ACCELERATING MIXTURE OF BLOCK ATTENTION VIA DYNAMIC ROUTING, KV-CENTRIC EXECUTION AND RUNTIME PIPELINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Mixture of Block Attention (MoBA) reduces long-context computation by selecting key-value (KV) blocks for each query. FlashMoBA implements MoBA efficiently but retains a fixed Top-k budget and query-centric schedule. Fixed Top-k applies one budget across different score distributions and couples routing to task granularity; query-range ownership fragments K/V reuse when several ranges select the same column. We present **LightningMoBA**, which co-designs selection and execution through dynamic routing, KV-centric ownership, and runtime pipelining. Dynamic routing thresholds MoBA's per-query block scores and directly emits variable-cardinality sparse metadata. KV-centric ownership groups selected rows by $(\text{query head}, \text{KV block})$ column, splits high-fanout columns into bounded tasks, and reuses each task's loaded KV tile. On Hopper, pipelining overlaps regular K/V transfers and irregular query gathers with Tensor Core computation. Together these mechanisms decouple retention from GPU scheduling granularity. At matched Top-16 retention across five checkpoints, **LightningMoBA** accelerates end-to-end generation by 1.20 to 1.60$\times$ while raising mean RULER score from 74.42 to 75.28. In controlled two-model replays, complete device-path speedup grows from 1.56 to 1.65$\times$ at Top-16 to 2.16 to 2.56$\times$ at Top-32, while routing and CSC construction reach 6.99 to 11.25$\times$ at Top-64. The code of **LightningMoBA** is available at `anonymous.4open.science/r/LightningMoba-A090`.

1 INTRODUCTION

Dense self-attention evaluates all pairwise query-key interactions in a sequence, whereas sparse attention reduces this work by routing each query to only a subset of keys. On GPUs, however, fewer interactions do not automatically yield proportionally lower latency: score selection, sparse-index construction, irregular data movement, and load imbalance can offset the savings in arithmetic. An efficient sparse attention method must therefore jointly optimize the routing decision and the GPU workload that follows from it.

Mixture of Block Attention (MoBA) (Lu et al., 2025) routes at block granularity: it scores each KV block against the query, retains a fixed number k of blocks, and computes softmax attention over their tokens. FlashMoBA (Xiao et al., 2025) implements this efficiently, maintaining an ordered Top-k candidate set per query and assigning each CTA a logical query range whose selected rows are gathered into dense Tensor Core tiles.

Nevertheless, two limitations remain. Fixed Top-k routing assigns the same block budget to queries with different score distributions, retaining weakly related blocks for concentrated scores while discarding relevant blocks for diffuse ones. Maintaining the ordered candidate set also adds routing latency. FlashMoBA organizes metadata by KV block but assigns CTA ownership by logical query range, so several CTAs may claim one $(\text{query head}, \text{KV block})$ column and leave cross-CTA reuse to the cache hierarchy. Coupling query-group size to k also makes a routing parameter determine scheduling granularity.

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.

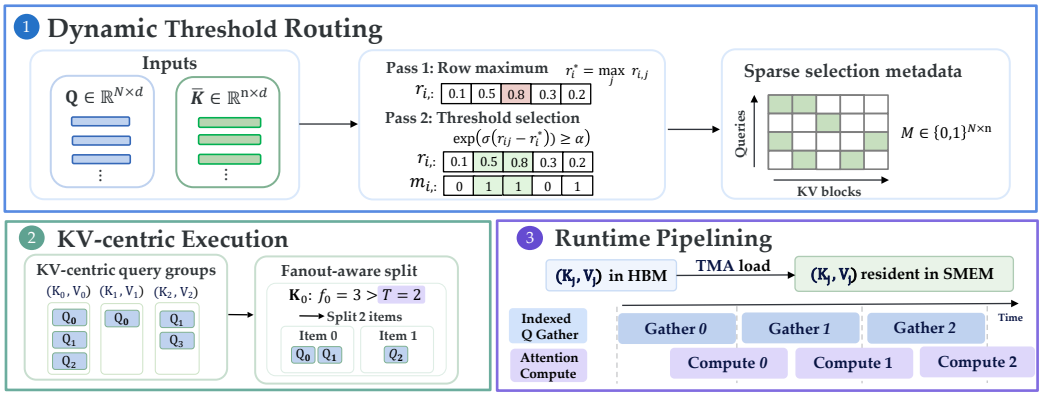

Figure 1: Overview of **LightningMoBA**. Dynamic threshold routing adapts the retained block count per query and constructs sparse metadata directly; KV-centric execution organizes selected rows by (query head, KV block) column and splits high-fanout columns into bounded work items; runtime pipelining keeps each work unit's KV tile resident while its query gathers overlap Tensor Core work.

We present **LightningMoBA**, which preserves MoBA's parameter-free query-block scoring rule and computes standard softmax attention over the retained tokens, but reorganizes routing and execution along three axes (Fig. 1).

Our contributions are as follows:

- **Dynamic routing. LightningMoBA** replaces the fixed per-query Top-k budget with a threshold-based variable retention rule and emits sparse metadata during score evaluation, without an ordered candidate set or a learned auxiliary indexer.
- **KV-centric execution and runtime pipelining.** Work is organized by columns indexed by query head and KV block, so each unit reuses its loaded KV tile across the selected rows assigned to it. Fanout-aware partitioning bounds the rows per task, while an asymmetric Hopper pipeline overlaps TMA-based K/V transfers and irregular query gathers with Tensor Core computation.
- **Evaluation and analysis.** At matched Top-16 retention across five checkpoints, **LightningMoBA** accelerates end-to-end generation by 1.20 to 1.60× while increasing mean RULER score from 74.42 to 75.28. Controlled complete-device-path replays on two models reach 2.16 to 2.56× speedup at Top-32, while routing and CSC construction reach 6.99 to 11.25× at Top-64.

2 Related Work

The efficiency of sparse attention is determined jointly by its selection rule and by the execution path that consumes the resulting sparse pattern.

2.1 Scoring, Granularity, and Budgeting in Sparse Attention

Prescribed patterns (Beltagy et al., 2020; Zaheer et al., 2020) fix connectivity by design; input-derived patterns (Kitaev et al., 2020; Roy et al., 2021) cluster or hash tokens, and SSA (Shen et al., 2025) aligns sparse- with full-attention representations during training. Within content-dependent selection, granularity, scoring, and budget are separate choices: MoBA (Lu et al., 2025) scores blocks with mean-pooled keys under a fixed per-query budget; NSA (Yuan et al., 2025b) combines compressed, selected, and sliding-window branches; DSA (DeepSeek-AI, 2025) and MSA (Lai et al., 2026) use learned indexers, the latter per GQA group; HiLS (Hu et al., 2026), HySparse (Gao et al., 2026) and HySparse2 (Wei et al., 2026) learn summaries or derive routing from a preceding full-attention layer. Candidate discovery is amortized along different axes (Qiu et al., 2026; Liu et al., 2026; Bai et al., 2026; DeepSeek-AI, 2026), and adaptive budgets also appear (Yuan et al., 2025a; Wu et al., 2026). Thresholding avoids sorting: FlashPrefill (Fan et al., 2026) shares one block-level cutoff per query block, while **LightningMoBA** thresholds MoBA's per-query block scores,

so retention differs across individual queries. Retrofitting pretrained checkpoints through a learned selector is another route (Zhou et al., 2026; Ahmad & Yun, 2026). **LightningMoBA** changes neither weights nor scorer; it replaces only the rule that maps scores to a retained set and co-designs the representation and execution of the resulting variable pattern.

2.2 Co-Design of Sparse Attention and Systems

The cost of a pattern depends on how its interactions are represented and scheduled: FlashMoBA (Xiao et al., 2025) assigns CTA ownership by logical query range; FAST-Prefill (Jayanth & Prasanna, 2026) co-designs index generation and caching for an FPGA. KV-outer execution is established (Lu et al., 2025; Yan et al., 2025; Lai et al., 2026): fixed Top-k fixes the query-side row degree but leaves the KV-side column degree varying, so reverse indices and splitting popular columns are existing techniques.

LightningMoBA extends KV-outer execution to routing graphs with variable query-side degree: under per-query thresholding, both the row degree of each query and the total number of retained pairs are set at run time. It therefore co-designs the routing representation and execution interface, decoupling the per-query budget from the kernel's logical task size and dividing high-fanout columns into tasks with bounded query rows.

2.3 High-Performance Kernels on Modern GPUs

FlashAttention (Dao et al., 2022; Dao, 2023) tiles attention with online softmax; FlashAttention-3 (Shah et al., 2024) and FlashAttention-4 (Zadouri et al., 2026) add warp-specialized producer-consumer pipelines on Hopper and Blackwell. PagedAttention (Kwon et al., 2023) manages KV-cache storage at the serving level, orthogonal to the organization of data movement inside one attention call. These kernels show that tile shape, work ownership, and operand access determine what can overlap. **LightningMoBA** applies these principles to a KV-centric sparse workload with asymmetric operand access: contiguous K/V tiles move through TMA while indexed query rows move through a dedicated producer path, both overlapping Tensor Core computation, with tile shapes and warp roles independent of how many blocks each query retains.

3 Methodology

3.1 Preliminaries: MoBA and FlashMoBA

We consider causal prefill over N tokens with head dimension d. Batch and head indices are omitted unless needed, and g query heads share each key/value head under GQA. We write $\mathbf{q}_i$, $\mathbf{k}_t$ and $\mathbf{v}_t$ for a token's query, key and value vectors, reserving the unsubscripted k for the number of blocks a query retains, with bold type keeping the two apart.

Mixture of Block Attention (MoBA) (Lu et al., 2025) partitions the sequence into $n = \lceil N/B \rceil$ blocks of at most B tokens; the keys of block j form $\mathbf{K}_j \in \mathbb{R}^{B\times d}$. For each query $\mathbf{q}_i$, MoBA scores each block against the query through the block's mean-pooled key and keeps the best k causally available blocks, including the query's own block $c_i = \lceil i/B \rceil$:

$$\tilde{\mathbf{k}}_j = \frac{1}{|\mathbf{K}_j|}\sum_{\mathbf{k}\in\mathbf{K}_j}\mathbf{k}, \qquad r_{ij} = \mathbf{q}_i^\top \tilde{\mathbf{k}}_j, \qquad \mathcal{S}_i = \{c_i\} \cup \operatorname*{TopK}_{\min\{k-1,\, c_i-1\}} \big(\{r_{ij}\}_{1\le j<c_i}\big). \tag{1}$$

Pooling is used only for routing; attention is computed over all tokens in the retained blocks. We write $Z = \sum_i |\mathcal{S}_i|$ for the number of selected query-block pairs, i.e., the sparse pattern executed by the attention stage. Under Eq. (1) each query retains $1 + \min\{k-1,\, c_i - 1\} \le k$ blocks, so the budget bounds the pattern size as well: $Z \le Nk$. Sec. 3.2 replaces the fixed budget by a score criterion, after which $|\mathcal{S}_i|$ and Z are set by the scores rather than by k. Sec. A.1 gives the corresponding worst-case bound.

