LASA: LEARNABLE ADAPTIVE SPARSE ATTENTION FOR EFFICIENT AND ACCURATE REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

In long chain-of-thought reasoning, every decoding step reads an ever-growing KV cache, and this memory traffic becomes the inference bottleneck. Sparse attention reduces this cost by accessing only a subset of the KV cache. Most existing methods focus on improving the quality of *ranking* KV tokens, while using a fixed token count (top-k) for *budgeting*. Some adaptive-budget methods instead vary token counts according to a fixed target for retained attention mass (top-p). Both rely on *a priori* targets shared across layers, KV heads, and queries, rather than learning from model output quality where more or less computation is needed. We introduce LASA (Learnable Adaptive Sparse Attention), which makes budgeting a learned and fine-grained decision. Given a block ranking distilled from dense attention, a $\sim$0.6M-parameter predictor sets a threshold for each layer, KV head, and decoding step, and uses simple elementwise comparisons to select blocks whose normalized ranker scores exceed it. The predictor is trained end-to-end through output-level distillation under a relaxed token budget, with both the backbone and ranker frozen. Our extensive evaluation shows that LASA substantially improves the accuracy–density Pareto frontier over existing sparse attention methods, retaining near-full-attention accuracy while under 1K average attended tokens. Our efficient kernels and vLLM integration translate these savings into up to $13\times$ attention-kernel and $4.1\times$ end-to-end decoding speedups over dense attention on H20 GPUs.

1 INTRODUCTION

Large reasoning models improve performance on complex tasks through test-time scaling (OpenAI, 2024; Guo et al., 2025; Yang et al., 2025a), often generating tens of thousands of tokens to solve difficult mathematical and coding problems. As the sequence grows, decoding attention comes to dominate inference cost: its share of per-step decoding latency rises from 36% at 8K context to 80% at 64K.[1] Sparse attention reduces this cost by reading only a subset of the KV cache. Selecting this subset involves two tasks, which we term ❶ *ranking*, ordering KV tokens by their importance to the current query, and ❷ *budgeting*, deciding how many tokens to access.

Existing methods have substantially improved ranking quality, progressing from training-free heuristics (Xiao et al., 2024; Zhang et al., 2023; Tang et al., 2024; Jiang et al., 2024) to training-based *attention map distillation* (Gao et al., 2024; 2025; DeepSeek-AI, 2025), which trains an auxiliary module to approximate full attention weights. However, budgeting typically relies on *a priori* targets shared across *attention rows*:[2] a fixed token count (top-k) (Tang et al., 2024; Gao et al., 2025; Yang et al., 2026; DeepSeek-AI, 2025) or a fixed target for retained attention mass (top-p) (Zhu et al., 2025; Lin et al., 2025; Zhu et al., 2026). A fixed token count ignores whether attention is concentrated on a few KV tokens or spread across many, while a fixed mass target overlooks how differently sparsifying each attention row affects the model's output (Figure 1). These policies apply the same preset target across attention rows, rather than learning each attention row's budget from model output quality, thus causing computation misallocation.

[1] Qwen3-8B at batch 8 on one H20 GPU.

[2] We use *attention row* to denote a budgeting unit at a given layer, KV head (since our design follows the common sparse attention practice of sharing one KV selection across query heads within each GQA group for efficient KV reuse), and decoding step.

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.

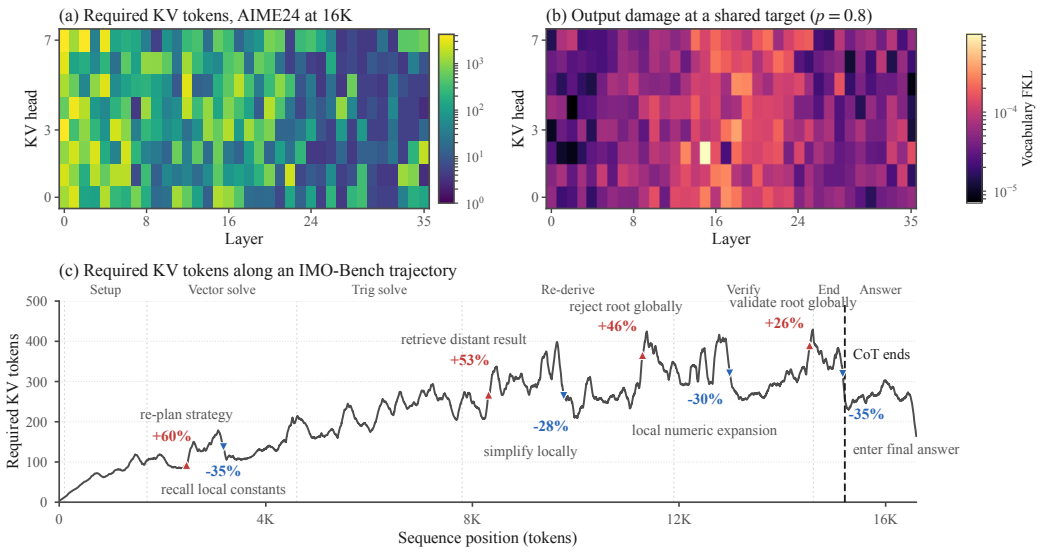

Figure 1: **Attention budget demand in reasoning** (Qwen3-4B). *(a)* KV tokens needed to reach 85% of the attention mass of each (layer, KV head) at one AIME24 decoding step. *(b)* forward KL divergence of the model's output distribution from dense when only one (layer, KV head) group is sparsified with top-p ($p = 0.8$), on an OpenThoughts3 trajectory. *(c)* KV tokens needed to reach 85% attention mass along an IMO-Bench (Luong et al., 2025) trajectory; Table 6 lists annotated events.

We therefore propose LASA, an adaptive sparse attention framework designed to accelerate long decoding while preserving accuracy by learning a fine-grained budgeting policy end-to-end from model output quality. We design LASA across module architecture, training strategy, and inference deployment: **(1) Fine-grained adaptive budgeting.** For each layer and KV head, LASA pairs a KV-block ranker (Gao et al., 2025) with a *threshold predictor*. The predictor maps the decoding query representation to a threshold for each attention row. Blocks whose normalized ranker scores exceed this threshold are selected through efficient elementwise comparisons. **(2) End-to-end training with hard selections.** With the backbone and ranker frozen, the predictor sparsifies attention and learns dynamic, fine-grained budget allocation from model output quality. The training is end-to-end under a relaxed budget constraint and uses hard block selections, matching the discrete decisions made at inference. A straight-through estimator (Bengio et al., 2013) and *Multiplicative Gating Softmax* (Section 3.3) propagate gradients through these hard selections and preserve a recovery signal for excluded blocks. **(3) High-performance decoding.** We develop a fused indexer and a dynamic sparse decoding attention kernel that efficiently computes attention over ragged, per-KV-head lists of selected KV blocks. We integrate both kernels into vLLM's paged KV cache (Kwon et al., 2023), remaining CUDA-graph friendly.

We evaluate LASA across four reasoning models and six math, code, and science benchmarks. LASA retains near-full-attention accuracy while averaging 610–1,040 attended tokens per attention row[3] at block size 64 and 230–650 at block size 16. Compared with existing sparse attention baselines, LASA improves the Pareto frontier and shows less generation-length inflation at low densities. At block size 64, the attention kernel delivers up to a 13× speedup over dense attention, while the indexer takes about 2–5% of dense decoding time. Integrated into vLLM, these kernels deliver up to a 4.1× end-to-end decoding speedup on H20 GPUs. The benefit also holds for complete reasoning workloads: LASA achieves 3.4× speedup while preserving accuracy on AIME25 workload.

2 Motivation

Sparse attention reduces KV cache accesses by selecting a subset $\mathcal{S}$ of N candidate tokens for each attention row. A ranker f uses the inference state $\mathcal{I}$ to score KV tokens and order their indices

[3]Within each benchmark run (one response per problem), we divide the total attended KV-token count by the total number of decoding attention rows. We report the mean across runs.

$j_1, \ldots, j_N$, while budgeting determines how many to access:

$$\underbrace{\mathbf{s} = f(\mathcal{I}), \quad s_{j_1} \geq \cdots \geq s_{j_N}}_{\text{ranking}} \qquad \underbrace{\mathcal{S} = \{j_1, \ldots, j_k\}}_{\text{budgeting: choose } k}. \tag{1}$$

Beyond ranking, effective sparse attention needs *judicious budgeting* across attention rows, allocating limited computation where it matters most for output quality. However, common budgeting policies can misallocate computation by applying the same preset target across attention rows.

Top-k assigns the same token count whether attention is concentrated on a few KV tokens or spread across many. Yet the count needed to cover 85% of an attention row's mass varies by three orders of magnitude across rows within a single decoding step (Figure 1*(a)*). Demand also changes throughout the reasoning trajectory (Figure 1*(c)*). It shrinks when the model uses local information (recalling constants, line-by-line simplification) and expands when it reaches back to distant context (retrieving an earlier result, re-planning); Table 6 details these events. A fixed k therefore risks either unnecessary computation or degraded output quality.

Top-p adapts token counts to attention concentration but uses an *a priori* and shared retained-mass target. For a row with dense attention output $\mathbf{o}$, retaining mass p and renormalizing gives a sparse output $\mathbf{o}_p$ satisfying

$$\mathbf{o}_p - \mathbf{o} \;=\; \frac{1-p}{p}\big(\mathbf{o} - \boldsymbol{\mu}\big), \qquad \|\mathbf{o}_p - \mathbf{o}\| \;\leq\; \frac{1-p}{p}\, D, \tag{2}$$

where $\boldsymbol{\mu}$ is the attention-weighted mean of the dropped values and D bounds the value spread. Increasing p tightens this local bound, but a shared mass target does not imply similar effects on the model's final output. Sparsifying one (layer, KV head) group at a time at $p = 0.8$, with all others dense, the resulting output-distribution shift varies by $136\times$ across groups (Figure 1*(b)*). These observations suggest that budgeting should vary across attention rows according to how sparsification affects model output quality.

3 LASA

LASA combines a ranker that estimates KV-block importance with a threshold predictor that adapts how many blocks each attention row retains (Figure 2). We train LASA in a two-stage manner, keeping the backbone frozen throughout. In Stage 1, we train the ranker by distilling dense attention maps (Section 3.1); in Stage 2, we freeze the ranker and train the predictor through output-level distillation under a relaxed budget constraint (Section 3.3). To translate the learned sparsity into faster decoding, we develop fused indexing and dynamic sparse decoding attention kernels and integrate them into vLLM (Section 3.4).

3.1 Block ranking

We adopt the SeerAttention-R attention gate (Gao et al., 2025) as LASA's ranker, keeping its architecture unchanged. It learns ranking by distilling dense attention maps, the same supervision used by DSA (DeepSeek-AI, 2025). The KV cache is partitioned into B-token blocks for hardware efficiency. At layer ℓ, KV head h, and decoding step t, the ranker scores block b as

$$\begin{aligned} \mathbf{q}^{\text{rank}}_{\ell,h,t} = \mathbf{q}_{\ell,h,t}\, \mathbf{W}^{q,\text{rank}}_{\ell,h}, \qquad \mathbf{k}^{\text{rank}}_{\ell,h,b} = \text{pool}_{j\in b}\left(\mathbf{k}_{\ell,h,j}\right) \mathbf{W}^{k,\text{rank}}_{\ell,h}, \\ s_b = \left\langle \text{RoPE}(\mathbf{q}^{\text{rank}}_{\ell,h,t}), \text{RoPE}(\mathbf{k}^{\text{rank}}_{\ell,h,b}) \right\rangle / \sqrt{d_r}, \end{aligned} \tag{3}$$

where $\mathbf{q}_{\ell,h,t}$ concatenates the pre-RoPE queries in GQA group h, $\mathbf{k}_{\ell,h,j}$ is the pre-RoPE key vector for token j, and pool concatenates the block's elementwise key max, min, and mean. Both projections have width d_r, with RoPE using query and block-start positions, respectively (Gao et al., 2025). In Stage 1, the backbone remains frozen and we train only the ranker as an auxiliary branch, minimizing KL divergence from the teacher's normalized block attention map to $\text{softmax}(s)$. Besides ranking, $\hat{p}_b = \text{softmax}(s)_b$ gives the *estimated attention mass* of block b.

3.2 Budgeting with a learned threshold

Given the block ranking, LASA needs to decide how many KV blocks to access. It predicts a threshold for each attention row and selects blocks whose normalized ranker scores exceed it.

