YOU ONLY INDEX ONCE: CROSS-LAYER SPARSE ATTENTION WITH SHARED ROUTING

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-context inference in modern LLMs is increasingly constrained by decoding efficiency, especially in reasoning-heavy settings where models generate long intermediate chains of thought. Existing sparse attention methods often face a practical efficiency-quality trade-off. Structured block sparse methods typically provide stronger acceleration but incur noticeable quality loss, while token sparse methods are usually more accurate yet deliver limited end-to-end speedup because top-k routing over the full cache remains expensive. In this work, we propose cross-layer sparse attention (CLSA), which is built on top of KV-sharing architectures such as YOCO. The core idea is to share not only the KV cache across cross-decoder layers, but also the routing index. A single indexer computes token-level top-k selection once and reuses the resulting index across layers, thereby preserving the fine-grained selectivity of token sparse attention while amortizing the routing overhead. The resulting architecture improves all major inference bottlenecks jointly, including pre-filling, KV-cache storage, and long-context decoding. Experiments across short-context and long-context benchmarks show that CLSA is both accurate and efficient, achieving up to 7.6$\times$ decoding speedup and 17.1$\times$ overall throughput improvement at 128K context. These results suggest a more complete architectural solution for long-context LLMs that jointly advances model quality and inference efficiency.

1 INTRODUCTION

Long-context inference has become a common operating regime for modern LLMs, especially in reasoning-heavy settings such as chain-of-thought generation and test-time scaling (Jaech et al., 2024; Guo et al., 2025). In these scenarios, models often need to decode long intermediate reasoning traces while repeatedly attending to a large context, making inference increasingly decoding-bound as the sequence grows. At the same time, pre-filling becomes more expensive and the KV cache grows with context length. Designing an architecture that remains efficient under long contexts is therefore now a core requirement rather than a niche optimization.

Sparse attention is a natural direction for reducing the cost of long-context inference, but existing methods often face a difficult efficiency-quality trade-off. In practice, block-sparse attention (Yuan et al., 2025; Lu et al., 2025; Tang et al., 2024; Gao et al., 2024) usually delivers larger wall-clock speedups because its structured sparsity maps better to GPU execution, but it also tends to introduce a coarser approximation and more noticeable quality loss. Token-sparse attention (Liu et al., 2025) is often more accurate because it can preserve finer-grained salient tokens, yet its end-to-end acceleration is usually limited. A key reason is that token sparse methods still require a routing stage based on top-k selection over the full cache, and this step is irregular and expensive on modern GPUs, especially when it is recomputed independently across many layers during decoding.

In this work, we propose cross-layer sparse attention (CLSA), which is built on top of the KV sharing design of cross-attention architectures such as YOCO (Sun et al., 2024). The central idea is to extend sharing from memory to routing. When multiple cross-decoder layers read from the same KV cache, they also share the same routing index. Concretely, a single indexer computes token-level top-k routing once and the resulting index is reused across layers. In this way, the model

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.

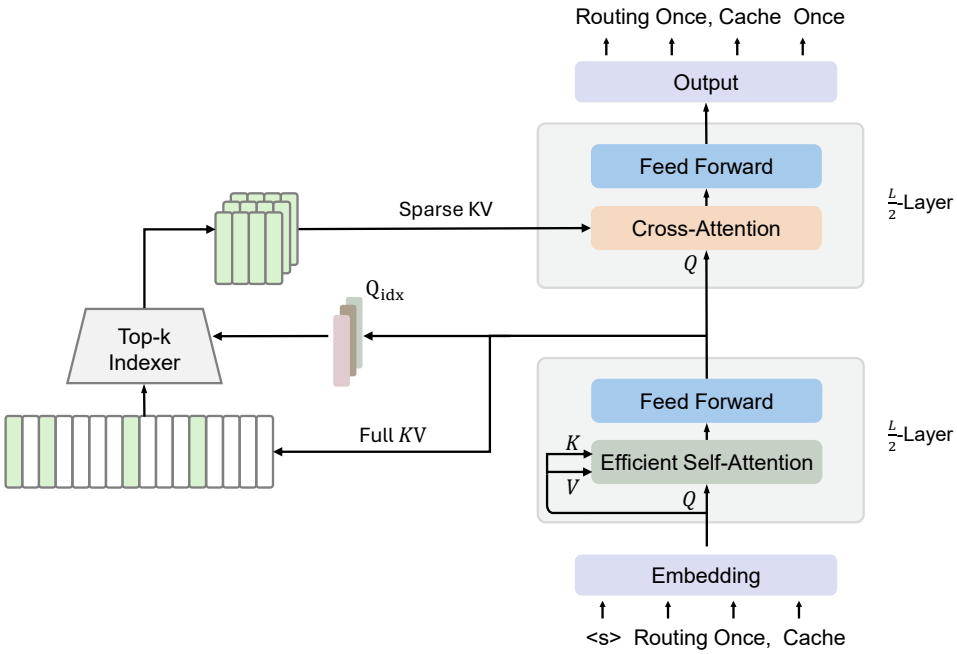


Figure 1: Overview of cross-layer sparse attention. The self-decoder first produces a shared KV cache, which is computed only once and then reused by all subsequent cross-decoder layers. During this stage, a shared query-aware indexer jointly generates the routing queries and keys and computes a token-level sparse top-k index for each query token. This sparse index is also produced only once and is shared across the following cross-decoder layers, allowing them to reuse the same selected KV positions instead of recomputing layer-specific sparse routing.

preserves the main advantage of token-sparse attention, namely selecting a compact active subset of informative tokens without sacrificing quality, while substantially reducing the practical cost of sparse decoding by amortizing the routing overhead.

The resulting architecture provides a unified treatment of the major inference bottlenecks in long-context LLMs. Through KV sharing, it retains YOCO's advantages in pre-filling and KV-cache storage, while shared-index sparse retrieval improves decoding efficiency by avoiding repeated dense global attention and repeatedly recomputed routing. Consequently, the overall system approaches a favorable efficiency frontier across pre-filling, KV-cache footprint, and long-context decoding, rather than improving only one aspect at the expense of the others.

Our experiments show that CLSA preserves model quality across both short and long context benchmarks spanning multiple domains, while maintaining nearly lossless behavior relative to dense baselines. Attention-pattern analysis further shows that sharing one routing index across layers has only a minor effect on the resulting attention behavior, supporting the central assumption behind our design. At the same time, CLSA delivers substantial acceleration. At 128K context, it improves decoding throughput by up to $7.6\times$ over the Transformer baseline and improves overall end-to-end throughput by up to $17.1\times$. Taken together, these results suggest that CLSA provides a potentially promising LLM architecture that better reconciles model quality and inference efficiency.

2 Method

Figure 1 gives an overview of our method. We build on the cross-attention architecture of YOCO (Sun et al., 2024), which naturally decomposes the model into a self-decoder and a cross-decoder. The self-decoder first encodes the input sequence into shared hidden states and constructs a single KV cache. On top of these shared states, we add a lightweight query-aware indexer that computes a token-level top-k routing index once. The cross-decoder layers then reuse both the

shared KV cache and the shared routing index, so each layer keeps its own query states and FFN transformation while attending only to the selected KV positions.