FlashMoBA (Xiao et al., 2025) provides an efficient GPU implementation of MoBA. It evaluates the fixed Top-k routing rule, constructs the corresponding sparse metadata, and executes block-sparse attention over the selected query-block pairs. Its reference router maintains a running Top-k

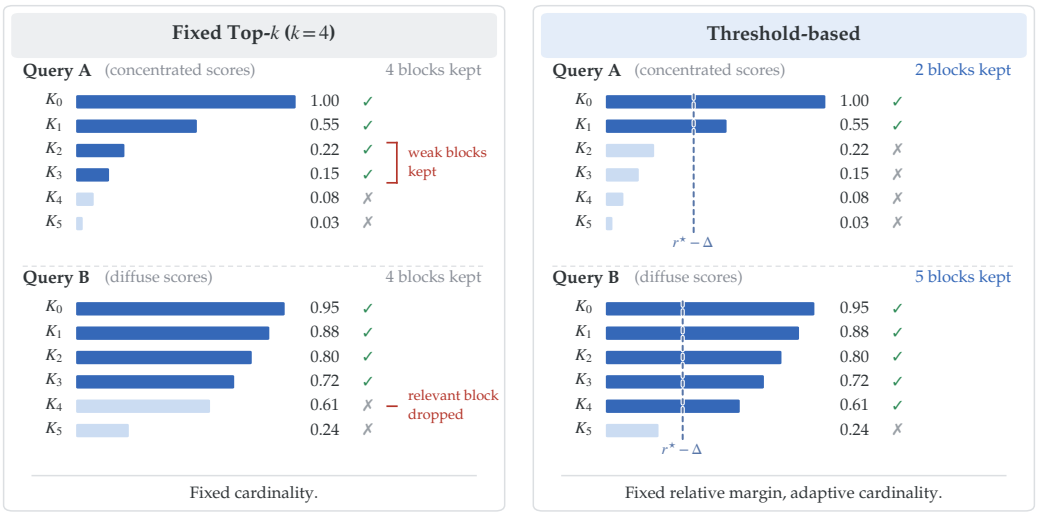

Figure 2: Fixed Top-k and threshold-based selection applied to the same block scores of two queries. A fixed budget keeps the same number of blocks for both, retaining weakly scored blocks when the scores are concentrated and discarding a comparably scored one when they are diffuse. The threshold rule keeps every block within a fixed margin $\Delta = -\log \alpha/\sigma$ of each query's strongest preceding-block score, so the retained count follows the score distribution. Compulsory blocks are omitted so that the comparison isolates the selection rule.

candidate set for each query, while the attention kernel organizes execution around logical query ranges and gathers selected query rows into dense Tensor Core tiles.

We retain MoBA's query-block score based on mean-pooled keys and its standard softmax attention over retained tokens, while redesigning block selection and the sparse workload that follows.

3.2 Dynamic Routing

MoBA assigns the same routing budget k to every query, regardless of how its block scores are distributed. This fixed-cardinality constraint need not match the amount of relevant context available to each query: for a concentrated score distribution, selecting exactly k blocks may retain weakly related blocks only to satisfy the budget, whereas for a more diffuse distribution the same budget may discard additional blocks with non-negligible scores. Top-k therefore fixes how many blocks are retained rather than what score level is sufficient for retention.

Inspired by FlashPrefill (Fan et al., 2026), we instead apply a query-local threshold to MoBA's original block scores. The scoring function is unchanged; only the rule that converts scores into a retained set is replaced. As illustrated in Fig. 2, this fixes a relative score criterion while allowing the retained cardinality to adapt independently for each query.

3.2.1 Threshold-Based Block Selection

Query $\mathbf{q}_i$ may attend to the causally admissible blocks $\mathcal{C}_i = \{j : 1 \le j \le c_i\}$. We retain unconditionally up to n_{sink} initial sink blocks and a local window of $w \ge 1$ blocks ending at the query's own block,

$$\mathcal{F}_i = \{1, \dots, \min(n_{\text{sink}}, c_i)\} \cup \{\max(1, c_i - w + 1), \dots, c_i\} \subseteq \mathcal{C}_i. \tag{2}$$

Let $\sigma > 0$ denote the softmax scale and $r_i^\star = \max_{1 \le j < c_i} r_{ij}$ the largest score over complete blocks preceding the query's own block. If $c_i = 1$, set $r_i^\star = 0$; there are no optional blocks. We retain

$$\mathcal{S}_i = \mathcal{F}_i \cup \Big\{ j \in \{1, \dots, c_i - 1\} \setminus \mathcal{F}_i : \exp\big(\sigma(r_{ij} - r_i^\star)\big) \ge \alpha \Big\}, \qquad \alpha \in (0, 1]. \tag{3}$$

Equivalently, taking logarithms on both sides of the inequality in Eq. (3) and dividing by σ, an optional block is retained when

$$r_i^\star - r_{ij} \le -\frac{\log \alpha}{\sigma} =: \Delta, \tag{4}$$

i.e., when its score lies within a fixed margin Δ of the strongest preceding block for that query. The decision therefore requires **no softmax normalization**. Unlike fixed-budget routing, which fixes cardinality and induces a query-dependent score cutoff, our rule fixes a query-relative score criterion and lets $|\mathcal{S}_i|$ vary across queries and heads.

The current block is retained with token-level causal masking, but its pooled key is excluded from the reference maximum because it can contain future tokens. Preceding forced blocks still contribute to the maximum. The rule depends only on score differences relative to $r_i^\star$, so it is invariant to an additive shift of the preceding-block scores for a query.

The router emits this representation with a two-pass streaming construction and a prefix-sum scatter into compact CSC metadata; we defer those details to Appendix E and only use their output here.

3.3 KV-Centric Execution

The sparse pattern produced by routing is content-dependent, whether or not its per-query cardinality is fixed. FlashMoBA assigns work by logical query ranges, so queries in different ranges that select the same KV block are handled by different CTAs, causing repeated KV loads and fragmented reuse across query-owned tasks. This limits the useful work each loaded KV tile can amortize.

We instead organize the sparse pattern by $(\text{query head}, \text{KV block})$ columns and assign execution around these columns (Sec. 2.2). Queries selecting the same block under a query head are therefore grouped under the same KV-owned column, allowing each work unit to reuse its loaded KV tile across its assigned rows. Because a column's task size is proportional to its fanout, assigning an entire column to one CTA can leave a long-running tail. We therefore separate two decisions: *ownership*, which determines which KV column a task belongs to, and *task size*, which partitions high-fanout columns into bounded work units. Alg. 1 illustrates the KV-centric ownership pattern before fanout-aware splitting: execution is organized by KV columns rather than query ranges. Sec. 3.3.2 then partitions high-fanout columns into bounded work items without changing this ownership rule.

3.3.1 KV-Centric Ownership

In FlashMoBA a CTA, scheduled as a unit onto an SM, owns a band of M consecutive query rows and walks the blocks those rows selected, gathering only that band's rows so that routing induces no masked lanes. Write

$$\mathcal{Q}_{h,j} = \{\, i : j \in \mathcal{S}_{h,i} \,\}, \qquad \lambda_{h,j}(M) = \big|\{\, b : \mathcal{Q}_{h,j} \cap [bM, (b{+}1)M) \neq \emptyset \,\}\big|, \tag{5}$$

for the queries that selected column (h, j) and the number of bands claiming it, where a column is block $\mathcal{B}_j$ as seen by query head h. With C selected columns, the mean over columns

$$\Lambda(M) = \frac{1}{C}\sum_{h,j} \lambda_{h,j}(M) \;=\; \frac{|\mathcal{V}(M)|}{C}, \qquad \mu(M) = \frac{Z}{|\mathcal{V}(M)|} = \frac{\bar{f}}{\Lambda(M)}, \qquad \bar{f} = \frac{Z}{C}, \tag{6}$$

counts how many times a column is claimed, where $\mathcal{V}(M)$ is the set of (band, column) pairs carrying work and $\bar{f}$ is the mean fanout. These two quantities determine the costs of band ownership. A column's K/V is requested $\lambda_{h,j}(M)$ times rather than once. Since a physical KV block may appear in up to g query-head columns, band fragmentation compounds with GQA sharing: $\Lambda(M)$ measures the mean multiplicity at the column level, while the request count of a physical block is the sum of the multiplicities of its participating columns, and how many of those requests reach HBM depends on cache behaviour. Independently, each request exposes only $\mu(M)$ selected rows with which to fill an MMA tile that holds m_{MMA} rows. The two are not independent: Z and C are fixed by the routing pattern, and M changes only $|\mathcal{V}(M)|$, so dividing numerator and denominator by C gives

$$\mu(M) = \frac{Z}{|\mathcal{V}(M)|} = \frac{Z/C}{|\mathcal{V}(M)|/C} = \frac{\bar{f}}{\Lambda(M)}.$$

At fixed $\bar{f}$, raising Λ multiplies the requests per column and divides the work each request carries by the same factor. The arithmetic is unchanged: fragmenting leaves the routing pattern untouched, so the same Z query-block pairs must still be evaluated, and C, $\bar{f}$ and the $\mathcal{O}(ZBd)$ term of Sec. A.1 are unaffected. What changes is how often the K/V operand is fetched and how much of each MMA tile is filled with work that is actually useful.

Within fixed query-band ownership, increasing M is the direct way to reduce this fragmentation (shared memory caps it, but the cap is not what makes M the wrong knob): a band claims a column alone only if it contains every query that selected it. The resulting necessary condition is stated below, with $s_{h,j} = \max \mathcal{Q}_{h,j} - \min \mathcal{Q}_{h,j} + 1$ denoting the query span of a column.

Lemma 3.1 (Span bound for unit multiplicity)**.** *Unit multiplicity requires*

$$\Lambda(M) = 1 \quad \Longrightarrow \quad M \;\geq\; \max_{h,j} s_{h,j}. \tag{7}$$

The converse does not hold.