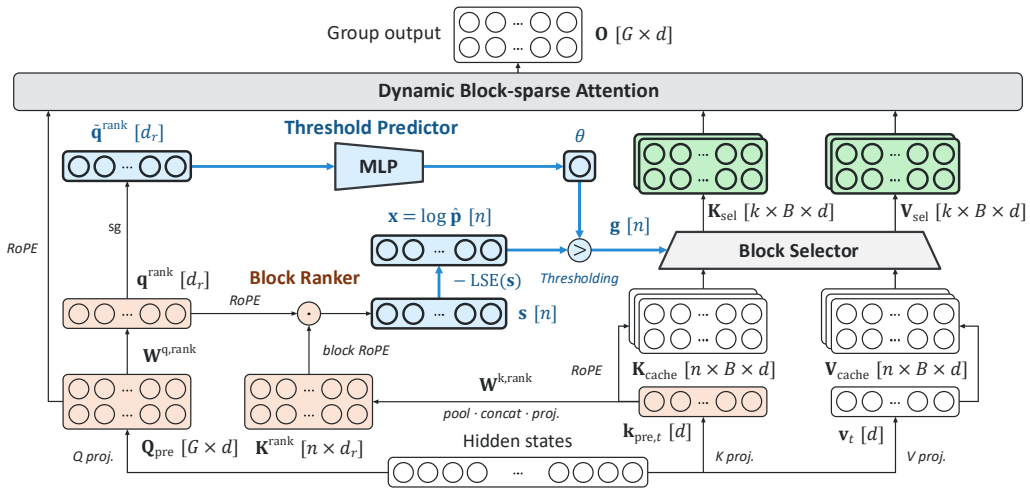

Figure 2: **Data flow of LASA for one layer and KV head at a decoding step.** The ranker scores KV blocks, and the layer–head-specific threshold predictor sets a cutoff from the ranker's query representation $\mathbf{q}^{\text{rank}}$. The resulting block selection is shared by all query heads in the GQA group.

Why a threshold. Adaptive budgeting can be achieved by predicting a token count k or a retained mass p for each attention row. However, even optimized GPU implementations of top-k and top-p incur overhead from reductions and synchronization (PyTorch Contributors, 2025; Lin et al., 2025). To reduce this selection overhead, LASA instead predicts a threshold and selects blocks through parallel elementwise comparisons.

What the threshold acts on. Stage 1 supervises scores through the shift-invariant softmax, leaving an arbitrary offset for each attention row that a raw-score threshold would have to account for. We remove this offset by subtracting a statistic of the score vector, the *anchor*. Using log-sum-exp (LSE) over causally valid blocks $\mathcal{C}_t$ gives log estimated mass:

$$x_b = s_b - \text{LSE}_{j \in \mathcal{C}_t}(s_j) = \log \hat{p}_b, \qquad \text{open block } b \iff x_b > \theta_{\ell,h,t}. \tag{4}$$

Other anchors also remove the offset, but mix a block's estimated mass with a statistic of the score vector, such as its mean, maximum, or length (Appendix B). Unless these statistics are also fed to the threshold predictor, it cannot compensate for them, and alternative anchors underperform LSE in our ablations (Table 1). The comparison between LSE-normalized scores and the predicted threshold has an intuitive benefit–cost interpretation. A block's benefit can be viewed as its estimated mass $\hat{p}_b$ weighted by the final output's sensitivity to sparsifying its row, while every block has the same access cost. The threshold in $\log \hat{p}_b > \theta_{\ell,h,t}$ thus incorporates both row sensitivity and access cost, and is learned in Stage 2 from the model's final output.

Threshold predictor. To capture both layer–head differences and query-dependent variation in budget demand (Section 2), we use a single-hidden-layer MLP for each (layer, KV head). Its input is the ranker's query representation $\mathbf{q}^{\text{rank}}_{\ell,h,t}$, a compact representation of the GQA queries learned for KV-block selection. With d_r=128 and hidden width d_f=16, the predictor is lightweight, containing $\sim$0.6M parameters in total. The threshold is computed as

$$\theta_{\ell,h,t} = b_{\ell,h} + \mathbf{w}_{\ell,h}^{\top} \text{SiLU}\big(\mathbf{W}^{q,\text{pred}}_{\ell,h} \tilde{\mathbf{q}}^{\text{rank}}_{\ell,h,t} + \mathbf{u}_{\ell,h}\big), \tag{5}$$

where $\tilde{\mathbf{q}}^{\text{rank}} = \text{sg}(\mathbf{q}^{\text{rank}})$, with sg denoting stop-gradient, $\mathbf{W}^{q,\text{pred}}_{\ell,h} \in \mathbb{R}^{d_f \times d_r}$ and bias $\mathbf{u}_{\ell,h}$ project to width $d_f \ll d_r$, and $\mathbf{w}_{\ell,h} \in \mathbb{R}^{d_f}$ maps to a scalar with bias $b_{\ell,h}$.

3.3 End-to-end learning of discrete sparse selection

Output-quality-supervised budgeting. In Stage 2, with the backbone and ranker frozen, we train the threshold predictor through output distillation under a relaxed KV token budget constraint (K:

soft cap on the mean selected block count):

$$\mathcal{L} = \underbrace{D_{\mathrm{KL}}\big(P_{\mathrm{dense}} \,\|\, P_{\mathrm{sparse}}\big)}_{\text{vocabulary distillation}} + \lambda_B \underbrace{\operatorname*{avg}_{\text{length bins}} \phi\Big([\bar{k} - K]_+ / K\Big)}_{\text{relaxed budget penalty}}. \tag{6}$$

The vocabulary-level KL aligns the sparse model's output distribution P_{sparse} with the dense model's original distribution P_{dense}, providing richer supervision than a single target token (Hinton et al., 2015).

Length-binned relaxed budget constraint. To encourage sparsity with a *predictable average budget* at inference, we penalize mean block usage above a soft cap K during training, while allowing budgets to adapt across attention rows. A straightforward approach is to apply the budget penalty to the mean selected block count across all attention rows, regardless of context length. However, queries with short contexts have fewer KV blocks available and tend to select fewer. Their lower usage can keep the overall mean below K even when queries with longer contexts select more than K blocks on average. This leaves long-context overspending unpenalized, risking budget overruns during longer generations. We therefore group attention rows by context length and apply the same soft cap K separately to the mean selected block count within each length bin. Here, ϕ is the Huber penalty (Huber, 1992), and $[u]_+ = \max(u, 0)$. In Figure 10, LASA's average allocation stays largely within the K target as context grows, while adapting across reasoning events.

Training with hard block selections. Stage 2 uses the same binary selections as deployment (Eq. (4)), avoiding a soft-to-hard training–inference gap. We differentiate through them with a straight-through estimator (STE) (Bengio et al., 2013):

$$\pi_b = \sigma(x_b - \theta_{\ell,h,t}), \qquad g_b = \mathbb{I}[\pi_b > \tfrac{1}{2}] + \pi_b - \mathrm{sg}(\pi_b), \tag{7}$$

where σ is the sigmoid activation. How the hard gates are incorporated into attention affects both the output-level learning signal and numerical stability. A natural approach is to add $\log g_b$ to the attention scores through the additive-mask interface of existing attention libraries (PyTorch Contributors, n.d.) (*Additive Log Masking*). However, backpropagation through this mask gives excluded blocks zero or undefined output-loss gradients (Appendix A.1). We therefore use *Multiplicative Gating Softmax*, multiplying the exponentiated attention scores by the gates and then normalizing:

$$\widetilde{P}_{t,j} = \frac{g_{b(j)} \exp(S_{t,j})}{Z_t}, \qquad Z_t = \sum_k g_{b(k)} \exp(S_{t,k}), \tag{8}$$

where $S_{t,j}$ is the raw attention score, $b(j)$ maps key j to its block, and $\widetilde{P}_{t,j}$ is the normalized sparse attention weight. This yields the same hard forward output while allowing excluded blocks to receive output-loss gradients, so the predictor can learn to readmit useful blocks (Appendix A.2). Because excluded scores do not contribute to Z_t, the gate-gradient factor $\exp(S_{t,j})/Z_t$ can become large for high-scoring excluded tokens. We therefore clip this factor only when computing gradients with respect to the gates (Appendix A.3). For efficient training, we extend FlashAttention-2 to support the forward and backward passes of Multiplicative Gating Softmax, preserving its IO-efficient tiled execution (Dao et al., 2022; Dao, 2024) (Appendix C).

3.4 LASA DECODING

We integrate LASA with vLLM's paged KV cache (Kwon et al., 2023). Alongside the KV cache, we maintain a compact indexer cache with one entry for each block representation $\mathbf{k}^{\mathrm{rank}}_{\ell,h,b}$, aligned with its corresponding B-token KV page. The two caches share page IDs, block tables, and allocation lifetimes.

Fused indexing. LASA-Index fuses $\mathbf{k}^{\mathrm{rank}}$ updates, ranking, threshold prediction, and selection in one kernel, reducing kernel launches and intermediate memory traffic. For each (request, KV head), it returns the selected KV-page IDs to the attention kernel. The execution schedule is chosen during CUDA graph capture based on the batch size and number of KV heads. With few requests, a 2-CTA Thread Block Cluster (introduced with SM90) partitions the work to increase parallelism. For larger batches, requests sharing a KV head are grouped to reuse projection weights, with one CTA per group to increase concurrency and reduce scheduling waves.

Dynamic sparse decoding. LASA-Decode is a dynamic sparse decoding kernel that extends the Dynamic Decode Attention kernel of `HPC-Ops` (Tencent Hunyuan AI Infra Team, 2026) to support LASA's dynamic, per-KV-head block selection. For each (request, KV head), it divides LASA-Index's selected-page list into fixed-size chunks and distributes them across persistent CTAs to balance work across rows with different selected counts. A lightweight GPU-side kernel rebuilds this chunk-to-CTA mapping from LASA-Index's counts at each layer, keeping the decode path CUDA-graph friendly.

4 EVALUATION

4.1 EXPERIMENT SETUP

Models, benchmarks, and protocol. We evaluate four GQA-based reasoning models, Qwen3-4B/8B/14B (Yang et al., 2025a) and DeepSeek-R1-Distill-Llama-8B (R1-Llama-8B) (Guo et al., 2025), on six reasoning benchmarks: AIME24 and AIME25 (Art of Problem Solving, n.d.), and MATH-500 (Lightman et al., 2024) for mathematics, GPQA-Diamond (Rein et al., 2024) for science, and LiveCodeBench (Jain et al., 2025) V5 and V6 for coding. All methods use each model's official inference configuration with a maximum output length of $32{,}768$ tokens, extended to $38{,}912$ on AIME24/25. We average pass@1 over 64 responses per problem on AIME24/25, 16 on GPQA-Diamond, and 8 elsewhere (Appendix E.1).

Density as the cost axis. Most prior work uses the reported top-k value as the cost axis, but this measure is undefined for adaptive-count methods and can omit KV accesses in dense layers. For each benchmark, we aggregate attended and causally available KV-token counts over all samples, decoding steps, layers, and KV heads within each run. We report KV density as the mean of the resulting run-level ratios (Appendix G). To account for the extra cost of longer generations, we also report mean output length at each operating point (Appendix E.2).

Baselines and operating points. We compare with two fixed-count methods, SeerAttention-R (Gao et al., 2025) and LessIsMore (Yang et al., 2026), and one adaptive-count method, Twilight (Lin et al., 2025). We retain the released dense layers: four in LessIsMore for its selection pipeline, and Twilight's first two since all-sparse accuracy collapses (Appendix E.1). Since their dense layers count toward density, we sweep lower budgets to cover low-density regime. SeerAttention-R is evaluated at token budgets of $\{1, 2, 4, 6, 8\}$K with 64-token blocks, and LessIsMore at $\{0.25, 0.5, 1, 2, 4, 6, 8\}$K. Twilight is evaluated at $p \in \{0.4, 0.7, 0.95, 0.99\}$ on the Qwen3 models and $p \in \{0.7, 0.95, 0.99\}$ on R1-Llama-8B. LASA has five budget variants, LASA-K15 through LASA-K127, trained with soft budgets of $K \in \{15, 31, 63, 95, 127\}$ 64-token blocks (960–8,128 tokens). Since Twilight prefilters 16-token pages before token-level top-p, we also train 16-token-block LASA at $K \in \{15, 31\}$ to match its granularity, using LASA-K15 on MATH-500 and GPQA-Diamond and LASA-K31 on AIME24/25 and LiveCodeBench.

Training. For each model, Stage 1 distills the ranker for 800 steps on 16K-token packed OpenThoughts3 sequences (Guha et al., 2026) with the backbone frozen; Stage 2 then freezes the ranker as well and trains only the threshold predictor for 400 steps, once per budget variant. Across the four models, the predictor contains 0.53–0.67M parameters, accounting for 0.005–0.015% of the dense model's parameter count. On $8\times$MI300X, Stage 1 takes 1.2–2.2 hours per model and each Stage-2 variant 1.4–2.7 hours. Other training details are provided in Appendix D.

4.2 EVALUATION ON REASONING TASKS

Figure 3 compares the accuracy–density frontiers across four models and six benchmarks.

LASA tracks full-attention accuracy with extremely low budgets. With 64-token blocks, LASA-K15 retains near-full-attention accuracy across all benchmarks while averaging 610–1,040 attended tokens per attention row (6.4–14.6% KV density). The 16-token-block LASA variants generally retain near-full-attention accuracy at lower budgets: LASA-K15 uses 230–270 tokens per attention row on MATH-500 and GPQA (3.5–4.4% density), while LASA-K31 uses 480–650 tokens on AIME and LiveCodeBench (4–6% density).