This overview highlights the main design principle of cross-layer sparse attention: when several decoder layers read from the same memory, the expensive routing decision should also be tied to that memory and shared across layers. As a result, CLSA preserves the fine-grained selectivity of token-sparse attention, but avoids recomputing layer-specific top-k indices during decoding. The following subsections detail the sparse attention formulation, the multi-layer distillation objective used to train the shared indexer, and the resulting inference advantages.

Efficient Self-decoder. The self-decoder is kept unchanged from YOCO. It performs an efficient attention mechanism and constructs the shared KV cache exactly once. This stage encodes the history into reusable memory while avoiding dense full-context attention throughout the whole stack. As a result, the model preserves YOCO's efficiency advantages in both pre-filling and KV-cache storage.

Sparse Cross-decoder. Each cross-decoder layer consists of a cross-attention module followed by an FFN block. Its role is to retrieve relevant information from the shared KV cache and refine the representation through the feed-forward transformation. In dense YOCO, the cross-attention module performs dense full attention over the shared cache. Our modification is to replace this dense retrieval with CLSA driven by the shared indexer, while keeping the FFN and the rest of the cross-decoder unchanged.

2.1 Cross-Layer Sparse Attention

As shown in Figure 1, we consider a stack of decoder layers that all query the same external memory through a cross-attention module. Let $H \in \mathbb{R}^{n\times d}$ denote the shared hidden states written into the memory, let $K, V \in \mathbb{R}^{n\times d}$ denote the corresponding shared KV cache, and let $Q^{(l)}$ be the query states of decoder layer l. A dense cross-attention module computes

$$O^{(l)} = \mathrm{Attn}(Q^{(l)}, K, V), \tag{1}$$

which means every decoder layer scans the full cache. Our goal is to keep the shared-memory structure, but replace dense retrieval with a routed sparse variant. To do so, we introduce an additional single-head indexing branch on top of the shared hidden states. Specifically, the index queries and keys are computed as:

$$Q_{\mathrm{idx}} = HW^{Q}_{\mathrm{idx}}, \quad K_{\mathrm{idx}} = HW^{K}_{\mathrm{idx}}, \tag{2}$$

where $Q_{\mathrm{idx}}, K_{\mathrm{idx}} \in \mathbb{R}^{n\times d_{\mathrm{idx}}}$. Unlike the multi-head attention projections used by the main attention module, this indexing branch uses only one head. We then form indexing scores and routing indices

$$I = Q_{\mathrm{idx}} K^{\top}_{\mathrm{idx}}, \quad S_t = \mathrm{TopK}(I_t, k) \tag{3}$$

and restrict attention to the selected tokens:

$$O^{(l)}_t = \mathrm{Attn}(Q^{(l)}_t, K_{S_t}, V_{S_t}) \tag{4}$$

Here the index query and index key are both derived from the same shared hidden states and have shape $[n, d_{\mathrm{idx}}]$, while the layer-wise query $Q^{(l)}$ still varies across decoder layers. The activated set size k is much smaller than n, so each layer only attends to a compact subset of the global memory.

The central idea is to bind the routing index to the shared KV cache, rather than treating them as separate objects. Once multiple decoder layers attend to the same KV cache, they should also reuse the same routing index, so inference only needs to compute top-k once.

2.2 Multi-Layer Distillation

Sharing one routing index across multiple decoder layers is efficient, but the selected tokens must be useful to several layers simultaneously rather than matching the preference of one layer in isolation. Follow prior sparse attention methods with distillation (Gao et al., 2024; Bai et al., 2026), we use a multi-layer distillation objective to train the shared indexer.

Table 1: Comparison of inference complexity. N, L, and D denote sequence length, number of layers, and hidden dimension. W_1 denotes the local window size, and W_2 denotes the number of selected tokens in sparse attention. γ denotes the fraction of global-attention layers in a hybrid model, and η denotes the per query-key routing cost of the indexer.

Model	KV Cache Memory	Prefilling Time	Decoding Time
Transformer	$\mathcal{O}(LND)$	$\mathcal{O}(LN^2D)$	$\mathcal{O}(LND)$
Hybrid TRM	$\mathcal{O}(L(\gamma N + (1-\gamma)W_1)D)$	$\mathcal{O}(L(\gamma N^2 + (1-\gamma)W_1 N)D)$	$\mathcal{O}(L(\gamma N + (1-\gamma)W_1)D)$
YOCO (Dense)	$\mathcal{O}((N + W_1 L)D)$	$\mathcal{O}(\frac{L}{2}W_1 ND)$	$\mathcal{O}(\frac{L}{2}(N + W_1)D)$
DSA	$\mathcal{O}(LND)$	$\mathcal{O}(LW_2 ND + \eta LN^2)$	$\mathcal{O}(LW_2 D + \eta LN)$
YOCO (CLSA)	$\mathcal{O}((N + W_1 L)D)$	$\mathcal{O}(\frac{L}{2}W_1 ND)$	$\mathcal{O}(\frac{L}{2}(W_1 + W_2)D + \eta N)$

For an input sequence, the dense cross-attention modules provide attention distributions from all decoder layers and all heads. We first aggregate them into a common target

$$\bar{A} = \frac{1}{LH} \sum_{l=1}^{L} \sum_{h=1}^{H} \mathrm{softmax}\left(Q^{(l,h)} {K^{(h)}}^{\top}\right), \tag{5}$$

and then match that target with the indexer distribution using

$$\mathcal{L}_{\mathrm{KD}} = \frac{1}{n} \sum_{t=1}^{n} \mathrm{KL}\left(\mathrm{sg}\left[\bar{A}_t\right] \,\middle\|\, \mathrm{softmax}(I_t)\right), \tag{6}$$

where L is the number of decoder layers, H is the number of attention heads in each layer, and $\mathrm{sg}[\cdot]$ denotes stop-gradient. Intuitively, the shared indexer learns to preserve the consensus salient tokens that remain important across the full decoder stack.

Stage 1: indexer warmup with frozen backbone. We first warm up the shared indexer using only the distillation objective while freezing the backbone parameters. In this stage, the indexer learn a stable routing pattern before it is coupled with language modeling.

$$\mathcal{L}_{\mathrm{stage1}} = \mathcal{L}_{\mathrm{KD}}. \tag{7}$$

Stage 2: joint sparse adaptation. After the indexer warmup, we optimize the model with both language modeling and distillation losses. The main purpose of this stage is to let the backbone adapt to the sparse attention distribution induced by the shared indexer:

$$\mathcal{L}_{\mathrm{stage2}} = \mathcal{L}_{\mathrm{LM}} + \lambda \mathcal{L}_{\mathrm{KD}}. \tag{8}$$

where λ is a fixed weighting coefficient for the KD loss, and we set $\lambda = 0.1$ in all experiments.

2.3 Inference Advantages

Decoding efficiency, pre-filling efficiency, and KV-cache footprint cover most of the practical bottlenecks in LLM inference. Our design improves these three aspects in a complementary way.

