SCALING UP FHE-BASED LLM INFERENCE: HIGHER THROUGHPUT AND LONGER INPUTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Fully homomorphic encryption (FHE) enables non-interactive confidential inference of large language models (LLMs), but existing solutions scale poorly: they are limited to small models or to a few input tokens, and activation outliers force high-degree polynomial approximations of non-linear layers. We scale up FHE-based LLM inference in two directions. First, for fully encrypted prompts, we suppress outliers without retraining via attention-sink prefixes and additional orthogonal rotations, introduce a numerically stable sum-of-squares polynomial evaluation for sparsely packed ciphertexts that speeds up SoftMax, and combine these with fast BLAS-based homomorphic linear algebra whose encoding conversions are mostly fused into bootstrapping. Second, when a long context is public and only the query is sensitive, we process the public prefix in the clear and encrypt only the query; the resulting wide plaintext-ciphertext attention is evaluated without bootstrapping on its wide path, using a depth-1 matrix multiplication and a shallow SoftMax. Our implementation, Sylph, runs the prefill of Llama-3.1-8B on a 128-token encrypted prompt in 20 s on eight RTX PRO 6000 GPUs, versus 134 s for the prior state of the art on eight B200 GPUs ($5.8\times$ faster on the same four B200 GPUs), and processes a 4096-token prompt whose last 128 tokens are encrypted in 64 s.

1 INTRODUCTION

When a client queries a Large Language Model (LLM), the query and its response often contain sensitive data. Secure Multi-Party Computation (MPC) (Hou et al., 2023; Wu et al., 2024; Kei & Chow, 2025; Cheng et al., 2025; Gupta et al., 2024) enables private inference but requires many communication rounds and non-collusion assumptions. Fully Homomorphic Encryption (FHE) needs no interaction, but its cost has confined CKKS-based Transformer inference (Chen et al., 2022; Moon et al., 2025; Park et al., 2025; Zhang et al., 2025a;b) to small models or tiny inputs (e.g., $\leq$8 input tokens for Llama-3-8B (Zhang et al., 2025a;b)); even the state-of-the-art multi-GPU system (Jayashankar et al., 2026) needs 134 s for a 128-token prefill on eight B200 GPUs.

We present *Sylph*, a multi-GPU FHE system for LLM inference that addresses three bottlenecks:

- **Activation outliers.** FHE evaluates non-linear layers with polynomials that are accurate only on bounded intervals, whereas LLM activations exceed 2200 in magnitude (Sun et al., 2024); recent 8B-scale systems (Zhang et al., 2025a;b; Jayashankar et al., 2026) report neither activation ranges nor fidelity to plaintext, and replaying the released approximations of two of them on Llama-3 activations shows inputs outside their approximation domains (Appendix L). We suppress outliers without retraining using attention-sink prefixes, additionally applying orthogonal rotations; together they bring every approximation interval below 40 and enable lower-degree approximations that match plaintext perplexity.
- **Linear layers and SoftMax.** Linear projections dominate the cost at 8B scale, and fast BLAS-based algorithms (Bae et al., 2024; Cheon et al., 2025; Park, 2025) each expect a different ciphertext encoding. We fuse most encoding conversions into bootstrappings that are needed anyway, choose packing layouts whose bit-reversal permutations produce the layouts these kernels expect, and extend ring switching to the SinC encoding. For SoftMax normalization on sparsely packed ciphertexts, we split a polynomial into a sum of two squares of half-degree polynomials evaluated in otherwise idle slots, which keeps intermediate values bounded under CKKS.

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

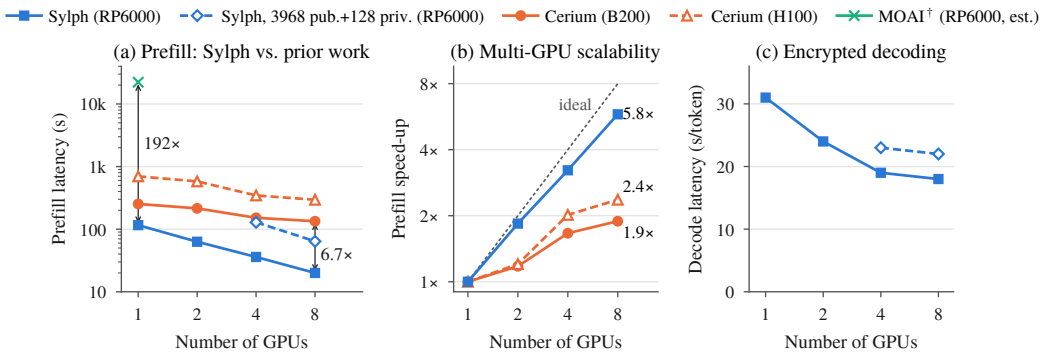

Figure 1: End-to-end latency of encrypted Llama-3.1-8B inference (128-token encrypted prompt unless noted): **(a)** prefill vs. prior work, **(b)** prefill speed-up over one GPU, **(c)** per-token decoding over the 32 decoder layers. Cerium timings are from Jayashankar et al. (2026); †MOAI timings are estimates as the workload exceeds GPU memory; its released approximations do not produce meaningful outputs on Llama-3 activations (Appendix L). Raw numbers are in Table 7.

- **Long contexts.** Encrypting every token makes the cost grow with the whole prompt, so published results stop at 128 tokens. When a long context is public and only a short query is private (e.g., RAG pipelines (Lewis et al., 2020), agentic workflows, shared system prompts), Sylph prefills the public context in the clear and only the query under FHE; the query and every generated token remain encrypted (Section 2.3). The public context then enters only through plaintext–ciphertext attention (PC-attention), whose wide intermediate state we evaluate without bootstrapping, using a depth-1 adaptation of the matrix multiplication of Jiang et al. (2018) and a calibrated shallow SoftMax.

Sylph also selects output tokens homomorphically and scales to eight GPUs with a $5.8\times$ prefill speed-up over one GPU (Section 5). On a 128-token encrypted prompt, it completes prefill in 20 s on eight RTX PRO 6000 GPUs, versus 134 s for Cerium (Jayashankar et al., 2026) on eight B200 GPUs, and is $5.8\times$ faster than Cerium on the same four B200 GPUs (Figure 1); with 3968 public and 128 encrypted tokens, prefill takes 64 s, a regime no prior FHE Transformer reaches.

2 Preliminaries

We denote plaintext–ciphertext and ciphertext–ciphertext matrix multiplications by PCMM and CCMM, matrix–vector ones by PCMv and CCMv, and coordinate-wise multiplication by $\odot$; further notation is in Appendix B.

2.1 The CKKS Scheme and Encrypted Linear Algebra

CKKS (Cheon et al., 2017) is an RLWE-based FHE scheme enabling approximate arithmetic over real vectors encrypted in the ring $\mathcal{R}_N = \mathbb{Z}[X]/(X^N+1)$. Data vectors are embedded into plaintext polynomials via an *encoding*, which dictates how operations act on the underlying messages. We leverage three encoding formats: (1) *Slot encoding* ($\mathsf{Ecd}_{\mathrm{slot}}$), where arithmetic acts coordinate-wise (SIMD), ideal for evaluating non-linear functions; (2) *Coefficient encoding* ($\mathsf{Ecd}_{\mathrm{coeff}}$), where multiplication acts as convolution, required by fast BLAS-accelerated PCMM (Bae et al., 2024); and (3) *Slots-in-Coefficients encoding* ($\mathsf{Ecd}_{\mathsf{SinC}_{k,N}}$) (Cheon et al., 2025), an intermediate format designed for batched CCMM. Ciphertexts encrypting fewer slots than capacity ($d \ll N$) are termed *sparsely packed* (or *slim*). Details on cryptographic primitives (encryption, rotation, key-switching) and more formal definitions are deferred to Appendix B.

Homomorphic multiplications consume modulus levels, eventually requiring costly *bootstrapping* (Cheon et al., 2018). Because bootstrapping evaluates homomorphic DFTs, coefficient and SinC formats can be accessed mid-stream, so format conversions can be fused into bootstrapping instead of being performed separately. Prior FHE Transformers rely on slot-based matrix multiplication (Jiang et al., 2018) with high multiplicative depth, whereas BLAS-based algorithms reduce

PCMM (Bae et al., 2024), CCMM (Park, 2025) and batched {P,C}CMM (Cheon et al., 2025) to cleartext matrix products on small moduli; for decoding, BSGS-optimized PCMv and CCMv (Halevi & Shoup, 2014; He et al., 2024) minimize key-switchings. We target Llama-3.1-8B-Instruct (Dubey et al., 2024) (Llama-3.1-8B for short; RMSNorm, RoPE, grouped-query attention, SwiGLU), whose prefill is dominated by matrix–matrix and decoding by matrix–vector products (Appendix C).

2.2 Homomorphic SoftMax Evaluation

SoftMax is costly under CKKS because exponentiation and division require deep polynomial approximations. We build upon the two-track algorithm of Cho et al. (2024). For a SoftMax instance with inputs $(x_i)_{0\le i<d}$ shifted into $[-M, 0]$, it approximates $y_i^{(0)} \approx \exp(x_i/2^k)$ by a low-degree polynomial over $[-M/2^k, 0]$ and then applies k normalize-and-square steps $y_i^{(j)} = \left(y_i^{(j-1)}/\|\mathbf{y}^{(j-1)}\|_2\right)^2$ for $1 \le j \le k$, so that $\mathbf{y}^{(k)} = \mathsf{SoftMax}(\mathbf{x})$. The computation splits into two tracks: (1) a *wide main track* that computes the polynomial exponential and the squarings on all d entries, and (2) a *narrow auxiliary track* that evaluates the normalization factors ($\|\mathbf{y}^{(j-1)}\|_2^{-1}$) via polynomial inverse square roots. Because the auxiliary track holds only one value per SoftMax instance (far fewer than N), its polynomials are evaluated over *sparsely packed ciphertexts*. We accelerate this auxiliary track with a new slim polynomial evaluation algorithm (Section 3.3) and reduce the main track's multiplicative depth through tight calibration (Section 4.2).

2.3 Threat Model

We adopt the standard single-server outsourced-inference setting. A client holds the CKKS secret key and a private query; a semi-honest server holds the model weights in the clear and evaluates the network homomorphically—with no interaction beyond the initial upload and no non-collusion assumption.

Protected: The client's query and all generated tokens remain encrypted throughout the computation: decoding is fully homomorphic, as the server selects each next token by an encrypted argmax over the logits and retrieves its embedding homomorphically (Section 3.2). The server never sees a plaintext of any client-supplied or model-generated value, and decryption occurs exclusively on the client, so the passive key-recovery attacks on approximate FHE (Li & Micciancio, 2021; Li et al., 2022) do not apply.

Leakage: the server learns the lengths of the public prefix and of the private query, the number of decoding steps, and which public content is used as the prefix.

Public prefixes. In heterogeneous prompt processing (Section 4), the client splits a prompt into a public prefix and a private suffix. The prefix is content the server already holds in the clear, so evaluating it in plaintext reveals nothing beyond its identity; the private suffix is protected as above regardless of the prefix length, and an empty prefix recovers the fully encrypted setting.

3 Correct and Efficient Encrypted Inference at 8B Scale

Figure 2 gives an overview of Sylph. Table 2 (Appendix A) lists its components and marks which ones are new. CKKS-based inference imposes two numerical contracts: inputs to the polynomial approximating each non-linear function f must lie within its fitted interval $D_f = [a_f, b_f]$, and ciphertexts entering bootstrapping must satisfy $\|x\|_\infty \le B_{\text{boot}}$. Violations fail silently rather than gracefully (outside D_f a degree-d polynomial grows as $|x|^d$, and larger inputs break the modular reduction inside bootstrapping), yet Llama-3.1-8B activations exceed 2200 (Table 1). Section 3.1 enforces both contracts without retraining, which also allows lower-degree approximations and cheaper bootstrapping.

3.1 Model Calibration and Outlier Mitigation

Activation outliers in LLMs arise from two mechanisms (Table 1): token-level spikes at attention sinks (Xiao et al., 2024), where initial tokens attract disproportionately large attention, and persistent channel-wise outliers in specific hidden dimensions (Ashkboos et al., 2024; Xiao et al., 2023).

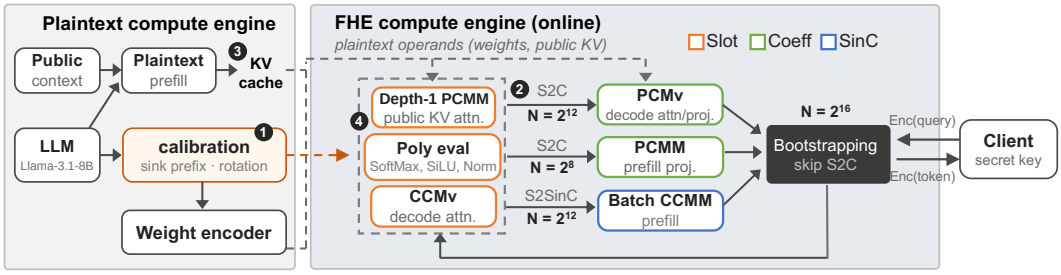

Figure 2: Overview of Sylph. The public context is prefilled in the clear, and its KV cache and the encoded weights enter the FHE engine only as plaintext operands. Bootstrapping ($N = 2^{16}$, final SlotToCoeff skipped) outputs slot-encoded ciphertexts for PC-attention, decode attention, and polynomial evaluation of non-linear layers; these are converted to coefficient (S2C) or SinC (S2SinC) encoding at smaller ring degrees for PCMM, PCMv, and batch CCMM. Model calibration fixes the value ranges and precision used throughout the encrypted computation. Most conversions are fused into bootstrapping; the remaining ones are discussed in Section 3.2. Circled numbers mark calibration (1, Section 3.1), fused encoding conversions (2, Section 3.2), the public KV cache (3, Section 4), and the kernels we design or adapt (4, Sections 3.3 and 4.2).

Table 1: Maximum magnitudes at the two contract points of Llama-3.1-8B before and after calibration. †Rotation is fused into the weights and is function-preserving, so non-linear inputs are unchanged. The measurement protocol and the rotation-only ablation are given in App. D.3.

Contract	Operator	Baseline	+Prefix	+Prefix+Rot.
Non-linear input (D_f)	SoftMax	39.24	32.78	32.78†
	SiLU	23.00	10.82	10.82†
	RMSNorm	2243.97	7.65	7.65†
Bootstrapping input (B_{boot})	`down_proj`	310.56	12.14	**1.92**
	`o_proj`	10.12	9.91	**1.08**
	`v_proj`	5.89	5.88	**4.74**

Attention-sink suppression. We prepend a static KV cache computed offline from sink-inducing prefix tokens (Xiao et al., 2024). As the prefix is model-dependent but not data-dependent, it leaks no client information and can be preprocessed in plaintext. For Llama-3.1-8B, the prefix is the `[BOS]` token alone, adding zero latency overhead (App. D.1).

