Following the exact modular reduction of the plaintext-reduction line [3], let $\mathbf{A} \in \mathbb{Z}_q^{m \times \kappa}$ collect coefficient vectors and let $\mathbf{W} \in \mathbb{Z}_q^{\kappa \times n}$ be the modular-integer weight matrix. To compute $\mathbf{C} = \mathbf{A} \mathbf{W} \bmod q$ exactly, decompose both operands into small bit chunks:

$$\mathbf{A} = \sum_{i=0}^{c_L-1} \mathbf{A}_i 2^{ib_L}, \quad \mathbf{W} = \sum_{j=0}^{c_R-1} \mathbf{W}_j 2^{jb_R}. \quad (11)$$

Here $\mathbf{A}_i$ and $\mathbf{W}_j$ are the i -th and j -th bit chunks with widths b_L and b_R . Algorithm 1 takes $b_L = b_R = b$, hence $c_L = c_R = n_{\text{chk}}$. If

$$\kappa \cdot 2^{b_L+b_R} < 2^{53}, \quad (12)$$

then each partial product $\mathbf{A}_i \mathbf{W}_j$ is exactly representable in IEEE-754 double precision and one `dgemm` call computes it without rounding error. Recombining the partial products gives

$$\mathbf{C} = \left(\sum_{i,j} \mathbf{A}_i \mathbf{W}_j \cdot 2^{ib_L+jb_R} \right) \bmod q. \quad (13)$$

Thus each floating-point GEMM stays in the integer-precise range, and the final reduction recovers the exact modular product.

G Detailed CCMM Derivations

$QK^\top$. In the interleaved layout, a diagonal displacement preserves its batch residue under the rotation

$$\text{rot}(\delta) = \delta \cdot B. \quad (14)$$

The real-only computation evaluates every diagonal-column pair and has

$$N_{\text{rot}}^{\text{naive}} = Ld_h, \quad N_{\text{mul}}^{\text{naive}} = Ld_h. \quad (15)$$

A generic use of the imaginary lane first constructs

$$K_j^{\text{merge}} = K_j + i \cdot \text{Rot}(K_j, \frac{L}{2}B), \quad (16)$$

which halves the diagonal loop but adds one pre-rotation per column:

$$N_{\text{rot}}^{\text{base}} = \frac{L}{2}d_h + d_h, \quad N_{\text{mul}}^{\text{base}} = \frac{L}{2}d_h. \quad (17)$$

Under a BSGS split with $BG = L/2$, this generic merge costs $d_h(B + G)$ rotations, against $\frac{d_h}{2}(B_Q + G_Q)$ with $B_Q G_Q = L$ for COLPAIR in (22). COLPAIR instead pairs the column-contraction halves,

$$Q_j^{\text{pair}} = Q_j + iQ_{j+d_h/2}, \quad K_j^{\text{pair}} = K_j - iK_{j+d_h/2}, \quad (18)$$

and evaluates

$$Q_j^{\text{pair}} \cdot \text{Rot}(K_j^{\text{pair}}, \delta B). \quad (19)$$

Its real component is

$$Q_j \cdot \text{Rot}(K_j, \delta B) + Q_{j+d_h/2} \cdot \text{Rot}(K_{j+d_h/2}, \delta B), \quad (20)$$

because the cross terms occupy the imaginary component. Hence

$$N_{\text{rot}}^{\text{colpair}} = L \cdot \frac{d_h}{2}, \quad N_{\text{mul}}^{\text{colpair}} = L \cdot \frac{d_h}{2}. \quad (21)$$

Writing $\delta = gB_Q + b$ with $L = B_Q G_Q$ reuses baby rotations and restores the giant displacement after accumulation, giving

$$N_{\text{rot}}^{\text{QK,bsgs}} = \frac{d_h}{2}(B_Q + G_Q), \quad N_{\text{mul}}^{\text{QK,bsgs}} = L \cdot \frac{d_h}{2}. \quad (22)$$