Decoding efficiency from shared top-k routing. In a standard sparse design, each decoder layer still needs to run its own routing procedure to identify the top-k tokens, and that routing must still be computed over the full-length cache. More importantly, the top-k operator is not well matched to modern GPU execution. Unlike dense matrix multiplications, it cannot effectively leverage Tensor Core acceleration, so its wall-clock cost can account for a large fraction of decoding time. Our shared-indexer design removes this redundancy by computing the top-k result once and reusing it across the decoder stack. As a result, sparse retrieval becomes practically useful for decoding. While the model still attends to only a small active set, but inference no longer wastes time recomputing the same expensive top-k step in every layer.

Table 2: Main downstream benchmark results for the 4B Transformer baseline, dense YOCO, and YOCO with cross-sparse attention. YOCO (CLSA) maintains the overall capability profile of the dense baselines while improving ARC-Challenge (ARC-C), GSM8K, and DROP, and matching the best HumanEval score.

Model	ARC-C	BBH	GSM8K	HellaSwag	HumanEval	MMLU	DROP	WinoGrande
Transformer	0.453	**0.420**	0.434	0.667	0.384	**0.527**	0.366	**0.638**
YOCO (Dense)	0.461	0.411	0.430	**0.676**	**0.396**	0.519	0.387	0.630
YOCO (CLSA)	**0.465**	0.418	**0.470**	0.674	**0.396**	0.513	**0.391**	0.616

Pre-filling and KV-cache efficiency inherited from YOCO. YOCO (Sun et al., 2024) already makes pre-filling efficient by avoiding dense full-context attention, and it reduces KV-cache memory by letting the cross-decoder layers reuse a single shared cache. Our method preserves these benefits completely, because it only changes how the shared cache is read by the cross-attention modules in the cross-decoder. The two components are complementary: YOCO addresses pre-filling and KV-cache storage, while cross-sparse attention addresses decoding. Together they cover nearly all major bottlenecks in LLM inference, yielding a unified design that is efficient in pre-filling, memory footprint, and long-context generation.

Complexity comparison with other architectures. Table 1 highlights why improving only one component is insufficient for long-context inference. Pure dynamic sparse attention Liu et al. (2025) reduces the number of tokens read by each attention layer, but it still maintains a layer-wise KV cache of size $\mathcal{O}(LND)$ and therefore does not address the memory bottleneck. Its decoding cost can also be dominated by the indexer term ηLN, because token selection must be performed over the full cache and repeated across layers. Hybrid architectures Yang et al. (2025a); Team et al. (2025b); Gao et al. (2026) reduce part of the attention cost by mixing efficient-attention layers with global retrieval layers, but their gains are constrained by the hybrid ratio γ. In contrast, CLSA combines KV sharing with a shared routing index, reducing cache storage through YOCO-style memory sharing while paying the expensive indexer cost only once across the cross-decoder stack.

3 Experiments

3.1 Setup

Model configuration We compare a Transformer baseline, a dense YOCO model, and YOCO with cross-sparse attention, all at the 4B scale. The baseline uses RoPE with `base`=500,000, and both YOCO variants use RNoPE (Yang et al., 2025b), where RoPE (Su et al., 2024) is activated in SWA with `base`=10,000 and NoPE in global attention. Both YOCO variants use sliding window attention in the self-decoder with window size 512. QK normalization is enabled in the Transformer and the self-decoder of YOCO. The maximal activated tokens in CLSA is 2048. Across models we keep width and depth aligned with hidden size 2560, 7680 FFN width, 32 layers, 20 heads and 4 KV heads, no weight tying. For the YOCO variants, the 32 layers are split into 16 self-decoder layers and 16 cross-decoder layers. The full field-by-field layout is in Appendix C.

Training hyper-parameters Dense pretraining runs in two stages. In dense stage 1 we use batches of 8M tokens per step, maximum sequence length 8192, peak learning rate 3×10^{-4} with minimum 3×10^{-5}, 2000 warmup iterations, and 125,000 optimizer updates. Dense stage 2 increases the context cap to 32,768 tokens, fixes the learning rate at 3×10^{-5}, and runs for 10,000 optimizer steps. Sparse adaptation also uses two stages on 32,768-token sequences: sparse stage 1 likewise uses 8 M tokens per step, learning rate 3×10^{-4} with the same minimum 3×10^{-4}, for 2500 steps, and sparse stage 2 continues for another 2500 steps at 3×10^{-5}. Further settings and stage-wise details are given in Appendix B.

Evaluation benchmarks We evaluate models with BBH (Suzgun et al., 2023) and MMLU (Hendrycks et al., 2020) for heterogeneous knowledge and reasoning, DROP (Dua et al., 2019) and ARC-Challenge (Clark et al., 2018) for reading-style reasoning, HellaSwag (Zellers

Table 3: RULER results at 16K and 32K context lengths. TRM denotes the standard Transformer. CLSA maintains strong single-needle retrieval performance and achieves the best average score at 32K, with gains mainly from the harder multi-needle settings.

Ctx	Model	Single Needle			Multi Needle					RULER Tasks				Avg
		S1	S2	S3	MK1	MK2	MK3	MQ	MV	QH	QS	CWE	FWE	
16K	TRM	100.0	99.8	98.4	88.2	71.4	14.4	85.7	85.6	28.8	33.2	15.6	52.1	**64.4**
	YOCO (Dense)	100.0	99.8	96.4	69.4	91.6	61.2	45.8	49.3	30.8	31.4	9.4	67.0	62.7
	YOCO (CLSA)	100.0	100.0	98.4	70.4	92.4	58.4	53.0	47.2	31.2	32.7	9.8	61.6	62.9
32K	TRM	100.0	98.8	83.4	57.0	38.8	0.8	45.6	42.6	21.2	20.2	1.8	43.8	46.2
	YOCO (Dense)	100.0	90.2	74.8	53.2	84.0	43.6	27.0	29.0	30.6	30.6	4.6	60.3	52.3
	YOCO (CLSA)	100.0	93.6	83.2	58.4	88.8	38.0	31.6	29.8	29.2	29.2	5.1	50.2	**53.1**

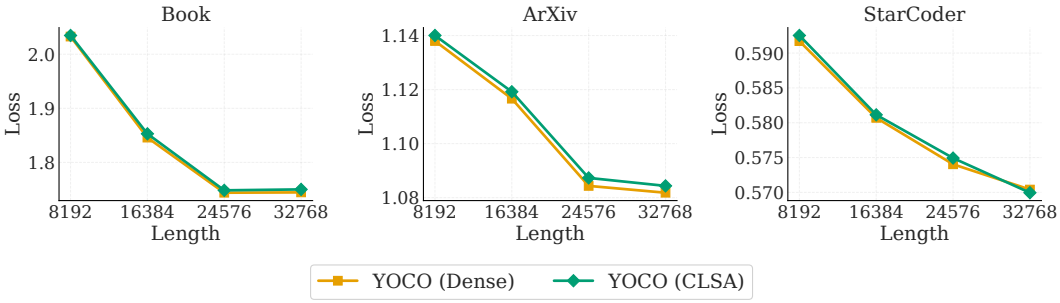


Figure 2: Long-context validation loss for dense and cross-sparse attention on Books, ArXiv, and StarCoder. The two curves track each other closely from 8K to 32K tokens.

et al., 2019) and WinoGrande (Levesque et al., 2012) for commonsense multiple-choice completion, GSM8K (Cobbe et al., 2021) for grade-school math word problems, HumanEval (Chen et al., 2021) for Python function synthesis, and RULER (Hsieh et al., 2024) for long-context synthetic retrieval.