The requirement is not an artifact of how M is chosen. Band ownership groups queries by position, whereas routing groups them by content, so the two partitions coincide only by accident; a column selected by both an early and a late query spans the sequence, and no band shorter than that span can own it alone. This is a property of content-based routing rather than of thresholding, so it also applies to a fixed budget. It explains why query-band ownership works for local or structured sparsity, where each block's consumers are contiguous by construction, but fragments content-dependent patterns. Fig. 8 evaluates Eq. (6) on captured routing patterns; we use it to quantify how much fragmentation remains at practical values of M.

Making the selected column (h, j) the ownership domain removes band fragmentation: all selected rows of a column are grouped under the same KV owner before tasks are formed, and M leaves the ownership decision entirely. This gives unit multiplicity at the column level before deliberate fanout splitting, $\lambda_{h,j} = 1$, and makes every selected row available for tile formation. It does not fix how many CTAs serve the column: Sec. 3.3.2 partitions it into several work units that each load the tile for their assigned rows, emit a partial output and local log-sum-exp, and are combined by the online-softmax merge, which is exact in real arithmetic.

3.3.2 Fanout-Aware Workload Decomposition

KV-centric ownership removes band fragmentation by grouping all consumers of a column under one ownership domain. A column's work grows with its fanout $f_{h,j} = |\mathcal{Q}_{h,j}|$, the number of queries selecting it. A task covering the column serially processes

$$w_{h,j} = \left\lceil f_{h,j} / m_{\mathrm{MMA}} \right\rceil \tag{8}$$

MMA tiles, so fanout becomes task size and dispersion in fanout becomes dispersion in task duration. Fixed Top-k bounds how many blocks a query keeps, not how many queries keep a block, so this dispersion is a property of the routing rather than of thresholding.

Dispersion in task size affects scheduling differently from total work. Lemma E.1 in Appendix E.1 specializes the classical list-scheduling bound to these work units: Eq. (11) separates an aggregate throughput term W/P from a largest-work-item term $\max_{h,j} w_{h,j}$. Fanout dispersion raises the latter. Additional hardware parallelism reduces the first term but cannot subdivide the largest item, and tensor parallelism sharpens this tail because a synchronized step waits for its slowest device.

Fig. 6 in Appendix F characterizes the source and control of that tail on captured routing patterns. Panel (a) reports the mean and selected column-fanout quantiles for one high-tail prefill trace, normalized by its mean of 2,031 query rows. The median is only $0.064\times$ the mean, whereas P99 and the maximum reach $28.1\times$ and $64.5\times$ the mean. This dispersion raises the largest-work-item term in Eq. (11). Panel (b) shows the corresponding control trade-off: decreasing the task-size cap T reduces the largest work item while increasing load multiplicity. Both panels use the same actual all-pair columns, including compulsory sink and local selections; they characterize execution work rather than content-selected fanout alone. Sec. B.1.4 separately reports the optional-only tail.

Fanout-aware splitting To prevent a high-fanout column from defining the critical path, we decouple ownership from task size. Ownership remains KV-centric; only the granularity of individual work units changes. Capping the number of selected query rows a unit may claim, we partition each column into work units $(h, j, i_{\text{begin}}, i_{\text{count}})$ subject to $i_{\text{count}} \leq T$. The cap is on the count of selected rows in the column's compact row list, not on the sequence positions those rows occupy, so it is unrelated to the span $s_{h,j}$ of Sec. 3.3.1. A count cap suffices because the metadata of Sec. E.1 lists a column's selected rows contiguously even when those queries are scattered over the input sequence.

The exchange is explicit in both directions: $\max_{h,j} w_{h,j}$ is bounded by $\lceil T/m_{\mathrm{MMA}} \rceil$, while a column of fanout $f_{h,j}$ becomes $\lceil f_{h,j}/T \rceil$ units that each independently load the column's K/V operand and produce the outputs for their assigned query rows, raising the multiplicity to

$$\Lambda_T = \frac{1}{C} \sum_{h,j} \lceil f_{h,j}/T \rceil \; < \; 1 + \bar{f}/T. \tag{9}$$

Fanout-aware splitting therefore trades within-unit KV reuse for finer-grained parallelism and improved load distribution, rather than obtaining them alongside it, and both sides of the trade are computable from the routing metadata before the attention work is launched: lowering T reduces the largest-task term in Eq. (11) while increasing Λ_T. Taking T to be a multiple of m_{MMA} preserves the total number of MMA row tiles for each column while redistributing those tiles among independently schedulable work units. Sec. B.1 sweeps T and reports the setting we use.

3.4 Runtime Pipelining

KV-centric execution creates asymmetric operand access: K/V tiles remain contiguous, whereas Q rows must be gathered from the routing metadata. We assign regular K/V transfers to the Tensor Memory Accelerator (TMA) and irregular Q gathers to a producer path, allowing both paths to be staged concurrently with Tensor Core computation. The implementation targets Hopper's TMA, WGMMA and warp-specialized execution model; Sec. B.3 discusses portability.

4 Experiments

4.1 Experimental Setup

The evaluation separately measures execution cost and model quality under two training-free comparisons. Model configurations, benchmark settings, and timing scopes are specified below and in the corresponding result sections.

Models The evaluation uses five checkpoints: Qwen3-4B-Instruct-2507 (Qwen3-4B), Qwen3-235B-A22B, Qwen3-32B-YaRN4, Llama-3.1-8B-Instruct (Llama-3.1-8B), and GQA-30B-A3B, an internal GQA model (30B total, 3B active) whose architecture follows Qwen3 and is trained from scratch. These span dense and mixture-of-experts architectures as well as different GQA grouping ratios; checkpoint sources are listed with the full setup in Appendix D. Each reported configuration identifies the checkpoint revision, tested context lengths, head configuration, and the layers to which sparse attention is applied.

We compare against an optimized implementation of the same block-attention method; its baseline definition, training-free protocol and model sources are in Appendix D.

4.2 Main Results

4.2.1 End-to-End Quality and Latency

Table 1 reports the end-to-end comparison on RULER. The baseline is FlashMoBA with fixed Top-16 routing, one sink block and a two-block local window; **LightningMoBA** uses dynamic threshold prefill with the same compulsory blocks and a matched average Top-16 retention budget. Both variants use dense decode with CUDA Graph enabled. Every entry aggregates the full 6,500-example RULER suite at tensor parallelism eight and batch size one, and times are generation-loop wall times from the first request to the completion of the last, excluding model loading, data preparation and result serialization. The comparison is made at Top-16 because that is the routing budget the deployed baseline uses, and neither method is given an increased retention budget here. How the two routers behave at larger budgets is a separate question with its own timing scope, reported in Sec. B.1.

Table 1: End-to-end RULER results over the full 6,500-example suite. Each cell reports Base / Ours; wall times are in minutes with speedup in parentheses. Averages are over the six context lengths for each model, and the better value is bolded.

Length	Qwen3-4B	GQA-30B-A3B	Qwen3-235B-A22B	Llama-3.1-8B	Qwen3-32B-YaRN4
			(a) RULER score: Base / Ours		
4K	93.75 / 94.29	94.40 / 94.10	95.26 / 94.86	95.27 / 94.96	89.15 / 88.98
8K	85.98 / 91.87	88.20 / 89.15	94.27 / 94.65	91.82 / 93.54	85.90 / 87.80
16K	79.45 / 87.18	82.47 / 80.88	89.18 / 93.26	83.62 / 87.25	74.97 / 74.11
32K	69.98 / 79.93	74.09 / 70.82	79.72 / 84.03	70.73 / 77.76	67.44 / 69.96
64K	59.08 / 53.41	57.14 / 55.72	70.87 / 68.78	61.05 / 57.57	61.58 / 56.32
128K	46.35 / 46.51	34.92 / 35.40	63.78 / 62.90	41.45 / 41.75	50.85 / 50.54
Average	72.43 / **75.53**	**71.87** / 71.01	82.18 / **83.08**	73.99 / **75.47**	**71.65** / 71.29
			(b) Wall time: Base / Ours (speedup)		
4K	145.9 / 60.5 (2.41×)	119.1 / 36.8 (3.24×)	325.6 / 123.9 (2.63×)	135.6 / 70.9 (1.91×)	270.6 / 171.6 (1.58×)
8K	153.1 / 69.5 (2.20×)	118.7 / 41.1 (2.89×)	368.6 / 150.8 (2.44×)	142.7 / 75.6 (1.89×)	290.3 / 195.0 (1.49×)
16K	165.0 / 87.7 (1.88×)	116.5 / 59.3 (1.96×)	458.2 / 206.0 (2.22×)	150.7 / 88.2 (1.71×)	317.8 / 240.0 (1.32×)
32K	174.8 / 114.3 (1.53×)	146.2 / 103.9 (1.41×)	579.3 / 341.2 (1.70×)	166.1 / 117.0 (1.42×)	394.1 / 327.7 (1.20×)
64K	206.7 / 153.1 (1.35×)	178.8 / 122.5 (1.46×)	785.0 / 537.1 (1.46×)	192.8 / 158.3 (1.22×)	504.1 / 446.8 (1.13×)
128K	284.2 / 251.6 (1.13×)	293.0 / 253.1 (1.15×)	1180.1 / 957.4 (1.19×)	277.9 / 241.3 (1.15×)	812.3 / 770.9 (1.05×)
Average	188.3 / **122.8** (**1.53**×)	162.1 / **102.8** (**1.58**×)	616.1 / **386.1** (**1.60**×)	177.6 / **125.2** (**1.42**×)	431.5 / **358.7** (**1.20**×)

Key result. At the Top-16 budget used by the deployed baseline, **LightningMoBA** shortens the total generation-loop time of all five models, with per-model Average speedups of 1.20 to 1.60× (Table 1). Mean RULER score rises for three models and falls for two, by 0.86 points on GQA-30B-A3B and 0.36 points on Qwen3-32B-YaRN4. Quality also varies with length: the largest gains occur between 8K and 32K, the two methods stay within one point of each other at 4K and 128K, and at 64K **LightningMoBA** scores 1.42 to 5.67 points lower on all five models.

4.2.2 Cumulative Implementation Ablation

Adding one component at a time to FlashMoBA Top-k gives the cumulative device-path latency of Table 2, each variant timed on the same captured BF16 Q/K/V with routing, CSC construction, attention and merge included. The budget exposes the scaling difference: the final path is 1.56 to 1.65× faster than FlashMoBA at Top-16 and 2.16 to 2.56× at Top-32, because ordered Top-k maintenance grows with k while the threshold stage applies one scalar test per recomputed score.

Table 2: Cumulative device-path latency (ms): FlashMoBA, specialized unsplit KV kernel, split-4096 worklist, SM90 pipeline, and matched-density threshold routing. The first four variants share saved Top-k CSC metadata. Speedup compares the final variant with FlashMoBA.