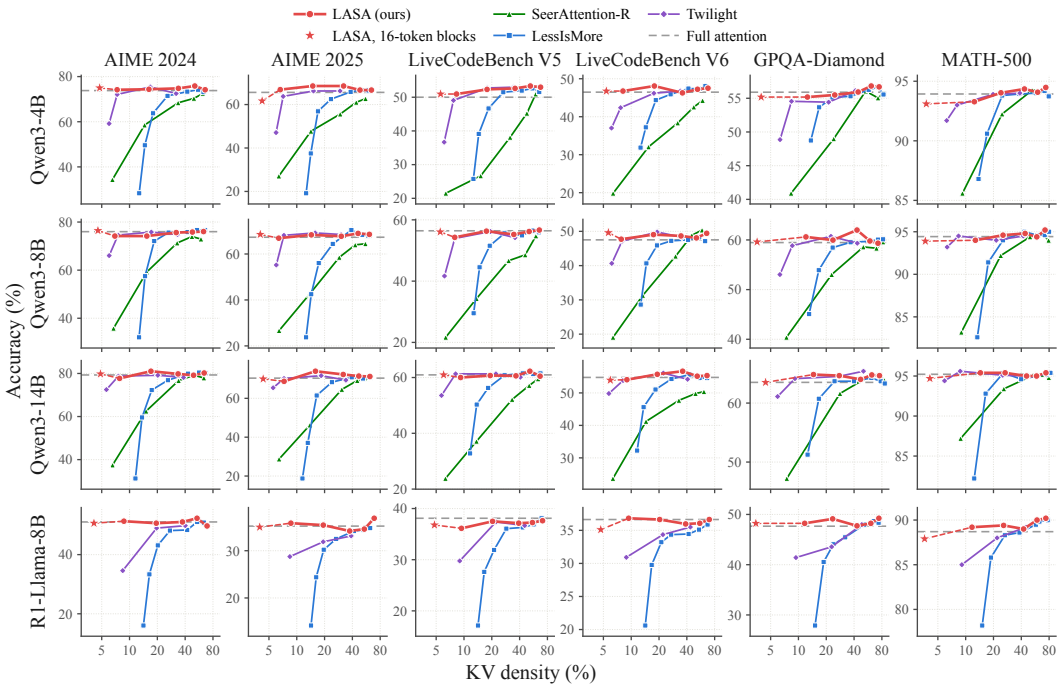

Figure 3: **Accuracy versus efficiency (KV density).** Pass@1 versus measured KV density (log scale) across four models and six benchmarks. Dashed lines denote full-attention accuracy, and stars ($\star$) mark 16-token-block LASA variants. LASA approaches full-attention accuracy at lower densities.

LASA also outperforms the sparse attention baselines at low densities. LessIsMore's accuracy collapses at low densities as its dense layers consume much of the budgets. Even its nominal 1K setting (excluding dense-layer KV accesses) trails LASA-K15, whose budget is typically below 1K, by up to 10.8 points on AIME25. SeerAttention-R uses a distilled ranker but falls behind LASA at middle and low densities due to fixed top-k budgeting. Twilight is more robust with 16-token-block scoring and adaptive token-level top-p pruning. It nevertheless trails 16-token-block LASA at low densities, suggesting that its shared, preset retained mass target misallocates computation.

At low density, sparse attention methods may require longer outputs, increasing the total decoding cost. While fixed-count baselines show pronounced length inflation, LASA keeps generation lengths close to full attention with both block sizes. Its 16-token-block variants also produce shorter outputs than Twilight at its sparsest settings (Appendix E.2).

4.3 Efficiency evaluation

To translate sparsity into faster decoding, we integrate LASA into vLLM using the kernels described in Section 3.4. We evaluate kernel-level and end-to-end decoding latency. The experiments use NVIDIA H20 GPUs (a decoding-oriented GPU), whose $\sim$4 TB/s HBM bandwidth makes dense decoding fast and sparse speedups harder-won. We sweep sequence lengths from 8K to 32K, batch sizes from 1 to 32, and all five LASA budget variants for Qwen3-8B with 64-token blocks. Measurement details and the 64K extension are provided in Appendix E.3.

Kernel-level latency. Our dense baseline, `HPC-Ops` (Tencent Hunyuan AI Infra Team, 2026), achieves 3.28 TB/s effective KV bandwidth, versus 1.56 TB/s for FlashAttention-3 (Shah et al., 2024) and 2.77 TB/s for FlashInfer (Ye et al., 2025) (Appendix E.3). LASA-Decode consistently outperforms it, with gains increasing as context and batch size grow in I/O-bound decoding (Figure 4, top). With LASA-K15 at 32K context and batch 32, it reaches 13$\times$ speedup. LASA-Index takes only 6.3–24.6 μs per layer per decoding step (Figure 4, middle). Even with LASA-K15, where attention is fastest, indexing remains 3.5–9.3% of decode-layer latency.

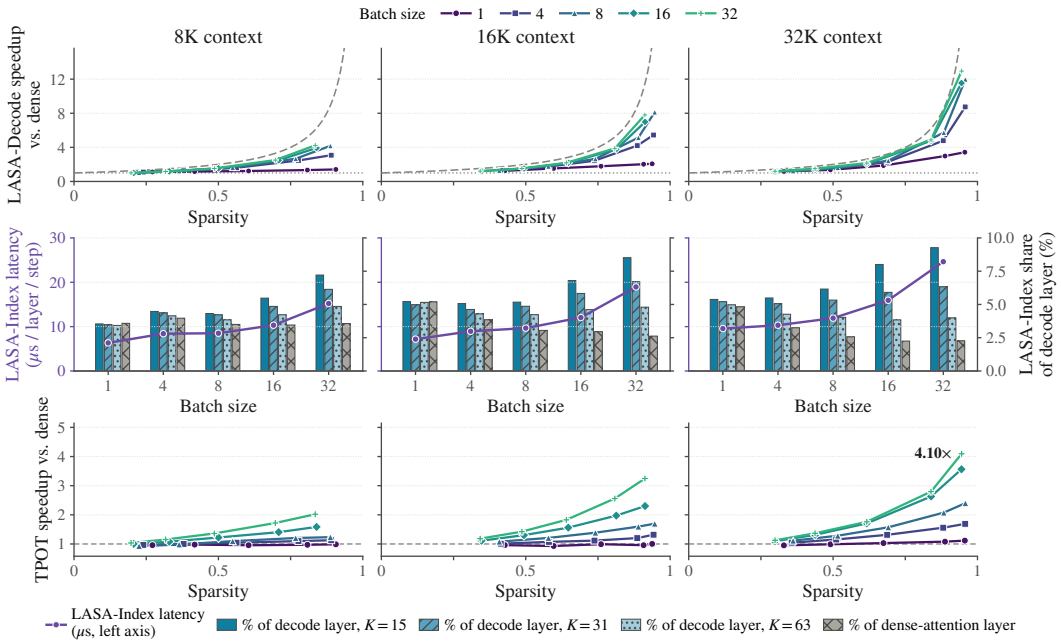

Figure 4: **Efficiency of LASA with Qwen3-8B on H20.** *Top:* LASA-Decode speedup over dense attention; dashed curves show the ideal $1/\text{density}$ ceiling. *Middle:* LASA-Index latency (line, left axis) and its share of decode-layer latency (bars, right axis). *Bottom:* end-to-end TPOT speedup over dense. In the top and bottom rows, points on each curve correspond to LASA-K$\{127, 95, 63, 31, 15\}$ from left to right. For TPOT, both methods use tensor parallelism (TP) of 1, except at 32K context and batch 32 (TP=2).

End-to-end decoding. Against vLLM + `HPC-Ops` dense decoding, LASA-K15 achieves TPOT speedups of $2.0\times$, $3.25\times$, and $4.1\times$ at 8K, 16K, and 32K contexts with batch 32 (Figure 4, bottom). On the complete AIME25 workload (30 problems in one batch, output budget 38,912 tokens), mean generation time is 5.59 minutes for LASA versus 19.11 minutes for dense attention while preserving the accuracy (Table 4). These results show that LASA's speedup survives the full serving stack and holds through practical reasoning tasks.

5 Ablation Study

We ablate predictor inputs and score anchors (Table 1) and compare budgeting rules on the same ranker (Figure 5).

Predictor input. Holding all other settings fixed, we compare three input variants. Layer–head bias removes the query-dependent branch and trains a separate bias for each layer–KV-head pair using the same Stage 2 objective as LASA. We also test whether observing the score distribution helps the predictor set its threshold, using a Deep Sets summary (Zaheer et al., 2017) of the LSE-normalized scores as an additional input alongside the query.

$$\begin{aligned}
\text{Layer–head bias:} \quad & \theta_{\ell,h,t} = b_{\ell,h}, \\
\text{+ query (LASA):} \quad & \theta_{\ell,h,t} = b_{\ell,h} + \mathbf{w}_{\ell,h}^{\top} \operatorname{SiLU}\big(\mathbf{W}_{\ell,h}^{q,\text{pred}}\, \tilde{\mathbf{q}}_{\ell,h,t}^{\text{rank}} + \mathbf{u}_{\ell,h}\big), \\
\text{+ Deep Sets scores:} \quad & \theta_{\ell,h,t} = b_{\ell,h} + \mathbf{w}_{\ell,h}^{\top} \operatorname{SiLU}\big(\mathbf{W}_{\ell,h}^{q,\text{pred}}\, \tilde{\mathbf{q}}_{\ell,h,t}^{\text{rank}} + \mathbf{W}_{\ell,h}^{z,\text{pred}}\, \tilde{\mathbf{z}}_{\ell,h,t} + \mathbf{u}_{\ell,h}\big),
\end{aligned} \tag{9}$$

where $\tilde{\mathbf{z}}_{\ell,h,t}$ is an eight-channel learned histogram of LSE-normalized scores (Appendix B). Layer–head bias loses 5.3 AIME24 and 7.9 AIME25 points relative to LASA, despite higher density and longer outputs, supporting query-dependent allocation. Adding the Deep Sets input does not improve performance, supporting the effectiveness of query-only threshold prediction (Table 1).

Table 1: **Threshold predictor ablations on Qwen3-4B.** All variants use the same Stage-1 ranker. Each ablation changes either the predictor input or the score anchor relative to LASA (top row). FKL: mean Stage-2 vocabulary forward KL (final 100 steps). Acc/Dens.: pass@1 (%) / measured KV density (%).

		AIME24		AIME25	
	FKL ↓	Acc	Dens.	Acc	Dens.
LASA	**0.0155**	**74.22**	**7.34**	**66.93**	**6.63**
Predictor input					
Layer–head bias	0.0237	68.91	8.13	59.01	7.82
+ Deep Sets scores	0.0157	73.18	7.41	64.22	6.65
Score anchor					
Max	0.0193	71.82	10.02	59.95	9.69
LME	0.0302	72.97	9.99	60.78	9.77
Mean	0.0437	57.24	13.82	48.70	12.76

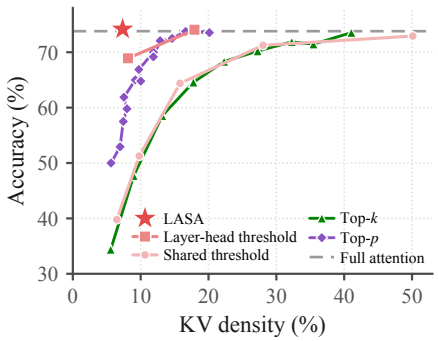

Figure 5: **Same-ranker budgeting.** Qwen3-4B, AIME24; block size 64. Top-k: $k = 16$–80 blocks (step 8); top-p: $p = .66$–$.92$ (step .02); Shared threshold: $\tau = .002$–$.010$ (step .002); Layer–head threshold: $K = 15, 31$; LASA-K15.

Score anchor. With the predictor input fixed, we compare LSE (LASA), max, log-mean-exp (LME), and mean anchors. LSE achieves the lowest FKL and the highest accuracy while using the lowest KV density (Table 1).

Budgeting on the same ranker. We compare LASA with top-k, top-p, Shared threshold, and Layer–head threshold (Layer–head bias in Table 1) using the same Stage-1 ranker at block size 64 on AIME24 (Figure 5). Shared threshold applies $\hat{p}_b > \tau$, where $\hat{p}_b = \mathrm{softmax}(s)_b$ comes from the ranker's block logits and τ is a preset threshold shared across all attention rows. Top-p accumulates the same estimated attention mass to a fixed target. LASA achieves 74.2% accuracy at 7.3% KV density, while the baselines approach comparable accuracy at much higher densities. Top-p offers a better trade-off than top-k. Layer–head threshold improves over Shared threshold, highlighting the value of fine-grained layer–head budgeting. LASA's query-dependent threshold adjustment further improves accuracy by 5.3 points over Layer–head threshold at $K = 15$ despite lower density.