3.2 General Benchmark

Table 2 summarizes the main downstream results for the Transformer baseline, YOCO (Dense), and YOCO (CLSA). Overall, CLSA preserves the broad capability profile of the dense models while improving performance on several tasks that require selective evidence aggregation. In particular, YOCO (CLSA) obtains the best scores on ARC-Challenge, GSM8K, and DROP, and matches the best HumanEval result. On BBH, MMLU, HellaSwag, and WinoGrande, its performance remains close to the dense baselines, indicating that sparsifying the global attention path does not introduce a systematic degradation of general reasoning or knowledge. More broadly, hybrid architectures may also yield quality gains by combining complementary modeling capabilities and by leveraging multiple positional-encoding configurations.

3.3 Long Context

Table 3 reports our main long-context comparison on RULER at 16K and 32K tokens. At 16K, the Transformer baseline achieves the best overall average, while CLSA remains competitive with dense attention and preserves near-perfect performance on the single-needle retrieval tasks. At 32K, where long-range interference becomes more pronounced, CLSA achieves the best average score among all models. The improvement is mainly driven by stronger robustness on the more difficult multi-needle settings, especially MK1 and MK2, while maintaining comparable performance on the single-needle tasks. Results at longer contexts of 64K and 128K on the single-needle and multi-key retrieval tasks are provided in Appendix F.

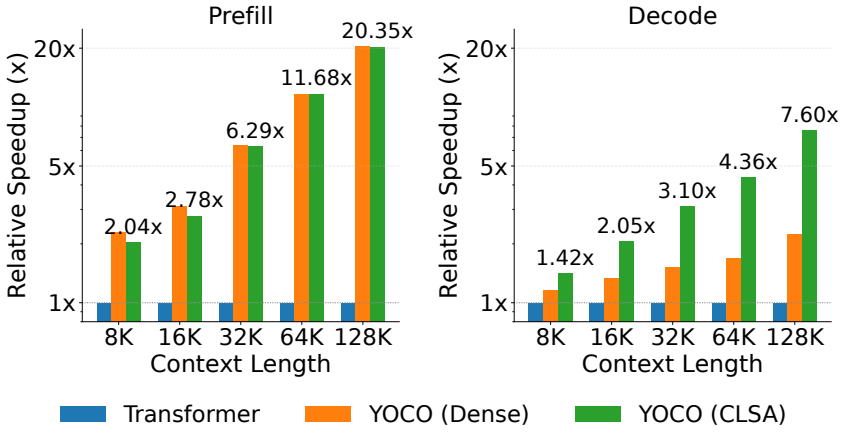


Figure 3: Inference throughput relative to the Transformer for prefill and decode across different context lengths. Both YOCO variants substantially accelerate prefill, while CLSA provides the largest decoding gains and widens its advantage as the context grows.

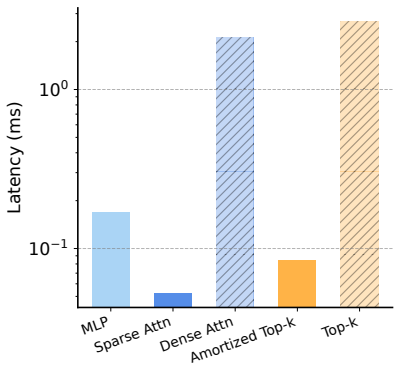


Figure 4: 128K latency analysis for different components. After amortizing routing, the amortized top-k becomes efficient. Without amortization, the unamortized top-k stage can be comparable to or even larger than dense attention.

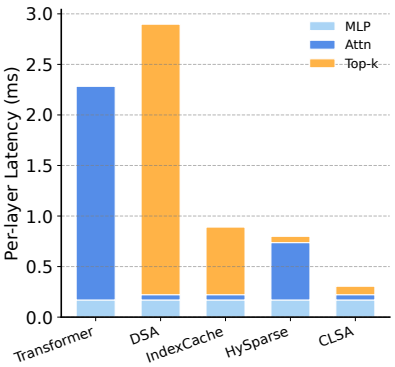


Figure 5: Per-layer latency comparison across representative sparse attention methods and dense baselines at 128K context. CLSA achieves the lowest latency by amortizing routing across cross-decoder layers.

As shown in Figure 2, beyond these synthetic retrieval benchmarks, we also measure language modeling quality on long validation slices from Books, ArXiv, and StarCoder at context lengths from 8K to 32K tokens. YOCO (Dense) and YOCO (CLSA) exhibit nearly overlapping cross-entropy loss across domains and lengths, indicating that CLSA is effectively lossless for long-context modeling. As context grows from 8K to 32K, CLSA follows the same loss trend as dense attention across all three domains. This indicates that CLSA preserves the same context-scaling behavior.

3.4 Inference Efficiency

We integrate CLSA into vLLM (Kwon et al., 2023) and measure end-to-end serving performance on NVIDIA B200 GPUs.

Figure 3 compares prefill and decode throughput; raw measurements are in Appendix E. Dense YOCO is the matched control with the same backbone and KV sharing. At 128K, it provides 2.23× Transformer decoding throughput, and CLSA adds a further 3.41× gain, reaching 7.6× relative to the Transformer. End-to-end throughput improves by 1.78× over Dense YOCO and 17.1× over the Transformer. Prefill throughput remains nearly unchanged from Dense YOCO; the prefill and KV-cache complexity advantages are inherited from YOCO.

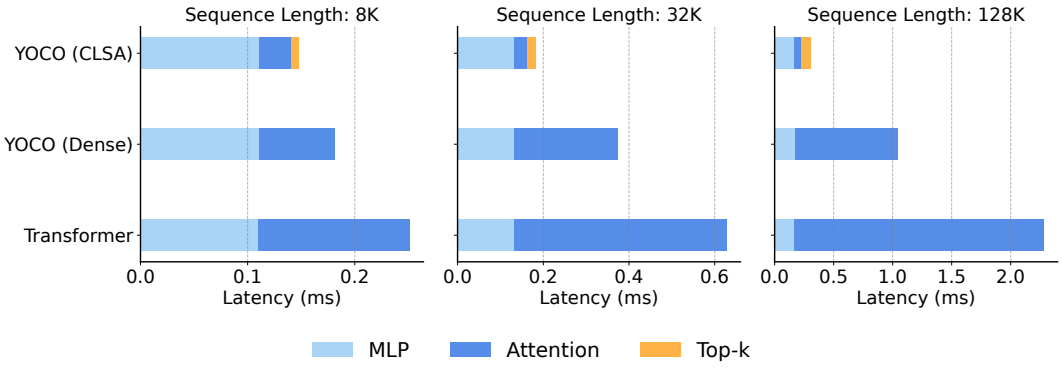


Figure 6: Per-layer latency breakdown at 8K, 32K, and 128K context. For YOCO (Dense), the attention cost is averaged over SWA and dense attention layers. For YOCO (CLSA), the attention cost is averaged over SWA and CLSA layers, and the top-k cost is amortized across cross-decoder layers. At 128K context, the amortized top-k stage takes about 0.08 ms per layer.