Model	Length	k	FlashMoBA	+KV	+Split	+Pipeline	+Threshold	Speedup
Qwen3-4B	32K	16	25.529	22.091	20.564	16.520	15.632	1.63×
Qwen3-4B	128K	16	137.243	104.740	97.489	81.813	83.284	1.65×
GQA-30B-A3B	32K	16	24.543	21.574	19.961	15.866	15.786	1.56×
GQA-30B-A3B	128K	16	131.002	102.606	95.244	79.751	82.249	1.59×
Qwen3-4B	32K	32	56.083	48.102	47.797	37.821	24.291	2.31×
Qwen3-4B	128K	32	312.387	262.775	261.526	221.020	121.845	2.56×
GQA-30B-A3B	32K	32	52.660	45.178	45.107	35.154	24.380	2.16×
GQA-30B-A3B	128K	32	287.236	243.851	242.549	202.464	120.512	2.38×

4.2.3 Routing Cost Across Budgets

Isolating routing cost shows why the budget matters. Against FlashMoBA's ordered Top-k candidate set at matched realized pair counts, threshold routing is faster by 3.15 to 4.13× at $k = 32$ and 6.99 to 11.25× at $k = 64$, and the gap widens with k (Table 3).

4.2.4 Ownership Ablation

On identical Top-16 metadata and captured tensors, column ownership cuts attention-stage latency by 18.9% at 32K and 33.8% at 128K (16.9% and 31.0% on GQA-30B-A3B) by removing repeated K/V loads (Table 4). Validation replays of the same routing give maximum absolute and relative

Table 3: Complete routing sweep including CSC construction. Each group reports Top-k and threshold latency (ms) with their ratio; each threshold setting matches the valid CSC pair count within 0.05%. Timings use CUDA Graph replay.

		$k = 16$			$k = 32$			$k = 64$		
Model	Length	Top-k	Thresh.	Speedup	Top-k	Thresh.	Speedup	Top-k	Thresh.	Speedup
Qwen3-4B	16K	1.86	1.40	1.33×	6.32	1.91	3.31×	20.63	2.88	7.17×
Qwen3-4B	32K	4.06	3.39	1.19×	17.93	4.34	4.13×	64.01	6.02	10.64×
Qwen3-4B	64K	10.25	10.36	0.99×	47.47	12.65	3.75×	177.28	17.26	10.27×
Qwen3-4B	128K	30.70	32.28	0.95×	135.33	36.81	3.68×	514.42	45.74	11.25×
GQA-30B-A3B	16K	1.66	1.35	1.23×	5.58	1.77	3.15×	19.22	2.75	6.99×
GQA-30B-A3B	32K	3.52	3.26	1.08×	15.28	4.12	3.71×	54.65	6.22	8.79×
GQA-30B-A3B	64K	10.01	9.68	1.03×	44.82	11.62	3.86×	159.34	15.99	9.96×
GQA-30B-A3B	128K	28.57	31.85	0.90×	116.96	35.48	3.30×	416.01	44.56	9.34×

L_2 output errors below 1.6×10^{-2} and 9.4×10^{-4}, matching the level reported in Sec. B.1.1. KV-centric units serve whole columns, so each unit's task size follows its column's fanout, and fanout is strongly skewed: the largest column carries 64.5× the mean and the median only 0.064× (Fig. 3). Sec. B.1.4 bounds the resulting tail by splitting.

Table 4: Ownership ablation on identical Top-16 routing metadata and complete 32/8-head model tensors; latencies are attention-stage milliseconds. $\Lambda(M)$ is the mean number of query-centric units touching a selected (h, j) column; $\mu(M)$ and μ_{KV} are the mean selected rows served per query-centric and KV-centric unit.

Model	Length	Query	KV unsplit	Speedup	$\Lambda(M)$	$\mu(M)$	μ_{KV}
Qwen3-4B	32K	20.056	16.263	1.23×	13.66	145.50	1988.00
Qwen3-4B	128K	100.843	66.727	1.51×	20.91	97.24	2033.00
GQA-30B-A3B	32K	19.644	16.316	1.20×	10.88	182.72	1988.00
GQA-30B-A3B	128K	96.207	66.380	1.45×	16.03	126.79	2032.99

4.2.5 SM90 Query-Gather Pipelining

With Top-16 routing, $T = 4096$ partitioning, and the same TMA/WGMMA configuration and shared-memory allocation, the two-stage query-gather pipeline yields 2.53 to 2.65× speedup and zero measured output error on 8,192 sampled CSC rows per case (Table 5).

Model	Len.	Serial	Piped	Speedup
Qwen3-4B	32K	17.642	6.868	2.57×
Qwen3-4B	128K	71.412	26.951	2.65×
GQA-30B-A3B	32K	17.053	6.741	2.53×
GQA-30B-A3B	128K	69.586	26.806	2.60×

Table 5: SM90 query-gather pipeline ablation with the same Top-16 routing, $T = 4096$ work partition and CUDA Graph timing scope; latencies are milliseconds. "Serial" denotes serial Q gathering in the same kernel.

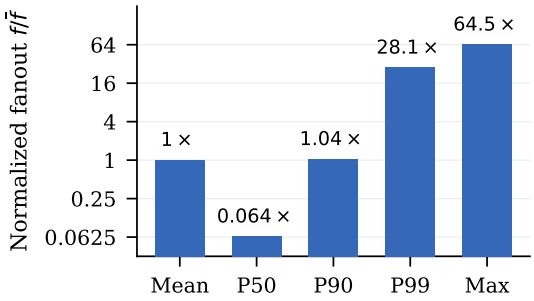

Figure 3: Column fanout imbalance across 4,096 nonempty query-head/KV-block columns of a high-tail GQA-30B-A3B TP8 RULER-128K prefill trace, normalized by the mean.

5 Conclusion

LightningMoBA keeps MoBA's scoring function and retained-token softmax while decoupling retention from task formation: threshold routing adapts retention to each score distribution, KV-centric ownership and fanout-aware splitting form reusable bounded tasks, and Hopper pipelining overlaps K/V movement, query gathering and Tensor Core work. At matched Top-16 retention these mechanisms accelerate end-to-end generation by 1.20 to 1.60× across five checkpoints while increasing mean RULER score from 74.42 to 75.28; in controlled replays, complete device-path speedup grows to 2.16 to 2.56× at Top-32, while routing and CSC construction reach 6.99 to 11.25× at Top-64.

AI USE STATEMENT

Generative AI tools were used in three ways, each reviewed by the authors. First, to aid and polish the writing: phrasing, structure and typographic consistency of the manuscript were edited with AI assistance, and all technical content was checked by the authors. Second, for retrieval and discovery: AI-assisted search helped identify and locate prior work on sparse attention and GPU attention kernels, and every citation was verified against the original paper. Third, for research ideation and execution: AI tools were used to discuss presentation and analysis choices (for example, which results belong in the main text and how each ablation isolates one design axis), and to assist with manuscript preparation tasks such as compiling, checking references and formatting tables and figures. All experiments, measurements and reported numbers were produced and verified by the authors, who take full responsibility for the content of this submission.

ETHICS STATEMENT

The study is algorithmic and systems-level: it changes the selection rule, task formation and data movement of block-sparse attention and reports measurements on public benchmarks. No human subjects, human annotations, or personally identifiable or sensitive data are involved, and the study does not target a high-risk application domain. We foresee no immediate ethical concerns arising from this line of systems work.

REPRODUCIBILITY STATEMENT

The threshold rule, ownership model, task decomposition and pipeline schedule are specified in Section 3, with the streaming metadata construction and the forward-pass pseudocode deferred to Appendix E and Algorithm 1. Section 4.1 lists the checkpoints, context lengths and benchmark; the end-to-end comparison and the ablations isolating each axis are in Sections 4.2, 4.2.2, 4.2.4 and 4.2.5; the appendix reports the cost breakdown and profiler counters. Proofs of both lemmas appear in Appendix G. Four of the five checkpoints are publicly released; GQA-30B-A3B is an internal model that cannot be distributed.

REFERENCES

Huzama Ahmad and Se-Young Yun. SpotAttention: Plug-in block-sparse routing for pretrained long-context transformers. *arXiv preprint arXiv:2606.22874*, 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.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *arXiv preprint arXiv:2205.14135*, 2022.

DeepSeek-AI. Deepseek-v3.2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.

DeepSeek-AI. DeepSeek-V4.1-Flash: Pushing the limits of KV cache compression. *arXiv preprint arXiv:2609.19969*, 2026.

Qihang Fan, Huaibo Huang, Zhiying Wu, Juqiu Wang, Bingning Wang, and Ran He. FlashPrefill: Instantaneous pattern discovery and thresholding for ultra-fast long-context prefilling. *arXiv preprint arXiv:2603.06199*, 2026.

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

Ronald L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45(9):1563–1581, 1966. doi: 10.1002/j.1538-7305.1966.tb01709.x.

Xiang Hu, Xinyu Wei, Hao Gu, Minshen Zhang, Tian Liang, Huayang Li, Lei Zhu, Yan Wang, Sirui Han, Yushi Bai, Kewei Tu, Haitao Mi, and Leo Liang. Hierarchical sparse attention done right: Toward infinite context modeling. *arXiv preprint arXiv:2607.02980*, 2026.

Rakshith Jayanth and Viktor K. Prasanna. FAST-Prefill: FPGA accelerated sparse attention for long context LLM prefill. In *2026 IEEE 34th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2026. doi: 10.1109/FCCM68464.2026.00067.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *International Conference on Learning Representations*, 2020.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. *arXiv preprint arXiv:2309.06180*, 2023.

Xunhao Lai, Weiqi Xu, Yufeng Yang, Qiaorui Chen, Yang Xu, Lunbin Zeng, Xiaolong Li, Haohai Sun, Haichao Zhu, Vito Zhang, Jinkai Hu, Jiayao Li, Rui Gao, Zekun Li, Songquan Zhu, Jingkai Zhou, and Pengyu Zhao. Minimax sparse attention. *arXiv preprint arXiv:2606.13392*, 2026.

Hong Liu, Yuan Cheng, Lin Niu, Yi Su, Yufei Xue, Anmin Liu, Guanghua Yu, and Jianchen Zhu. PIVOT: Efficient query-group indexing for token-level sparse attention. *arXiv preprint arXiv:2607.24593*, 2026.

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, Yutao Zhang, Zhilin Yang, Xinyu Zhou, Mingxing Zhang, and Jiezhong Qiu. MoBA: Mixture of block attention for long-context LLMs. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

NVIDIA. CUDA Programming Guide, Version 12.4. Technical report, NVIDIA Corporation, 2024a. [Online]. Available: `https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html`.