Selective orthogonal rotation. On top of the prefix, we additionally apply the Walsh–Hadamard rotations of Ashkboos et al. (2024) that fuse offline into the weights (on the residual stream, and per head between `v_proj` and `o_proj`), omitting the online ones (App. D.2). With RMSNorm's scale folded into the next projections, normalization commutes with the rotation, which flattens the bootstrapping inputs after `v_proj`, `o_proj` and `down_proj` at zero parameter or latency cost.

In Table 1, the prefix bounds the inputs of non-linear layers (e.g., RMSNorm $2244 \to 7.65$), and the additional rotation further bounds the bootstrapping inputs (e.g., `down_proj` $12.14 \to 1.92$), bringing every contract below 40.

Calibration protocol. Each approximation interval is set offline from the global minimum m and maximum M (not percentiles) of its input over WikiText-2, widened to $[1.2m, 1.2M]$ ($[0.3m, 3M]$ for RMSNorm). With the intervals frozen, none of the more than 10^7 observations per activation on held-out C4 and UltraChat fell outside its interval (App. D.3). This is cross-dataset evidence rather than a worst-case guarantee.

Precision requirements. Simulating CKKS noise at every operation (App. D.4), we observe that 12 bits of precision match the perplexity of plaintext FP16 evaluation (6.13; Table 4), as does actual FHE execution.

Non-linear layers. Unlike FHE-friendly networks that replace activations with low-degree polynomials and must be retrained, we approximate the original non-linearities to 12-bit precision on the calibrated intervals, so the pretrained model is used as is. Narrow ranges lower degree and cost: SiLU, computed as $x(1/2 + q(x/R))$ with a polynomial q on $[-1, 1]$ and input range R, needs $\deg q = 191$ only in the last layer (inputs down to -45.3, $R = 64$), so the other 31 layers use $\deg q = 53$, cutting their SiLU time from 349 to 198 ms. The same statistics bound the SoftMax normalization range, enabling an 8-level main track in all but the last layer (Section 4.2); approximation parameters are in App. D.5.

Bootstrapping precision. CKKS bootstrapping is accurate only for inputs in $[-1, 1]$. Instead of costlier bootstrapping over larger ranges (Bae et al., 2022; Choe et al., 2025), we scale inputs by $1/B_{\mathrm{boot}}$, which leaves $p - \log_2 B_{\mathrm{boot}}$ bits of a p-bit bootstrapping. With $p = 19.5$, we use $B_{\mathrm{boot}} = 128$ at every bootstrapping, leaving at least the required 12 bits (App. D.5).

3.2 Homomorphic Execution Pipeline

Figure 3 shows how Sylph maps each operator (its PC-attention path is described in Section 4): linear projections to BLAS-based PCMM (Bae et al., 2024) (PCMv (He et al., 2024) in decoding), attention products to batch CCMM (Cheon et al., 2025) (CCMv (Halevi & Shoup, 2014) in decoding), RoPE to plaintext–ciphertext multiplications, and non-linear layers to polynomial approximations. Decoding is fully homomorphic: after the LM head, the next token is selected by an encrypted argmax over the logits and its embedding is retrieved homomorphically, so that autoregressive generation requires no client interaction (hierarchical argmax (Masalha et al., 2026) with 28 bootstrappings per token; App. F).

Challenge: incompatible ciphertext formats. These kernels expect different ring degrees (2^{16} for non-linear layers, 2^{12} for batch CCMM) and encodings (slot, coefficient, SinC), marked by box colors in Figure 3, as well as different packing layouts; explicit conversions between them would consume levels and time, erasing the gains of the specialized kernels.

Solution 1: encoding and ring conversion fusion. Sylph minimizes dedicated conversions by fusing format transformations into preceding bootstrappings wherever possible. First, we exploit the fact that CKKS bootstrapping internally computes homomorphic DFT and iDFT maps to switch between slot and coefficient spaces. By truncating or adapting these linear transforms mid-routine (Bae et al., 2024; Cheon et al., 2025), we extract ciphertexts directly in coefficient or SinC encodings at the lowest modulus level, simultaneously saving multiplicative depth and reducing plaintext memory footprint. Second, we bridge disparate ring degrees depth-free: we apply standard coefficient ring-switching (Gentry et al., 2012) for PCMM/PCMv, and our generalized ring-switching, which preserves the SinC encoding (App. G), for SinC-encoded batch-CCMM. Conversions whose inputs are not produced directly by a bootstrapping, namely those of batch CCMM after RoPE and SoftMax, are performed separately (App. E).

Solution 2: packing layouts absorbing bit reversal. Encoding conversions permute slots in bit-reversed order, which would normally have to be undone by costly homomorphic permutations. Instead, we choose packing layouts on which this permutation, together with ring switching, directly produces the layout each kernel expects. For attention tensors, the bit reversal in SlotToCoeff and SlotToSinC transposes the token and head dimensions exactly as batch CCMM requires, with no explicit transposition or rotation (App. E).

3.3 Slim Polynomial Evaluation for SoftMax

The auxiliary track of homomorphic SoftMax evaluates normalization factors on sparsely packed (slim) ciphertexts, where only $2^t \ll N$ slots contain active values (Cho et al., 2024). While Okada et al. (2023) utilized idle SIMD slots to evaluate polynomials in $O(\log d)$ key-switchings, their multiplicative root factorization causes severe numerical instability under CKKS due to extreme dynamic range variations. To resolve this, we propose a numerically stable evaluation algorithm based on recursive Sum-of-Squares (SOS) decomposition.

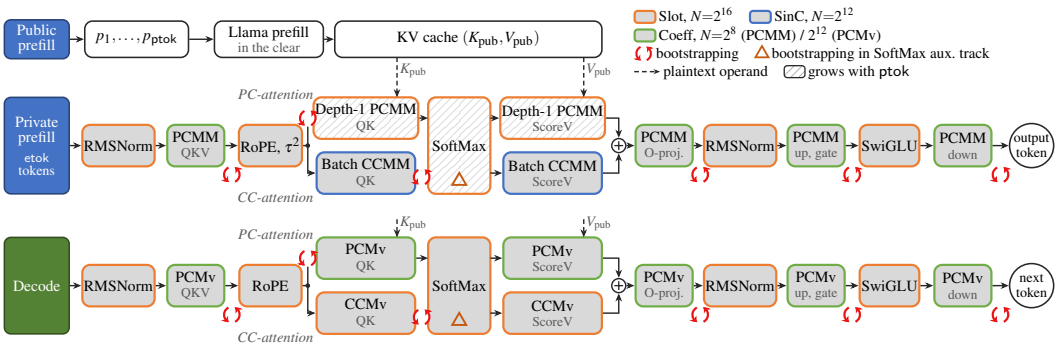

Figure 3: Encrypted computation with a public context (one of 32 decoder layers shown). The public KV cache enters only as plaintext operands of PC-attention (dashed arrows); a single SoftMax normalizes the concatenated PC/CC-attention scores, and the two ScoreV outputs are summed. Box colors give the ciphertext encoding and ring degree. Red circular arrows mark bootstrapping; orange triangles mark bootstrapping inside SoftMax, which occurs only in its narrow auxiliary track (Section 4.2). PCMM (Bae et al., 2024), PCMv (He et al., 2024), batch CCMM (Cheon et al., 2025), CCMv (Halevi & Shoup, 2014).

Sum-of-squares decomposition. Our algorithm builds upon the following property:

Lemma 1. *Let $P \in \mathbb{R}[X]$ be a polynomial of even degree d with a positive leading coefficient. There exist two polynomials U, V of degree $\leq d/2$ and a scalar m such that $P(x) = U(x)^2 + V(x)^2 - m$.*

In practice, we construct U and V by pairing conjugate roots of $P + m$ into quadratic SOS components $(X - \mathrm{Re}(\alpha))^2 + \mathrm{Im}(\alpha)^2$ and merging them with the two-squares identity $(U_0^2 + V_0^2)(U_1^2 + V_1^2) = A^2 + B^2$ (App. H). Unlike a multiplicative factorization $P = P_1 \cdot P_2$, which causes dramatic scale variations in CKKS, this decomposition keeps values bounded: as $U^2 + V^2 = P + m$, $|U|, |V| \leq \sqrt{\max_{\mathcal{I}} |P| + m}$ on the input interval $\mathcal{I}$.

Encrypted evaluation algorithm. Given a slim ciphertext ct with 2^t active slots, we copy its inputs into the next 2^t unused slots and evaluate U on the first block and V on the second in a single polynomial evaluation, yielding ct'; one squaring and one rotation then recover $P(x)$ as $\mathsf{ct}' \odot \mathsf{ct}' + \mathsf{Rot}_{2^t}(\mathsf{ct}' \odot \mathsf{ct}') - m$. For $\deg P = 2^k$, we apply Lemma 1 recursively up to depth $j < k$: starting from $U_0^{(0)} = P$, each level $\ell \leq j$ splits the previous one as

$$U_i^{(\ell)^2} + U_{i+2^{\ell-1}}^{(\ell)^2} = U_i^{(\ell-1)} + m_i^{(\ell-1)}, \quad 0 \leq i < 2^{\ell-1}, \tag{1}$$

and the 2^j leaves $U_i^{(j)}$, of degree 2^{k-j}, occupy 2^j blocks of 2^t slots ($2^{t+j} \leq N$). Algorithm 1 evaluates all leaves at once with the Paterson–Stockmeyer method and climbs the tree with j squaring-and-rotation steps, using $k + 1$ levels, $O(2^{(k-j)/2}) + j$ ciphertext multiplications and j rotations (proofs and stability bounds in App. H).

4 Scaling to Long Public Contexts with Private Queries

As outlined in Section 1, we process the public context in the clear and only the short private query under FHE (Section 4.1); the public context then enters the encrypted computation only through attention, whose wide path needs no bootstrapping thanks to a depth-1 PCMM and a shallow SoftMax (Section 4.2).

4.1 Heterogeneous Public/Private Prefill

We split a prompt of ntok tokens into a public prefix $P = (p_1, \ldots, p_{\mathsf{ptok}})$ and a private suffix $S = (s_1, \ldots, s_{\mathsf{etok}})$. The *public prefill* runs the standard Llama prefill on P in the clear and produces a public KV cache $(K_{\text{pub}}, V_{\text{pub}})$. The *private prefill* homomorphically processes S against this cache and produces the private KV cache $(K_{\text{priv}}, V_{\text{priv}})$ and the first output token; decoding then

proceeds as usual over both caches. This is the standard prefill applied to two chunks of lengths ptok and etok instead of a single chunk of length ptok + etok, and is thus exact in the clear (App. I.1).

Only attention sees the public context. QKV-projection, RoPE, O-projection and FFN act on each token independently, so the private prefill evaluates them exactly as in Section 3. Attention is the only operator that mixes tokens: the QK step computes $K_{\mathrm{pub}}Q^T$ and $K_{\mathrm{priv}}Q^T$, a single SoftMax normalizes the concatenated scores over ptok+etok positions, with a causal mask among the private tokens (each of which attends to the whole public prefix), and the ScoreV step applies them to V_{pub} and V_{priv} and sums the two results. We call the two tracks *PC-attention* (encrypted queries against the plaintext public cache) and *CC-attention* (encrypted queries against the encrypted private cache). CC-attention involves only the etok private tokens and reuses the pipeline of Section 3.2. PC-attention is where the long context shows up: its inputs and outputs fit in a few ciphertexts, but its intermediate state grows with ptok (256 ciphertexts for ptok $= 3968$ and etok $= 128$, versus 8 for the private tokens alone). Bootstrapping this wide state would dominate the cost, so we design PC-attention to need no bootstrapping on its wide path (Section 4.2).

Decoding. During decoding, matrix–matrix products become matrix–vector products: the attention scores form a single vector, so PC-attention no longer widens the circuit and bootstrapping is not a bottleneck. We use He et al. (2024) for all PCMv steps, including PC-attention, and the CC-attention flow of Section 3.2.

4.2 PC-ATTENTION WITHOUT WIDE BOOTSTRAPPING

PC-attention consists of a PCMM (QK step), a SoftMax, and a second PCMM (ScoreV step). Bootstrapping or switching encodings (SlotToCoeff/CoeffToSlot) on its wide intermediate state would process every one of its ciphertexts, so we keep the whole chain in slot encoding, in which bootstrapping already outputs the queries, and evaluate it within the levels left by that bootstrapping. This relies on two components: a PCMM that consumes a single level, and a SoftMax whose wide part is shallow and which bootstraps only on a narrow auxiliary track (both described below).

Depth-1 PCMM. We build on the JKLS algorithm (Jiang et al., 2018), which computes $C = AB$ for $A, B \in \mathbb{R}^{d\times d}$ as $\sum_k \mathsf{rot}_C^k(\sigma(A)) \odot \mathsf{rot}_R^k(\tau(B))$ for fixed permutations σ, τ and column and row rotations rot_C, rot_R, and consumes three levels when both operands are encrypted (App. I.2). Two observations bring this down to one level. First, when A is the plaintext operand, $\sigma(A)$ and its column rotations are plaintext operations and cost no depth. Second, the level spent on $\tau(B)$ disappears if ciphertexts are kept in a τ-shifted layout: our variant takes $\tau^{\ell+1}(\mathsf{ct}_B)$ as input and returns $\tau^{\ell}(\mathsf{ct}_C)$, for any integer ℓ. With a baby-step giant-step decomposition, the resulting PCMM uses one multiplicative level and $\mathcal{O}(\sqrt{d})$ ciphertext rotations (App. I.3). Instead of storing the d rearranged plaintexts it requires, we store a single rearranged copy of A and derive each of them on the fly with two plaintext rotations, versus eight for Powerformer (Park et al., 2025).

Since each PCMM removes one τ, we apply τ^2 right after RoPE, while the computation is still narrow, so that PC-attention ends in the layout expected by the rest of the pipeline. The SoftMax between the two PCMMs acts on $\tau(M)$, in which each SoftMax instance (a column of M) is cyclically rotated; since SoftMax commutes with cyclic shifts, its output is directly $\tau(\mathsf{SoftMax}(M))$ (App. I.4).

Shallow SoftMax over long instances. The public context lengthens each SoftMax instance from etok to ptok + etok entries, while the number of instances stays etok per head. In the two-track SoftMax of Cho et al. (2024) (Section 2.2), this growth falls entirely on the shallow main track, which now processes etok(ptok + etok) entries; the deep auxiliary track still holds one value per instance and remains the only part of SoftMax that bootstraps (orange triangles in Figure 3). The main track must therefore fit within the level budget of PC-attention, and each normalize-and-square iteration adds to its depth. With the tight value ranges obtained by calibration (Section 3.1), two iterations, and hence a main track of 8 levels, suffice in all layers except the last, which uses five. With ptok $= 31$etok, the $32\times$ longer instances increase SoftMax time by only about $1.9\times$ (App. I.5).