Table 4: Attention coverage and cross-entropy loss under sparse selection across selected-token budgets. Larger budgets recover more dense attention mass. Importantly, sparse selection introduces negligible cross-entropy loss degradation. 2048 selected tokens provide a favorable trade-off across domains.

	Attn. Coverage (% ↑)				**Cross-Entropy Loss (↓)**					
Domain	**512**	**1024**	**2048**	**4096**	**512**	**1024**	**2048**	**4096**	**Dense**	Δ_{2048}
StarCoder	62.34	76.28	84.12	90.73	0.5789	0.5717	0.5699	0.5703	0.5703	−0.0004
Books	51.37	65.11	76.29	84.10	1.7720	1.7560	1.7500	1.7480	1.7446	+0.0054
ArXiv	55.77	67.73	80.67	89.55	1.1070	1.0900	1.0844	1.0837	1.0818	+0.0026

Figure 4 shows that a standalone top-k pass at 128K can take time comparable to dense attention. Sharing this routing cost across cross-decoder layers makes sparse decoding efficient in CLSA.

Figure 5 compares per-layer decode latency with DSA (Liu et al., 2025), IndexCache (Bai et al., 2026), and HySparse (Gao et al., 2026). DSA computes token-level top-k independently at every layer and is slower than the dense Transformer at 128K because routing dominates its cost. IndexCache reuses routing across four layers, while HySparse mixes block-sparse layers with a fraction of dense layers. Both reduce this overhead but still incur substantial attention or routing costs. CLSA achieves the lowest latency by combining lightweight sparse attention with routing shared across all cross-decoder layers. Appendix G reports latency projections for sharing every 2, 4, or 8 layers and cross-layer recall results.

Figure 6 decomposes latency across context lengths, with values in Appendix D. For Dense YOCO, the attention term averages SWA in the 16 self-decoder layers and dense attention in the 16 cross-decoder layers. For YOCO (CLSA), it averages the same SWA layers and 16 CLSA layers. Routing runs once on the shared KV cache, and all 16 cross-decoder layers reuse the selected indices. To compare models consistently, all components are normalized by the full 32-layer depth. The plotted top-k cost is therefore the one-off routing cost divided by 32, although sharing occurs across 16 layers. Dense attention increasingly dominates Transformer latency as context grows, while CLSA keeps attention costs low.

3.5 Attention Sparsity Analysis

Table 4 summarizes attention behavior on books, ArXiv, and StarCoder data. As we increase the number of activated tokens, sparse attention recovers a growing fraction of the dense attention mass. At a 1:16 activation ratio, which corresponds to 2048 selected tokens, the routed set already captures roughly 80% of the dense attention score, showing that a small active subset can approximate most of the global allocation.

Second, matching full dense attention is not the same as matching language-modeling quality. Across Books, ArXiv, and StarCoder, sparse attention reaches dense-level quality, with loss differences around 0.006 or smaller. On StarCoder, the sparse curve even slightly improves over dense YOCO. This shows that selected-token sparse attention can preserve dense quality without recovering 100% of the attention mass.

This analysis also clarifies the relationship among block sparse attention, token sparse attention, and shared routing. Block sparse attention gains efficiency from structured memory access, but it relies on a block-level inductive bias that is not well aligned with long-context retrieval. Nearby tokens may serve very different semantic roles and exhibit very different attention patterns, so a block can contain both irrelevant tokens and a few crucial ones. This makes it difficult for block sparsity to reach the same long-context quality as fine-grained selection. Token sparse attention avoids this bias by estimating saliency at token granularity, but recomputing token-level top-k independently in every layer is expensive. CLSA keeps the token-level routing index and shares it across layers. The validity of this sharing comes from the empirical similarity of cross-layer attention scores induced by the data distribution and shared KV memory, rather than from an imposed block structure, which helps preserve quality while amortizing routing cost.

4 Related Work

4.1 Sparse Attention

Training-aware methods such as NSA (Yuan et al., 2025) and MoBA (Lu et al., 2025), together with post-training and training-free approaches (Gao et al., 2025; Sun et al., 2025; Gao et al., 2024; Tang et al., 2024; Liu et al., 2025), reduce attention cost by selecting relevant tokens or blocks. Their practical gains depend on both selection quality and routing overhead: coarse selection can lose useful context, while fine-grained token routing can be expensive.

Cross-layer reuse reduces repeated selection by exploiting similar attention patterns across layers. Training-free methods propagate token importance across layers (Deshmukh et al., 2025; Hao et al., 2025; Yang et al., 2024a; Entezari Zarch et al., 2025), while HySparse (Gao et al., 2026) and IndexCache (Bai et al., 2026) learn to reuse salient tokens through training or adaptation. We build on this direction by sharing one routing index across the full YOCO cross-decoder, where all layers read the same KV cache.

4.2 Hybrid Architecture

Hybrid architectures combine softmax attention with efficient sequence operators such as Mamba (Gu & Dao, 2023), RetNet (Sun et al., 2023), and Gated DeltaNet (Yang et al., 2024b), using layer schedules that balance efficiency and model quality (Lieber et al., 2024; Li et al., 2025; Yang et al., 2025a; Team et al., 2025b). Cross-layer KV sharing offers a complementary way to reduce memory cost. YOCO (Sun et al., 2024) lets cross-decoder layers reuse a shared cache, and related designs appear in CLA (Brandon et al., 2024), Gemma 3n (Team et al., 2025a), and Phi-4-mini-Flash (Ren et al., 2025). Our method retains YOCO's prefill and KV-cache advantages and extends sharing to routing, reducing sparse decoding cost.

5 Conclusion

We presented cross-layer sparse attention for KV-sharing architectures that extends sharing from memory to routing. By reusing one shared routing index across cross-decoder layers, CLSA preserves the fine-grained selectivity of token sparse attention while amortizing the practical cost of top-k routing. This yields an architecture that jointly improves the three major inference bottlenecks in long-context LLMs, namely pre-filling, KV-cache storage, and decoding. Empirically, CLSA remains nearly lossless across short-context and long-context evaluations while delivering substantial end-to-end acceleration, including up to $7.6\times$ decode speedup and $17.1\times$ overall throughput improvement at 128K context. We hope this result provides a more complete architectural direction for long-context LLMs that better reconciles model quality and inference efficiency.

AI USE STATEMENT

We used generative AI tools only to edit manuscript text, organize tables, and assist with LaTeX formatting. We reviewed all AI-assisted changes and take responsibility for the final content of this work.

ETHICS STATEMENT

This work studies efficient inference for large language models. Lower inference costs can make long-context models more accessible, but can also make harmful uses easier to scale. The proposed attention mechanism does not by itself address risks such as biased or misleading outputs, privacy leakage, or misuse. Deployments should assess these risks for the intended application and respect the terms governing the underlying models and data.

REPRODUCIBILITY STATEMENT

The main text describes the shared routing mechanism, the distillation objective, and the two-stage sparse adaptation procedure. Appendices B and C report the training hyperparameters and model configurations. Appendices D and E provide the latency breakdown and raw throughput measurements used in the efficiency figures, including the hardware and normalization conventions.