NVIDIA. Parallel Thread Execution ISA, Version 8.4. Technical report, NVIDIA Corporation, 2024b. [Online]. Available: `https://docs.nvidia.com/cuda/pdf/ptx_isa_8.4.pdf`.

Zihan Qiu, Zekun Wang, Xiao Li, Yanpeng Li, Yang Xu, Yixuan Wang, Huaqing Zhang, Rui Men, Bochao Mao, Chengruidong Zhang, Fan Zhou, Hao Luo, Haofeng Huang, Haoran Lian, Haoyan Huang, Hongqing Chen, Jianwei Zhang, Jing Xu, Junjie Wang, Langshi Chen, Liangyu Wang, Linlang Jiang, Man Yuan, Minmin Sun, Peng Jin, Siqi Zhang, Siyu Wang, Xingzhang Ren, Yakai Wang, Yi Zhang, Yiming Dong, Yizhong Cao, Yubo Ma, Yunfei Mao, Bo Zheng, and Dayiheng Liu. On the design of Qwen3.8-Next architecture: Evaluation, efficiency, and training stability. *arXiv preprint arXiv:2608.30320*, 2026.

Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics*, 9:53–68, 2021.

Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. Flashattention-3: Fast and accurate attention with asynchrony and low-precision. *arXiv preprint arXiv:2407.08608*, 2024.

Zhenyi Shen, Junru Lu, Lin Gui, Jiazheng Li, Yulan He, Di Yin, and Xing Sun. SSA: Sparse sparse attention by aligning full and sparse attention outputs in feature space. *arXiv preprint arXiv:2511.20102*, 2025.

Jianyu Wei, Yizhao Gao, Qihao Zhang, Shimao Chen, Zhengju Tang, Yu Cheng, Shengjie Zhou, Zihan Jiang, Yifan Song, Hailin Zhang, Liang Zhao, Bo Yang, Gang Wang, Shijie Cao, and Fuli Luo. HySparse2: Hybrid sparse attention with two-level KV sharing. *arXiv preprint arXiv:2609.26368*, 2026.

Jianzong Wu, Liang Hou, Haotian Yang, Ye Tian, Pengfei Wan, Di Zhang, and Yunhai Tong. VMoBA: Mixture-of-block attention for video diffusion models. In *International Conference on Learning Representations*, 2026.

Guangxuan Xiao, Junxian Guo, Kasra Mazaheri, and Song Han. Optimizing mixture of block attention. *arXiv preprint arXiv:2511.11571*, 2025.

Ran Yan, Youhe Jiang, Zhuoming Chen, Haohui Mai, Beidi Chen, and Binhang Yuan. FSA: An alternative efficient implementation of native sparse attention kernel. *arXiv preprint arXiv:2508.18224*, 2025.

Jiayi Yuan, Cameron Shinn, Kai Xu, Jingze Cui, George Klimiashvili, Guangxuan Xiao, Perkz Zheng, Bo Li, Yuxin Zhou, Zhouhai Ye, Weijie You, Tian Zheng, Dominic Brown, Pengbo Wang, Markus Hoehnerbach, Richard Cai, Julien Demouth, John D. Owens, Xia Hu, Song Han, Timmy Liu, and Huizi Mao. Blasst: Dynamic blocked attention sparsity via softmax thresholding. *arXiv preprint arXiv:2512.12087*, 2025a.

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. In *Annual Meeting of the Association for Computational Linguistics*, 2025b.

Ted Zadouri, Markus Hoehnerbach, Jay Shah, Timmy Liu, Vijay Thakkar, and Tri Dao. Flashattention-4: Algorithm and kernel pipelining co-design for asymmetric hardware scaling. *arXiv preprint arXiv:2603.05451*, 2026.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences. *arXiv preprint arXiv:2007.14062*, 2020.

Yanke Zhou, Yiduo Li, Hanlin Tang, Maohua Li, Kan Liu, Tao Lan, Lin Qu, Yuan Yao, and Xiaoxing Ma. Full attention strikes back: Transferring full attention into sparse within hundred training steps. *arXiv preprint arXiv:2605.16928*, 2026.

A Additional Methodology

A.1 Complexity Analysis

A call has four stages, and under the reference design of Sec. 3.1 they scale as

$$\mathcal{O}(N^2 d/B) \text{ (scoring)}, \quad \mathcal{O}(N^2 k/B) \text{ (selection)}, \quad \mathcal{O}(Z) \text{ (index)}, \quad \mathcal{O}(ZBd) \text{ (attention)}, \quad (10)$$

where scoring and selection remain dense over the candidate blocks, while the emitted metadata volume and the attention work scale with the realized pattern. The candidate-set update within each query is comparison-bound and involves sequential dependence, with cost increasing in both n and k, since the reference router maintains an ordered candidate set. The three design axes act on different terms of Eq. (10); beyond arithmetic work, the proposed execution also changes memory-traffic multiplicity and scheduling granularity. Table 6 summarizes each component against its corresponding reference design, which differs by component.

Threshold selection removes that k-dependent ordered-set maintenance, so the improvement is against the procedure Eq. (10) prices rather than against Top-k in general, while requiring two score passes to obtain the per-query maximum before any comparison. Neither pass materializes a floating-point score matrix. Index construction additionally scans the packed candidate mask, giving $\mathcal{O}(Nn + Z)$ work at candidate granularity. Because threshold routing does not impose a fixed per-query retention budget, Z is data-dependent and can reach $\Theta(N^2/B)$ in the dense-block

Table 6: Arithmetic, movement, and scheduling costs for one attention call. Each row states its own reference design, and data-dependent quantities appear as realized rather than bounded.

Component	Reference design	Proposed design
Scoring	one pass, $\mathcal{O}(N^2 d/B)$	two passes, same asymptotic order
Selection	ordered k-set, $\mathcal{O}(N^2 k/B)$	threshold test, $\mathcal{O}(N^2/B)$
Index construction	count + prefix-sum + scatter, $\mathcal{O}(Z)$	scan + emit, $\mathcal{O}(Nn + Z)$
Attention arithmetic	$\mathcal{O}(ZBd)$	unchanged, $\mathcal{O}(ZBd)$
K/V movement	$\mathcal{O}(\Lambda(M)\, CBd)$	$\mathcal{O}(\Lambda_T\, CBd)$
Longest work item	$\max_{h,j} w_{h,j}$	$\leq \lceil T/m_{\mathrm{MMA}} \rceil$

limit, so attention complexity must be compared at matched realized sparsity; Sec. B.1 provides this controlled comparison.

KV-centric ownership and fanout splitting leave the $\mathcal{O}(ZBd)$ arithmetic unchanged; they instead reduce fetch multiplicity and bound the longest serial work item, trading reuse against parallelism. Every bound here is per attention call and does not by itself determine end-to-end latency, which Sec. 4 measures.

B Additional Results and Discussion

B.1 Ablation Studies

Unless otherwise stated, we replay captured BF16 tensors from Qwen3-4B and GQA-30B-A3B at 32K and 128K on a single rank, retaining the complete 32-query-head/8-KV-head shape with head dimension 128 and block size 128. Comparisons use identical causal and compulsory-block rules. Latencies are medians of 50 measurements after 10 warmups; each subsection specifies its timing scope and CUDA Graph usage. These device replays exclude host-side work and model inference, and remove the cross-rank synchronization present in the TP8 main experiments. Execution-only comparisons hold routing metadata fixed; routing-policy comparisons match the realized number of selected query-block pairs. We first evaluate the cumulative implementation, then each component.

B.1.1 Cumulative Implementation Ablation

Table 2 adds KV-centric ownership, fanout splitting, query-gather pipelining, and threshold routing to FlashMoBA. At each Top-k budget, the first four variants consume one identical saved CSC pattern. Their timed replays execute the Top-k router but feed the saved pattern to downstream kernels, controlling run-to-run changes in column counts. The threshold variant uses a separate saved pattern with valid pair count matched within 0.05%; selected pair identities may differ.

Timing includes pooling, routing, required reverse-index construction, device-side worklist construction, attention, and merge. We use eager CUDA events because the baseline's segmented-sort path is not graph-capturable in this replay. These fixed-pattern device-path costs are distinct from live router-to-attention invocations and the CUDA Graph kernel timings below.

The final path achieves 1.56 to 1.65× speedup at Top-16 and 2.16 to 2.56× at Top-32. At Top-16, threshold routing helps slightly at 32K but regresses at 128K, where the gain comes primarily from execution; its contribution is larger at Top-32. Adjacent columns are implementation steps rather than single-switch causal tests: +Split replaces the specialized unsplit kernel with a worklist kernel, and +Pipeline also changes the kernel family and merge path. Secs. B.1.4 and 4.2.5 isolate these effects within fixed kernels. On sampled rows, the same-pattern Top-k variants have relative L_2 output error below 9.3×10^{-4} against FlashMoBA; this check excludes the threshold arm, which selects different pairs.

B.1.2 Top-k vs. Threshold Selection Cost

We compare the cost of producing attention-ready CSC metadata. FlashMoBA scores candidates once while maintaining an ordered k-element set; threshold routing scores them twice, first for the per-query maximum and then for threshold selection and packed-mask updates. This tests the extra score pass against ordered-set maintenance.

We replay captured Q and mean-pooled K at 16K, 32K, 64K, and 128K for $k \in \{16, 32, 64\}$. A separate α is calibrated outside timing for each model, length, and k, matching total attention-visible valid CSC pairs within 0.05% across all 24 comparisons. Compulsory blocks count toward this total and occupy slots within FlashMoBA's fixed budget. Threshold CSC columns use device-resident counts to delimit their valid prefixes. CUDA Graph timings include scoring, selection, and CSC construction, excluding mean pooling and attention.

Table 3 shows threshold routing faster at 16K and 32K, near parity at 64K, and slower at 128K for $k = 16$. At $k = 32$ and 64, its full-router speedups reach 3.15 to 4.13× and 6.99 to 11.25×, respectively. The growing gap is consistent with ordered-set maintenance becoming more expensive, although threshold routing also pays more to emit the larger matched pattern. The two sides scale along different variables, which is what the crossover reflects. Threshold routing pays for a second scoring pass, a cost set by the candidate count and independent of k, whereas the reference router pays for ordered-set maintenance, a cost that grows with k (Sec. A.1). At $k = 16$ the two are close enough that the extra pass decides the comparison as the sequence grows, so the advantage is lost by 128K; at larger k the ordered-set term dominates instead.