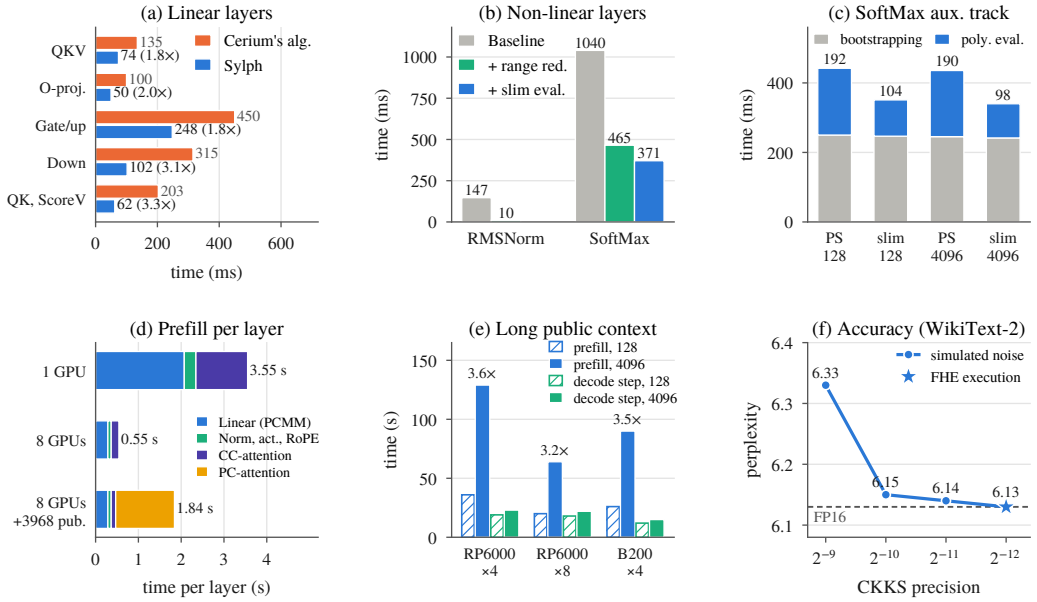

Figure 4: Evaluation of Sylph on Llama-3.1-8B (RTX PRO 6000 unless noted). **(a)** Linear layers: our algorithms vs. a reimplementation of Cerium's (with Jiang et al. (2018) for QK and ScoreV) in the same library. **(b)** Non-linear layers from a baseline over prior-work input ranges, adding range reduction (Section 3.1) and slim polynomial evaluation (Section 3.3) cumulatively. **(c)** SoftMax auxiliary track with Paterson–Stockmeyer (PS) or slim evaluation, for SoftMax instances of 128 and 4096 entries. **(d)** Per-layer prefill time by component; with 3968 public tokens, SoftMax is counted in PC-attention. **(e)** Prefill and one decoding step for a 128-token encrypted prompt vs. 3968 public + 128 encrypted tokens; labels give the prefill ratio. **(f)** WikiText-2 perplexity under simulated CKKS noise and actual FHE execution (Table 4). Raw numbers are in Appendix J. Panels (a)–(c) are measured on one GPU for layer 1 and exclude format conversions, unlike Table 11; (a) and (b) also exclude bootstrapping, which (c) reports separately.

5 Evaluation

Setup. We implement Sylph as a multi-GPU runtime on top of the GPU version of the HEaaN library (CryptoLab, 2025), which provides the CKKS primitives, and evaluate it with Llama-3.1-8B-Instruct on eight NVIDIA RTX PRO 6000 GPUs (PCIe Gen5) and on four NVLink-connected B200 GPUs (Appendix K.4). Each PCMM is distributed with a block partitioning of the weight matrix whose shape depends on the projection; inter-GPU communication consists of reduce-scatters on bottom-level ring-degree-2^{12} ciphertexts, scalar reductions in RMSNorm and SoftMax, and all-gathers of activations (Appendix K). The CKKS parameters (Table 10, Appendix K.4) provide 128-bit security (lattice estimator, MATZOV cost model).

Benchmarks. The *fully encrypted* benchmark, referred to as the 128-token encrypted prompt, processes 128 positions: the first is the `[BOS]` sink token, whose KV pair is precomputed in plaintext (Section 3.1), and the remaining 127 are encrypted. The *heterogeneous* benchmark uses $\mathsf{ptok} = 3968$ public and $\mathsf{etok} = 128$ encrypted tokens. Decoding times below cover the 32 decoder layers; the final RMSNorm, LM head, homomorphic token selection and embedding lookup add about 2.2 s per generated token on one RTX PRO 6000 GPU (App. F). Cerium timings are taken from Jayashankar et al. (2026); MOAI timings are estimated, as its workload exceeds GPU memory.

End-to-end latency. On the fully encrypted benchmark, prefill takes 20 s on eight RTX PRO 6000 GPUs and 26 s on four B200 GPUs, versus 152 s for Cerium on the same four B200 GPUs ($5.8\times$) and 134 s on eight (Figure 1). Per-layer prefill time falls from 3.55 s on one GPU to 0.55 s on eight ($6.5\times$), with linear layers taking the largest share in both cases (Figure 4d).

Where the speedup comes from. Each technique is compared with its prior-work counterpart implemented in our library with a comparable engineering effort. Our linear algebra is 1.8–$3.3\times$ faster than a reimplementation of Cerium's algorithms (Figure 4a). Range reduction speeds up RMSNorm by $15\times$ and SoftMax by $2.2\times$, and slim evaluation brings SoftMax a further $1.25\times$ (Figure 4b) by halving the polynomial-evaluation part of its auxiliary track (Figure 4c). To separate library performance from algorithmic gains, we ran bootstrapping in the same shape as Cerium ($N = 2^{16}$, 2^{15} complex slots) on one RTX PRO 6000 GPU: it takes 16.96 ms with HEaaN2 and 23.5 ms with Cerium's released artifact, a $1.39\times$ difference, compared with $5.8\times$ end to end.

Long public contexts. Extending the prompt from 128 to 4096 tokens ($\mathsf{ptok} = 3968$) increases prefill time by only 3.2–$3.6\times$ and the decoding step by about $1.2\times$ across configurations (Figure 4e). The extra cost is almost entirely PC-attention, while linear layers are unaffected (Figure 4d); the auxiliary track of SoftMax, the only part that bootstraps, costs the same for instances of 128 and 4096 entries (Figure 4c).

Accuracy and calibration robustness. With simulated CKKS noise, perplexity matches FP16 at 12 bits of precision (Figure 4f), and encrypted FHE inference on WikiText-2 reaches the same perplexity (6.13; inputs longer than 128 tokens use the public KV cache). The calibrated intervals held on held-out C4 and UltraChat, with no out-of-range observation among more than 10^7 per activation (Section 3.1).

6 Related Work

MPC-based private inference (Hou et al., 2023; Wu et al., 2024; Kei & Chow, 2025; Cheng et al., 2025; Gupta et al., 2024), including KV-cache eviction for long contexts (Zeng et al., 2026), needs many communication rounds and non-colluding servers, whereas FHE-based Transformer inference (Chen et al., 2022; Moon et al., 2025; Park et al., 2025; Zhang et al., 2025a;b; Jayashankar et al., 2026) is non-interactive but, at 8B scale, has been limited to a few tokens (Zhang et al., 2025a;b) or to more than 100 s per 128-token prefill (Jayashankar et al., 2026), without evaluating fidelity to plaintext inference (Appendix L). Sylph builds on BLAS-based PCMM and batched CCMM (Bae et al., 2024; Cheon et al., 2025; Park, 2025), BSGS matrix–vector products (Halevi & Shoup, 2014; He et al., 2024), the two-track SoftMax of Cho et al. (2024) and slim polynomial evaluation (Okada et al., 2023), and reuses outlier mitigation from LLM quantization (Xiao et al., 2024; Sun et al., 2024; Ashkboos et al., 2024; Xiao et al., 2023) to shrink the approximation intervals of FHE evaluation.

7 Conclusion

We presented Sylph, which scales FHE-based inference of Llama-3.1-8B in two directions: faster fully encrypted inference, through FHE-aware outlier mitigation, encoding-co-designed linear algebra and slim SoftMax evaluation, and longer prompts, by processing a public context in the clear and evaluating the resulting plaintext–ciphertext attention without bootstrapping on its wide path. Decoding scales less than prefill with the number of GPUs ($1.7\times$ vs. $5.8\times$ from one to eight GPUs); speculative decoding could turn its matrix–vector products into matrix–matrix products. The calibrated approximation intervals are validated empirically on held-out datasets rather than guaranteed for adversarial inputs, whose out-of-range activations would silently corrupt the output.

Reproducibility Statement

Sylph builds on the GPU version of the HEaaN library (CryptoLab, 2025), whose license does not allow us to redistribute its source code. For the review process, executable binaries are available at `https://anonymous.4open.science/r/sylph-71CE/`. To support verification, we provide full descriptions and proofs of our algorithms (Appendices H, I, and G), the packing layouts (Appendix E), the calibration protocol and measurements (Appendix D), the multi-GPU partitioning and communication schedule (Appendix K), and the CKKS parameters (Table 10, Appendix K.4).

Ethics Statement

This work concerns the cryptographic optimization and performance evaluation of an open-weight language model (Llama-3.1-8B-Instruct) under fully homomorphic encryption. It involves no human subjects, no collection of personally identifiable information, and no proprietary data.

Use of Large Language Models

LLM-based assistants were used to help with the programming implementation, and to edit and restructure the text, figures and tables of this paper, including consistency checks between the text and the reported numbers. All technical content, experiments and results are the authors' own; the authors reviewed all LLM outputs and take full responsibility for the code and the text.

References

S. Ashkboos, A. Mohtashami, M. L. Croci, B. Li, P. Cameron, M. Jaggi, D. Alistarh, T. Hoefler, and J. Hensman. QuaRot: Outlier-free 4-bit inference in rotated LLMs. In *NeurIPS*, 2024.

Y. Bae, J. H. Cheon, W. Cho, J. Kim, and T. Kim. META-BTS: bootstrapping precision beyond the limit. In *CCS*, 2022.

Y. Bae, J. H. Cheon, G. Hanrot, J. H. Park, and D. Stehlé. Plaintext-ciphertext matrix multiplication and FHE Bootstrapping: Fast and fused. In *CRYPTO*, 2024.

Y. Benoist. Writing positive polynomials as sums of (few) squares. *EMS Newsletter*, 2017.

Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Trans. Comput. Theory*, 2014.

T. Chen, H. Bao, S. Huang, L. Dong, B. Jiao, D. Jiang, H. Zhou, J. Li, and F. Wei. THE-X: Privacy-preserving transformer inference with homomorphic encryption. *Findings of the Association for Computational Linguistics: ACL 2022*, 2022.

K. Cheng, Y. Xia, A. Song, J. Fu, W. Qu, Y. Shen, and J. Zhang. Mosformer: Maliciously secure three-party inference framework for large transformers. In *CCS*, 2025.

J. H. Cheon, A. Kim, M. Kim, and Y. Song. Homomorphic encryption for arithmetic of approximate numbers. In *ASIACRYPT*, 2017.

J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song. Bootstrapping for approximate homomorphic encryption. In *EUROCRYPT*, 2018.

J. H. Cheon, M. Kang, and J. Lee. Fast batch matrix multiplication in ciphertexts. *IACR Cryptology ePrint Archive*, 2025. Paper 2025/1957, available at `https://eprint.iacr.org/2025/1957`.

W. Cho, G. Hanrot, T. Kim, M. Park, and D. Stehlé. Fast and accurate homomorphic Softmax evaluation. In *CCS*, 2024.

H. Choe, J. Kim, D. Stehlé, and E. Suvanto. Leveraging discrete CKKS to bootstrap in high precision. In *CCS*, 2025.

CryptoLab. HEaaN2 library, 2025.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The Llama 3 herd of models, 2024. Available at `https://arxiv.org/abs/2407.21783`.

C. Gentry, S. Halevi, C. Peikert, and N. P. Smart. Ring switching in BGV-style homomorphic encryption. In *SCN*, 2012.

K. Gupta, N. Jawalkar, A. Mukherjee, N. Chandran, D. Gupta, A. Panwar, and R. Sharma. Sigma: Secure GPT inference with function secret sharing. *PETS*, 2024.

S. Halevi and V. Shoup. Algorithms in HElib. In *CRYPTO*, 2014.

J. He, K. Yang, G. Tang, Z. Huang, L. Lin, C. Wei, Y. Yan, and W. Wang. Rhombus: Fast homomorphic matrix-vector multiplication for secure two-party inference. In *CCS*, 2024.

X. Hou, J. Liu, J. Li, Y. Li, W.-j. Lu, C. Hong, and K. Ren. CipherGPT: Secure two-party GPT inference. *IACR Cryptology ePrint Archive*, 2023. Paper 2023/1147, available at `https://eprint.iacr.org/2023/1147`.

S. Jayashankar, J. Kim, M. B. Sullivan, W. Zheng, and D. Skarlatos. Cerium: A multi-GPU framework for terabyte-scale encrypted inference. In *Proceedings of the 32nd ACM Symposium on Operating Systems Principles (SOSP)*, 2026. Preprint: `https://arxiv.org/abs/2512.11269`.

X. Jiang, M. Kim, K. Lauter, and Y. Song. Secure outsourced matrix computation and application to neural networks. In *CCS*, 2018.

Daniel Jurafsky and James H Martin. *Speech and Language Processing*. Stanford University, 3rd draft edition, 2024.

A. Y. L. Kei and S. S. M. Chow. SHAFT: Secure, handy, accurate, and fast transformer inference. In *NDSS*, 2025.

D. Kim and Y. Song. Approximate homomorphic encryption over the conjugate-invariant ring. In *ISISC*, 2018.

A. Langlois and D. Stehlé. Worst-case to average-case reductions for module lattices. *Des. Codes Cryptogr.*, 75, 2015.

P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *NeurIPS*, 2020.

Baiyu Li and Daniele Micciancio. On the security of homomorphic encryption on approximate numbers. In *Advances in Cryptology – EUROCRYPT 2021*. Springer, 2021. doi: 10.1007/978-3-030-77870-5_23.

Baiyu Li, Daniele Micciancio, Mark Schultz, and Jessica Sorrell. Securing approximate homomorphic encryption using differential privacy. In *Advances in Cryptology – CRYPTO 2022*. Springer, 2022. doi: 10.1007/978-3-031-15802-5_20.

Z. Liu, C. Zhao, I. Fedorov, B. Soran, D. Choudhary, R. Krishnamoorthi, V. Chandra, Y. Tian, and T. Blankevoort. Spinquant: LLM quantization with learned rotations. In *ICLR*, 2025.

V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. In *EUROCRYPT*, 2010.

R. Masalha, A. Akavia, A. Adir, E. Aharoni, and E. Kushnir. Argmax and XGBoost training over fully homomorphic encryption. *Proceedings on Privacy Enhancing Technologies*, 2026(1):175–200, 2026. doi: 10.56553/popets-2026-0010.

J. Moon, D. Yoo, X. Jiang, and M. Kim. THOR: Secure transformer inference with homomorphic encryption. In *CCS*, 2025.

H. Okada, R. Player, and S. Pohmann. Homomorphic polynomial evaluation using galois structure and applications to BFV bootstrapping. In *ASIACRYPT*, 2023.

D. Park, E. Lee, and J. Lee. Powerformer: Efficient and high-accuracy privacy-preserving language model with homomorphic encryption. In *ACL*, 2025.