REFERENCES

Yushi Bai, Qian Dong, Ting Jiang, Xin Lv, Zhengxiao Du, Aohan Zeng, Jie Tang, and Juanzi Li. Indexcache: Accelerating sparse attention via cross-layer index reuse. *arXiv preprint arXiv:2603.12201*, 2026.

William Brandon, Mayank Mishra, Aniruddha Nrusimha, Rameswar Panda, and Jonathan Ragan-Kelley. Reducing transformer key-value cache size with cross-layer attention. *Advances in Neural Information Processing Systems*, 37:86927–86957, 2024.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Dhruv Deshmukh, Saurabh Goyal, Nipun Kwatra, and Ramachandran Ramjee. Kascade: A practical sparse attention method for long-context llm inference. *arXiv preprint arXiv:2512.16391*, 2025.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2368–2378, 2019.

Hossein Entezari Zarch, Lei Gao, Chaoyi Jiang, and Murali Annavaram. Delta: Dynamic layer-aware token attention for efficient long-context reasoning. *arXiv preprint arXiv:2510.09883*, 2025.

Yizhao Gao, Zhichen Zeng, Dayou Du, Shijie Cao, Peiyuan Zhou, Jiaxing Qi, Junjie Lai, Hayden Kwok-Hay So, Ting Cao, Fan Yang, et al. Seerattention: Learning intrinsic sparse attention in your llms. *arXiv preprint arXiv:2410.13276*, 2024.

Yizhao Gao, Shuming Guo, Shijie Cao, Yuqing Xia, Yu Cheng, Lei Wang, Lingxiao Ma, Yutao Sun, Tianzhu Ye, Li Dong, Hayden Kwok-Hay So, Yu Hua, Ting Cao, Fan Yang, and Mao Yang. Seerattention-r: Sparse attention adaptation for long reasoning, 2025. URL https://arxiv.org/abs/2506.08889.

Yizhao Gao, Jianyu Wei, Qihao Zhang, Yu Cheng, Shimao Chen, Zhengju Tang, Zihan Jiang, Yifan Song, Hailin Zhang, Liang Zhao, et al. Hysparse: A hybrid sparse attention architecture with oracle token selection and kv cache sharing. *arXiv preprint arXiv:2602.03560*, 2026.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Jitai Hao, Yihang Zhu, Tianle Wang, Jia Yu, Xiaowen Xin, Bo Zheng, Zhong Ren, and Sheng Guo. Omnikv: Dynamic context selection for efficient long-context llms. In *The Thirteenth International Conference on Learning Representations*, 2025.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

Hector Levesque, Ernest Davis, and Leora Morgenstern. The winograd schema challenge. In *Proceedings of KR*, 2012. URL https://dl.acm.org/doi/10.5555/3031843.3031909.

Aonian Li, Bangwei Gong, Bo Yang, Boji Shan, Chang Liu, Cheng Zhu, Chunhao Zhang, Congchao Guo, Da Chen, Dong Li, et al. Minimax-01: Scaling foundation models with lightning attention. *arXiv preprint arXiv:2501.08313*, 2025.

Opher Lieber, Barak Lenz, Hofit Bata, Gal Cohen, Jhonathan Osin, Itay Dalmedigos, Erez Safahi, Shaked Meirom, Yonatan Belinkov, Shai Shalev-Shwartz, Omri Abend, Raz Alon, Tomer Asida, Amir Bergman, Roman Glozman, Michael Gokhman, Avashalom Manevich, Nir Ratner, Noam Rozen, Erez Shwartz, Mor Zusman, and Yoav Shoham. Jamba: A hybrid Transformer-Mamba language model. *CoRR*, abs/2403.19887, 2024.

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, et al. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.

Enzhe Lu, Zhejun Jiang, Jingyuan Liu, Yulun Du, Tao Jiang, Chao Hong, Shaowei Liu, Weiran He, Enming Yuan, Yuzhi Wang, Zhiqi Huang, Huan Yuan, Suting Xu, Xinran Xu, Guokun Lai, Yanru Chen, Huabin Zheng, Junjie Yan, Jianlin Su, Yuxin Wu, Neo Y. Zhang, Zhilin Yang, Xinyu Zhou, Mingxing Zhang, and Jiezhong Qiu. Moba: Mixture of block attention for long-context llms, 2025. URL https://arxiv.org/abs/2502.13189.

Liliang Ren, Congcong Chen, Haoran Xu, Young Jin Kim, Adam Atkinson, Zheng Zhan, Jiankai Sun, Baolin Peng, Liyuan Liu, Shuohang Wang, et al. Decoder-hybrid-decoder architecture for efficient reasoning with long generation. *arXiv preprint arXiv:2507.06607*, 2025.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models. *arXiv preprint arXiv:2307.08621*, 2023.

Yutao Sun, Li Dong, Yi Zhu, Shaohan Huang, Wenhui Wang, Shuming Ma, Quanlu Zhang, Jianyong Wang, and Furu Wei. You only cache once: Decoder-decoder architectures for language models, 2024. URL https://arxiv.org/abs/2405.05254.