$S \cdot V$. For diagonal ciphertexts S_δ and a column ciphertext V_j , the direct output is

$$O_j = \sum_{\delta=0}^{L-1} S_\delta \cdot \text{Rot}(V_j, \delta B), \quad (23)$$

with Ld_h rotations and products. DIAGPAIR forms

$$S_\delta^{\text{pair}} = S_\delta + iS_{\delta+L/2}, \quad \delta \in \{0, \dots, L/2-1\}, \quad (24)$$

and pairs the corresponding value positions through

$$V_j^{\text{merge}} = V_j - i \cdot \text{Rot}(V_j, \frac{L}{2}B). \quad (25)$$

The real component collects both diagonals, so

$$N_{\text{rot}}^{\text{SV,dp}} = d_h + \frac{L}{2}d_h, \quad N_{\text{mul}}^{\text{SV,dp}} = \frac{L}{2}d_h. \quad (26)$$

Table 11: Cumulative construction of the two attention CCMM kernels over their unpaired baselines.

Stage	#KS		Speedup	
	COLPAIR	DIAGPAIR	COLPAIR	DIAGPAIR
Real (unpaired)	8192	8192	1.00×	1.00×
+ complex-lane merge	4096	4160	1.98×	1.89×
+ BSGS	768	1024	4.67×	3.82×
+ hoisting (deployed)	944	1592	4.71×	4.06×

#KS is a static per-head count. The first three rows give the logical rotation schedule of each variant, in which every rotation performs one automorphism key switch. The deployed row gives the key switches the hoisted implementation issues, which additionally include the per-output relinearizations and final rotations every variant performs, so it is not on the same scale as the rows above it. Hoisting issues no additional rotations and reuses one digit decomposition across the targets of the same source. Speedups are ratios of mean measured kernel times over each kernel's unpaired variant. BSGS uses (16, 8) for $QK^\top$ and (8, 8) for $S \cdot V$.

With $\delta = gB_S + b$ and $L/2 = B_S G_S$, pre-rotate $\hat{S}_{g,b} = \text{Rot}(S_{gB_S+b}^{\text{pair}}, -gB_S B)$. Then the BSGS accumulation is

$$O_j = \text{Re}_2 \sum_{g=0}^{G_S-1} \text{Rot}\left(\sum_{b=0}^{B_S-1} \hat{S}_{g,b} \boxtimes \text{Rot}(V_j^{\text{merge}}, bB), gB_S B\right), \quad (27)$$

where each baby rotation of V_j^{merge} is shared across giant steps. The resulting dominant counts are

$$N_{\text{rot}}^{\text{SV,bsgs}} = d_h(B_S + G_S), \quad N_{\text{mul}}^{\text{SV,bsgs}} = \frac{L}{2}d_h, \quad (28)$$

minimized at $B_S = G_S = \sqrt{L/2}$.

G.1 Cumulative CCMM Ablation

Table 11 shows a cumulative ladder. Merging the complex lanes halves the key-switch count and yields $1.98\times$ for $QK^\top$ and $1.89\times$ for $S \cdot V$; it cuts $S \cdot V$ time by 47.2%. BSGS, the largest step, reaches $4.67\times$ and $3.82\times$ and cuts $S \cdot V$ time by another 50.5%. Hoisting adds 0.8% and 5.9%. That margin is small by construction, because Figure 2 places the hoisting advantage near $1.29\times$ at the chain bottom, which is where these kernels run. Generic merging reaches $2.07\times$, or $3.28\times$ with BSGS(8, 8). COLPAIR's count of 4096 is only 1.5% below the 4160 that generic merging needs for $QK^\top$, and the 4.6% timing gap lies within run-to-run variation. The advantage is structural. The wider decomposable index enables (16, 8), whose endpoint is 29.8% faster, matching Section 4.2's $1.33\times$ reduction.

In Table 12, hoist-off is best at (16, 8) and $4.67\times$; hoist-on at (8, 16) and $5.00\times$. Deployed (16, 8)-on reaches $4.71\times$, 6.3% below the optimum, but uses the fewest hoist-on key switches (944). Deployed $S \cdot V$ split (8, 8) is best off and on. Hoisting is not universal. $QK^\top$ (32, 4) changes from $4.05\times$ to $3.57\times$, and $S \cdot V$ (4, 16) from $3.23\times$ to $3.13\times$, matching Section 4.2.