J. H. Park. Ciphertext-ciphertext matrix multiplication: Fast for large matrices. In *EUROCRYPT*, 2025.

D. Stehlé, R. Steinfeld, K. Tanaka, and K. Xagawa. Efficient public key encryption based on ideal lattices. In *ASIACRYPT*, 2009.

M. Sun, X. Chen, J. Z. Kolter, and Z. Liu. Massive activations in large language models. In *COLM*, 2024.

H. Wu, W. Fang, Y. Zheng, J. Ma, J. Tan, Y. Wang, and L. Wang. Ditto: Quantization-aware secure inference of transformers upon MPC. In *ICML*, 2024.

G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han. SmoothQuant: Accurate and efficient post-training quantization for large language models. In *ICML*, 2023.

G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis. Efficient streaming language models with attention sinks. In *ICLR*, 2024.

Wenxuan Zeng, Ye Dong, Jinjin Zhou, Jin Tan, Lei Wang, Tao Wei, Runsheng Wang, and Meng Li. MPCache: MPC-friendly KV cache eviction for efficient private LLM inference. *Advances in Neural Information Processing Systems*, 38:97254–97284, 2026.

J. Zhang, X. Yang, L. He, K. Chen, W.-j. Lu, Y. Wang, X. Hou, J. Liu, K. Ren, and X. Yang. Secure transformer inference made non-interactive. In *NDSS*, 2025a.

L. Zhang, X. Wang, J. J. Sim, Z. Huang, J. Zhong, H. Wang, P. Duan, and K. Y. Lam. MOAI: Module-optimizing architecture for non-interactive secure transformer inference, 2025b. Available at `https://eprint.iacr.org/2025/991`.

A SUMMARY OF COMPONENTS

Table 2: Components of Sylph: where each is used, the ciphertext encoding and ring degree it operates on, and whether it is new, adapted from, or reused as in prior work.

Component	Used for	Encoding, N	Status	Sec.
Attention-sink prefix (Xiao et al., 2024)	outlier mitigation	plaintext	reused	3.1
Selective orthogonal rotation (Ashkboos et al., 2024)	outlier mitigation	fused in weights	adapted	3.1
Range calibration, input scaling for bootstrapping	all non-linear layers	–	new	3.1
BLAS PCMM (Bae et al., 2024)	QKV, O, FFN (prefill)	Coeff, 2^8	reused	3.2
BSGS PCMv (He et al., 2024)	QKV, O, FFN (decode)	Coeff, 2^{12}	reused	3.2
Batch CCMM (Cheon et al., 2025)	QK, ScoreV (prefill)	SinC, 2^{12}	reused	3.2
CCMv (Halevi & Shoup, 2014)	QK, ScoreV (decode)	Slot, 2^{16}	reused	3.2
Conversions fused in bootstrapping (Bae et al., 2024; Cheon et al., 2025)	all linear algebra	Slot $\rightarrow$ Coeff/SinC	adapted	3.2
Packing layout absorbing bit reversal	attention, projections	Slot, 2^{16}	new	3.2
Ring switching in SinC encoding	batch CCMM input	SinC, $2^{16} \rightarrow 2^{12}$	new	App. G
Two-track SoftMax (Cho et al., 2024)	attention	Slot, 2^{16}	reused	2.2
Slim sum-of-squares polynomial evaluation	SoftMax auxiliary track	sparse Slot	new	3.3
Hierarchical argmax (Masalha et al., 2026)	token selection (decode)	Slot, 2^{16}	adapted	App. F
Embedding lookup as one-hot PCMv	next-token input (decode)	as BSGS PCMv	adapted	App. F
Heterogeneous prefill	long public contexts	plaintext + Slot	new	4.1
Depth-1 PCMM (from Jiang et al. (2018))	PC-attention	Slot, 2^{16}	adapted	4.2
Shallow calibrated SoftMax main track	PC-attention	Slot, 2^{16}	adapted	4.2
Multi-GPU partitioning and communication	all layers	–	new	5

B THE CKKS SCHEME

Notation. Boldface lowercase letters denote vectors and uppercase letters denote matrices. We let $\lfloor \cdot \rceil$ denote rounding to the nearest integer (with an arbitrary rule for ties), applied coefficient-wise to polynomials, and $\log$ denote the base-2 logarithm.

CKKS Cheon et al. (2017) is an RLWE-based FHE scheme implementing arithmetic over approximate (real or complex) numbers. Its main parameter is the ring degree $N = 2^n$ (typically $n = 16$). It has message space $\mathbb{C}^{N/2}$, while its plaintext space is $\mathcal{R}_N = \mathbb{Z}[X]/(X^N + 1)$ – we shall simply use $\mathcal{R}$ when there is no ambiguity on the ring degree.

As all our data are real, Sylph uses the conjugate-invariant variant of CKKS (Kim & Song, 2018) throughout: a ciphertext of ring degree N (the conjugate-invariant subring of rank N of the cyclotomic ring of degree $2N$) carries N real slots instead of $N/2$ complex ones, i.e., 2^{16} real values for $N = 2^{16}$, and slot encoding remains an approximate ring homomorphism. For readability, we state the encodings below for standard CKKS over $\mathbb{C}^{N/2}$; their conjugate-invariant counterparts follow Kim & Song (2018), and all slot counts in this paper refer to real slots.

B.1 ENCODINGS

Let $\zeta = \exp(\pi\sqrt{-1}/N)$ be a primitive $(2N)$-th root of unity. We define the map $\mathrm{iDFT}_N : \mathbb{C}^{N/2} \rightarrow \mathbb{R}[X]/(X^N + 1)$ by

$$P = \mathrm{iDFT}_N((\mathbf{m})_{0 \le i < N/2}) \Leftrightarrow P(\zeta^{5^i}) = m_i,\ 0 \le i < N/2 \ .$$

CKKS maps the message space to the plaintext space using one of the following two encoding functions, parameterized by a large integer Δ called the *scaling factor* and which drives the numerical precision. On input $\mathbf{m} = (m_i)_{0 \le i < N/2}$, we define

$$\mathsf{Ecd}_{\mathrm{slot}}(\mathbf{m}) = \lfloor \Delta \cdot \mathrm{iDFT}(\mathbf{m}) \rceil \ ,$$

$$\mathsf{Ecd}_{\mathrm{coeff}}(\mathbf{m}) = \sum_{i=0}^{N/2-1} \left(\lfloor \Delta \mathrm{Re}(m_i) \rceil + X^{N/2} \lfloor \Delta \mathrm{Im}(m_i) \rceil \right) X^i \ .$$

Besides slot and coefficient encoding, we also use an intermediate encoding called *Slots-in-Coefficients* (SinC), introduced in Cheon et al. (2025).

Let k be a divisor of N, and let us identify $\mathcal{R}_k = \mathbb{Z}[Y]/(Y^k+1)$ with a subring of $\mathcal{R}_N$ by mapping Y to $X^{N/k}$. The iDFT$_k$ map, with root of unity $\xi = \zeta^{N/k}$ sends $\mathbb{C}^{k/2}$ to $\mathcal{R}_k$. Given $\mathbf{z} \in \mathbb{C}^{N/2}$, we define $\mathbf{z}^{(i)} = (z_{ik/2}, \ldots, z_{(i+1)k/2-1}) \in \mathbb{C}^{k/2}$ and

$$\mathsf{Ecd}_{\mathsf{SinC}_{k,N}}(z) = \sum_{i=0}^{N/k-1} \lfloor \Delta \cdot \text{iDFT}_k(\mathbf{z}^{(i)}) \rceil X^i \in \mathcal{R}_N \ . \tag{2}$$

Conversion between encodings can be performed by homomorphically evaluating the relevant DFT or iDFT maps; we discuss these transforms in Section 3.2.

When the message to be encoded has fewer coordinates than the number of slots ($N/2$ complex slots, or N real slots in the conjugate-invariant variant), we call it *sparsely-packed* or, for short, *slim*. This terminology is extended to plaintexts (and ciphertexts) if their underlying messages are slim. Algebraically, if N' is the smallest power of two that is no smaller than the number of coordinates, we have the plaintext live in a subring $\mathcal{R}_{N'}$ of $\mathcal{R}_N$; equivalently, when using slot-encoding the underlying message is periodic, of the form

$$\underbrace{(x_0, \ldots, x_{N'/2-1}, x_0, \ldots, x_{N'/2-1} \cdots, x_0, \ldots, x_{N'/2-1})}_{N/N' \text{ times}} \ .$$

B.2 Encryption and decryption

CKKS uses RLWE ciphertext formats Stehlé et al. (2009); Lyubashevsky et al. (2010)[1]

- SetUp: chooses a tuple $(N, P, Q, h, \mathcal{D})$, where N, P, Q, and h are positive integers (respectively ring degree, auxiliary modulus, top ciphertext modulus, and secret key Hamming weight) and $\mathcal{D}$ is a distribution on $\mathcal{R}$. These parameters are chosen so that RLWE with ring degree N and modulus PQ with sparse ternary secrets of Hamming weight h and error distribution $\mathcal{D}$ has 128-bit security. Setup also picks and returns a uniform $\mathsf{sk} \in \mathbb{Z}[X]/(X^N+1)$ with coefficients in $\{-1, 0, 1\}$ and h non-zero coefficients. The choice of (N, P, Q) among secure values takes into account the circuit to be implemented, in particular its shape in terms of width (related to N), depth (related to Q), and number of key switchings (the cost of which is related to an integer parameter $\mathsf{dnum} \approx (\log Q)/(\log P)$).
- $\mathsf{Enc}_{\mathsf{sk},\mathcal{D},Q}(\mathsf{pt})$: on input $\mathsf{pt} \in \mathbb{Z}[X]/(X^N+1)$, return a pair $(a, -a \cdot \mathsf{sk} + \mathsf{pt} + e \bmod Q)$, where $e \leftarrow \mathcal{D}$ and a is uniformly chosen in $\mathcal{R}_Q = \mathcal{R}/Q\mathcal{R}$;
- $\mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct})$: on input $\mathsf{ct} = (a, b) \in \mathcal{R}_Q^2$, return $a \cdot \mathsf{sk} + b \bmod Q$.

In particular, the CKKS ciphertext space is $\mathcal{R}_Q^2$. For simplicity, we omit the possibility to enable public key encryption, and encryption/decryption with various moduli. In the sequel, we shall simply write Enc and Dec, and assume that all their parameters are fixed at setup time.

An important property of RLWE ciphertexts for homomorphic encryption is the ability to switch keys. A *key-switching key* from the secret key sk to the secret key sk' is an encryption of $P \cdot \mathsf{sk}$ under sk' in $\mathcal{R}_{PQ}^2$, namely a pair $(a, -a \cdot \mathsf{sk}' + P \cdot \mathsf{sk} + e)$, where $e \leftarrow \mathcal{D}$ and a is uniformly chosen in $\mathcal{R}_{PQ}$ (assuming that $\mathsf{dnum} = 1$). This key-switching key, which can be made public, is used in the key-switching operation:

- $\mathsf{KeySwitch}_{\mathsf{ksk}}(\mathsf{ct})$: on input a key-switching key ksk from sk to sk' in $\mathcal{R}_{PQ}^2$ and a ciphertext $\mathsf{ct} \in \mathcal{R}_Q^2$ encrypted under sk, returns a ciphertext $\mathsf{ct}' \in \mathcal{R}_Q^2$ under sk' such that $\mathsf{Dec}_{\mathsf{sk}'}(\mathsf{ct}') \approx \mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct})$. $\mathsf{KeySwitch}$ reduces to arithmetic in $\mathcal{R}_{PQ}$, with a number of operations depending on dnum.
- $\mathsf{Add}(\mathsf{ct}_1, \mathsf{ct}_2)$: on input two ciphertexts ct_1 and ct_2 in $\mathcal{R}_Q^2$, return a ciphertext $\mathsf{ct}' \in \mathcal{R}_Q^2$ such that $\mathsf{Dec}(\mathsf{ct}') \approx \mathsf{Dec}(\mathsf{ct}_1) + \mathsf{Dec}(\mathsf{ct}_2)$; this operation is implemented as componentwise addition in $\mathcal{R}_Q^2$;

[1] Some functionalities require moving to MLWE formats Brakerski et al. (2014); Langlois & Stehlé (2015) temporarily.

- $\mathsf{PCMult}(\mathsf{pt}, \mathsf{ct})$: on input a plaintext $\mathsf{pt} \in \mathcal{R}$ and a ciphertext $\mathsf{ct} \in \mathcal{R}_Q^2$, both with scaling factor Δ, return a ciphertext $\mathsf{ct}' \in \mathcal{R}_{Q'}^2$ with $Q' \approx Q/\Delta$, such that $\mathsf{Dcd}(\mathsf{Dec}(\mathsf{ct}')) \approx \mathsf{Dcd}(\mathsf{Dec}(\mathsf{ct})) \odot \mathsf{Dcd}(\mathsf{pt})$.
- $\mathsf{Mult}_{\mathsf{rlk}}(\mathsf{ct}_1, \mathsf{ct}_2)$: on input two ciphertexts ct_1 and ct_2 in $\mathcal{R}_Q^2$ with scaling factors Δ, and a key-switching key $\mathsf{rlk} \in \mathcal{R}_{PQ}^2$ from sk^2 to sk, return a ciphertext $\mathsf{ct}' \in \mathcal{R}_{Q'}^2$ with $Q' \approx Q/\Delta$, such that $\mathsf{Dcd}(\mathsf{Dec}(\mathsf{ct}')) \approx \mathsf{Dcd}(\mathsf{Dec}(\mathsf{ct}_1)) \odot \mathsf{Dcd}(\mathsf{Dec}(\mathsf{ct}_2))$.
- $\mathsf{Rot}_{\mathsf{rk}_k}(\mathsf{ct}, k)$: on input a ciphertext ct, an integer $0 \leq k < N/2$, and a switching key from $\mathsf{sk}(X^{5^k})$ to sk, return a ciphertext ct' such that $\mathsf{Dcd}(\mathsf{Dec}(\mathsf{ct}'))$ is a cyclic rotation of $\mathsf{Dcd}(\mathsf{Dec}(\mathsf{ct}))$ by k positions.

B.3 Rescaling, precision, and modulus handling

The use of a scaled encoding with scaling factor Δ implies that the arithmetic result of an encoded multiplication must be corrected via division by Δ in order to restore the correct scaling factor. This is implemented homomorphically as Rescale:

- $\mathsf{Rescale}(\mathsf{ct})$: on input a ciphertext $\mathsf{ct} \in \mathcal{R}_Q^2$, returns $\mathsf{ct}' \in \mathcal{R}_{Q'}^2$ with $Q' \approx Q/\Delta$, such that $\mathsf{Dcd}(\mathsf{Dec}(\mathsf{ct}')) \approx \Delta^{-1} \cdot \mathsf{Dcd}(\mathsf{Dec}(\mathsf{ct}))$.