6 Related Work

Fixed-budget sparse attention. Training-free methods exploit observed attention patterns (Xiao et al., 2024; Jiang et al., 2024) or estimate KV importance (Zhang et al., 2023; Tang et al., 2024). Training-based approaches incorporate sparse attention into model training, as in NSA and MoBA (Yuan et al., 2025; Lu et al., 2025), or learn through attention-map distillation (Gao et al., 2024; 2025; DeepSeek-AI, 2025). Sharing sparse indices further reduces selection overhead (Yang et al., 2026; 2025b; Bai et al., 2026). These advances improve KV selection quality and efficiency, but fixed budgets can still waste computation or sacrifice accuracy as attention demands change.

Adaptive-budget sparse attention. Top-p-based methods vary KV budgets with the attention distribution (Lin et al., 2025; Zhu et al., 2026; 2025; Lai et al., 2025). KV-cache compression methods distribute a fixed total budget across layers using preset schedules (Cai et al., 2024) or across heads using observed attention weights (Feng et al., 2025). Without learning dynamic, fine-grained budget allocation from model output quality, these methods can still misallocate computation.

7 Conclusion

LASA improves the accuracy–density Pareto frontier by learning both layer–head-specific budgets and query-dependent threshold adjustments from output quality, with the backbone and ranker frozen. Across four reasoning models and six math, code, and science benchmarks, it retains near-full-attention accuracy with substantially fewer KV accesses. Our vLLM integration translates these savings into up to 4.1× end-to-end decoding speedup on H20 GPUs.

AI USE STATEMENT

The authors used ChatGPT by OpenAI (OpenAI, 2026) for grammatical refinements of the manuscript. All resulting edits were manually reviewed, revised, and finalized by the authors.

ETHICS STATEMENT

This work studies the computational efficiency of language-model inference using existing pretrained models and publicly available research datasets. It involves no human-subject experiments or collection of personal data.

REPRODUCIBILITY STATEMENT

The model architecture and objectives are specified in Section 3. Appendix D reports the training data, optimization settings, budget variants, initialization, precision, parallelism, parameter counts, and convergence curves; Appendix E.1 gives the inference and sampling protocol; and Appendix G defines the density metric and its aggregation. Kernel pseudocode and efficiency protocols are provided in Appendix C and Appendix E.3. Code, configurations, evaluation scripts, and model checkpoints will be released upon publication.

REFERENCES

Art of Problem Solving. AIME problems and solutions. AoPS Wiki. https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions, n.d. Accessed September 25, 2026.

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.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, et al. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*, 2024.

Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations*, volume 2024, pp. 35549–35562, 2024.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35: 16344–16359, 2022.

DeepSeek-AI. DeepSeek-V3.2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025. URL https://arxiv.org/abs/2512.02556.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S Kevin Zhou. Ada-kv: Optimizing kv cache eviction by adaptive budget allocation for efficient llm inference. *Advances in Neural Information Processing Systems*, 38:113152–113188, 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, et al. Seerattention-r: Sparse attention adaptation for long reasoning. *arXiv preprint arXiv:2506.08889*, 2025.

Etash Guha, Ryan Marten, Sedrick Keh, Negin Raoof, Georgios Smyrnis, Hritik Bansal, Marianna Nezhurina, Jean Mercat, Trung Vu, Zayne Sprague, et al. Openthoughts: Data recipes for reasoning models. In *International Conference on Learning Representations*, volume 2026, pp. 108059–108130, 2026.

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

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Peter J Huber. Robust estimation of a location parameter. In *Breakthroughs in statistics: Methodology and distribution*, pp. 492–518. Springer, 1992.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=chfJJYC3iL.

Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H Abdi, Dongsheng Li, Chin-Yew Lin, et al. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention. *Advances in Neural Information Processing Systems*, 37:52481–52515, 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.

Xunhao Lai, Jianqiao Lu, Yao Luo, Yiyuan Ma, and Xun Zhou. Flexprefill: A context-aware sparse attention mechanism for efficient long-sequence inference. *arXiv preprint arXiv:2502.20766*, 2025.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024.

Chaofan Lin, Jiaming Tang, Shuo Yang, Hanshuo Wang, Tian Tang, Boyu Tian, Ion Stoica, Song Han, and Mingyu Gao. Twilight: Adaptive attention sparsity with hierarchical top-p pruning. *Advances in Neural Information Processing Systems*, 38:119406–119432, 2025.

Enzhe Lu, Zhejun Jiang, Jingyuan Liu, Yulun Du, Tao Jiang, Chao Hong, Shaowei Liu, Weiran He, Enming Yuan, Yuzhi Wang, et al. Moba: Mixture of block attention for long-context llms. *Advances in Neural Information Processing Systems*, 38:17790–17815, 2025.

Minh-Thang Luong, Dawsen Hwang, Hoang H Nguyen, Golnaz Ghiasi, Yuri Chervonyi, Insuk Seo, Junsu Kim, Garrett Bingham, Jonathan Lee, Swaroop Mishra, et al. Towards robust mathematical reasoning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 35418–35442, 2025.

OpenAI. OpenAI o1 system card. *arXiv preprint arXiv:2412.16720*, 2024. URL https://arxiv.org/abs/2412.16720.

OpenAI. ChatGPT. Software. https://chatgpt.com/, 2026. Accessed September 16, 2026.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. Yarn: Efficient context window extension of large language models. In *International Conference on Learning Representations*, volume 2024, pp. 31932–31951, 2024.

PyTorch Contributors. PyTorch CUDA top-k implementation. Source code, version 2.8.0. https://github.com/pytorch/pytorch/blob/v2.8.0/aten/src/ATen/native/cuda/TensorTopK.cu, 2025. Accessed September 23, 2026.

PyTorch Contributors. PyTorch documentation: scaled_dot_product_attention. Documentation, version 2.14. https://docs.pytorch.org/docs/2.14/generated/torch.nn.functional.scaled_dot_product_attention.html, n.d. Accessed September 23, 2026.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=Ti67584b98.

Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. Flashattention-3: Fast and accurate attention with asynchrony and low-precision. *Advances in Neural Information Processing Systems*, 37:68658–68685, 2024.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context llm inference. *arXiv preprint arXiv:2406.10774*, 2024.

Tencent Hunyuan AI Infra Team. HPC-Ops: High performance LLM inference operator library. Software repository. https://github.com/Tencent/hpc-ops, 2026. Accessed September 25, 2026.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, volume 2024, pp. 21875–21895, 2024.

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.

Lijie Yang, Zhihao Zhang, Zhuofu Chen, Zikun Li, and Zhihao Jia. Tidaldecode: Fast and accurate llm decoding with position persistent sparse attention. In *International Conference on Learning Representations*, volume 2025, pp. 5823–5844, 2025b.

Lijie Yang, Zhihao Zhang, Arti Jain, Shijie Cao, Baihong Yuan, Yiwei Chen, Zhihao Jia, and Ravi Netravali. Less is more: Fast and accurate reasoning with cross-head unified sparse attention. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=trSWJ99WzS.

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, et al. Flashinfer: Efficient and customizable attention engine for llm inference serving. *Proceedings of Machine Learning and Systems*, 7, 2025.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Yuxing Wei, Lean Wang, Zhiping Xiao, et al. Native sparse attention: Hardware-aligned and natively trainable sparse attention. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 23078–23097, 2025.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in neural information processing systems*, 36:34661–34710, 2023.

Kan Zhu, Tian Tang, Qinyu Xu, Zhan Jin, Yile Gu, Zhichen Zeng, Rohan Kadekodi, Liangyu Zhao, Ang Li, Arvind Krishnamurthy, et al. Tactic: Adaptive sparse attention with clustering and distribution fitting for long-context llms. In *International Conference on Learning Representations*, volume 2026, pp. 30626–30644, 2026.

Qianchao Zhu, Jiangfei Duan, Chang Chen, Siran Liu, Xiuhong Li, Guanyu Feng, Xin Lv, Xiao Chuanfu, Dahua Lin, and Chao Yang. Sampleattention: Near-lossless acceleration of long context LLM inference with adaptive structured sparse attention. In *Eighth Conference on Machine Learning and Systems*, 2025. URL https://openreview.net/forum?id=RuZ80yl71h.

A HARD-GATED ATTENTION: GRADIENT DERIVATIONS

Let $\mathcal{L}$ be the scalar LLM output loss, and let ℓ_{ij} be the gate logit for query position i and key position j. The query, key, and value vectors are $\mathbf{q}_i, \mathbf{k}_j, \mathbf{v}_j \in \mathbb{R}^d$, where d is the attention head dimension. We use $\mathbb{I}[\cdot]$ for the indicator function and $\mathrm{sg}(x)$ for stop-gradient: it equals x in the forward pass and has zero derivative. Define

$$S_{ij} = \frac{\mathbf{q}_i^\top \mathbf{k}_j}{\sqrt{d}}, \qquad E_{ij} = \exp(S_{ij}), \qquad W_{ij} = \frac{E_{ij}}{\sum_k E_{ik}}, \tag{10}$$

where W_{ij} is the dense, ungated softmax weight and k indexes all keys visible to query i. Let $\widetilde{W}_{ij}$ denote the normalized attention weight after applying a gate by either method below. The gated attention output and its weight gradient are

$$\mathbf{o}_i = \sum_k \widetilde{W}_{ik} \mathbf{v}_k, \qquad \frac{\partial \mathcal{L}}{\partial \widetilde{W}_{ij}} = \frac{\partial \mathcal{L}}{\partial \mathbf{o}_i} \cdot \mathbf{v}_j. \tag{11}$$

We inspect $\partial \mathcal{L} / \partial \ell_{ij}$ under the two gating methods.

A.1 ADDITIVE LOG MASKING

Using an additive log mask is a natural way to implement sparse attention with existing attention APIs such as `scaled_dot_product_attention` (SDPA), because the log gate can be passed directly as the additive attention mask. Its closed-gate backward depends on where the STE is placed: applying the STE to the mask gives zero gradient, whereas applying it to the gate before $\log$ gives a NaN. Neither provides a stable, nonzero recovery signal.

Hard gate and mask. For our sigmoid activation, $\sigma(x) = 1/(1+e^{-x})$, define

$$\pi_{ij} = \sigma(\ell_{ij}), \qquad h_{ij} = \mathbb{I}[\pi_{ij} > 1/2], \qquad m_{ij}^{\mathrm{hard}} = \log h_{ij} \in \{0, -\infty\}. \tag{12}$$

Both implementations below use m_{ij}^{hard} in the forward pass but place the STE on opposite sides of the logarithm.

Mask-space STE: zero gradient. Let $a_{ij} = \log \pi_{ij} = \log \sigma(\ell_{ij})$ and apply the STE directly to the additive mask:

$$m_{ij} = a_{ij} + \mathrm{sg}\big(m_{ij}^{\mathrm{hard}} - a_{ij}\big). \tag{13}$$

Thus $m_{ij} = 0$ for an open gate and $m_{ij} = -\infty$ for a closed gate in the forward pass. Its derivative in the backward pass is

$$\frac{\partial m_{ij}}{\partial \ell_{ij}} = \frac{\partial \log \sigma(\ell_{ij})}{\partial \ell_{ij}} = 1 - \pi_{ij}. \tag{14}$$

Implementing a_{ij} with `F.logsigmoid` evaluates this bounded derivative without separately forming $1/\pi_{ij}$.

Forward.

$$\widetilde{S}_{ij} = S_{ij} + m_{ij}, \qquad \widetilde{W}_{ij} = \frac{\exp(\widetilde{S}_{ij})}{\sum_k \exp(\widetilde{S}_{ik})}. \tag{15}$$

For an open gate, $m_{ij} = 0$ and its unnormalized attention weight remains $\exp(S_{ij}) = E_{ij}$. For a closed gate, $m_{ij} = -\infty$ and $\widetilde{W}_{ij} = 0$.

Gradient.

$$\frac{\partial \mathcal{L}}{\partial \ell_{ij}} = \left(\sum_k \frac{\partial \mathcal{L}}{\partial \widetilde{W}_{ik}} \frac{\partial \widetilde{W}_{ik}}{\partial \widetilde{S}_{ij}} \right) \frac{\partial \widetilde{S}_{ij}}{\partial m_{ij}} \frac{\partial m_{ij}}{\partial \ell_{ij}} \tag{16}$$

$$= \left(\sum_k \frac{\partial \mathcal{L}}{\partial \widetilde{W}_{ik}} \frac{\partial \widetilde{W}_{ik}}{\partial \widetilde{S}_{ij}} \right) (1 - \pi_{ij}). \tag{17}$$

With Kronecker delta $\delta_{kj} = 1$ for $k = j$ and 0 otherwise, the softmax Jacobian is

$$\frac{\partial \widetilde{W}_{ik}}{\partial \widetilde{S}_{ij}} = \widetilde{W}_{ik}(\delta_{kj} - \widetilde{W}_{ij}). \tag{18}$$

When gate j is closed, $m_{ij} = -\infty$ and $\widetilde{W}_{ij} = 0$. Therefore,

$$k = j: \quad \widetilde{W}_{ij}(1 - \widetilde{W}_{ij}) = 0 \cdot 1 = 0, \tag{19}$$

$$k \neq j: \quad \widetilde{W}_{ik}(\delta_{kj} - \widetilde{W}_{ij}) = \widetilde{W}_{ik} \cdot 0 = 0. \tag{20}$$

Hence

$$\boxed{\frac{\partial \mathcal{L}}{\partial \ell_{ij}} = 0} \qquad \text{(closed gate).} \tag{21}$$

Consequently, a closed gate receives no quality gradient from the LLM loss, so the model cannot directly correct that gate when it excludes a useful key.