Table 12: BSGS and hoisting sweep for the two attention CCMMs.

Kernel	(B, G)	#KS		Speedup	
		off	on	off	on
$QK^\top$	(4, 32)	1152	1340	$3.86\times$	$4.62\times$
	(8, 16)	768	952	$4.40\times$	$5.00\times$
	(16, 8) [†]	768	944	$4.67\times$	$4.71\times$
	(32, 4)	1152	1312	$4.05\times$	$3.57\times$
$S\cdot V$	(4, 16)	1280	2364	$3.23\times$	$3.13\times$
	(8, 8)	1024	1592	$3.82\times$	$4.06\times$
	(16, 4)	1280	1584	$3.76\times$	$4.00\times$

#KS and Speedup follow Table 11. [†] marks the deployed $QK^\top$ split; the deployed $S\cdot V$ split is (8, 8). The off column is the logical rotation schedule and the on column is the key switches the hoisted implementation issues, so the two are not on the same scale. Speedup is the measured quantity.

G.2 BSGS and Hoisting Sweep

Table 12 covers the tested split range for both attention kernels, with [†] marking the deployed $QK^\top$ split.

H Polynomial Approximation Details

This appendix records the approximation choices used by the active THEMIS pipeline. Softmax uses Chebyshev polynomials and aSOR as described below, while LayerNorm and GELU use the same polynomial evaluation infrastructure without square-and-normalize.

H.1 Exponential Approximation

The exponential is approximated directly by Chebyshev interpolation and evaluated with BSGS. The deployed path evaluates the polynomial on the post-offset logit scaled by $2^{-K_\ell-1}$; the first square step then produces the effective 2^{-K_ℓ} factor. This is equivalent in exact arithmetic and keeps the polynomial input range milder. No separate scale-and-square decomposition is used inside the exponential polynomial.

H.2 Low-Precision Relative Reciprocals

The non-final normalization stages use low-degree Chebyshev polynomials calibrated by relative reciprocal error. For a denominator interval I_{nf} , THEMIS chooses a polynomial P_t by the objective

$$P_t^* = \arg \min_{P_t} \max_{D \in I_{\text{nf}}} |DP_t(D) - 1|. \quad (29)$$

This objective matches the role of a non-final reciprocal. If

$$D\hat{r}(D) \in [1 - \eta, 1 + \eta], \quad (30)$$

then the normalized row is scaled by a factor in the same interval. The next square-and-normalize stage removes this

row-wise scale, while the denominator range can inflate by at most

$$\left(\frac{1+\eta}{1-\eta}\right)^2 \quad (31)$$

relative to the exact normalized path. Non-final stages therefore require reciprocal-like conditioning, but not high-fidelity inversion.

H.3 High-Precision Final Reciprocal

The final normalization has no later stage that can cancel row-wise scale error. THEMIS therefore uses a Chebyshev initializer followed by a small number of aSOR refinement rounds [31]. For a denominator $z > 0$,

$$y_0 = P_d(z), \quad z_0 = zy_0, \quad (32)$$

and the refinement is

$$y_{t+1} = k_t y_t (2 - k_t z_t), \quad z_{t+1} = k_t z_t (2 - k_t z_t), \quad (33)$$

where k_t is a plaintext relaxation parameter and $k_t = 1$ gives the standard Goldschmidt iteration. The final reciprocal approximation is

$$\text{ARECIP}(z) := y_T. \quad (34)$$

The interval, degree, and refinement count are calibrated jointly with the softmax iteration schedule.

H.4 LayerNorm and GELU Approximation

Compared with softmax, the remaining nonlinearities are simpler because they do not combine exponential amplification with repeated row-wise denominators. For an input vector x , LayerNorm computes $\text{LayerNorm}(x) = ((x - \mu(x))/\sqrt{\text{Var}(x) + \epsilon})\gamma + \beta$, where $\mu(x)$ and $\text{Var}(x)$ are the mean and variance of x , and γ, β are the standard affine parameters. THEMIS keeps this standard target and approximates the inverse square root on a calibrated nonnegative variance interval. THEMIS approximates GELU directly by a Chebyshev polynomial on a calibrated asymmetric interval. Neither operation requires offset-conditioned square-and-normalize. Both effects are included in Table 7.