As a consequence, every multiplication reduces the modulus of a ciphertext by a factor $\approx \Delta$. This is usually accounted via the notion of *level*: given integers $Q_0, Q_1, \ldots, Q_L$ where $Q_i/Q_{i-1} \approx \Delta$ for $i \geq 1$, we define level-j ciphertexts as elements of $\mathcal{R}_{Q_j}^2$. Multiplication (both PCMult and Mult) can then be seen as taking its inputs in $\mathcal{R}_{Q_j}^2$ and returning its result in $\mathcal{R}_{Q_{j-1}}^2$.

The numerical model of CKKS is close to a fixed-point number system, meaning that the precision of an input value x is absolute and does not depend on the order of magnitude of x. The precision of CKKS computations is driven by the scaling factor Δ. Choosing Δ is a tradeoff between precision needs and modulus usage; a larger Δ provides better accuracy but exhausts ciphertext modulus faster.

C Visualization of Llama Inference Pipeline

Figure 5 illustrates the Llama inference pipeline.

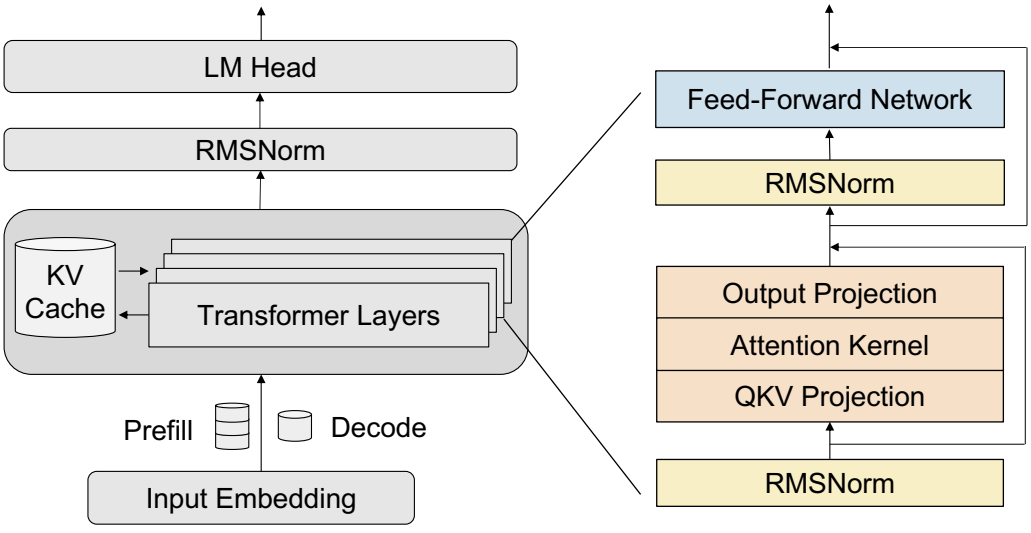

Figure 5: Llama inference pipeline with KV caching. The left figure shows the overall autoregressive inference flow, while the right figure illustrates the structure of a Transformer layer.

D Model Calibration Details

This section complements Section 3.1.

D.1 Attention-Sink Prefix

Following Xiao et al. (2024), we prepend a static KV cache computed offline from sink-inducing prefix tokens. The prefix depends only on the model, not on client data, so it is computed once in plaintext and reused for every query. In general LLMs, the sink-inducing prefix amounts to 4 tokens; for Llama-3.1-8B, it consists of the `[BOS]` token alone, adding zero latency overhead. The prefix is selected following Xiao et al. (2024).

D.2 Selective Orthogonal Rotation

Of the Walsh–Hadamard rotations of Ashkboos et al. (2024), we keep only those that can be fused offline into the weights, and omit the online Hadamard transforms (on queries and keys, and before `down_proj`), which would require additional homomorphic linear transforms. The first rotation acts on the residual stream: RMSNorm's learned scale is folded into the subsequent projections, so that the remaining scale-free normalization commutes with orthogonal matrices, and the rotation is absorbed into the weights that read from or write to the residual stream. The second acts per attention head between `v_proj` and `o_proj`. Both are function-preserving: the inputs of non-linear layers (SoftMax, SiLU, and the inverse square root in RMSNorm) are unchanged, while channel-wise spikes at the bootstrapping inputs after `v_proj`, `o_proj`, and `down_proj` are flattened. The fusion follows Ashkboos et al. (2024) and Liu et al. (2025): the two rotations correspond to R_1 and R_2 of Liu et al. (2025), and we omit the online rotations R_3 and R_4.

D.3 Measurement Protocol and Ablation

Each approximation interval is calibrated offline on WikiText-2 from the global minimum m and maximum M of its input over all layers and tokens (not percentiles), and widened to $[1.2m, 1.2M]$, or $[0.3m, 3M]$ for RMSNorm. The intervals are then frozen and tested on held-out C4 and UltraChat prompts: across more than 10^7 scalar observations per activation, none falls outside its interval. This is cross-dataset evidence rather than a worst-case guarantee for adversarial or out-of-distribution inputs. Table 3 isolates the contribution of each technique at the bootstrapping inputs.

Table 3: Maximum activation magnitude at bootstrapping inputs with prefix and rotation optimizations. Bold marks the best value per row.

Operator	**Baseline**	**+Prefix**	**+Rotation**	**+Both**
`down_proj`	310.56	12.14	14.86	**1.92**
`o_proj`	10.12	9.91	1.14	**1.08**
`v_proj`	5.89	5.88	**4.60**	4.74

D.4 Precision Calibration

For a given precision, we run the forward pass with injection of modeled CKKS noise at each operation, and compute the perplexity score (Jurafsky & Martin, 2024). We report perplexity on WikiText-2 under the precision settings in Table 4; for inputs longer than 128 tokens, the preceding tokens enter through the public KV cache (Section 4).

Table 4: Perplexity under different precision settings (simulated CKKS noise), plaintext FP16, and actual FHE execution.

Precision	2^{-9}	2^{-10}	2^{-11}	2^{-12}	FP16	FHE
Perplexity	6.33	6.15	6.14	**6.13**	6.13	6.13

D.5 Approximation Parameters and Bootstrapping Input Scaling

SiLU is evaluated as $x\,(1/2 + q(x/R))$, where q approximates the sigmoid shifted by $-1/2$ on the calibrated interval: $\deg q = 53$ in all layers except the last, and $\deg q = 191$ with $R = 64$ in the last layer, whose gate inputs reach -45.3. On one GPU, the SiLU of a layer takes 198 ms and 349 ms, respectively. The SoftMax main track uses two normalize-and-square iterations (8 levels) in all layers except the last, which uses five (Section 4.2). Table 5 summarizes the degrees of the polynomial approximations.

Table 5: Degrees of the polynomial approximations of the non-linear layers.

Layer	Approximated function	Degree
SiLU	shifted sigmoid q	53 (191 in the last layer)
SoftMax, main track	exponential	16
SoftMax, auxiliary track	inverse square root	128 and 256
RMSNorm	inverse square root	31

CKKS bootstrapping assumes inputs in $[-1, 1]$. Given a p-bit bootstrapping procedure and an input bound B_{boot}, we scale a ciphertext ct by $1/B_{\mathrm{boot}}$ so that all slot values lie within $[-1, 1]$. This scaling reduces the effective precision by $\log_2 B_{\mathrm{boot}}$ bits, yielding $p - \log_2 B_{\mathrm{boot}}$ bits of precision after bootstrapping. Our bootstrapping procedure provides $p = 19.5$ bits, so meeting the 12-bit target requires $\log_2 B_{\mathrm{boot}} \leq 7.5$. We therefore use a single value $B_{\mathrm{boot}} = 128 = 2^7$ at every bootstrapping position, which leaves at least 12 bits of precision.

E Encoding Conversions and Packing Layouts

Adopting specialized kernels introduces format incompatibilities across the pipeline: intermediate ciphertexts differ in (i) ring degree (e.g., 2^{16} for non-linear layers vs. 2^{12} for batch CCMM), (ii) encoding (slot, coefficient, or SinC), and (iii) packing layout (e.g., row-major, row-split, or diagonal). Inserting explicit homomorphic format conversions and transpositions between incompatible operators would consume levels and time, and quickly erase the efficiency gains of the specialized kernels.

Conversions outside bootstrapping. Sylph fuses an encoding conversion into bootstrapping whenever the ciphertext to convert is the output of a bootstrapping (Section 3.2). The inputs of batch CCMM are produced after RoPE (for Q and K) or after SoftMax (for the attention scores) rather than directly by a bootstrapping, so they require a separate partial SlotToCoeff or SlotToSinC followed by ring switching (App. G). These conversions act only on the narrow inputs of CC-attention, and their cost is included in the per-layer times of Table 11.

Packing layouts absorbing bit reversal. Homomorphic encoding conversions inherently induce bit-reversal permutations arising from internal FFT decompositions, which would normally disrupt the underlying tensor coordinates. Rather than undoing these permutations with expensive homomorphic permutations, Sylph chooses the initial packing so that the deterministic permutation induced by encoding conversions and ring switching maps each tensor directly into the layout required by the downstream kernel (Figure 6).

Llama-3.1-8B uses 32 query heads and 8 key/value heads; with zero-based logical head indexing, query head i uses key/value head $\lfloor i/4 \rfloor$ in both QK and ScoreV. For attention tensors $(t_{i,j,k}) \in \mathbb{R}^{n_{\text{heads}} \times 128 \times 128}$ (indexed by head i, channel j, and token k), we pack message slots in the lexicographical order of (j, i, k) across $n_{\text{heads}}/4 = 8$ ciphertexts of ring degree 2^{16}, each holding 2^{16} real values: token index k varies fastest within contiguous 128-slot blocks, followed by head index i. Under this layout, the bit-reversal map occurring during SlotToCoeff and SlotToSinC transposes the token and head dimensions (Figure 6). Subsequent ring switching and coefficient slicing resolve this swapped geometry, so that the token sequence of a given channel, or the matrix rows of a given attention head, remain co-located. Sylph thereby satisfies the layout preconditions of BLAS-based PCMM (Bae et al., 2024) and batch CCMM (Cheon et al., 2025) without any explicit transposition or rotation.

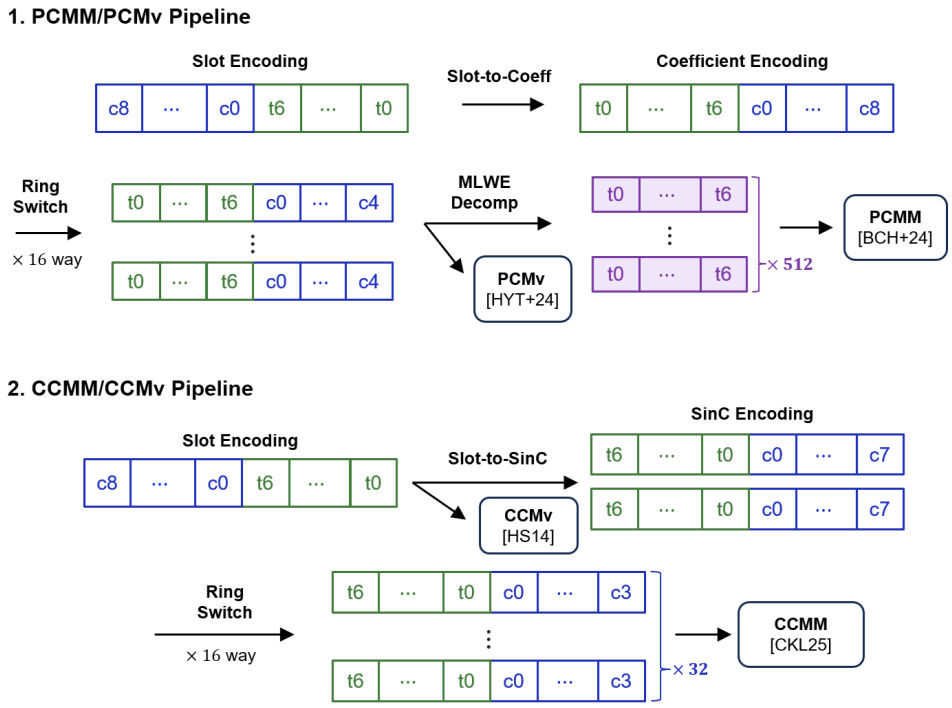

Figure 6: Ciphertext formats, packing layouts, and conversion paths used for Llama-3.1-8B evaluation. The labels $t_0, \ldots, t_6$ denote bit positions of the token coordinate for a 128-token block, not individual tokens. The labels c_i correspond to (part of) the channel number.

F Encrypted Token Selection and Embedding Lookup

During decoding, Sylph selects the next token and retrieves its embedding without decryption, so that generation proceeds without client interaction. The output of the last decoder layer passes through the final RMSNorm and the LM head, evaluated as a BSGS PCMv (He et al., 2024) over the vocabulary of $128{,}256 = 31 \cdot 4096 + 1280$ tokens, split into 32 blocks of 4096 logits. The selector outputs an approximate one-hot vector over the vocabulary, with ties resolved towards the smallest token index, which is then multiplied by the embedding table to form the input of the next decoding step.

Hierarchical argmax. Within each block, we follow the hierarchical argmax of Masalha et al. (2026). A tournament round ($\mathrm{League}^*_{2\times1}$) and the special $\mathrm{League}^*_{4\times2}$ reduce 4096 values to 512 candidates; the survivors are then replicated into the idle slots, so that the all-pairs leagues of size 8×8 and 64×64 each need a single SIMD comparison. A block of 4096 logits thus takes 4 SIMD comparisons, instead of 12 for a tournament, and the comparison indicators yield both the one-hot vector of the block and its maximum. We depart from Masalha et al. (2026) in three ways. (i) Every comparison is oriented from the later to the earlier index, and its reverse is obtained as $1 - c$; this preserves the smallest-index tie-breaking rule, including in $\mathrm{League}^*_{4\times2}$. (ii) The reduction sums the positive weights $v + B$ and subtracts the public bound B afterwards, so that a league without a clear winner does not output 0 when all its inputs are negative. (iii) As the vocabulary exceeds the block capacity of 4096 values, we replace the tournament pre-reduction of Masalha et al. (2026, §4.2) by a single all-pairs 32×32 league over the 32 block maxima (the *head*); the products of its rows ($\log_2 32 = 5$ multiplications) give one gate per block, and multiplying the one-hot vector of each block by its gate yields the one-hot vector over the full vocabulary.

Comparator and batching. A comparison $x > y$ normalizes $x - y$ by a public bound and applies a composite sign polynomial in monic form, i.e., with the scalar factors moved into the amplitude chain. The block comparator has 10 stages and consumes about 29 levels. The head comparator has

12 stages of degrees $(7, 3, 3, 3, 5, 5, 5, 5, 5, 5, 5, 7)$ and depth 33; the amplitude of its first stage is folded into the input scaling, which saves one multiplication. The coefficients of both comparators are certified in 80-digit arithmetic. As a block uses 4096 slots, 16 blocks are packed as independent lanes of one ciphertext with 2^{16} slots (league rotations stay within a lane), and the 32 blocks are processed in two calls.

Depth and cost. With about 15 levels available after each bootstrapping, the depth of the selector translates into bootstrappings: each block call uses 12 (mostly inside the comparators, the others for league masks and the final polish), and the head uses 4, for a total of $2 \cdot 12 + 4 = 28$ bootstrappings per generated token. Table 6 reports the resulting costs on one RTX PRO 6000 GPU. Bootstrapping, at about 10.2 ms each, accounts for more than half of the selector time.