Gate-space STE followed by $\log$**: NaN gradient.** An alternative is to apply the STE to the sigmoid gate first,

$$g_{ij} = h_{ij} + \pi_{ij} - \mathrm{sg}(\pi_{ij}), \qquad m_{ij} = \log g_{ij}. \tag{22}$$

For a closed gate, the forward values are $g_{ij} = 0$, $m_{ij} = -\infty$, and $\widetilde{W}_{ij} = 0$. The softmax backward therefore gives $\partial \mathcal{L}/\partial m_{ij} = 0$, while the next local derivative is

$$\left.\frac{\partial m_{ij}}{\partial g_{ij}}\right|_{g_{ij}=0} = \left.\frac{1}{g_{ij}}\right|_{g_{ij}=0}, \tag{23}$$

which is undefined. PyTorch autograd evaluates the resulting $0 \times \infty$ (or $0/0$) operation as NaN, and this NaN propagates through $\partial g_{ij}/\partial \ell_{ij} = \pi_{ij}(1 - \pi_{ij})$. For $g_{ij} > 0$, the factors can be simplified algebraically as

$$\frac{\partial \mathbf{o}_i}{\partial g_{ij}} = \frac{\partial \mathbf{o}_i}{\partial m_{ij}} \frac{\partial m_{ij}}{\partial g_{ij}} = \frac{\widetilde{W}_{ij}}{g_{ij}}(\mathbf{v}_j - \mathbf{o}_i) = \frac{E_{ij}}{Z_i}(\mathbf{v}_j - \mathbf{o}_i). \tag{24}$$

This equality does not justify cancellation at $g_{ij} = 0$, where the logarithm is singular and the node-wise autograd computation is undefined.

A.2 Multiplicative Gating Softmax

Gate.

$$\pi_{ij} = \sigma(\ell_{ij}), \qquad g_{ij} = \mathbb{I}[\pi_{ij} > 1/2] + \pi_{ij} - \mathrm{sg}(\pi_{ij}), \qquad \frac{\partial g_{ij}}{\partial \ell_{ij}} = \pi_{ij}(1 - \pi_{ij}). \tag{25}$$

Forward.

$$\widetilde{W}_{ij} = \frac{g_{ij} E_{ij}}{Z_i}, \qquad Z_i = \sum_k g_{ik} E_{ik}. \tag{26}$$

We assume at least one visible key remains open, so $Z_i > 0$. The normalized output is then well-defined at $g_{ij} = 0$, and direct differentiation with respect to g_{ij} gives its finite right derivative at this boundary. This is equivalent to post-softmax multiplication followed by renormalization. For an open gate, $g_{ij} = 1$ and its unnormalized attention weight is E_{ij}. For a closed gate, $g_{ij} = 0$ and $\widetilde{W}_{ij} = 0$.

Gradient. Differentiating $\widetilde{W}_{ik} = g_{ik} E_{ik}/Z_i$ with respect to g_{ij} gives

$$\frac{\partial \widetilde{W}_{ik}}{\partial g_{ij}} = \begin{cases} \dfrac{E_{ij}}{Z_i}(1 - \widetilde{W}_{ij}), & k = j, \\ -\widetilde{W}_{ik} \dfrac{E_{ij}}{Z_i}, & k \neq j. \end{cases} \tag{27}$$

Therefore,

$$\frac{\partial \mathcal{L}}{\partial \ell_{ij}} = \sum_k \frac{\partial \mathcal{L}}{\partial \widetilde{W}_{ik}} \frac{\partial \widetilde{W}_{ik}}{\partial g_{ij}} \frac{\partial g_{ij}}{\partial \ell_{ij}} \tag{28}$$

$$= \left\{ \frac{\partial \mathcal{L}}{\partial \widetilde{W}_{ij}} \frac{E_{ij}}{Z_i}(1 - \widetilde{W}_{ij}) - \sum_{k \neq j} \frac{\partial \mathcal{L}}{\partial \widetilde{W}_{ik}} \widetilde{W}_{ik} \frac{E_{ij}}{Z_i} \right\} \pi_{ij}(1 - \pi_{ij}) \tag{29}$$

$$= \frac{E_{ij}}{Z_i} \left[\frac{\partial \mathcal{L}}{\partial \widetilde{W}_{ij}} - \sum_k \widetilde{W}_{ik} \frac{\partial \mathcal{L}}{\partial \widetilde{W}_{ik}} \right] \pi_{ij}(1 - \pi_{ij}) \tag{30}$$

$$= \frac{E_{ij}}{Z_i} \left[\frac{\partial \mathcal{L}}{\partial \mathbf{o}_i} \cdot \mathbf{v}_j - \sum_k \widetilde{W}_{ik} \left(\frac{\partial \mathcal{L}}{\partial \mathbf{o}_i} \cdot \mathbf{v}_k \right) \right] \pi_{ij}(1 - \pi_{ij}). \tag{31}$$

For a closed gate, Z_i excludes E_{ij}. Therefore E_{ij}/Z_i can become dangerously large if the dropped S_{ij} is much larger than the retained scores, making the surrogate gradient numerically unstable.

A.3 Ratio Clipping for Multiplicative Gating Softmax

At a closed gate, Additive Log Masking gives either a zero gradient under a mask-space STE or a NaN when a gate-space STE is followed by $\log$. Multiplicative Gating Softmax instead provides a finite gate-space derivative that is not structurally zero, but its E_{ij}/Z_i factor can become dangerously large. To retain this recovery path while controlling the risk, LASA clips the factor in the gate backward. Let $C > 0$ be the clipping threshold:

$$\left(\frac{\partial \mathcal{L}}{\partial \ell_{ij}} \right)_{\text{clipped}} = \min\left(\frac{E_{ij}}{Z_i}, C \right) \left[\frac{\partial \mathcal{L}}{\partial \mathbf{o}_i} \cdot \mathbf{v}_j - \sum_k \widetilde{W}_{ik} \left(\frac{\partial \mathcal{L}}{\partial \mathbf{o}_i} \cdot \mathbf{v}_k \right) \right] \pi_{ij}(1 - \pi_{ij}). \tag{32}$$

For an open gate, $g_{ij} = 1$ implies $E_{ij}/Z_i = \widetilde{W}_{ij} \leq 1$, so $C = 1$ preserves its exact gradient. For a closed gate, clipping bounds this factor without structurally eliminating the recovery path. The complete gradient can still be zero when the loss-weighted value difference is zero. This ratio cap is distinct from the optimizer's global gradient-norm clipping.

B Score anchors

Let $r = (\ell, h, t)$ identify an attention row and let $\mathbf{s} = (s_1, \ldots, s_n)$ contain its n causally valid block scores.

B.1 Why an anchor is needed

Stage 1 supervises the ranker through its softmax distribution, not the absolute score values. For any scalar offset ξ_r shared by all block scores in attention row r, with $\mathbf{1}$ denoting the n-dimensional all-ones vector,

$$\mathrm{softmax}(\mathbf{s} + \xi_r \mathbf{1}) = \mathrm{softmax}(\mathbf{s}). \tag{33}$$

The distillation objective therefore leaves this offset unconstrained. A threshold on raw scores would nevertheless have to shift by the same ξ_r to preserve its selection. This adds a nuisance offset, unrelated to the teacher's importance distribution, that a predictor driven by the query and layer–head identity would have to estimate.

We remove it by subtracting a statistic of the score vector, $A(\mathbf{s})$, the *anchor*, and comparing $x_i^A = s_i - A(\mathbf{s})$ with the learned threshold θ_r. An anchor satisfying $A(\mathbf{s} + \xi_r \mathbf{1}) = A(\mathbf{s}) + \xi_r$ cancels the offset exactly:

$$(s_i + \xi_r) - A(\mathbf{s} + \xi_r \mathbf{1}) = s_i - A(\mathbf{s}) = x_i^A. \tag{34}$$

All four anchors below satisfy this property and preserve block ranking within each attention row. Their difference is the reference against which the threshold measures block importance after the arbitrary offset has been removed.

B.2 Anchor choices and equivalent thresholds

Define $Z = \sum_j e^{s_j}$ and the estimated attention mass of block i as $\hat{p}_i = e^{s_i}/Z$ (estimated mass for short), with $\hat{p}_{\max} = \max_j \hat{p}_j$ and $\overline{s} = n^{-1}\sum_j s_j$. The anchors ablated in Table 1 are

$$x_i^{\mathrm{LSE}} = s_i - \log Z = \log \hat{p}_i, \qquad \text{log estimated mass,} \tag{35}$$
$$x_i^{\max} = s_i - \max_j s_j = \log(\hat{p}_i/\hat{p}_{\max}), \qquad \text{log winner-relative mass,} \tag{36}$$
$$x_i^{\mathrm{LME}} = s_i - \log(Z/n) = \log(n\hat{p}_i), \qquad \text{log mass relative to } 1/n, \tag{37}$$
$$x_i^{\mathrm{mean}} = s_i - \overline{s}, \qquad \text{mean-relative score.} \tag{38}$$

LSE gives log estimated mass. Max uses the strongest block as its reference, LME uses the uniform reference $1/n$, and mean uses the average raw score. A given numerical threshold therefore has a different meaning in each coordinate.

To reproduce the LSE selection $\log \hat{p}_i > \theta_r$ using an anchor A, observe that

$$\log \hat{p}_i = x_i^A - \delta_A(\mathbf{s}), \qquad \delta_A(\mathbf{s}) = \log Z - A(\mathbf{s}). \tag{39}$$

The equivalent comparison is therefore $x_i^A > \theta_r + \delta_A(\mathbf{s})$, with

$$\begin{aligned} \delta_{\mathrm{LSE}} &= 0, & \delta_{\max} &= -\log \hat{p}_{\max}, \\ \delta_{\mathrm{LME}} &= \log n, & \delta_{\mathrm{mean}} &= \log Z - \overline{s}. \end{aligned} \tag{40}$$

These corrections are shared across blocks within an attention row but vary across attention rows. Max uses the peak estimated mass, LME the candidate count, and mean the difference between the log partition and mean score. A predictor that recovers these scalars can reproduce the LSE mask exactly. Our query-only predictor has no explicit score-distribution input, so these corrections are not directly available to it. LSE instead provides log estimated mass directly, leaving the threshold to model each attention row's budget demand.

B.3 Behavior as the candidate set changes

The same coordinates also behave differently as decoding increases the number of causally valid blocks.

LSE uses the score partition. With the existing logits fixed, suppose newly appended blocks contribute total exponentiated score weight ηZ. The new partition is $(1+\eta)Z$, so every existing LSE score changes by

$$x_i^{\mathrm{LSE}\prime} - x_i^{\mathrm{LSE}} = -\log(1+\eta). \tag{41}$$

Candidates with negligible total score weight therefore induce only a negligible common shift. Moreover, if k_r blocks satisfy $\hat{p}_i > e^{\theta_r}$, then $k_r e^{\theta_r} < 1$ and hence $k_r < e^{-\theta_r}$. For a fixed threshold, this count bound is independent of n, while learned thresholds still allow budgets to vary across attention rows.

Max is winner-relative. With max anchoring, the comparison $x_i^{\max} > \theta_r$ is equivalent to $\hat{p}_i > e^{\theta_r}\hat{p}_{\max}$. Its effective estimated-mass cutoff therefore changes with the concentration of the score distribution unless the predictor supplies the missing $-\log \hat{p}_{\max}$ correction. Max also depends on one order statistic: for a unique winning score s_j, $\partial\,\mathrm{LSE}(\mathbf{s})/\partial s_j = \hat{p}_j$, whereas $\partial \max(\mathbf{s})/\partial s_j = 1$. Perturbing the winner consequently moves every non-winning max-relative score by the full perturbation, and the coordinate is non-smooth when the winner changes.

LME carries a fixed-fraction tendency. LME selects $\hat{p}_i > e^{\theta_r}/n$. If the complete score multiset is duplicated m times, then $n' = mn$ and each copy has estimated mass $\hat{p}'_i = \hat{p}_i/m$, leaving $n'\hat{p}'_i = n\hat{p}_i$. A fixed LME threshold therefore selects all m copies of every previously selected block: its selected count grows with the candidate set while its selected fraction remains fixed. Recovering the LSE cutoff requires the explicit $\log n$ correction in Eq. (40).

Mean is controlled by the low-score tail. Appending m candidates with mean score $\overline{u}$ shifts every original mean-relative score by

$$\Delta x_i^{\text{mean}} = \frac{m}{n+m}(\overline{s} - \overline{u}). \tag{42}$$

This shift need not be small even when the appended candidates have negligible estimated mass. Mean anchoring can therefore change the selected set because low-score candidates exist, rather than because the estimated mass of the original blocks changed appreciably.

B.4 Does LSE still need score-distribution input?