A separate 128K decision-stage measurement excludes final CSC offset/scatter construction but retains scoring, selection, and output/count updates. Here threshold routing uses one fixed α calibrated at Top-16, rather than matching each k separately. Its latency stays near 25 ms; FlashMoBA is faster at $k = 8$, near parity at $k = 16$, and rises to 109.7 to 125.9 ms at $k = 32$ and 398.4 to 491.3 ms at $k = 64$. This supports the k dependence discussed in Sec. A.1 for the measured FlashMoBA implementation, but does not isolate set-update instructions from other effects of kernel specialization.

B.1.3 KV-Centric Ownership

We compare query-centric and KV-centric execution using identical Q/K/V and Top-16 CSC metadata, with one sink block and a two-block local window. The captures contain post-normalization, post-RoPE Q/K and corresponding V. Qwen3-4B is captured with one tensor-parallel rank; GQA-30B-A3B's eight rank captures are concatenated along the head dimension for the full-head replay.

The KV-centric variant assigns one CTA per (h, j) column, using an unsplit 64×64 kernel and online-softmax merge, with fanout splitting, query splitting, and worklist scheduling disabled. Timing covers the query-centric attention kernel or the KV-centric kernel plus its required merge; routing and reverse-index construction are excluded from both. The latter costs 0.62 ms at 32K and 3.36 to 3.48 ms at 128K across the models.

KV-centric ownership reduces attention-stage latency by 16.9 to 18.9% at 32K and 31.0 to 33.8% at 128K (Table 4). At 128K, it replaces query-centric multiplicities of 16.03 to 20.91 with one owner per column, serving about 2,033 rows per owner. This is consistent with the reuse mechanism in Sec. 3.3.1, although latency alone does not isolate memory traffic from other kernel costs. Selected pairs and the attention definition are preserved, while accumulation order changes; relative L_2 output error stays below 9.32×10^{-4} across the four validation cases.

B.1.4 Fanout-Aware Splitting

Using the captures of Sec. B.1.3, we fix one Top-16 CSC pattern per capture and vary only the selected-row cap $T \in \{\infty, 8192, 4096, 2048\}$. All variants use the same flat-worklist kernel and merge, without work-item sorting or query-gather pipelining. The $T = \infty$ control assigns one item per nonempty column through this same path, unlike the specialized unsplit kernel in the ownership experiment. Timing includes attention and merge; routing, reverse-index construction, and host-built worklists are outside timing, so the measurements exclude production device-side worklist construction costs.

Fig. 4 characterizes optional fanout $f^{\mathrm{opt}}_{h,j} = |\{i : j \in \mathcal{S}_{h,i} \setminus \mathcal{F}_i\}|$, excluding compulsory sink/local selections. Only 0.44% to 2.89% of nonempty optional columns exceed $T = 4096$, yet maximum optional fanout at 128K differs by 10.3× between the captures (117,179 versus 11,337 rows). This describes the sampled patterns, not a general model-architecture ranking.

Splitting the actual all-pair columns at $T = 4096$ reduces attention-stage latency by 15.0 to 16.0% (Table 7). The largest work item falls from 512 tiles at 32K and 2,046 to 2,048 tiles at 128K to 64 tiles, while structural K/V load multiplicity Λ_{4096} rises to 1.034 to 1.094. This count is not

Table 7: Fanout-aware splitting. Latencies include the attention kernel and merge, in ms, on the same CSC pattern within each row. Columns → items gives nonempty (h, j) columns and work items at $T = 4096$; their ratio is Λ_{4096}. Max tiles gives $\max w$ for $T = \infty \to 4096$ with $m_{\mathrm{MMA}} = 64$. Speedup compares unsplit with $T = 4096$.

Model	Length	Columns → items	Max tiles	Unsplit	$T = 8192$	$T = 4096$	$T = 2048$	Speedup
Qwen3-4B	32K	8192 → 8960	512 → 64	17.406	15.176	14.795	14.683	1.18×
Qwen3-4B	128K	32768 → 35040	2048 → 64	71.189	59.832	59.901	59.755	1.19×
GQA-30B-A3B	32K	8192 → 8467	512 → 64	17.464	15.176	14.803	14.664	1.18×
GQA-30B-A3B	128K	32736 → 34049	2046 → 64	71.053	59.680	59.666	59.539	1.19×

measured physical memory traffic. Similar speedups despite different optional tails indicate that optional fanout alone does not predict the gain: both all-pair patterns also contain full-length forced sink columns. All tested caps outperform unsplit; their latencies differ by at most 3.5% at 32K and 0.25% at 128K relative to each case's fastest cap. We use $T = 4096$ thereafter. Validation replays give zero maximum absolute and relative L_2 output errors.

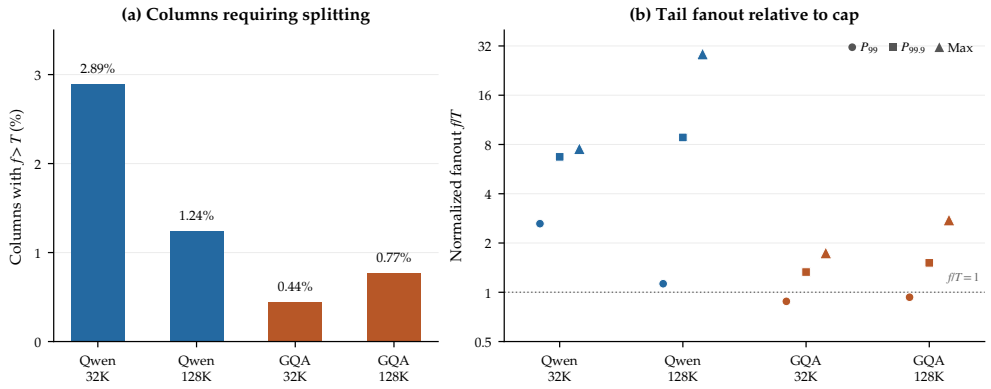

Figure 4: Selective splitting targets a small but potentially severe optional-route fanout tail. (a) Fraction of nonempty optional (h, j) columns with $f^{\mathrm{opt}}_{h,j} > T = 4096$. (b) P99, P99.9, and maximum optional fanout normalized by T; the dotted line marks $f/T = 1$. Qwen and GQA-30B-A3B denote Qwen3-4B and GQA-30B-A3B. Forced sink and local query-to-block pairs are excluded from both panels. Structural counts come from separate routing replays of the same captured Q/K inputs used for the latency comparison.

B.2 Analysis

Fig. 5 breaks down the complete attention path against FlashMoBA and dense FA3 at 32K and 128K.

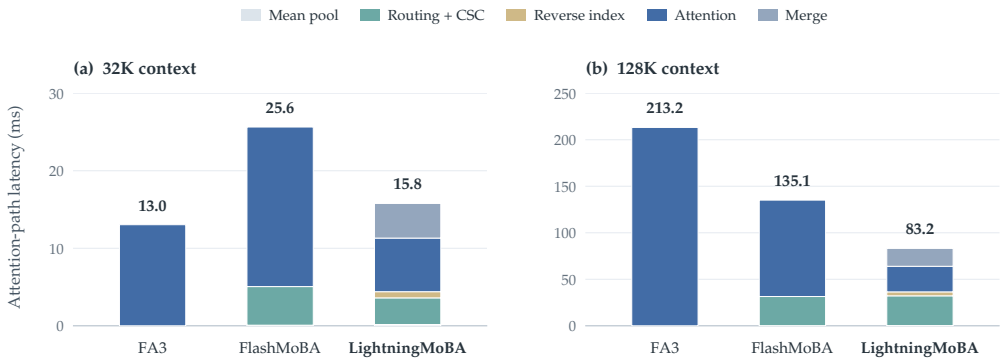

Figure 5: Complete forward attention-path timing breakdown for Qwen3-4B at (a) 32K and (b) 128K on one SM90 GPU. Each panel compares dense FA3, FlashMoBA Top-16, and the final LightningMoBA configuration. The panels use separate latency axes; sparse bars include pooling, routing/CSC construction, attention, reverse-index work, and merge.

B.2.1 Pipeline Stall and Activity Analysis

We profile the attention kernel of the serial and pipelined variants from Sec. 4.2.5 on the two 128K captures using Nsight Compute. These profiler runs do not supply the CUDA Graph timings.

With pipelining, long-scoreboard stall falls from about 63% to below 8%, while tensor-pipe activity rises from about 7% to 17% and SM issue activity from about 15% to 36% (Table 8). These whole-kernel counters support overlap of irregular Q gathers with matrix work, but do not resolve producer/consumer timelines or attribute the latency gain to individual warps. Counter definitions appear in the caption.

Table 8: Kernel-level Nsight Compute counters at 128K for the serial and pipelined variants of Sec. 4.2.5. Tensor-pipe and SM issue activity are percentages of peak sustained rate during unit-active cycles; long scoreboard is the percentage of active warps per cycle waiting on an L1TEX scoreboard dependency (local, global, surface, or texture access).

Model	Variant	Long scoreboard (%)	Tensor pipe (%)	SM issue (%)
Qwen3-4B	No pipeline	63.33	7.33	15.05
Qwen3-4B	Pipelined	7.87	17.08	36.01
GQA-30B-A3B	No pipeline	62.56	7.49	15.33
GQA-30B-A3B	Pipelined	7.67	17.14	36.03

B.3 Discussion

B.3.1 Architectural Discussion

The design separates architecture-agnostic work organization from Hopper-specific execution mechanisms. KV-centric ownership, fanout-aware splitting, and reverse-index metadata determine how routed work is represented and scheduled, independent of TMA or WGMMA. Our Hopper implementation maps the regular K/V path to TMA, the irregular query-gather path to a dedicated producer warp, and matrix computation to WGMMA, allowing data movement and Tensor Core computation to overlap. Porting to another accelerator therefore requires hardware support, or an alternative software mechanism, for concurrent contiguous transfers, indexed gathers, and matrix operations, but does not require changing the routing rule or ownership abstraction. Consequently, the pipelining gains reported here are tied to Hopper, whereas the routing and ownership analyses are hardware-independent.

B.3.2 Impact on Tensor-Parallel Load Balance

Fanout-aware splitting bounds the work assigned to one KV column within a rank, but it does not redistribute work between tensor-parallel ranks. We analyze an instrumented Qwen3-4B RULER-128K prefill profile with eight ranks, each owning four query heads and one KV head. The verified routing configuration uses an eight-block minimum and a 9% density cap. Excluding the first request, 12 requests and 36 layers provide 432 complete request-layer groups. For each group, we divide the maximum rank-local value by the mean across ranks; Table 9 aggregates these ratios. The same profile confirms that fanout imbalance, not randomness, drives the slowest rank: its selected pair count is also the highest in almost every group.