Table 6: Cost of the final RMSNorm, LM head, encrypted token selection and embedding lookup per generated token on one RTX PRO 6000 GPU. The selector time is the median of 7 warm runs.

Step	Bootstrappings	Time
Final RMSNorm	–	0.45 s
LM head	–	0.6 s
Selector (32 blocks and head)	28	0.54 s
of which head	4	0.10 s
Embedding lookup	33	0.6 s
Total	–	2.2 s

Accuracy. The logits satisfy $|\ell| \leq 64$. The resolution of the selector is about $5.2 \cdot 10^{-3}$: when the two largest logits differ by less, the selected token may differ from the plaintext argmax. In a 32-layer end-to-end run, the maximum error of the encrypted logits against FP32 is $6.5 \cdot 10^{-4}$, and the errors of the one-hot vector and of the maximum are $3.4 \cdot 10^{-5}$ and $5.2 \cdot 10^{-6}$, well below their budgets of $5 \cdot 10^{-3}$ and $1.3 \cdot 10^{-3}$. Across 37 test inputs, the selected token always matches the reference, and an independent audit of $2.1 \cdot 10^{8}$ decrypted slots found no mismatch.

Embedding lookup. Instead of indexing the embedding table, we compute $\mathbf{e} = \sum_{b=0}^{31} \mathbf{o}_b E_b$, where $\mathbf{o}_b$ is the b-th block of the one-hot vector and $E_b \in \mathbb{R}^{4096 \times 4096}$ the corresponding rows of the table, with the same BSGS PCMv and block plan as the LM head and the table as plaintext operand. Each one-hot block is bootstrapped once before its product (32 bootstrappings); the 32 products are summed before a single final bootstrapping, whose output (at level 15) directly serves as the input of the next decoding step. Since the one-hot vector is approximate, $\mathbf{e}$ is a weighted sum of all rows, in which losing tokens keep a residual weight of order 10^{-5}. Including this effect, the maximum error against the exact row is $5.3 \cdot 10^{-6}$ (table entries are at most 0.077 in magnitude), with gain 1.000017 and correlation 0.99999998.

G Ring-Switching for SinC Encodings

While PCMM operates over low-degree coefficient encodings, the fast batch-CCMM of Cheon et al. (2025) requires inputs in low-degree SinC encoding. As noted in Cheon et al. (2025), conversion to SinC encoding can be performed opportunistically during bootstrapping by truncating the internal linear transformations. However, deploying batch-CCMM on modern LLMs requires reducing the ciphertext ring degree from N to N' ($N' \mid N$) while preserving this intermediate SinC structure.

Standard ring-switching decomposes a ciphertext in $\mathcal{R}_N$ under coefficient encoding into $k = N/N'$ ciphertexts over the subring $\mathcal{R}_{N'}$ via key-switching and polynomial slicing. We show that this property generalizes naturally to SinC-encoded ciphertexts. Specifically, assuming N, N', k' are powers of two with $k' \mid N'$ and $k = N/N'$ (where k' denotes the block size of SinC), decomposing the key-switched polynomial yields k ciphertexts in $\mathcal{R}_{N'}$, each preserving the $\mathsf{SinC}_{k',N'}$ encoding, so that no further format conversion is needed; the cost is that of a standard ring switching, i.e., one key switching followed by polynomial slicing. The formal derivation follows.

In Sylph, we leverage this generalized ring-switching to feed the batch-CCMM pipeline during self-attention. We begin with 8 ciphertexts in $\mathcal{R}_{2^{16}}$ that together pack the 32 matrices of dimension

128×128 (2^{16} real values per ciphertext). During the slot-to-coefficient step of bootstrapping, we extract ciphertexts in $\mathsf{SinC}_{2^7,2^{16}}$ encoding and ring-switch them to $\mathcal{R}_{2^{12}}$ ($k = 16$). This yields 256 ciphertexts in $\mathsf{SinC}_{2^7,2^{12}}$, each containing a single row across 16 different matrices—matching the exact layout required by Cheon et al. (2025) to execute 32 matrix multiplications concurrently via GPU BLAS.

We present the formal derivation showing that standard RLWE ring-switching preserves the Slots-in-Coefficients (SinC) encoding structure (Cheon et al., 2025).

Setting and Parameters. Let N, N', k' be powers of two such that $N' \mid N$ and $k' \mid N'$ (in our implementation, $N = 2^{16}$, $N' = 2^{12}$, and $k' = 2^7$). We define the ring degree reduction ratio as $k = N/N'$. Standard ring-switching maps a ciphertext $\mathsf{ct} = (a, b) \in \mathcal{R}_N^2$ to k ciphertexts $(a_j, b_j)_{0 \le j < k} \in \mathcal{R}_{N'}^2$ under the subring isomorphism $Y = X^k \in \mathcal{R}_{N'} = \mathbb{Z}[Y]/(Y^{N'} + 1)$. Specifically, key-switching to $\mathsf{sk}' \in \mathcal{R}_{N'}$ and decomposing the polynomial into k slices yields:

$$a(X) = \sum_{j=0}^{k-1} a_j(X^k) X^j, \quad b(X) = \sum_{j=0}^{k-1} b_j(X^k) X^j. \tag{3}$$

Preservation of SinC Encoding. Let $\mathbf{z} \in \mathbb{C}^{N/2}$ be encoded under $\mathsf{SinC}_{k',N}$, split into N/k' blocks of length $k'/2$: $\mathbf{z}^{(i)} = (z_{i \cdot k'/2}, \ldots, z_{(i+1) \cdot k'/2 - 1}) \in \mathbb{C}^{k'/2}$ for $0 \le i < N/k'$. By definition of SinC encoding, the plaintext polynomial $m(X) \in \mathcal{R}_N$ is:

$$m(X) = \sum_{i=0}^{N/k'-1} \lfloor \Delta \cdot \mathrm{iDFT}_{k'}(\mathbf{z}^{(i)}) \rceil X^i. \tag{4}$$

Setting $i = j' \cdot k + j$ with $0 \le j < k$ and $0 \le j' < N'/k'$, we rearrange the summation as:

$$m(X) = \sum_{j=0}^{k-1} \left(\sum_{j'=0}^{N'/k'-1} \lfloor \Delta \cdot \mathrm{iDFT}_{k'}(\mathbf{z}^{(j'k+j)}) \rceil (X^k)^{j'} \right) X^j. \tag{5}$$

Under the subring mapping $Y = X^k$, each component polynomial $m_j(Y) \in \mathcal{R}_{N'}$ corresponds to:

$$m_j(Y) = \sum_{j'=0}^{N'/k'-1} \lfloor \Delta \cdot \mathrm{iDFT}_{k'}(\mathbf{z}^{(j'k+j)}) \rceil Y^{j'}. \tag{6}$$

By inspection, $m_j(Y)$ exactly matches the definition of $\mathsf{SinC}_{k',N'}$ encoding over the subring $\mathcal{R}_{N'}$, packing the interleaved sub-vectors:

$$m_j = \mathsf{Ecd}_{\mathsf{SinC}_{k',N'}} \left(\mathbf{z}^{(j)}, \mathbf{z}^{(j+k)}, \mathbf{z}^{(j+2k)}, \ldots, \mathbf{z}^{(j+N/k'-k)} \right) \in \mathcal{R}_{N'}. \tag{7}$$

Therefore, coefficient slicing via ring-switching preserves the SinC encoding structure without requiring explicit homomorphic permutation or encoding conversion layers.

H Details and Analysis of Slim Polynomial Evaluation

In this section, we provide the complete mathematical proofs, implementation details, and numerical error analysis for Algorithm 1 (Slim Polynomial Evaluation for SoftMax).

H.1 Proof and Construction of Lemma 1

Proof of Lemma 1. Let $P \in \mathbb{R}[X]$ be a polynomial of even degree d with a positive leading coefficient. We define the scalar offset as:

$$m = \max_{x \in \mathbb{R}} (-P(x)). \tag{8}$$

Because the leading coefficient is positive and the degree d is even, $P(x) \to +\infty$ as $x \to \pm\infty$, guaranteeing that $-P(x)$ is bounded from above and m is finite. Consequently, the shifted polynomial $P(x) + m \ge 0$ is non-negative everywhere on $\mathbb{R}$. By the classical representation theorem of positive polynomials (e.g., Benoist (2017)), any real non-negative polynomial can be expressed as a sum of two squares of real polynomials: $P + m = U^2 + V^2$ with $\deg(U), \deg(V) \le d/2$. Rearranging terms yields $P(x) = U(x)^2 + V(x)^2 - m$. □

Algorithm 1 Slim Polynomial Evaluation for Degree-2^k with Depth-j Recursion

Input: Ciphertext $\mathsf{ct} \in \mathcal{R}^2_{Q_\ell}$ with 2^t active slots ($2^{t+j} \leq N$)
Input: Precomputed plaintexts $\mathbf{v}^{(0)}, \ldots, \mathbf{v}^{(2^{k-j})}$ and constants $\mathbf{m}^{(0)}, \ldots, \mathbf{m}^{(j-1)}$
Input: Relinearization key rlk, rotation keys $(\mathsf{rk}_{2^{t+i}})_{0 \leq i < j}$
Output: Evaluated ciphertext $\mathsf{ct}' \approx P(\mathsf{ct})$

1: $\mathsf{ct} \leftarrow \mathsf{Evaluate}(\mathsf{ct}, \mathbf{v}^{(0)}, \ldots, \mathbf{v}^{(2^{k-j})})$ {Paterson–Stockmeyer evaluation}
2: **for** $\ell = j$ down to 1 **do**
3: $\quad \mathsf{ct} \leftarrow \mathsf{Mult}_{\mathsf{rlk}}(\mathsf{ct}, \mathsf{ct})$
4: $\quad \mathsf{ct} \leftarrow \mathsf{ct} + \mathsf{Rot}_{\mathsf{rk}_{2^{t+\ell-1}}}(\mathsf{ct}, 2^{t+\ell-1}) - \mathbf{m}^{(\ell-1)}$
5: **end for**
6: **return** ct

Algorithmic Construction from Polynomial Roots. To explicitly construct U and V, we factor $P + m$ over $\mathbb{C}$ as:

$$P(X) + m = p_d \prod_{i=0}^{d-1} (X - \alpha_i). \tag{9}$$

Since $P + m$ is non-negative on $\mathbb{R}$, any real roots must have even multiplicity, and non-real roots appear in complex conjugate pairs. We can thus order the roots such that $\alpha_{2i} = \overline{\alpha_{2i+1}}$ for all $0 \leq i < d/2$. Each quadratic factor can be written as a sum of two squares:

$$(X - \alpha_{2i})(X - \alpha_{2i+1}) = (X - \mathrm{Re}(\alpha_{2i}))^2 + (\mathrm{Im}(\alpha_{2i}))^2 . \tag{10}$$

Hence, $P + m$ is expressed as a product of $d/2$ base sums of two squares. We then iteratively apply the Brahmagupta–Fibonacci identity:

$$(U_0^2+V_0^2)(U_1^2+V_1^2) = \left(\frac{U_0U_1 - V_0V_1 + U_0V_1 + U_1V_0}{\sqrt{2}}\right)^2 + \left(\frac{U_0U_1 - V_0V_1 - U_0V_1 - U_1V_0}{\sqrt{2}}\right)^2, \tag{11}$$

to compose pairs of polynomials hierarchically until we obtain U and V satisfying $P+m = U^2+V^2$.

H.2 Plaintext Packing and Algorithmic Correctness

Structure of Precomputed Plaintexts. In Algorithm 1, we recursively decompose the binary tree of polynomials of degree $2^{k-\ell}$ up to depth $j < k$. At the tree leaves (depth j), we have 2^j polynomials $U_0^{(j)}, \ldots, U_{2^j-1}^{(j)}$ of degree 2^{k-j}. Given an input slim ciphertext with 2^t active message slots such that $2^{t+j} \leq N$:

- For $0 \leq \delta \leq 2^{k-j}$, the plaintext vector $\mathbf{v}^{(\delta)}$ contains, in its slot range $[i \cdot 2^t,\ (i+1) \cdot 2^t - 1]$, the degree-$\delta$ coefficient of the sub-polynomial $U^{(j)}_{i \bmod 2^j}$.
- For each level $1 \leq \ell \leq j$, the constant plaintext $\mathbf{m}^{(\ell-1)}$ contains, in its slot range $[i \cdot 2^t,\ (i+1) \cdot 2^t - 1]$, the scalar offset $m^{(\ell-1)}_{i \bmod 2^{\ell-1}}$ associated with the decomposition at that level.

Theorem 1 (Correctness and Complexity of Algorithm 1)**.** *Algorithm 1 correctly evaluates $P(\mathsf{ct})$ on a slim ciphertext ct using $k+1$ multiplicative levels. Its computational complexity is dominated by $O(2^{(k-j)/2}) + j$ ciphertext-ciphertext multiplications and j homomorphic rotations.*

Proof. The evaluation function $\mathsf{Evaluate}$ computes the polynomial evaluations of the leaves $U_i^{(j)}(\mathsf{ct})$ simultaneously across the 2^j slot partitions using the Paterson–Stockmeyer algorithm. This consumes $k - j + 1$ multiplicative levels and $O(2^{(k-j)/2})$ ciphertext multiplications. The subsequent loop executes j iterations. In iteration ℓ, line 3 computes the pointwise square $\mathsf{ct} \odot \mathsf{ct}$ (1 multiplicative level), and line 4 evaluates the shift and offset subtraction:

$$\mathsf{ct} \leftarrow \mathsf{ct} \odot \mathsf{ct} + \mathsf{Rot}(\mathsf{ct} \odot \mathsf{ct}, 2^{t+\ell-1}) - \mathbf{m}^{(\ell-1)}. \tag{12}$$

By the recursive identity in Equation (1), this exactly computes the sum of squares $U_i^{(\ell-1)} = (U_i^{(\ell)})^2 + (U_{i+2^{\ell-1}}^{(\ell)})^2 - m_i^{(\ell-1)}$ across the corresponding slots. Accumulating over all j steps

requires j levels, j multiplications, and j key-switching rotations. Summing the stages gives $(k-j+1)+j = k+1$ total multiplicative levels and $O(2^{(k-j)/2})+j$ multiplications. □

Optimization to k Multiplicative Levels. If the leading coefficient $v_{2^{k-j}}$ of the leaf polynomials is scaled out such that the target polynomial is monic, the initial input ciphertext can be pre-multiplied by $\left(v_i^{(2^{k-j})}\right)^{2^{j-k}}$ during a preceding linear step in the Transformer layer (e.g., fused into RoPE or projection). In this configuration, Algorithm 1 can be evaluated in exactly k multiplicative levels rather than $k+1$.