Yutao Sun, Tianzhu Ye, Li Dong, Yuqing Xia, Jian Chen, Yizhao Gao, Shijie Cao, Jianyong Wang, and Furu Wei. Rectified sparse attention. *arXiv preprint arXiv:2506.04108*, 2025.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed Chi, Denny Zhou, et al. Challenging big-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 13003–13051, 2023.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context llm inference, 2024. URL https://arxiv.org/abs/2406.10774.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, Gaël Liu, Francesco Visin, Kathleen Kenealy, Lucas Beyer, Xiaohai Zhai, Anton Tsitsulin, Robert Busa-Fekete, Alex Feng, Noveen Sachdeva, Benjamin Coleman, Yi Gao, Basil Mustafa, Iain Barr, Emilio Parisotto, David Tian, Matan Eyal, Colin Cherry, Jan-Thorsten Peter, Danila Sinopalnikov, Surya Bhupatiraju, Rishabh Agarwal, Mehran Kazemi, Dan Malkin, Ravin Kumar, David Vilar, Idan Brusilovsky, Jiaming Luo, Andreas Steiner, Abe Friesen, Abhanshu Sharma, Abheesht Sharma, Adi Mayrav Gilady, Adrian Goedeckemeyer, Alaa Saade, Alex Feng, Alexander Kolesnikov, Alexei Bendebury, Alvin Abdagic, Amit Vadi, András György, André Susano Pinto, Anil Das, Ankur Bapna, Antoine Miech, Antoine Yang, Antonia Paterson, Ashish Shenoy, Ayan Chakrabarti, Bilal Piot, Bo Wu, Bobak Shahriari, Bryce Petrini, Charlie Chen, Charline Le Lan, Christopher A. Choquette-Choo, CJ Carey, Cormac Brick, Daniel Deutsch, Danielle Eisenbud, Dee Cattle, Derek Cheng, Dimitris Paparas, Divyashree Shivakumar Sreepathihalli, Doug Reid, Dustin Tran, Dustin Zelle, Eric Noland, Erwin Huizenga, Eugene Kharitonov, Frederick Liu, Gagik Amirkhanyan, Glenn Cameron, Hadi Hashemi, Hanna Klimczak-Plucińska, Harman Singh, Harsh Mehta, Harshal Tushar Lehri, Hussein Hazimeh, Ian Ballantyne, Idan Szpektor, Ivan Nardini, Jean Pouget-Abadie, Jetha Chan, Joe Stanton, John Wieting, Jonathan Lai, Jordi Orbay, Joseph Fernandez, Josh Newlan, Ju yeong Ji, Jyotinder Singh, Kat Black, Kathy Yu, Kevin Hui, Kiran Vodrahalli, Klaus Greff, Linhai Qiu, Marcella Valentine, Marina Coelho, Marvin Ritter, Matt Hoffman, Matthew Watson, Mayank Chaturvedi, Michael Moynihan, Min Ma, Nabila Babar, Natasha Noy, Nathan Byrd, Nick Roy, Nikola Momchev, Nilay Chauhan, Noveen Sachdeva, Oskar Bunyan, Pankil Botarda, Paul Caron, Paul Kishan Rubenstein, Phil Culliton, Philipp Schmid, Pier Giuseppe Sessa, Pingmei Xu, Piotr Stanczyk, Pouya Tafti, Rakesh Shivanna, Renjie Wu, Renke Pan, Reza Rokni, Rob Willoughby, Rohith Vallu, Ryan Mullins, Sammy Jerome, Sara Smoot, Sertan Girgin, Shariq Iqbal, Shashir Reddy, Shruti Sheth, Siim Põder, Sijal Bhatnagar, Sindhu Raghuram Panyam, Sivan Eiger, Susan Zhang, Tianqi Liu, Trevor Yacovone, Tyler Liechty, Uday Kalra, Utku Evci, Vedant Misra, Vincent Roseberry, Vlad Feinberg, Vlad Kolesnikov, Woohyun Han, Woosuk Kwon, Xi Chen, Yinlam Chow, Yuvein Zhu, Zichuan Wei, Zoltan Egyed, Victor Cotruta, Minh Giang, Phoebe Kirk, Anand Rao, Kat Black, Nabila Babar, Jessica Lo, Erica Moreira, Luiz Gustavo Martins, Omar Sanseviero, Lucas Gonzalez, Zach Gleicher, Tris Warkentin, Vahab Mirrokni, Evan Senter, Eli Collins, Joelle Barral, Zoubin Ghahramani, Raia Hadsell, Yossi Matias, D. Sculley, Slav Petrov, Noah Fiedel, Noam Shazeer, Oriol Vinyals, Jeff Dean, Demis Hassabis, Koray Kavukcuoglu, Clement Farabet, Elena Buchatskaya, Jean-Baptiste Alayrac, Rohan Anil, Dmitry, Lepikhin, Sebastian Borgeaud, Olivier Bachem, Armand Joulin, Alek Andreev, Cassidy Hardin, Robert Dadashi, and Léonard Hussenot. Gemma 3 technical report, 2025a. URL https://arxiv.org/abs/2503.19786.

Kimi Team, Yu Zhang, Zongyu Lin, Xingcheng Yao, Jiaxi Hu, Fanqing Meng, Chengyin Liu, Xin Men, Songlin Yang, Zhiyuan Li, et al. Kimi linear: An expressive, efficient attention architecture. *arXiv preprint arXiv:2510.26692*, 2025b.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

Bowen Yang, Bharat Venkitesh, Dwarak Talupuru, Hangyu Lin, David Cairuz, Phil Blunsom, and Acyr Locatelli. Rope to nope and back again: A new hybrid attention strategy. *arXiv preprint arXiv:2501.18795*, 2025b.

Lijie Yang, Zhihao Zhang, Zhuofu Chen, Zikun Li, and Zhihao Jia. Tidaldecode: Fast and accurate llm decoding with position persistent sparse attention. *arXiv preprint arXiv:2410.05076*, 2024a.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule. *arXiv preprint arXiv:2412.06464*, 2024b.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Y. X. Wei, Lean Wang, Zhiping Xiao, Yuqing Wang, Chong Ruan, Ming Zhang, Wenfeng Liang, and Wangding Zeng. Native sparse attention: Hardware-aligned and natively trainable sparse attention, 2025. URL https://arxiv.org/abs/2502.11089.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of ACL*, 2019. URL https://aclanthology.org/P19-1472/.

A Dense-Stage Training Curves

Figure 7 shows the dense-stage training curves on representative benchmarks. YOCO remains competitive with the Transformer throughout training. Since CLSA is coupled with the YOCO backbone, these curves also serve as a sanity check that YOCO provides a strong dense attention starting point, including on retrieval-style tasks such as DROP. This makes it possible to start from a good dense model and obtain our final model through a near-lossless sparse-attention adaptation, rather than relying on a substantially different pretraining recipe.

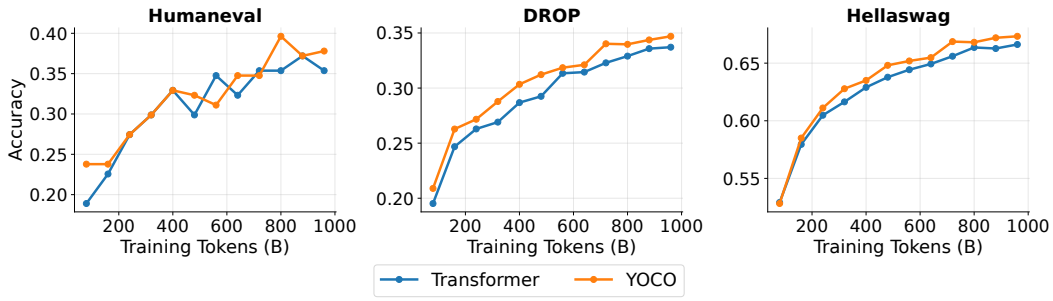


Figure 7: Dense-stage training curves on HumanEval, DROP, and HellaSwag as a function of training tokens. YOCO remains competitive with the Transformer throughout training, supporting its use as a stable dense backbone before sparse adaptation.

B Training Hyper-parameters

This appendix provides the exact optimization settings used for dense pretraining and sparse adaptation. Dense pretraining follows a two-stage schedule: stage 1 trains with a peak learning rate of 3×10^{-4} using 2000 warmup iterations on 8K contexts, and stage 2 increases the context cap to 32,768 while switching to a fixed learning rate of 3×10^{-5} for the remaining 10,000 updates. Sparse adaptation then reuses the same training recipe on 32,768-token sequences: stage 1 keeps the batch size and peak learning rate from dense stage 1, and uses the warmup iterations listed below; stage 2 continues for another 2,500 updates at the smaller fixed learning rate. For completeness, we also include the shared optimizer settings (Adam betas, epsilon, and weight decay), which are applied across all stages unless explicitly overridden.

Table 5: Stage-wise training hyperparameters for dense pretraining and sparse adaptation.

	Dense Stage 1	**Dense Stage 2**	**Sparse Stage 1**	**Sparse Stage 2**
Learning rate	3×10^{-4}	3×10^{-5}	3×10^{-4}	3×10^{-5}
Minimum LR	3×10^{-5}	3×10^{-5}	3×10^{-4}	3×10^{-5}
Max sequence length	8192	32768	32768	32768
Warmup iterations	2000	0	500	0
Training steps	125000	10000	2500	2500

Table 6: Shared optimization settings across all training stages.