Table 9: Observed TP rank imbalance on Qwen3-4B RULER-128K prefill. Ratios compare the maximum and mean of eight ranks within each of 432 request-layer groups. K1 denotes KV-centric attention execution; the complete stage includes routing, scheduling, K1, and merge.

Rank-local quantity	Mean max/mean	P90 max/mean
Selected query-block pairs	1.88×	2.65×
KV work items	1.87×	2.62×
K1 latency	1.86×	2.61×
Complete attention-stage latency	1.39×	1.71×

The rank with the most selected query-block pairs is also the slowest K1 rank in 428 of 432 groups, so the remaining K1 imbalance tracks rank-local workload even with within-rank splitting enabled. The profile times individual prefill stages with CUDA events and disables decode CUDA Graph for instrumentation; these ratios diagnose a limitation of the current head placement, not an end-to-end speedup estimate. Rank-aware budget control or cross-rank work redistribution could reduce the

slow-rank path, but the former can alter the selected pattern and the latter incurs communication. We leave both as future work rather than assuming a free TP balancing gain.

C Additional Results

Table 10: Retention sweep and quality-latency operating points on RULER-128K. Each row is a separate run; α is the routing threshold, and latency is the same end-to-end generation-loop wall time in minutes as Table 1. The 5%, 10%, and 20% rows denote the target retention densities used for the sweep. The full-run harness did not record realized routed density; the matched-budget rows instead report the fixed Top-16 routing's measured density. All points are measured, not interpolated. Full attention is a quality reference only; its latency is not matched.

Model	Operating point	α	Target density	Latency (min)	RULER score
Qwen3-4B	matched budget	0.169	3.13%	251.6	46.51
Qwen3-4B	5% density	0.08	5%	274.7	49.49
Qwen3-4B	10% density	0.015	10%	346.0	59.45
Qwen3-4B	20% density	0.003	20%	432.7	70.14
Qwen3-4B	full attention (reference)	—	100%	475.4	82.85
GQA-30B-A3B	matched budget	0.173	3.13%	253.1	35.40
GQA-30B-A3B	5% density	0.09	5%	290.6	35.49
GQA-30B-A3B	10% density	0.02	10%	356.8	41.70
GQA-30B-A3B	20% density	0.004	20%	440.6	54.09
GQA-30B-A3B	full attention (reference)	—	100%	489.7	68.11

C.1 Quality-Latency Trade-off

When the matched-budget **LightningMoBA** point leaves latency headroom relative to FlashMoBA, we increase the retention budget to test whether additional retained blocks improve quality while preserving a latency advantage. This is a budget sweep rather than an exact latency match: each operating point is measured independently, and we do not interpolate to force equal latency with the baseline. Table 10 reports the threshold α, target retention density where applicable, generation-loop latency, and RULER score for Qwen3-4B and GQA-30B-A3B. Full attention serves only as a quality reference.

D Additional Setup Details

Baselines The primary sparse baseline is MoBA's fixed Top-k routing (Lu et al., 2025) executed by the FlashMoBA kernel (Xiao et al., 2025), not an unoptimized reference implementation. This compares **LightningMoBA** against an optimized implementation of the same underlying block-attention method. The original-attention quality reference uses each checkpoint's native attention configuration; for hybrid models, this retains the prescribed sliding-window layers. It is a reference score, not an assumed upper bound on every task. Results identify the kernel behind each attention variant. The effect of each design component is examined through the cumulative ablation in Sec. B.1, rather than inferred solely from a shared implementation codebase.

Training-free protocol For each checkpoint, model weights are held fixed across the attention variants. Neither sparse method requires additional model fine-tuning or an auxiliary learned indexer for the comparison. **LightningMoBA** retains MoBA's block-scoring function and changes the block-selection rule and its execution. Training-free therefore describes the inference-time adaptation, not the training history of the checkpoint or equality of the resulting model outputs. Threshold calibration. For each model and target Top-k budget, we calibrate α on a held-out sample that is disjoint from the test evaluation. We sweep α and select the value whose mean realized number of retained blocks most closely matches the corresponding Top-k budget. We then verify the matched budget on a separate validation sample and fix α before running the reported evaluation. No task labels or benchmark scores are used during calibration.

Model sources Qwen3-4B[1], Qwen3-235B-A22B[2], Qwen3-32B-YaRN4[3], Llama-3.1-8B[4], and GQA-30B-A3B, an internal model with no external release.

Metrics and measurement protocol We report two timing quantities and keep them separate. End-to-end results use generation-loop wall time over a complete evaluation run, a system-level measurement in which both variants execute the same dense decode path. Controlled replays instead time the complete attention path, including routing, sparse-index construction and attention computation, with component timings where available. The two are not combined into a single speedup claim, and prefill, decode and full-model inference measurements are distinguished rather than merged. Quality comparisons report task scores alongside the original-attention reference and their differences. The run specification records hardware, parallelism, precision, software and kernel versions, batch and sequence lengths, warmup and timing repetitions, the aggregation statistic, and CUDA Graph usage. Common hardware, input, precision and timing conditions are held fixed within each comparison; method-specific kernel implementations and versions are recorded separately. Throughput, memory use and kernel-utilization measurements can provide additional diagnostics where collected, but are not substitutes for the stated latency and quality comparisons. Main experiments use tensor parallelism of eight unless noted, and ablations replay on one rank.

Comparison protocol The primary comparison uses a *matched block budget*: the two sparse methods use the same block size and the same mean realized number of retained blocks over the evaluated queries. The count includes compulsory blocks and uses the same causal-boundary convention for both methods. We compare both latency and quality at this operating point without assuming that the selected blocks are identical. When **LightningMoBA** retains latency headroom, we additionally increase its realized retention budget and report the measured quality and latency points. This supplementary sweep does not interpolate between runs or claim an exact latency match. The original-attention configuration supplies a quality reference; latency is not matched.

E Additional Method Details

E.1 Streaming Router and Sparse Metadata Construction

Eq. (3) defines what is selected, but a direct implementation appears to require the full score matrix: $r_i^\star$ must be reduced before the scores are reused for the threshold test, and materializing $\{r_{ij}\}$ costs $\mathcal{O}(Nn)$ floating-point storage per head.

We use the two-pass streaming construction of the routing stage in Fig. 1: the first pass computes the per-query maximum $r_i^\star$ over complete blocks preceding the query's own block, and the second recomputes optional-block scores against the equivalent cutoff $r_i^\star - \Delta$. Compulsory blocks are retained directly, emitting a candidate bitmask with per-query and per-column retained counts.

At one bit per candidate block the bitmask occupies $\mathcal{O}(Nn/32)$ machine words regardless of realized retention, and each candidate needs only a scalar comparison and a bit update, unlike an ordered Top-k candidate set per query.

The execution in Sec. 3.3 consumes the transpose of these decisions: for each KV block, the query rows that selected it. Since the per-column counts are already available, an exclusive prefix sum determines each column's contiguous range, after which a scatter pass writes the retained query indices into a compact buffer. The resulting representation contains exactly Z retained query-block pairs, requires no sorting, and directly exposes the per-column fanout used by Sec. 3.3.2.

Lemma E.1 (Schedule length under list scheduling)**.** *Model the P resident CTA slots as identical workers and measure time in MMA-tile service times. For column work sizes $w_{h,j}$ and $W = \sum_{h,j} w_{h,j}$, list scheduling produces a schedule of length $\mathcal{L}$ satisfying*

$$\max\Big(\frac{W}{P},\, \max_{h,j} w_{h,j}\Big) \;\le\; \mathcal{L} \;\le\; \frac{W}{P} + \Big(1 - \frac{1}{P}\Big) \max_{h,j} w_{h,j}. \tag{11}$$

[1] https://huggingface.co/Qwen/Qwen3-4B-Instruct-2507

[2] https://huggingface.co/Qwen/Qwen3-235B-A22B

[3] Qwen3-32B with YaRN-based context extension; https://huggingface.co/Qwen/Qwen3-32B

[4] https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct

Algorithm 1 KV-Centric Ownership and Forward Pass

Require: $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM; for each head h and KV block j, the retained row list $\mathcal{R}_{h,j}$ of the compact metadata of Sec. E.1, with fanout $f_{h,j} = |\mathcal{R}_{h,j}|$; logical block size B; physical tile sizes B_c and m_{MMA}.

Divide $\mathbf{K}, \mathbf{V}$ into $n = \lceil N/B \rceil$ logical blocks $\mathbf{K}_j, \mathbf{V}_j$ of size $B \times d$.
Allocate the partial-output workspace $\mathbf{O}_{\mathrm{tmp}}$ and the log-sum-exp workspace $\mathbf{L}$ in HBM.
for all (h, j) **in parallel do**
    Load $\mathbf{K}_j, \mathbf{V}_j$ from HBM into SRAM. ▷ Once per column in this conceptual formulation
    for $0 \le r < \lceil f_{h,j}/m_{\mathrm{MMA}} \rceil$ **do**
        Gather-load the r-th query tile $\mathbf{Q}_r^{(h,j)} \leftarrow \mathbf{Q}[\mathcal{R}_{h,j}[rm_{\mathrm{MMA}} : \min((r{+}1)m_{\mathrm{MMA}}, f_{h,j})], h, :]$.
        Initialize the on-chip softmax state (m, ℓ) and the partial-output accumulator $\mathbf{O}_r$.
        for $0 \le c < \lceil B/B_c \rceil$ **do**
            $\mathbf{S}_{rc} \leftarrow \mathbf{Q}_r^{(h,j)} \mathbf{K}_{j,c}^{\top}$, with causal masking.
            Update on-chip softmax states (m, ℓ) and partial outputs $\mathbf{O}_r$ using $\mathbf{V}_{j,c}$.
        end for
        Store the partial output $\mathbf{O}_r$ and $m + \log \ell$ of these rows to $\mathbf{O}_{\mathrm{tmp}}$ and $\mathbf{L}$.
    end for
end for
for all query rows i and heads h **in parallel do**
    Merge all partial outputs associated with (i, h) and write the result to $\mathbf{O}[i, h, :]$.
end for

E.2 Asymmetric Data Movement

The two paths differ in kind, not merely in volume. A selected block's keys and values occupy a contiguous window of the sequence, and the TMA moves affine tensor regions of exactly this form (NVIDIA, 2024a). Routed query rows do not form such a region: their indices come from the routing metadata and may fall anywhere in the sequence, so they require arbitrary indexed gathers. We accordingly move K and V with the asynchronous engine and gather query rows with vectorized indexed loads. Within a work unit, the block's keys and values are read by every query row the unit serves, so each CTA loads them once and keeps them resident for its lifetime, leaving the query side as the only recurring transfer. The matrix work follows the same asymmetry, using $64 \times 64 \times 16$ WGMMA tiles for $QK^{\top}$ and $64 \times 128 \times 16$ tiles for PV (NVIDIA, 2024b), with the probabilities held in registers so that the softmax between them introduces no shared-memory stage of its own.