H.3 Numerical Error and Stability Analysis

In approximate homomorphic encryption (CKKS), floating-point squarings amplify absolute approximation noise. Using the algebraic difference inequality:

$$|x^2 - y^2| \le 2\max(|x|, |y|) \cdot |x - y|, \tag{13}$$

where x and y denote the exact and the noisy value of the same quantity, each iteration of the recursive squaring loop in Algorithm 1 introduces an approximation error bound of approximately:

$$\approx 1 + \log_2\left(|U_i^{(\ell)}(x)| + |U_{i+2^{\ell-1}}^{(\ell)}(x)|\right) \quad \text{bits of precision loss in slot } i. \tag{14}$$

Comparison with Okada et al. (2023). In the exact-FHE algorithm of Okada et al. (2023), the polynomial is factored directly into linear root factors: $P(X) = p_d \prod_i (X - \alpha_i)$. Under CKKS, this product decomposition exhibits severe numerical instability: intermediate sub-products take arbitrarily large and small magnitudes across the complex plane, which destroys fixed-point precision under finite scaling factor Δ.

In contrast, our Sum-of-Squares decomposition strictly controls magnitude growth. Because $U^2 \le U^2 + V^2 = P + m$, for any input x in a bounded interval $\mathcal{I}$, the intermediate evaluation values are rigorously bounded by:

$$|U(x)|,\ |V(x)| \le \sqrt{\max_{x\in\mathcal{I}} |P(x)| + m}. \tag{15}$$

Optimal Decomposition Selection. Since the roots of $P + m$ can be paired into complex conjugates in multiple permutations, we can obtain different candidate polynomials U and V. During offline model setup, we perform a combinatorial search over root pairings to minimize the intermediate offset constants $m_i^{(\ell-1)}$ and minimize $\max_{x\in\mathcal{I}} |U_i^{(\ell)}(x)|$. Empirically, our optimized tree incurs only ≈ 1 bit of precision degradation per recursive level, which is easily accounted for by adjusting the CKKS scaling factor by 1 bit for the corresponding levels.

I Details of Long Public Context Processing

This appendix gives the full description of the techniques summarized in Section 4.

I.1 Heterogeneous prefill: plaintext algorithm and attention decomposition

For simplicity, we first describe the algorithm without FHE. We split the prompt of length ntok into two components: $P = (p_1, \ldots, p_{\mathsf{ptok}})$, the first public ptok tokens, and $S = (s_1, \ldots, s_{\mathsf{etok}})$, subsequent encrypted private etok tokens. Then, our plaintext algorithm is as follows.

- **Public prefill stage**: process the public prompt P in the clear and generate a public KV cache $(K_{\mathrm{pub}}, V_{\mathrm{pub}})$. It follows the standard Llama prefill.
- **Private prefill stage**: process the encrypted prompt S and the public KV cache $(K_{\mathrm{pub}}, V_{\mathrm{pub}})$ and generate the private KV cache $(K_{\mathrm{priv}}, V_{\mathrm{priv}})$ and the first output token.
- **Decode stages**: iteratively update the public and private KV caches and generate tokens; they are exactly the same as the standard Llama decode stages.

The public and private prefill stages follow the standard Llama prefill stages, but with different input lengths. The standard prefill algorithm uses $(\mathsf{ptok} + \mathsf{etok})$ tokens at once, and our prefill stages use ptok and etok tokens, respectively. Figure 3 visualizes the proposed algorithm.

Taken together with the clear public prefill, heterogeneous attention handles three types of interactions: plaintext-plaintext interactions within the public prefix, plaintext-ciphertext interactions between the encrypted queries and the public KV cache, and ciphertext-ciphertext interactions between the encrypted queries and the private KV cache.

I.2 The JKLS matrix multiplication algorithm

Let A and B be matrices in $\mathbb{R}^{d\times d}$ and C be their product. The JKLS matrix multiplication algorithm relies on the following (plaintext) matrix equation:

$$C \;=\; \sum_{k=0}^{d-1} \mathsf{rot}_C^k(\sigma(A)) \odot \mathsf{rot}_R^k(\tau(B)) \;\;, \tag{16}$$

where $\odot$ denotes the Hadamard (point-wise) multiplication between matrices stored as vectors by copying their rows one after another. rot_R and rot_C respectively represent rotation of row and column, and σ and τ represent relevant permutations. Specifically, we have, for all $0 \le i, j < d$:

$$\left(\mathsf{rot}_C^k(A)\right)_{i,j} = A_{i,j+k \bmod d} \;\;,\;\; (\sigma(A))_{i,j} = A_{i,i+j \bmod d} \;\;,$$
$$\left(\mathsf{rot}_R^k(A)\right)_{i,j} = A_{i+k \bmod d,j} \;\;,\;\; (\tau(A))_{i,j} = A_{i+j \bmod d,j} \;\;.$$

The original JKLS method homomorphically evaluates Equation (16) to obtain an encryption of C from encryptions of A and B. It consumes 3 multiplicative levels: one for σ and τ (performed in parallel), one for the column rotations (row rotations are implemented with a regular CKKS rotation and do not consume a level, if we write the matrix row after row in slots), and one for Hadamard multiplications.

I.3 Depth-optimized PCMM

While PCMM can be viewed as a special case of CCMM where the naive JKLS algorithm consumes three levels, we reduce this multiplicative depth to one by exploiting PCMM-specific properties. First, by setting A as the plaintext operand (in PC-attention, derived from the public KV cache) and B as the ciphertext operand, we eliminate one multiplicative level, as plaintext column rotations are depth-free.

Second, we adaptively update the packing structure of matrices to further reduce the depth. Observe that the naive JKLS algorithm consumes only a single level if the input matrix B is formatted as $\tau(\mathsf{ct}_B)$ instead of ct_B. Based on this observation, we modify the JKLS algorithm to take $\tau^{\ell+1}(\mathsf{ct}_B)$ as input and output $\tau^{\ell}(\mathsf{ct}_C)$ for an arbitrary integer ℓ.

We now describe our depth-optimized PCMM with BSGS optimization. Let pt_A and $\tau^{\ell+1}(\mathsf{ct}_B)$ be a plaintext and a ciphertext that respectively store the input matrices A and $\tau^{\ell+1}(B)$. For arbitrary integers g, b with $g \cdot b = d$, we compute $\tau^{\ell}(\mathsf{ct}_C)$ as

$$\sum_{j=0}^{g-1} \mathsf{rot}_R^{j\cdot b}\left(\sum_{i=0}^{b-1} \mathsf{pt}_{A,i,j,\ell} \odot \mathsf{rot}_R^{i}(\tau^{\ell+1}(\mathsf{ct}_B))\right) \;\;, \tag{17}$$

where the plaintext matrix $\mathsf{pt}_{A,i,j,\ell}$ is a rearranged version of pt_A defined as $\mathsf{pt}_{A,i,j,\ell} = \mathsf{rot}_R^{-\ell\cdot(i+j\cdot b)-j\cdot b} \circ \mathsf{rot}_C^{i+j\cdot b}(\tau^{\ell} \circ \sigma(\mathsf{pt}_A))$.

Eq. (17) yields an algorithm that computes $\tau^{\ell}(\mathsf{ct}_C)$ from pt_A and $\tau^{\ell+1}(\mathsf{ct}_B)$ using 1 multiplicative level and $\mathcal{O}(\sqrt{d})$ ciphertext rotations.

Equation (17) requires multiple rearranged plaintexts $\mathsf{pt}_{A,i,j,\ell}$. However, storing all of them is impractical due to the massive memory overhead, as pt_A ranges over the public KV cache of every layer and KV head. To tackle this issue, we store only $\tau^{\ell} \circ \sigma(\mathsf{pt}_A)$ and compute $\mathsf{pt}_{A,i,j,\ell}$ for each i,j and ℓ at runtime. This approach requires additional plaintext computations, but it significantly

reduces the memory requirements. This is applicable to other PCMM (and PCMv) algorithms such as Powerformer (Park et al., 2025) with a BSGS structure. However, our proposed PCMM algorithm requires fewer plaintext operations since each $\mathsf{pt}_{A,i,j,\ell}$ can be obtained with only two plaintext rotations from $\tau^\ell \circ \sigma(\mathsf{pt}_A)$ compared to 8 rotations required by Park et al. (2025) .

I.3.1 CORRECTNESS OF EQUATION (17)

We start with a lemma:

Lemma 2. *Let $M \in \mathbb{R}^{d\times d}$. For each integer k, we have:*

$$\tau\left(\mathsf{rot}_R^k(M)\right) = \mathsf{rot}_R^k(\tau(M)) \ ,$$

and

$$\tau\left(\mathsf{rot}_C^k(M)\right) = \mathsf{rot}_R^{-k} \circ \mathsf{rot}_C^k(\tau(M)) \ .$$

Proof. Let $0 \le i, j < d$. We have

$$\begin{aligned}\left(\tau\left(\mathsf{rot}_R^k(M)\right)\right)_{i,j} &= \left(\mathsf{rot}_R^k(M)\right)_{i+j,j} = M_{i+j+k,j} \\ &= (\tau(M))_{i+k,j} = \left(\mathsf{rot}_R^k(\tau(M))\right)_{i,j}\end{aligned}$$

and

$$\begin{aligned}\left(\tau\left(\mathsf{rot}_C^k(M)\right)\right)_{i,j} &= \left(\mathsf{rot}_C^k(M)\right)_{i+j,j} = M_{i+j,j+k} \\ = (\tau(M))_{i-k,j+k} &= \left(\mathsf{rot}_R^{-k} \circ \mathsf{rot}_C^k(\tau(M))\right)_{i,j} \ ,\end{aligned}$$

where all index operations are modulo d. □

Using Lemma 2, we proceed with Theorem 2, which modifies the original JKLS algorithm by applying τ^ℓ, to homomorphically compute $\tau^\ell(C)$ from $\tau^{\ell+1}(B)$.

Theorem 2. *Let $A, B \in \mathbb{R}^{d\times d}$ and $C = A \cdot B$. For each integer ℓ, we have*

$$\tau^\ell(C) = \sum_{k=0}^{d-1} \left(\mathsf{rot}_R^{-\ell\cdot k} \circ \mathsf{rot}_C^k(\tau^\ell \circ \sigma(A))\right) \odot \left(\mathsf{rot}_R^k(\tau^{\ell+1}(B))\right) \ .$$

Proof. We apply τ^ℓ to both sides of Equation (16):

$$\begin{aligned}\tau^\ell(C) &= \tau^\ell\left(\sum_{k=0}^{d-1} \mathsf{rot}_C^k(\sigma(A)) \odot \mathsf{rot}_R^k(\tau(B))\right) \\ &= \sum_{k=0}^{d-1} \tau^\ell\left(\mathsf{rot}_C^k(\sigma(A))\right) \odot \tau^\ell\left(\mathsf{rot}_R^k(\tau(B))\right) \ .\end{aligned}$$

Since the rot_R and τ operators commute by Lemma 2, we have:

$$\tau^\ell\left(\mathsf{rot}_R^k(\tau(B))\right) = \mathsf{rot}_R^k\left(\tau^{\ell+1}(B)\right) \ .$$

Moreover, Lemma 2 implies that

$$\begin{aligned}\tau^\ell\left(\mathsf{rot}_C^k(\sigma(A))\right) &= \tau^{\ell-1}\left(\tau\left(\mathsf{rot}_C^k(\sigma(A))\right)\right) \\ &= \tau^{\ell-1}\left(\mathsf{rot}_R^{-k} \circ \mathsf{rot}_C^k(\tau \circ \sigma(A))\right) \\ &= \mathsf{rot}_R^{-k}\left(\tau^{\ell-1}\left(\mathsf{rot}_C^k(\tau \circ \sigma(A))\right)\right) \\ &= \mathsf{rot}_R^{-2k}\left(\tau^{\ell-2}\left(\mathsf{rot}_C^k(\tau^2 \circ \sigma(A))\right)\right) \\ &= \cdots = \mathsf{rot}_R^{-\ell\cdot k} \circ \mathsf{rot}_C^k\left(\tau^\ell \circ \sigma(A)\right) \ .\end{aligned}$$

This completes the proof. □

Finally, decomposing $k = j \cdot b + i$ gives Equation (17):

$$\tau^\ell(C) = \sum_{j=0}^{g-1} \mathsf{rot}_R^{j\cdot b}\left(\sum_{i=0}^{b-1} (\mathsf{rot}_R^{-\ell\cdot(i+j\cdot b)-j\cdot b} \circ \mathsf{rot}_C^{i+j\cdot b}(\tau^\ell \circ \sigma(\mathsf{pt}_A))) \odot \mathsf{rot}_R^i(\tau^{\ell+1}(\mathsf{ct}_B))\right) \ .$$

I.4 LAYOUT HANDLING IN PC-ATTENTION

Both PCMM steps are performed using the previous algorithm, in slot-encoding, thus avoiding SlotToCoeff, CoeffToSlot, and bootstrapping during the wide PC-attention.

As each PCMM call simultaneously evaluates τ^{-1}, we apply $\tau^2(\cdot)$ right after RoPE, before the computation becomes wide. This allows us to obtain the correct layout at the end of the attention phase. For the remaining packing layouts, we use the same strategy as in Section 3.2.

As SoftMax is evaluated between the two PCMM steps, we perform it on a matrix M represented as $\tau(M)$:

$$(\mathsf{SoftMax}((\tau(M)_{j,0})_j), \cdots, \mathsf{SoftMax}((\tau(M)_{j,d-1})_j)) \quad .$$

Each vector $(\tau(M)_{j,i})_j$ is the i-th column of M rotated by i:

$$(M_{i,i}, \cdots, M_{d-1,i}, M_{0,i}, \cdots, M_{i-1,i}) \quad ,$$

and the value $\mathsf{SoftMax}((\tau(M)_{j,i})_j)$ is $\mathsf{SoftMax}((M_{j,i})_j)$ rotated by i, so that the output is

$$\tau\left(\mathsf{SoftMax}((M_{j,0})_j), \cdots, \mathsf{SoftMax}((M_{j,d-1})_j)\right) \quad .$$

allowing the next PCMM to proceed on SoftMax output.

I.5 SOFTMAX COST WITH LONG PUBLIC CONTEXTS

The heterogeneous prompt increases the dimension of each encrypted SoftMax instance from etok to $\mathsf{ptok} + \mathsf{etok}$, while the number of encrypted SoftMax instances to be handled in parallel remains equal to etok. In the context of the SoftMax algorithm from Cho et al. (2024) (see Section 2.2), the cheap and shallow main track thus operates on etok $(\mathsf{ptok} + \mathsf{etok})$ inputs while the deep and expensive auxiliary track operates only on etok inputs. The large public/private prompt, while increasing significantly the size of the input to SoftMax, has thus limited impact on SoftMax costs. In our experiments ($\mathsf{ptok} = 31\mathsf{etok}$), a $32\times$ increase on SoftMax input size leads only to about a $1.9\times$ increase in total SoftMax computation time on 8 GPUs (from 81 ms to 153 ms per layer; Table 11).

The main track, which for long prompts manipulates many ciphertexts during PC-attention, must fit within the level budget of PC-attention without bootstrapping; since each iteration of the normalize-and-square strategy (see Section 2.2) adds to its depth, we need to keep the number of iterations small. With the value ranges measured during calibration (see Section 3.1), two iterations suffice in all layers except the last, which uses five; the main track then uses only 8 levels in all other layers.