Hyper-parameter	**Value**
Batch size	8M
Adam β	(0.9, 0.95)
Adam ε	10^{-8}
Weight decay	0.1

C Model Configuration

Tables 7 and 8 list the architectural hyperparameters for all evaluated variants. Across models we keep the same overall width and depth, and (for the YOCO-based models) split the 32 layers into 16 self-decoder layers and 16 cross-decoder layers. The key practical difference lies in how positional information is handled: the Transformer applies RoPE throughout, while YOCO (CLSA) uses an RNoPE setting that restricts RoPE to the sliding-window self-decoder and removes positional encoding from the global cross-decoder attention path. We additionally enable QK normalization and use GQA in all models, as summarized in the table.

Table 7: Shared architectural hyperparameters across all evaluated models.

Hyperparameter	**Value**
Hidden size	2560
FFN width	7680
Layers	32
Heads	20
KV heads	4
Head dimension	128
QK Norm	enabled
Weight tying	disabled

Table 8: Key architectural differences among the Transformer baseline, YOCO (Dense), and YOCO (CLSA), including positional encoding and attention type.

	Transformer	**YOCO (Dense)**	**YOCO (CLSA)**
Positional encoding	RoPE	RNoPE	RNoPE
RoPE base	5×10^5	1×10^4	1×10^4
Attention type	GQA	GQA	GQA+CLSA

D Experimental Details of Latency Breakdown

This section lists the per-layer latency values used in Figure 6. The table reports the plotted values in milliseconds after the script's scaling and per-layer averaging. For YOCO (Dense), the reported attention term averages SWA in the 16 self-decoder layers and dense full attention in the 16 cross-decoder layers. For YOCO (CLSA), it averages the same 16 SWA layers and 16 CLSA layers. The top-k term is the amortized per-layer routing cost, obtained by dividing the one-off routing latency by the full 32-layer model depth to match the normalization used in the plot. The routing decision itself is shared by the 16 cross-decoder layers, and the unamortized one-off 128K top-k cost is shown separately in Figure 4.

E Experimental Details of Inference Throughput

This section lists the raw throughput measurements used to produce the plotted panels of Figure 3, and also reports overall end-to-end generation throughput for the same setup. In the main paper we plot throughput ratios relative to the Transformer; the absolute tokens/s values reported here correspond to the same measurements for prefill, decode, and overall end-to-end generation (all evaluated on NVIDIA B200 GPUs).

Table 9: Per-layer latency breakdown (ms) used in Figure 6. For the YOCO variants, attention terms are averaged across layer types, and the top-k column reports the amortized per-layer routing cost for YOCO (CLSA).

Context	Model	MLP	Dense Attn.	Sparse Attn.	Top-k	Total
8K	Transformer	0.11	0.14	–	–	0.25
8K	YOCO (Dense)	0.11	0.07	–	–	0.18
8K	YOCO (CLSA)	0.11	–	0.03	0.01	0.15
32K	Transformer	0.13	0.50	–	–	0.63
32K	YOCO (Dense)	0.13	0.24	–	–	0.37
32K	YOCO (CLSA)	0.13	–	0.03	0.02	0.18
128K	Transformer	0.17	2.11	–	–	2.28
128K	YOCO (Dense)	0.17	0.87	–	–	1.04
128K	YOCO (CLSA)	0.17	–	0.05	0.08	0.31

Table 10: Raw prefill throughput (tokens/s) used in the left panel of Figure 3.

Context Length	Transformer	YOCO (Dense)	YOCO (CLSA)
8K	4721.71	10884.35	9623.48
16K	4160.92	13033.56	11572.40
32K	2889.39	18450.18	18163.24
64K	1741.56	20343.94	20349.11
128K	1019.06	20864.85	20741.51

Table 11: Raw decode throughput (tokens/s) used in the right panel of Figure 3.

Context Length	Transformer	YOCO (Dense)	YOCO (CLSA)
8K	4762.79	5516.50	6742.39
16K	3091.32	4147.85	6350.39
32K	1761.72	2677.12	5461.33
64K	948.15	1600.00	4137.37
128K	431.16	960.94	3276.80

Table 12: Raw overall throughput (tokens/s) measured under the same setup as Figure 3.

Context Length	Transformer	YOCO (Dense)	YOCO (CLSA)
8K	2989.78	4311.58	4920.12
16K	1449.91	2904.96	3715.19
32K	570.27	1832.66	2805.48
64K	191.36	1042.90	1735.59
128K	62.53	599.05	1068.06

F RETRIEVAL AT LONGER CONTEXT LENGTHS

Table 13 extends the 16K and 32K comparison in the main text to 64K and 128K contexts. We report the three single-needle and three multi-key retrieval tasks from RULER; the average is computed over these six tasks. YOCO (CLSA) achieves average scores of 75.3 at 64K and 73.0 at 128K, compared with 76.0 and 72.0 for YOCO (Dense). Its single-needle scores remain strong at both lengths, while Multi-key-3 scores are lower than those of the dense YOCO model.

Table 13: RULER retrieval results at 64K and 128K context lengths. Avg is the mean over the six reported tasks.

Context	Model	Single-1	Single-2	Single-3	Multi-key-1	Multi-key-2	Multi-key-3	Avg
64K	Transformer	100	100	80	52	0	0	55.3
	YOCO (Dense)	100	84	78	64	88	42	**76.0**
	YOCO (CLSA)	100	92	92	70	88	10	75.3
128K	Transformer	100	38	2	34	0	0	29.0
	YOCO (Dense)	100	98	76	70	62	26	72.0
	YOCO (CLSA)	100	98	98	72	66	4	**73.0**

G Routing Sharing Granularity and Recall

Latency projection. Sharing every g cross-decoder layers requires $16/g$ routing passes. Using the measured 128K routing cost of 2.56 ms per pass and fixed MLP and sparse-attention costs gives

$$T_{\text{total}}(g) = \left(0.17 + 0.05 + \frac{(16/g) \times 2.56}{32}\right) \text{ ms}. \tag{9}$$

Costs are normalized by the full 32-layer depth, as in Appendix D. Table 14 reports projections; we did not train or benchmark separate grouped-sharing models. The rounded components sum to 0.30 ms for CLSA, versus the measured total of 0.31 ms.

Table 14: Projected per-layer latency at 128K and ratios to CLSA. IndexCache-style and DSA-style denote sharing patterns on the YOCO backbone.

Sharing granularity	**Routing passes**	**Amortized top-k (ms)**	**Total (ms)**	**vs. CLSA**
All 16 cross-decoder layers (CLSA)	1	0.08	0.30	1.0×
Every 8 layers	2	0.16	0.38	1.3×
Every 4 layers (IndexCache-style)	4	0.32	0.54	1.8×
Every 2 layers	8	0.64	0.86	2.9×
Per layer (DSA-style)	16	1.28	1.50	5.0×

Table 15: Recall of each layer's top-128 tokens within a top-2048 candidate set built from the same layer or another layer.

Domain	**Same-layer recall (%)**	**Cross-layer recall (%)**
StarCoder	94.26	85.00
Books	90.30	84.43
ArXiv	89.54	82.90
Average	91.37	84.11