I Additional Details of the Softmax Design

This appendix gives the softmax details omitted from Section 5, including offset instantiation, reciprocal calibration, correctness, and offline parameter selection.

I.1 Instantiation of the Layer-Wise Offset

In the main text, the layer-wise offset is written abstractly as $a_\ell(x)$. We instantiate it through the following form:

$$a_\ell(x; \theta_\ell) = \mu(x) + \beta_\ell \text{Var}(x) + c_\ell, \quad \theta_\ell = (\beta_\ell, c_\ell), \quad (35)$$

Table 13: Calibrated square-and-normalize iteration count by layer.

Layer	L0	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	L11
K_ℓ	1	1	2	1	1	1	1	1	1	1	1	1

where $\mu(x)$ and $\text{Var}(x)$ denote the row mean and variance. Both statistics are computed homomorphically by additions, rotations, and polynomial evaluation. The parameters β_ℓ and c_ℓ are calibrated offline for each layer and task.

Given the offset form, the post-offset logits are

$$u_i = x_i - a_\ell(x; \theta_\ell). \quad (36)$$

The offset is not trained to approximate a row maximum. Instead, it is chosen to compress the interval ratio of the first denominator and to align its empirical lower endpoint to a fixed numerical target. Let

$$\begin{aligned} \text{base}(x) &= \log \left(\frac{1}{L} \sum_i \exp \left(\frac{x_i - \mu(x)}{T_\ell} \right) \right), \\ v(x) &= \text{Var}(x), \end{aligned} \quad (37)$$

where $T_\ell = 4$ for zero-based Layer 2 and $T_\ell = 2$ for the other layers in our BERT-base deployment. For a candidate β , we evaluate

$$\begin{aligned} w_\ell(\beta) &= \max_x \left(\text{base}(x) - \frac{\beta v(x)}{T_\ell} \right) \\ &\quad - \min_x \left(\text{base}(x) - \frac{\beta v(x)}{T_\ell} \right), \end{aligned} \quad (38)$$

and select β_ℓ to minimize this empirical width on the calibration set.

After β_ℓ is fixed, the constant c_ℓ is chosen to align the empirical lower endpoint of the scaled first denominator to a fixed numerical target $d_\star = 0.195$. Writing C for the public scaling constant in D_0 and

$$z_\ell(x) = \text{base}(x) - \frac{\beta_\ell v(x)}{T_\ell}, \quad (39)$$

we set

$$c_\ell = T_\ell \left(\min_x z_\ell(x) + \log C - \log d_\star \right). \quad (40)$$

This target is not required by softmax semantics. It is a conditioning parameter that keeps the reciprocal input away from excessively small values while preserving the exact row-shift invariance of softmax. The selected β_ℓ and c_ℓ values used in our evaluation follow from Algorithm 5.

I.2 Relative-Error Calibration for Non-Final Reciprocals

For a non-final reciprocal stage, THEMIS records an empirical denominator interval $I_{t,\ell}$ after applying the previously

Algorithm 5 Offline calibration of the softmax offset.

Input: Calibration rows for layer ℓ , scale T_ℓ , public scale C , target $d_\star$

Output: Offset parameters (β_ℓ, c_ℓ)

- 1: Compute $\text{base}(x)$ and $v(x)$ for all calibration rows
 - 2: Select $\beta_\ell \leftarrow \arg \min_\beta [\max_x (\text{base}(x) - \beta v(x)/T_\ell) - \min_x (\text{base}(x) - \beta v(x)/T_\ell)]$
 - 3: Set $c_\ell \leftarrow T_\ell [\min_x (\text{base}(x) - \beta_\ell v(x)/T_\ell) + \log C - \log d_\star]$
 - 4: **return** (β_ℓ, c_ℓ)
-

calibrated stages. A candidate polynomial P_t is accepted when

$$\max_{D \in I_{t,\ell}} |DP_t(D) - 1| \leq \eta_t \quad (41)$$

and the induced next-stage denominator interval remains inside the calibration envelope for the following reciprocal. This rule is more appropriate than absolute reciprocal error because the next normalization removes row-wise scale. The final reciprocal is calibrated by ordinary approximation error to $1/D$, since its output directly determines the softmax probabilities.