LSE removes the anchor-dependent corrections above, but the threshold predictor might still benefit from observing the normalized score distribution. Even at a fixed threshold, the selected count $k_r = \sum_{i=1}^{n} \mathbb{I}[\hat{p}_i > e^{\theta_r}]$ already depends on that distribution through the estimated-mass comparisons. We test whether explicitly summarizing the same distribution gives the predictor useful additional information for setting its cutoff.

Score-aware predictor. The score-aware ablation leaves the frozen backbone, Stage-1 ranker, LSE anchor, query branch, fusion width, training objective, and all optimization settings unchanged. It adds an eight-channel Deep Sets summary (Zaheer et al., 2017) of the LSE-normalized scores. For channel c,

$$z_{\ell,h,t,c} = \frac{1}{|\mathcal{C}_t|} \sum_{b \in \mathcal{C}_t} \mathrm{SiLU}\big(a_{\ell,h,c} x_{\ell,h,t,b} + d_{\ell,h,c}\big), \qquad c = 1, \dots, 8, \tag{43}$$

where $(a_{\ell,h,c}, d_{\ell,h,c})$ are learned scalars. The nonlinear channels act as soft bins, and their mean-pooled outputs form a learned histogram $\mathbf{z}_{\ell,h,t}$ of the score vector. A scalar root-mean-square (RMS) scale $\rho^z_{\ell,h}$ for each layer–head pair, estimated from the first training batch and then fixed, gives $\tilde{\mathbf{z}}_{\ell,h,t} = \mathbf{z}_{\ell,h,t}/(\rho^z_{\ell,h} + \epsilon)$, where $\epsilon > 0$. The original predictor

$$\theta_{\ell,h,t} = b_{\ell,h} + \mathbf{w}_{\ell,h}^{\top} \mathrm{SiLU}\big(\mathbf{W}^{q,\text{pred}}_{\ell,h} \tilde{\mathbf{q}}^{\text{rank}}_{\ell,h,t} + \mathbf{u}_{\ell,h}\big) \tag{44}$$

is changed only by adding a linear projection of this histogram inside the same MLP:

$$\theta_{\ell,h,t} = b_{\ell,h} + \mathbf{w}_{\ell,h}^{\top} \mathrm{SiLU}\big(\mathbf{W}^{q,\text{pred}}_{\ell,h} \tilde{\mathbf{q}}^{\text{rank}}_{\ell,h,t} + \mathbf{W}^{z,\text{pred}}_{\ell,h} \tilde{\mathbf{z}}_{\ell,h,t} + \mathbf{u}_{\ell,h}\big). \tag{45}$$

Here $\mathbf{W}^{q,\text{pred}}_{\ell,h} \in \mathbb{R}^{d_f \times d_r}$ is the predictor's query-input projection, and $\mathbf{W}^{z,\text{pred}}_{\ell,h} \in \mathbb{R}^{d_f \times 8}$ is its added score-input projection. Thus the ablation gives the predictor direct access to score-distribution shape without changing the estimated mass used by the gate.

B.5 Reading the ablation

Table 1 separates two questions. The predictor-input variants hold the LSE anchor fixed: removing the query degrades both FKL and downstream accuracy, while adding the Deep Sets score summary does not improve over the query-only predictor. The score-anchor variants instead hold the query-only predictor fixed. Max and LME use about $1.5\times$ the density of LSE yet lose 6–7 AIME25 points, while mean uses about $1.9\times$ the density and loses 17–18 points. Because all variants share the ranker and the same relaxed $K = 15$ objective, the alternatives are not disadvantaged by a smaller average budget; the resulting higher densities are themselves outcomes of learning in those coordinates.

Together, these results favor LSE with the query-only predictor at the tested operating points. Score shape already affects the selected count through estimated-mass comparison; explicitly providing the tested eight-channel summary does not improve results under the same training setup.

C Fused gated attention kernel

We implement Multiplicative Gating Softmax (Appendix A.2) by fusing its forward normalization and block-gate derivative into FlashAttention-2's online forward and recomputation backward (Dao, 2024), avoiding materialization of token-level attention weights.

C.1 FORWARD: A GATE INSIDE ONLINE SOFTMAX

Let $\mathbf{q}_i, \mathbf{k}_j, \mathbf{v}_j \in \mathbb{R}^d$ be a query, key, and value, and let $g_{i,b} \in \{0,1\}$ be the hard gate for query i and key block b. The map $b(j)$ returns the block containing key token j. We always retain the KV block containing the query, subject to causal masking, ensuring a nonempty selection; these KV accesses are included in the reported density. For one row,

$$S_{ij} = \frac{\mathbf{q}_i^\top \mathbf{k}_j}{\sqrt{d}}, \qquad M_i = \max_{j \text{ causal}} S_{ij},$$
$$z_i = \sum_{j \text{ causal}} g_{i,b(j)} e^{S_{ij} - M_i}, \qquad L_i = M_i + \log z_i, \tag{46}$$
$$\widetilde{P}_{ij} = g_{i,b(j)} e^{S_{ij} - L_i}, \qquad \mathbf{O}_i = \sum_j \widetilde{P}_{ij} \mathbf{v}_j. \tag{47}$$

Algorithm 1 Gated causal attention forward (row-wise online form)

input $Q, K, V \in \mathbb{R}^{n \times d}$, hard block gates g, block size B
for each query row i **do**
  $M_i \leftarrow -\infty$; $z_i \leftarrow 0$; $\mathbf{a}_i \leftarrow \mathbf{0}$
  for each causally valid key block b **do**
    $S_{ij} \leftarrow \mathbf{q}_i^\top \mathbf{k}_j / \sqrt{d}$ for $j \in b$
    $M_i' \leftarrow \max(M_i, \max_{j \in b} S_{ij})$; $\alpha_i \leftarrow e^{M_i - M_i'}$
    $u_{ij} \leftarrow g_{i,b} e^{S_{ij} - M_i'}$
    $z_i \leftarrow \alpha_i z_i + \sum_{j \in b} u_{ij}$
    $\mathbf{a}_i \leftarrow \alpha_i \mathbf{a}_i + \sum_{j \in b} u_{ij} \mathbf{v}_j$; $M_i \leftarrow M_i'$
  end for
  $\mathbf{O}_i \leftarrow \mathbf{a}_i / z_i$; $L_i \leftarrow M_i + \log z_i$
end for
return O, L

C.2 BACKWARD: FLASHATTENTION GRADIENTS PLUS ONE GATE REDUCTION

Let $\mathcal{L}$ be the scalar output loss and write $dX \equiv \partial \mathcal{L} / \partial X$. The backward recomputes

$$r_{ij} = e^{S_{ij} - L_i}, \qquad \widetilde{P}_{ij} = g_{i,b(j)} r_{ij}, \qquad d\widetilde{P}_{ij} = d\mathbf{O}_i^\top \mathbf{v}_j, \qquad D_i = d\mathbf{O}_i^\top \mathbf{O}_i = \sum_j \widetilde{P}_{ij} d\widetilde{P}_{ij}. \tag{48}$$

The value and score gradients are

$$d\mathbf{v}_j = \sum_i \widetilde{P}_{ij}\, d\mathbf{O}_i, \qquad dS_{ij} = \widetilde{P}_{ij}(d\widetilde{P}_{ij} - D_i), \tag{49}$$
$$d\mathbf{q}_i = \frac{1}{\sqrt{d}} \sum_j dS_{ij} \mathbf{k}_j, \qquad d\mathbf{k}_j = \frac{1}{\sqrt{d}} \sum_i dS_{ij} \mathbf{q}_i. \tag{50}$$

For the block-gate gradient, write $u_{ij} = g_{i,b(j)} e^{S_{ij} - M_i}$ and $\widetilde{P}_{ij} = u_{ij} / \sum_k u_{ik}$. Applying the normalized-weight Jacobian to all tokens sharing $g_{i,b}$ gives

$$dg_{i,b} = \sum_{j:\, b(j) = b} r_{ij}(d\widetilde{P}_{ij} - D_i). \tag{51}$$

Because r_{ij} does not include the gate, this derivative remains available for a closed block. We cap r_{ij} at 1 only in this gate-gradient reduction:

$$\bar{r}_{ij} = \min(r_{ij}, 1), \qquad dg_{i,b}^{\text{clip}} = \sum_{j:\, b(j) = b} \bar{r}_{ij}(d\widetilde{P}_{ij} - D_i). \tag{52}$$

Algorithm 2 Gated causal attention backward (recomputation form)

input Q, K, V, g, O, L, dO
$dQ, dK, dV, dg \leftarrow 0$
for each query row i **do**
  $D_i \leftarrow d\mathbf{O}_i^\top \mathbf{O}_i$
end for
for each query tile and each causally valid key block b (open or closed) **do**
  recompute $S_{ij} \leftarrow \mathbf{q}_i^\top \mathbf{k}_j / \sqrt{d}$
  $\widetilde{P}_{ij} \leftarrow e^{S_{ij} - L_i}$ if $g_{i,b} = 1$, and 0 otherwise; $d\widetilde{P}_{ij} \leftarrow d\mathbf{O}_i^\top \mathbf{v}_j$
  $\bar{r}_{ij} \leftarrow \min(e^{S_{ij} - L_i}, 1)$
  $dS_{ij} \leftarrow \widetilde{P}_{ij}(d\widetilde{P}_{ij} - D_i)$
  $d\mathbf{v}_j \mathrel{+}= \sum_i \widetilde{P}_{ij} d\mathbf{O}_i$
  $d\mathbf{q}_i \mathrel{+}= d^{-1/2} \sum_{j \in b} dS_{ij} \mathbf{k}_j$; $d\mathbf{k}_j \mathrel{+}= d^{-1/2} \sum_i dS_{ij} \mathbf{q}_i$
  $dg_{i,b} \mathrel{+}= \sum_{j \in b} \bar{r}_{ij}(d\widetilde{P}_{ij} - D_i)$
end for
return dQ, dK, dV, dg

If one block gate is shared by a GQA query-head group, its dg contributions are summed over those query heads. Computing dg for closed blocks requires recomputing their scores and $d\widetilde{P}$, so Algorithm 2 traverses both open and closed blocks.

D Training Details

D.1 Data and Optimization

For the 64-token-block variants, both stages use variable-length 16K sequences packed from OpenThoughts3. Stage 1 trains one ranker per model; Stage 2 trains a separate threshold predictor for each budget while freezing the backbone and ranker. Table 2 lists the remaining settings.

Table 2: Optimization settings for the Stage-1 Seer-R ranker and the query-only LSE threshold predictor.

	Stage 1 (ranker)	Stage 2 (threshold)
Trainable	Q/block ranker projections and norms	threshold predictor
Frozen	backbone	backbone and Stage-1 ranker
Data / packing	OpenThoughts3, packed variable-length 16K	same
Training tokens	0.42B	52M per budget variant
Steps / warmup	800 / 20	400 / 20
Global batch	32 (8 GPUs, accum. 4)	8 (8 GPUs)
Optimizer / weight decay	AdamW / 0	same
Learning rate / schedule	10^{-3} / cosine	4×10^{-3} / cosine
Objective	dense block-map forward KL	Eq. (6), temperature 1, $\lambda_B = 0.1$, Huber $\delta = 0.1$
Budget	—	$K \in \{15, 31, 63, 95, 127\}$; block-count edges $(16, 32, 64, 128, 256)$
Block size / ranker width	64 / 128	same
Predictor width d_f	—	16
Gate	—	hard sigmoid STE; gate-gradient ratio cap 1
Threshold initialization	—	first-batch calibration near K
Precision / parallelism	bf16, global gradient clip 1, ZeRO-2 on 8 MI300X GPUs	

Budget-loss specification. For an attention row r, let n_r and k_r be its causal and deployed block counts. With 64-token blocks, define $\mathcal{R}_j = \{r : e_j < n_r \le e_{j+1}\}$ for $\mathbf{e} = (16, 32, 64, 128, 256)$ and

$j = 1, \ldots, 4$, where the edges are in blocks. For each non-empty bin, let $\bar{k}_j = \sum_{r \in \mathcal{R}_j} k_r / |\mathcal{R}_j|$. For $\mathcal{A} = \{j : |\mathcal{R}_j| > 0\} \neq \varnothing$, the exact penalty is

$$\mathcal{L}_B = \frac{1}{|\mathcal{A}|} \sum_{j \in \mathcal{A}} \phi_\delta \left(\left[\frac{\bar{k}_j - K}{K} \right]_+ \right), \qquad \phi_\delta(u) = \begin{cases} u^2/(2\delta), & 0 \leq u < \delta, \\ u - \delta/2, & u \geq \delta, \end{cases} \quad \delta = 0.1, \tag{53}$$

where $[z]_+ = \max(z, 0)$. Each bin pools examples, layers, KV heads, and query positions, with an all-reduce across data-parallel workers before division; non-empty bins are then weighted equally. Empty bins are omitted, and the loss is zero if all are empty. Within a covered bin, rows with fewer than K available blocks are not special-cased; rows outside the four intervals do not enter $\mathcal{L}_B$.