E.3 Pipeline Overlap and Latency Hiding

Gathering queries is the irregular half of the loop; assigning these gathers to the compute warps interleaves indexed-load stalls with matrix instruction issue. We therefore use one 128-thread consumer warpgroup and one 32-thread producer warp. The producer decodes the next query tile's routing metadata, including the selected query-row indices and, for each row, the extent of its causal mask within the block, and then issues the gathers, while the consumer executes $QK^{\top}$, the softmax, and PV. Queries flow through a two-stage ring buffer (Fig. 7), so stage $i+1$ is gathered while stage i is consumed, and index processing overlaps matrix work instead of preceding it. Once $QK^{\top}$ and the softmax have consumed a staged tile we release the stage before PV and the epilogue.

K/V residency removes the regular transfers from the steady-state loop, and early release shortens a stage's shared-memory lifetime, letting the producer refill that stage sooner. Sec. 4.2.5 compares serial and overlapped query gathering within the same SM90 kernel and reports the resulting latency and kernel-level stall changes. This comparison does not separately isolate producer specialization.

F ADDITIONAL FIGURES

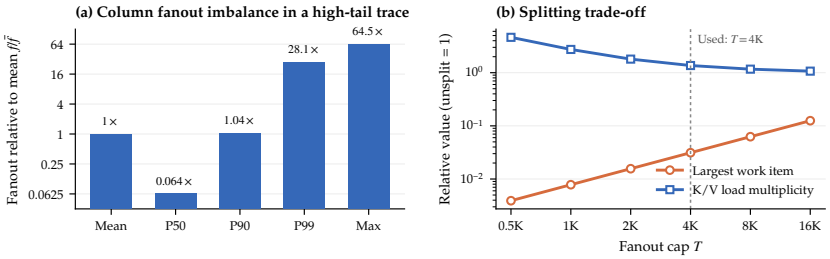

Figure 6: Fanout imbalance and fanout-aware splitting trade-off on a high-tail GQA-30B-A3B, TP8 RULER-128K prefill trace. (a) Fanout distribution over 4,096 nonempty query-head/KV-block columns, normalized by the mean. (b) Relative largest work item and K/V load multiplicity as the fanout cap T varies; the dashed line marks our $T = 4$K setting.

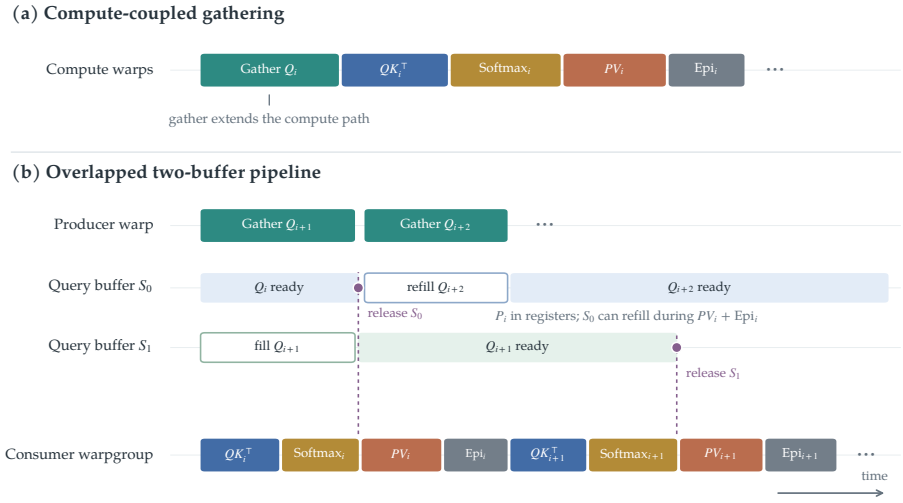

Figure 7: Two-stage query pipeline with warp specialization. (a) Without dedicated query-gather specialization, irregular query gathering remains on the compute path and serializes with matrix work. (b) A producer warp gathers query tile $i + 1$ while the consumer warpgroup processes tile i; early slot release lets the producer refill during PV and the epilogue.

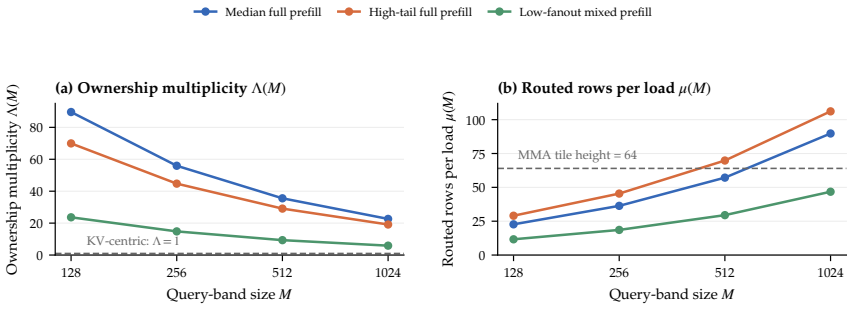

Figure 8: Ownership fragmentation under query-centric execution on three real Top-16 routing traces from TP8 RULER-128K inference with GQA-30B-A3B. (a) Mean ownership multiplicity; KV-centric ownership gives $\Lambda = 1$. (b) Mean routed rows per load; the dashed line marks the 64-row MMA tile height.

G PROOFS

Notation. Let $\beta_M(i) = \lfloor i/M \rfloor$ be the index of the band owning query row i, so band b is $\mathcal{B}_b(M) = \{bM, \dots, (b{+}1)M - 1\}$. For a selected column (h, j) write

$$i^{\min}_{h,j} = \min \mathcal{Q}_{h,j}, \qquad i^{\max}_{h,j} = \max \mathcal{Q}_{h,j}, \qquad s_{h,j} = i^{\max}_{h,j} - i^{\min}_{h,j} + 1.$$

Under this notation Eq. (5) becomes

$$\lambda_{h,j}(M) = \big|\beta_M(\mathcal{Q}_{h,j})\big|, \qquad \Lambda(M) = \frac{1}{C}\sum_{h,j} \lambda_{h,j}(M).$$

Proof of Lemma 3.1. Write $\lambda_{h,j} = \lambda_{h,j}(M)$. Since every selected column has $\mathcal{Q}_{h,j} \neq \emptyset$, we have $\beta_M(\mathcal{Q}_{h,j}) \neq \emptyset$ and hence

$$\lambda_{h,j} \;\geq\; 1 \qquad \text{for every selected } (h, j).$$

Assume $\Lambda(M) = 1$, i.e.

$$\sum_{h,j} \lambda_{h,j} = C.$$

Because each $\lambda_{h,j}$ is a positive integer and there are exactly C terms, the two displays force

$$\lambda_{h,j} = 1 \qquad \text{for every selected } (h, j).$$

Fix (h, j). Then $\big|\beta_M(\mathcal{Q}_{h,j})\big| = 1$ means that $\mathcal{Q}_{h,j} \subseteq \mathcal{B}_b(M)$ for a single b, hence

$$bM \;\leq\; i^{\min}_{h,j} \;\leq\; i^{\max}_{h,j} \;\leq\; (b{+}1)M - 1.$$

Consequently

$$s_{h,j} = i^{\max}_{h,j} - i^{\min}_{h,j} + 1 \;\leq\; \big((b{+}1)M - 1\big) - bM + 1 \;=\; M.$$

Taking the maximum over all selected columns gives

$$M \;\geq\; \max_{h,j} s_{h,j},$$

which is Eq. (7).

The implication cannot be reversed. Take $M = 128$ and a routing pattern with a single selected column ($C = 1$) and $\mathcal{Q} = \{127, 128\}$. Then

$$s = 128 - 127 + 1 = 2 \;\leq\; 128 = M,$$

whereas $\beta_M(127) = 0$ and $\beta_M(128) = 1$, so $\lambda = |\{0, 1\}| = 2$ and $\Lambda(M) = 2 \neq 1$. Hence $M \geq \max_{h,j} s_{h,j}$ is necessary but not sufficient for $\Lambda(M) = 1$. □

Proof of Lemma E.1. The classical list-scheduling bound of Graham (1966) gives the result; we restate its short proof for completeness. Index the work units by $\ell = 1, \dots, n$ with sizes $w_\ell \geq 0$, and write

$$W = \sum_{\ell=1}^{n} w_\ell, \qquad w_{\max} = \max_{1 \leq \ell \leq n} w_\ell.$$

All units are available at time 0, run non-preemptively on P identical workers, and list scheduling starts an available unit whenever a worker is idle.

Lower bound. The unit of size $w_{\max}$ occupies one worker for $w_{\max}$ time, so $\mathcal{L} \geq w_{\max}$; and P workers can complete at most $P\mathcal{L}$ work by time $\mathcal{L}$, so $W \leq P\mathcal{L}$. Together,

$$\mathcal{L} \;\geq\; \max\Big\{\frac{W}{P},\, w_{\max}\Big\}.$$

Upper bound. Let J be a unit that completes last, with start time s_J and size w_J, so that

$$\mathcal{L} = s_J + w_J.$$

For every $t \in [0, s_J)$ no worker is idle: if some worker were idle at such a t, then J would still be unstarted and available, and list scheduling would have started it at t, contradicting the definition of s_J. Hence all P workers execute work throughout $[0, s_J)$, while J itself is processed only after s_J. The corresponding work inequality is

$$P\, s_J \;\le\; W - w_J \qquad \Longrightarrow \qquad s_J \;\le\; \frac{W - w_J}{P}.$$

Substituting this into the expression for $\mathcal{L}$,

$$\mathcal{L} = s_J + w_J \;\le\; \frac{W - w_J}{P} + w_J \;=\; \frac{W}{P} + \Big(1 - \frac{1}{P}\Big) w_J \;\le\; \frac{W}{P} + \Big(1 - \frac{1}{P}\Big) w_{\max},$$

where the last step uses $w_J \le w_{\max}$. Identifying ℓ with (h, j) and substituting $w_{h,j} = \lceil f_{h,j}/m_{\mathrm{MMA}} \rceil$ from Eq. (8) yields Eq. (11). □