I.3 Correctness of the Precision-Staged Schedule

We now justify the square-and-normalize schedule used in Section 5.2. The exact mean-normalized target is

$$p_i^\star = \frac{\exp(u_i)}{\text{mean}_j \exp(u_j)}. \quad (42)$$

This differs from conventional sum-normalized softmax by the public factor L .

For a fixed layer ℓ , let the exact provisional normalization be

$$y_i^{(0)} = \frac{\exp(u_i/2^{K_\ell})}{\text{mean}_j \exp(u_j/2^{K_\ell})}. \quad (43)$$

The exact refinement update is

$$y_i^{(t)} = \frac{(y_i^{(t-1)})^2}{\text{mean}_j (y_j^{(t-1)})^2}, \quad t = 1, \dots, K_\ell. \quad (44)$$

Proposition 2. Under exact arithmetic, the K_ℓ -step update in (44) transforms (43) into the mean-normalized target:

$$y_i^{(K_\ell)} = p_i^\star. \quad (45)$$

Proof. We prove by induction that

$$y_i^{(t)} = \frac{\exp(u_i/2^{K_\ell-t})}{\text{mean}_j \exp(u_j/2^{K_\ell-t})}. \quad (46)$$

The base case $t = 0$ is exactly (43). Assume that (46) holds for some t . Then

$$\begin{aligned} (y_i^{(t)})^2 &= \frac{\exp(2u_i/2^{K_\ell-t})}{(\text{mean}_j \exp(u_j/2^{K_\ell-t}))^2} \\ &= \frac{\exp(u_i/2^{K_\ell-(t+1)})}{(\text{mean}_j \exp(u_j/2^{K_\ell-t}))^2}. \end{aligned}$$

After normalization over i , the common denominator cancels, and we obtain

$$y_i^{(t+1)} = \frac{\exp(u_i/2^{K_\ell-(t+1)})}{\text{mean}_j \exp(u_j/2^{K_\ell-(t+1)})}.$$

This proves the induction. Setting $t = K_\ell$ yields (45). $\square$

Proposition 3 (Scale cancellation in non-final stages). *Let a non-final normalization stage have denominator D_t and use an approximate reciprocal $\hat{r}_t(D_t)$ such that $D_t \hat{r}_t(D_t) = \alpha(D_t) > 0$. Then the output of this stage differs from the exact normalized output only by a row-wise positive scalar $\alpha(D_t)$. If a later stage performs an exact normalization, this scalar is canceled and does not change the softmax value.*

Proof. Let

$$y_i = \frac{(p_i^{(t-1)})^2}{D_t}$$

be the exact normalized output of the stage. The approximate output is

$$\hat{p}_i^{(t)} = (p_i^{(t-1)})^2 \hat{r}_t(D_t) = \alpha(D_t) y_i.$$

A later exact square-and-normalize stage removes this common scale:

$$\frac{(\alpha(D_t) y_i)^2}{\text{mean}_j (\alpha(D_t) y_j)^2} = \frac{y_i^2}{\text{mean}_j y_j^2}.$$

Thus non-final scale error is not a semantic error in the idealized schedule. It only affects whether subsequent denominators stay in a reciprocal-friendly interval. $\square$

The next denominator after a non-final approximate reciprocal satisfies

$$\hat{D}_{t+1} = (D_t \hat{r}_t(D_t))^2 D_{t+1}^{\text{exact}}. \quad (47)$$

Therefore arbitrary cheap conditioners are not sufficient. THEMIS calibrates non-final reciprocals by bounding $D_t \hat{r}_t(D_t)$, while reserving high-precision reciprocal accuracy for the final normalization.

I.4 Layer-wise deployment outputs

The active method produces three offline outputs for each layer, namely the offset parameters β_ℓ and c_ℓ and the iteration count K_ℓ . Algorithm 5 and Section I.1 define the offset calibration. After fixing the offset, THEMIS evaluates candidate square-and-normalize iteration counts on the calibration set and chooses the smallest stable value. Table 13 reports the resulting K_ℓ schedule.