D.2 Threshold-predictor initialization

We use attention rows from the first Stage-2 batch to initialize the input scale and threshold operating point. Neither procedure uses evaluation data or is repeated at deployment.

Query-input scale. For each layer–head pair, let $\mathcal{R}_{\ell,h}$ be the sampled rows and let e index the d_r query features. We fix

$$\rho^q_{\ell,h} = \sqrt{\underset{r \in \mathcal{R}_{\ell,h}, e}{\text{mean}} \left(q^{\text{rank}}_{r,e}\right)^2}, \qquad \mathbf{W}^{q,\text{pred}}_{\ell,h} = \frac{\mathbf{W}^{\text{train}}_{\ell,h}}{\rho^q_{\ell,h} + \epsilon}, \tag{54}$$

where $\epsilon > 0$. The implementation applies $\mathbf{W}^{\text{train}}_{\ell,h}$ to $\text{sg}(\mathbf{q}^{\text{rank}})/(\rho^q_{\ell,h} + \epsilon)$; folding the fixed scale into $\mathbf{W}^{q,\text{pred}}_{\ell,h}$ gives the parameterization in Eq. (5).

Threshold operating point. Let $x^{(K)}_r$ be the anchored-score cutoff corresponding to K open blocks on row r. We initialize

$$b^0_{\ell,h} = \underset{r \in \mathcal{R}_{\ell,h}}{\text{median}}\, x^{(K)}_r + \delta_\ell, \tag{55}$$

where a layer-wise shift δ_ℓ is found by bisection so that the aggregate sampled budget is near K. The output weights $\mathbf{w}_{\ell,h}$ and trainable bias residuals start at zero. This initialization sets only the starting point; training subsequently optimizes Eq. (6) without imposing a top-K rule.

D.3 Parameter and training cost

Stage 1 is trained once per model; Stage 2 costs are for one budget variant.

Table 3: Trainable parameters and wall-clock training time on eight MI300X GPUs.

	Trainable parameters		Hours	
Model	Stage 1	Stage 2	Stage 1	Stage 2
Qwen3-4B	33,039,360	599,328	1.23	1.39
Qwen3-8B	33,039,360	599,328	1.45	1.63
Qwen3-14B	41,953,280	665,920	2.16	2.70
R1-Llama-8B	29,368,320	532,736	1.30	1.49

D.4 Training convergence

Figure 6 shows the ranker and threshold-predictor training curves for both block sizes. All plotted runs use OpenThoughts3.

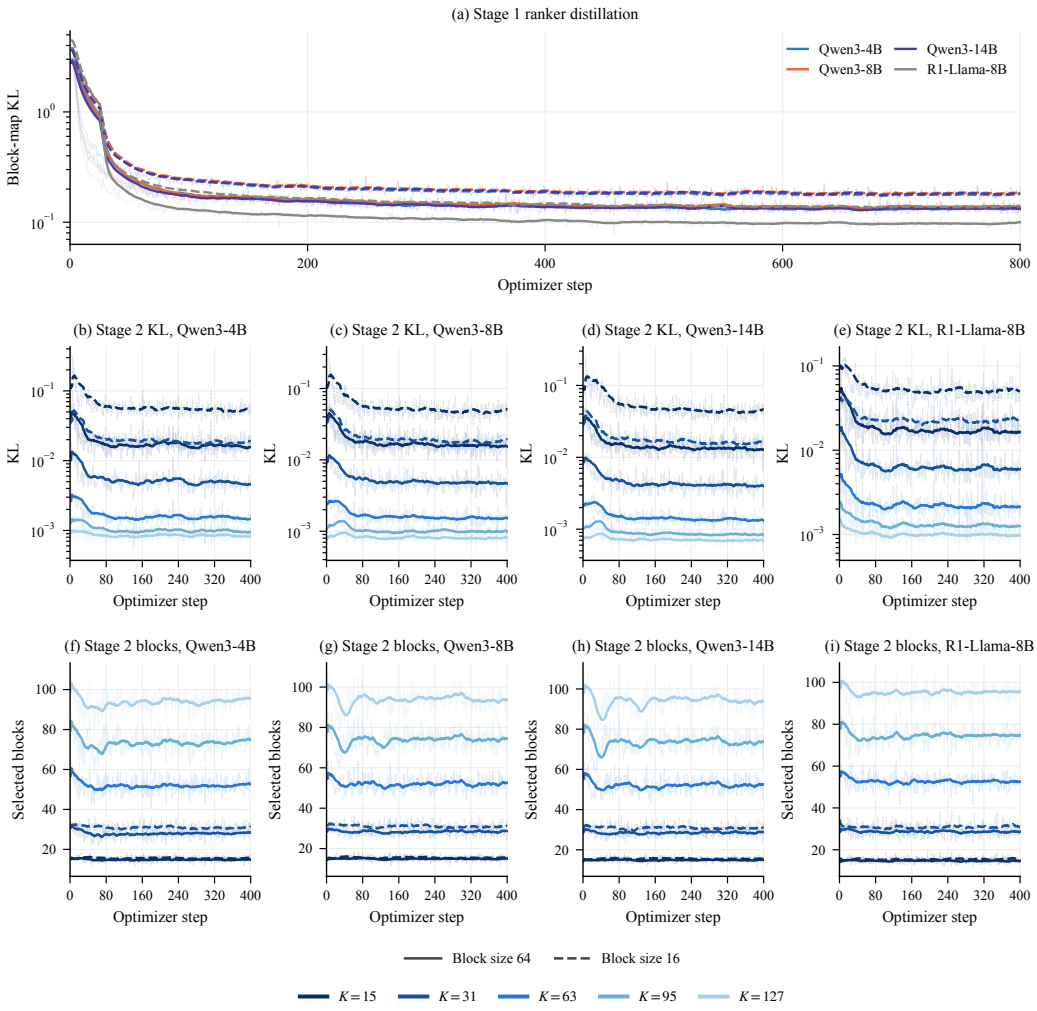

Figure 6: **Training convergence of LASA.** *(a)* Stage-1 block-map KL. *(b)–(e)* Stage-2 vocabulary KL and *(f)–(i)* selected blocks, one panel per model. Solid curves show 64-token blocks for $K \in \{15, 31, 63, 95, 127\}$; dashed curves show 16-token blocks for $K \in \{15, 31\}$. KL panels use logarithmic vertical axes.

E Inference evaluation configuration and results

E.1 Protocol for evaluation on reasoning tasks

On the Qwen3 models, all methods use the official sampling parameters: temperature 0.6, sampling top-p 0.95, and sampling top-k 20. The maximum output length is 32,768 tokens, extended to 38,912 on AIME24/25 following the Qwen3 technical report (Yang et al., 2025a). DeepSeek-R1-Distill-Llama-8B follows the official DeepSeek-R1 configuration with its native chat template (Guo et al., 2025). Prefill is dense for every method; sparse attention is applied only during decoding. Following the Qwen3 report, LiveCodeBench V5 denotes the problem window from October 2024 to January 2025 (210 problems) and V6 the window from February to April 2025 (131 problems). Sampling counts for the main evaluation are given in Section 4.1.

Twilight evaluation protocol. We evaluate Twilight at $p \in \{0.4, 0.7, 0.95, 0.99\}$ on the Qwen3 models and $p \in \{0.7, 0.95, 0.99\}$ on R1-Llama-8B, chosen to obtain a reasonably spaced sweep on the KV-density axis. Following the protocol in the Twilight paper and released implementation, we keep the first two layers dense and include all of their KV accesses in the reported density. Because

these dense layers impose a nonzero density floor, we include $p = 0.4$ to cover the lower-density regime.

We also tested an all-sparse variant that applies Twilight's Quest-style top-k prefilter over 16-token pages followed by INT4 token-level top-p filtering to the first two layers as well. On Qwen3-4B AIME24/25, accuracy is 0% at $p \in \{0.90, 0.95\}$ and only $11.7\%/13.3\%$ at $p = 0.99$ (20.26% density; $2{,}878/2{,}803$ average attended tokens per query); on LiveCodeBench V5/V6, it reaches only $7.1\%/6.9\%$ ($24.25\%/24.16\%$ density; $3{,}152/3{,}184$ tokens). At $p = 0.99$ on V5/V6, Qwen3-8B reaches only $28.5\%/29.6\%$ ($26.83\%/27.39\%$ density; $3{,}250/3{,}148$ tokens), and Qwen3-14B $38.8\%/37.4\%$ ($35.47\%/36.99\%$ density; $3{,}588/3{,}527$ tokens). This collapse may reflect high sensitivity of the first two layers to Twilight's sparse approximation, so we use the release-faithful protocol with the first two layers dense.

16-token-block variants. For the LASA-K15 and LASA-K31 points in Figure 3, Stage 1 distills a ranker over 16-token blocks and Stage 2 trains the corresponding threshold predictors with soft budgets of 15 and 31 blocks (240 and 496 tokens). For all four models, the 16-token-block stars in Figures 3 and 7 use K15 on GPQA and MATH-500 and K31 on AIME and LiveCodeBench.

E.2 Generation length under sparse decoding

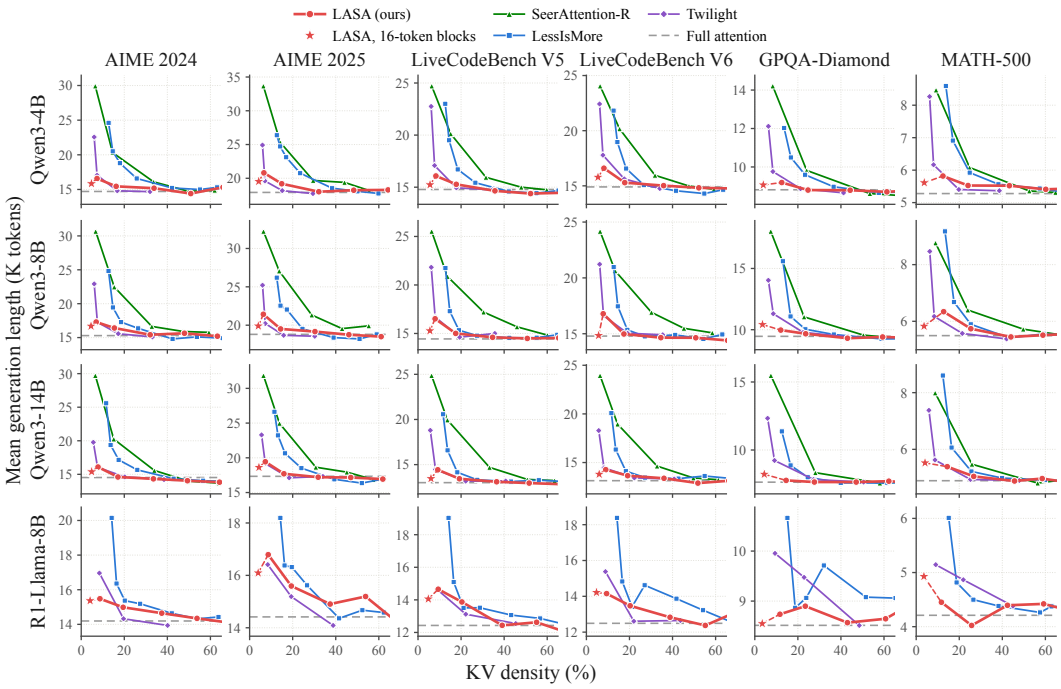

Figure 7: **Mean generation length versus KV density.** The same runs, models, benchmarks, and method colors as Figure 3; here density uses a linear axis. Low-budget fixed-count baselines show greater length inflation than LASA and Twilight. Stars denote the 16-token-block LASA variants.

E.3 LASA efficiency evaluation: protocol and results

Measurement. We use BF16 Qwen3-8B on H20 with CUDA Graph replay, 32 query heads, 8 KV heads, head dimension 128. Kernel timings include task-map construction and sparse attention across 36 layers and are reported per layer.

Dense-baseline bandwidth. Effective KV bandwidth is the KV cache size per layer divided by the per-layer attention latency. At batch 32 and 32K context, HPC-Ops achieves 3.28 TB/s, FlashAttention-3 1.56 TB/s, and FlashInfer 2.77 TB/s.

64K context. Figure 8 extends the efficiency evaluation to 64K context. Both methods use YaRN (Peng et al., 2024) with RoPE factor 1.625.

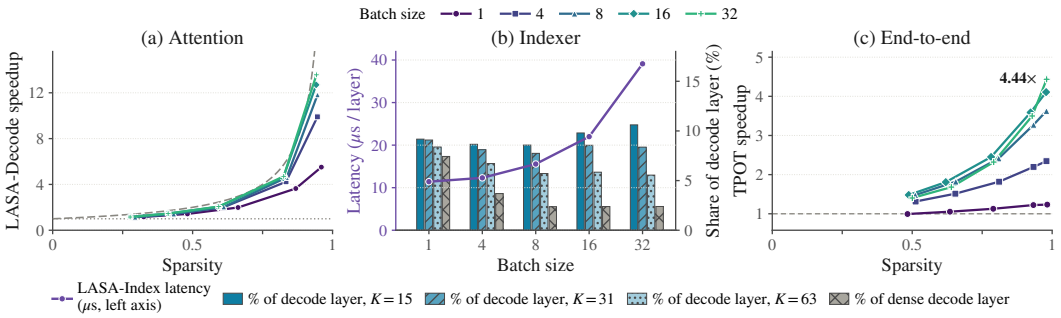

Figure 8: **LASA efficiency at 64K context on H20.** (a) LASA-Decode speedup over dense attention; (b) LASA-Index latency and its share of decode-layer latency; (c) TPOT speedup over dense. Colors and markers follow Figure 4; points run from K127 to K15 as sparsity increases. The attention and indexer latency lines in (a,b) use one GPU; bars show the corresponding in-model shares. TPOT uses TP=1 at batches 1–8, TP=2 at batch 16, and TP=4 at batch 32.