J FULL END-TO-END TIMINGS

Table 7 lists the raw timings plotted in Figure 1.

K MULTI-GPU IMPLEMENTATION DETAILS

This appendix details the multi-GPU aspects of Sylph summarized in Section 5. Sylph realizes the homomorphic operators introduced in the previous sections by distributing computation across GPUs so as to parallelize the matrix products while limiting inter-GPU traffic.

K.1 DATA PARTITIONING

We parallelize PCMM across multiple GPUs using a two-dimensional partitioning over the input and output dimensions. Our main goal is to parallelize the PCMM computation; as a byproduct, the plaintext weight matrices, which must reside in GPU memory throughout the computation, are distributed as well.

Let $W = (w_{i,j})_{0 \le i < d, 0 \le j < d'} \in \mathbb{R}^{d \times d'}$ be a plaintext weight matrix used during a PCMM step of the form $B = W \cdot A$, where $A = (a_{ij})_{0 \le i < d', 0 \le j < d''}$ and $B = (b_{ij})_{0 \le i < d, 0 \le j < d''}$.

Given g GPUs, we let p_r and p_c be two positive integers such that $p_r | d$, $p_c | d'$ and $p_r \cdot p_c = g$, and split W blockwise as

$$W^{(u,v)} = (w_{i+ud/p_r, j+vd'/p_c})_{0 \le i < d/p_r, 0 \le j < d'/p_c},$$

Table 7: Timings (in seconds) for HE inference systems under various hardware configurations. Cerium timings are extracted from Jayashankar et al. (2026). † MOAI results are estimates, as the workload exceeds the GPU memory capacity. Input settings per benchmark are noted in the first column. RP6000 refers to RTX PRO 6000. N/A means that the corresponding hardware was not accessible, and – that the configuration was not measured.

Benchmark	Phase	GPU	1×	2×	4×	8×
Sylph (Ours) 128 priv.	Decode	B200	24	17	12	N/A
	Decode	RP6000	31	24	19	18
	Prefill	B200	87	47	26	N/A
	Prefill	RP6000	116	63	36	20
Sylph (Ours) 3968 pub. +128 priv.	Decode	B200	-	-	15	N/A
	Decode	RP6000	-	-	23	22
	Prefill	B200	-	-	90	N/A
	Prefill	RP6000	-	-	129	64
Cerium 128 priv.	Prefill	B200	253	215	152	134
	Prefill	H100	698	580	346	295
MOAI† 128 priv.	Prefill	B200	5,963	-	-	-
	Prefill	RP6000	22,275	-	-	-

and accordingly

$$A^{(v)} = (a_{i+vd'/p_c,j})_{0\le i<d'/p_c,0\le j<d''},$$
$$B^{(u)} = (b_{i+ud/p_r,j})_{0\le i<d/p_r,0\le j<d''},$$

for $0 \le u < p_r$, $0 \le v < p_c$.

Each GPU is assigned a pair (u, v) as a logical coordinate. The GPU numbered (u, v) stores the plaintext block $W^{(u,v)}$ and the activation values $A^{(v)}$, and computes the term $W^{(u,v)} \cdot A^{(v)}$. In view of

$$B^{(u)} = \sum_{0\le v<p_c} W^{(u,v)} \cdot A^{(v)}, \tag{18}$$

the PCMM result is obtained by gathering, for all u, the products $W^{(u,v)} \cdot A^{(v)}$ across the nodes (u, v).

This partitioning distributes PCMM computation across all GPUs, with each GPU storing approximately $1/g$ of the weight matrix, and a proportion $1/p_c$ of the activation input. This partitioning describes the logical matrix-level view; the actual ciphertext representation follows the packing layouts defined in Section 3.2. For PCMV, we use the one-dimensional special case with $p_c = 1$, where the activation is not partitioned and only the plaintext matrix is distributed across GPUs. The shape (p_r, p_c) is chosen per projection: the O, Gate and Up projections split only the output dimension ($p_c = 1$), the Down projection splits only its input dimension, i.e., the FFN hidden dimension ($p_r = 1$), and the QKV projection splits only the output dimension on 2 GPUs and both dimensions on 4 and 8 GPUs (Table 8).

K.2 Inter-GPU Communication

Between decoder layers, the residual stream is partitioned along the hidden dimension: each GPU owns a slice of the hidden channels of every token. Attention heads are processed in independent *calls*; with 2 GPUs each call runs on a single GPU, and with 4 or 8 GPUs each call is shared by a group of GPUs (its *call group*). Table 8 lists, in execution order, the collective operations performed within one decoder layer during prefill. They fall into three groups.

Reduction of block products. As shown by Eq. (18), a PCMM whose input dimension is split ($p_c > 1$) requires accumulating block products: for each u, the GPUs (u, v), $0 \le v < p_c$, sum their products $W^{(u,v)} \cdot A^{(v)}$ through a reduce-scatter, after which each GPU holds the output blocks it

Table 8: Inter-GPU communication within one decoder layer during prefill, in execution order. "All" denotes all g GPUs; a call group is the set of GPUs sharing an attention call.

Position	Collective (participants)	Data and purpose
Attention RMSNorm	All-reduce (all)	Partial sums of squares over owned hidden channels, for the inverse RMS.
Before QKV PCMM	Send/Recv	Input channels of each GPU's QKV block (on 4/8 GPUs, only the required input range).
After QKV PCMM, before bootstrapping	Reduce-scatter (QKV group; 4/8 GPUs only)	Bottom-level ring-degree-2^{12} partial products, summed and sent to the owners of the output blocks.
After QKV bootstrapping	K/V exchange (GPU pairs)	One GPU of each pair computes the compact (GQA) K, the other the compact V; they swap results.
After RoPE and S2C of Q/K, before QK CCMM	Q/K exchange (pairs); all-gather (call group, 4/8 GPUs)	Coefficient-encoded Q/K ciphertexts for each GPU's QK tile; after restoring, score blocks of the partner's call are returned.
SoftMax, each normalization round	All-gather + HE addition (4 GPUs); all-reduce (8 GPUs); none (2 GPUs)	Partial row sums of the GPUs sharing a call.
After SoftMax and partial S2C of P, before ScoreV CCMM	All-gather (call group, 4/8 GPUs)	Probability ciphertexts P.
After ScoreV (fused restore and S2C), before O PCMM	All-gather (all)	The 8 coefficient-encoded ciphertexts of the O-projection input; each GPU computes its own output channels.
FFN RMSNorm	All-reduce (all)	As for the attention RMSNorm.
Before Gate/Up PCMM	All-gather (all)	Full normalized input, as Gate/Up split the output dimension only.
Gate/Up $\rightarrow$ SiLU $\rightarrow$ Hadamard $\rightarrow$ Down	None	Each GPU keeps its slice of the FFN hidden dimension.
After Down PCMM, before bootstrapping	Reduce-scatter (all)	Partial products over FFN hidden slices, summed and sent to the owners of the hidden channels; each GPU bootstraps only its outputs.

owns. This occurs after the QKV projection (on 4 and 8 GPUs) and after the Down projection. The block products are exchanged right after the PCMM, as bottom-level ciphertexts of ring degree 2^{12}, so each exchanged ciphertext is small; each GPU then composes the ring-degree-2^{12} ciphertexts it owns into ring-degree-2^{16} ciphertexts and bootstraps them locally.

Reduction of scalars. RMSNorm sums, across all GPUs, the partial sums of squares over the hidden channels each GPU owns. The SoftMax normalization sums the partial row sums of the GPUs sharing an attention call, once per normalize-and-square round. Both exchange a single value per token or per SoftMax row.

Redistribution of activations. The remaining exchanges bring each GPU the operands of its next block product: the input channels of its QKV block, the K/V and Q/K ciphertexts of its attention tiles, the SoftMax output P for ScoreV, the full input of the O projection, and the full normalized input of the Gate/Up projections, whose weights are split along the output dimension only. No communication occurs between the Gate/Up projections and the Down projection, since each GPU keeps its slice of the FFN hidden dimension throughout.

K.3 Model Conversion

We convert the Llama-3.1-8B weights into the layouts directly consumed by Bae et al. (2024) and He et al. (2024). The conversion is primarily a layout transformation: weights are partitioned and tiled. He et al. (2024) consumes coefficient-domain plaintext polynomials, hence further requires transforming the plaintext polynomials into the NTT domain.

Table 9 summarizes the resulting memory footprint for the converted Llama-3.1-8B projection weights across all transformer layers. As the original 16-bit weights are stored under a ≈ 64-bit modulus, we observe a $\approx \times 4$ storage expansion overall.

K.4 Experimental Setup

RTX PRO 6000 experiments are conducted on a server with eight NVIDIA RTX PRO 6000 GPUs (96 GB of memory each) and two Intel Xeon 6530P CPUs (PCIe Gen5, 512 GB host memory). B200 experiments are carried out on a server equipped with four B200 GPUs interconnected through

Table 9: Memory footprint of converted Llama-3.1-8B projection weights. The baseline stores weights in 16-bit precision.

Representation	Total memory	Expansion
Original 16-bit weights	13.0 GiB	1.00×
Bae et al. (2024) packed PCMM tiles	52.0 GiB	4.00×
He et al. (2024) NTT plaintexts	58.0 GiB	4.49×

NV18 NVLink and two AMD EPYC 9365 CPUs (2.2 TB host memory). Both systems use the HEaaN2 FHE library (CryptoLab, 2025) for CKKS primitive operations and cryptographic setup.

Table 10 lists the CKKS parameters.

Table 10: CKKS parameters of Sylph. They provide 128-bit security (lattice estimator, MATZOV cost model), estimated for $N = 2^{16}$, $\log QP = 1730$, $h = 1024$ and $\sigma = 3.2$.

Parameter	Value
Ring degree	$N = 2^{16}$; conjugate-invariant encoding with 2^{16} real slots
Ring degree per operation	2^{16} (bootstrapping, non-linear layers), 2^{12} (batch CCMM, decode PCMv), 2^8 (PCMM, module-LWE)
Scaling factor	$\Delta = 2^{38}$
Levels	15 after encryption, 14 after bootstrapping
Modulus	53-bit base prime; $\log Q = 735$ for computation and 1391 for bootstrapping (24 primes of 61 bits); $\log QP \leq 1730$
Key switching	$\mathrm{dnum} = 1$ for relinearization and rotation keys
Secret and error	ternary secret of Hamming weight $h = 1024$ ($h = 2048$ for the conjugate-invariant key of degree 2^{17}); sparse-secret encapsulation with $h = 32$; $\sigma = 3.2$
Bootstrapping	SlotToCoeff: 3 levels, scale 2^{28}; CoeffToSlot: 4 levels, scale 2^{44}, first level fused with modulus raising; EvalMod: 8 levels, scale 2^{60}, Chebyshev degree 56 with 2 double-angle iterations; 1 level each for the conjugate-invariant conversions

K.5 Per-layer Breakdown

Table 11 reports the per-layer execution time of Sylph on Llama-3.1-8B for prefill ($T = 128$) and decode ($T = 1$). The dominant costs come from PCMM and CCMM kernels. With 8 GPUs, prefill QKV projection decreases from 326 ms to 65 ms, O projection from 315 ms to 40 ms, and FFN gate/up projection from 1040 ms to 132 ms, showing effective scaling of large PCMMs. This (almost) linear speedup illustrates the efficiency of our distribution and communication strategies (Appendices K.1 and K.2).

The encrypted attention kernel also benefits from distribution. In prefill, QK, SoftMax, and ScoreV decrease from 316/440/452 ms to 41/81/63 ms, respectively. With 3968 public tokens on 8 GPUs, QK and ScoreV against the public KV cache (PC-operation) take 586 and 635 ms, versus 41 and 63 ms against the 128 private tokens (CC-operation), as their cost grows with the number of public tokens.

Overall, Sylph reduces per-layer prefill latency from 3.55 s on one GPU to 0.55 s on 8 GPUs, achieving a 6.5× speedup. Decode is less scalable because $T = 1$ limits parallelism, but still improves from 0.98 s to 0.56 s per layer.

L Fidelity of Prior Systems' Approximations

Recent 8B-scale FHE systems report latency but neither activation ranges nor fidelity to plaintext inference. We replay the non-linear approximations of the released implementations of Cerium (Jayashankar et al., 2026) and MOAI (Zhang et al., 2025b) in plaintext on Llama-3 activa-

Table 11: Per-layer GPU time on RTX PRO 6000 GPUs for Sylph's homomorphic Llama-3.1-8B evaluation. We use $T = 128$ for prefill and $T = 1$ for decode, $H = 4096$, $I = 14336$, $h = 32$, $d = 128$.

		Prefill (ms)		Decode (ms)	
Step	**Shape**	**1 GPU**	**8 GPU**	**1 GPU**	**8 GPU**
QKV projection					
Pre-attn norm	$T\times H$	20	19	9	10
QKV proj.	$(T\times H)(H\times 6144)$	326	65	135	70
RoPE	$T\times 5120$	35	12	2	2
Attention CC-operation					
QK scores	$(T\times d)(d\times T)\times h$	316	41	154	124
SoftMax	$T\times T\times h$	440	81	49	50
ScoreV	$(T\times d)(d\times T)\times h$	452	63	155	111
Attention PC-operation					
QK scores, pub.	$(T\times d)(d\times 3968)\times h$	-	586	-	70
SoftMax	$T\times 4096\times h$	-	153	-	123
ScoreV, pub.	$(T\times 3968)(3968\times d)\times h$	-	635	-	83
O projection					
O proj.	$(T\times H)(H\times H)$	315	40	80	40
Feed-forward network					
Pre-FFN norm	$T\times H$	20	20	20	20
Gate/up proj.	$(T\times H)(H\times 2I)$	1040	132	133	57
Gate act.	$T\times I$	198	27	4	4
Down proj.	$(T\times I)(I\times H)$	391	50	71	45

tions; as these emulations exclude CKKS noise, bootstrapping error and packing, they lower-bound the error of the encrypted programs.

Cerium. With the approximations and scaling constants of the artifact (commit `a98bf69`) on Llama-3.1-8B (1,024 C4 validation documents truncated to 128 tokens and the 5 artifact prompts), inputs of RMSNorm, SiLU and SoftMax leave their valid domains; for instance, SiLU inputs reach -55.3 against a limit of -48.9, where the sign polynomial returns NaN. As a result, 4 of 1,029 inputs produce non-finite logits, the next-token prediction changes on 10.4% of all positions, and only 23% of 32-token greedy generations match the exact model.

MOAI. With the Llama-path approximations of the released GPU code on Llama-3-8B (32 WikiText-2 sequences of 128 tokens), SiLU inputs reach 27.1 against a domain of $[-1, 1]$, RMSNorm inputs fall below the convergence region of the inverse square root, and 97% of SoftMax denominators lie where the reciprocal error exceeds 3%. Inserting all approximations raises perplexity from 18.07 to about 1.27×10^5.

These results reflect the approximations as released; recalibrating their constants would require the activation ranges, which neither system reports. In contrast, Sylph's encrypted inference matches the FP16 perplexity on WikiText-2 (Table 4).