E.4 Efficiency comparison with baselines

We integrate SeerAttention-R and Twilight into vLLM alongside LASA to study the realized efficiency–accuracy Pareto frontier, preserving their released algorithms and high-performance kernels as closely as possible.

Baseline integration. For SeerAttention-R, we adapt the released TileLang attention kernel's contiguous KV loads to vLLM's block-table-based paged access. At 16K context, batch size 1, and a 4K token budget on H20, this change increases attention GPU latency from 20.33 to 20.48 μs (+0.7%) under identical inputs and launch parameters. For Twilight, we build on its released CUDA selection kernels and adapt FlashInfer attention to read selected tokens directly from vLLM's KV cache without copying KV data. We move dynamic planning to the GPU to support CUDA Graph replay. In a matched H20 calibration at 16K context and batch size 1, GPU planning with graph replay reduces the dynamic attention component's wall-clock latency from 205.46 to 21.42 μs; adapting KV addressing then increases it to 21.60 μs. LessIsMore is omitted from the batched comparison because its released efficient implementation does not support batch sizes greater than one.

We offer this supplementary comparison as a best-effort evaluation of deployment efficiency, aiming to preserve the released algorithms and kernel implementations as closely as practical.

Evaluation. We pair Qwen3-8B accuracy from Figure 3 with H20 TPOT at matching settings, using 16K context for AIME24 and 20K for AIME25, near the mean generation-length range of most methods. We sweep LASA at $K \in \{15, 31, 63, 95, 127\}$ with block size 64 and $K = 31$ with block size 16, SeerAttention-R at $\{1, 2, 4, 6, 8\}$K tokens, and Twilight at $p \in \{0.4, 0.7, 0.95, 0.99\}$.

Results. The results show that LASA substantially advances the realized efficiency–accuracy Pareto frontier. The efficient TileLang attention kernel gives SeerAttention-R TPOT close to LASA at comparable sparsity, but its fixed top-k budget gives lower accuracy and requires more attended tokens to recover it, leaving its realized Pareto frontier behind. Twilight remains close to LASA in accuracy at low-density points, but its fine-grained selection and top-p computation add indexing overhead. It therefore generally requires roughly 2–4× LASA's TPOT to approach full-attention accuracy, whereas LASA reaches that region at lower TPOT.

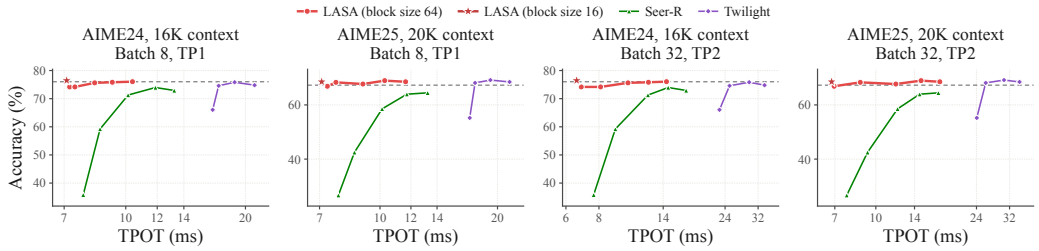

Figure 9: **Accuracy versus end-to-end decode TPOT on Qwen3-8B and H20.** Panels show AIME24 (16K context) and AIME25 (20K context) at batch 8 with TP1 (left pair) and batch 32 with TP2 (right pair), matched across methods. Horizontal lines denote full-attention accuracy. TPOT axes are logarithmic.

E.5 Complete reasoning workload

Table 4 reports accuracy, generation length, and generation time for Qwen3-8B on the complete AIME25 workload.

Table 4: **Complete AIME25 serving workload on one H20.** 64 runs each with 30 problems in one batch and a 38,912-token output budget. Generation time is the mean per pass; pass@1 averages all 64 passes.

Configuration	AIME25 pass@1 (%)	Mean gen. length	Gen. time (min)	Tokens/s per GPU
LASA	66.15	20,106	5.59	1793
Dense (full attention)	65.89	19,504	19.11	512

F Analysis of LASA's budget allocation

F.1 Variation across layers, heads, and queries

Top-k fixes a token count target, while top-p fixes a target for retained attention mass; LASA learns to vary both. We measure the selected KV token count and the attention mass covered by its selected tokens on assistant queries from OpenThoughts3 trajectories, using Qwen3-4B with block sizes 16 and 64 at $K = 15$. For variation along layer, KV head, or query, we fix the other two coordinates and compute the standard deviation (STD). For example, for the Query axis, we compute the STD across queries separately for each fixed layer and KV head pair within a trajectory. Table 5 reports the overall mean and summarizes the resulting STDs by their median, 90th percentile (P90), and maximum (Max).

Table 5: **Variation in selected KV token counts and retained attention mass.** Qwen3-4B, $K = 15$, block sizes (BS) 16 and 64. Overall mean averages all layers, KV heads, and queries. The remaining columns summarize the STDs along each axis with the other two coordinates fixed. For retained attention mass, means are percentages and standard deviations are percentage points (pp).

BS	Quantity	Overall mean	Axis	STD Median	STD P90	STD Max
16	KV token count	277.69	Layer	301.53	578.79	2705.23
			Head	198.28	513.60	4927.59
			Query	152.00	399.00	1610.42
16	Retained attention mass	71.76%	Layer	17.41	21.04	32.98
			Head	10.55	18.01	37.16
			Query	10.63	16.21	31.25
64	KV token count	890.03	Layer	768.56	1245.84	3342.31
			Head	544.80	1177.44	5109.39
			Query	459.84	995.28	2470.19
64	Retained attention mass	82.20%	Layer	13.97	18.31	29.94
			Head	6.74	14.27	40.83
			Query	7.80	15.09	27.60

F.2 Allocation across layers and heads

On Qwen3-4B, we measure each layer–KV-head cell's output sensitivity using the same procedure as in Figure 1*(b)*. We compare the resulting output FKL with the attention mass covered by LASA's selected blocks (block size 16, $K = 31$), averaging both quantities over assistant queries and OpenThoughts3 trajectories. Across all 288 cells, the two profiles have a Spearman correlation of 0.654: cells whose sparsification causes greater damage tend to receive higher retained mass. This supports preferential allocation to more sensitive cells.

F.3 Tracking demand along the sequence

We analyze a 16,594-token IMO-Bench (Luong et al., 2025) trajectory with Qwen3-4B. Dense top-p at $p = 0.85$ measures how many KV blocks are needed to cover 85% of the attention mass. We remove the slow positional trend and retain eleven sharp transitions for which token-level and block-level demand move in the same direction. Event selection uses only the dense measurements; Table 6 annotates the chain-of-thought activity at these boundaries.

We then teacher-force the same trajectory through LASA with 64-token blocks and a training budget target of 15 blocks. Dense demand and LASA allocation have Pearson and Spearman correlations of 0.866 and 0.849, and move in the same direction at all eleven transitions, with median layer–head directional agreement of 75.7%. LASA allocates more blocks at retrieval and re-planning events and fewer at local computation and presentation events.

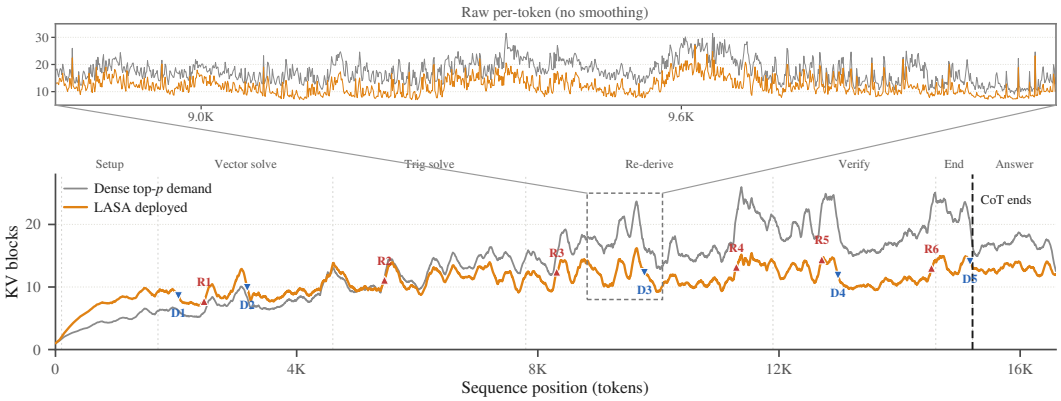

Figure 10: **Learned allocation and dense attention demand.** On the same fixed trajectory, dense top-p demand (gray) is the number of 64-token blocks needed to cover 85% of attention mass; LASA allocation (orange) is the number actually accessed. Both average over layers and KV heads. Event positions and labels are those of Table 6.

Table 6: **Chain-of-thought activity at sharp changes in dense attention demand.** Δtok. and Δblk. are relative changes between the 192-token windows before and after each boundary; Agr. is the fraction of the 36×8 layer/KV-head pairs moving in the same direction.

	Pos.	Δtok. (%)	Δblk. (%)	Agr. (%)	What the chain of thought does at this boundary
D1	2,040	-22.5	-16.7	78.5	Stops restating the geometric condition and instantiates the vectors $XP = (-1, p)$ and $XZ = (-1, \tan 50^\circ)$.
R1	2,465	$+59.6$	$+48.1$	78.1	Breaks off the algebraic expansion and reconsiders strategy: "this seems complicated … maybe there is a better way".
D2	3,181	-35.1	-30.9	83.3	Rote constant recall: $\tan 45^\circ = 1$, $\tan 60^\circ \approx 1.732$, hence $\tan 50^\circ \approx 1.1918$.
R2	5,457	$+44.6$	$+53.5$	76.7	Finishes the sub-result $PZ = 0.352a$ and replans against the global target YQ/ZP, noting that Q is still missing.
R3	8,313	$+52.8$	$+51.4$	88.5	Derives $PZ = 2a \tan 10^\circ$, retrieves the far earlier coordinate value $PZ \approx 0.238$, and detects the contradiction.
D3	9,768	-28.3	-27.0	77.1	Locally closed rewriting chain ("which is … which simplifies to …"), each line depending only on the previous one.
R4	11,293	$+45.6$	$+50.0$	86.5	Rejects the extraneous root $p = 1.743$ using the earlier global constraint $YZ = \tan 50^\circ \approx 1.1917$.
R5	12,710	$+39.6$	$+38.8$	78.8	Writes the boxed answer, then immediately restarts verification with a different method and rebuilds the coordinate setup.
D4	12,979	-30.1	-30.6	77.4	Substitutes decimals $\cos 50^\circ = 0.64279$ and $\cos 10^\circ = 0.98481$ and expands numerically.
R6	14,529	$+25.6$	$+29.0$	78.8	Reuses the earlier global constraint $YZ \approx 1.1917$ to validate $p \approx 0.841$ and reject $p \approx 1.717$ during verification.
D5	15,169	-34.5	-31.0	76.7	Leaves the exploratory chain of thought at </think> and writes the final solution.

G KV DENSITY MEASUREMENT

For sample i, decoding step t, layer ℓ, and KV head h, let $N_{\text{sel}}^{(i,t,\ell,h)}$ be the number of KV tokens actually attended and $N_{\text{causal}}^{(i,t,\ell,h)}$ the number of causally valid KV tokens. Within each benchmark run (one response per problem), we define the KV density as

$$d = \frac{\sum_{i,t,\ell,h} N_{\text{sel}}^{(i,t,\ell,h)}}{\sum_{i,t,\ell,h} N_{\text{causal}}^{(i,t,\ell,h)}}. \tag{56}$$

We report the arithmetic mean of KV density across runs.

Per-method measurement. For the adaptive-count methods, LASA and Twilight, a KV-access tracker in the serving runtime records the number of KV tokens attended at every decoding step, layer, and KV head for each sample. For the fixed-count methods, SeerAttention-R and LessIsMore, these counts are reconstructed from each sample's prompt and generation lengths together with the method's configured token budget and dense-layer pattern. KV accesses from layers that remain dense and tokens that a method always attends are included in the selected count.

Density and absolute KV reads. KV density depends on both the selected-token count and the causally available context along each generated trajectory. For LASA-K15 with block size 64, GPQA and MATH-500 have higher densities despite lower average KV reads per attention row, owing to their shorter trajectories. Density and average attended tokens therefore describe relative sparsity and absolute read budgets, respectively.