DOUBLE-P: HIERARCHICAL TOP-P SPARSE ATTENTION FOR LONG-CONTEXT LLMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Attention over a growing key-value (KV) cache dominates the cost of long-context decoding. Top-p sparse attention keeps the smallest set of tokens whose attention mass reaches a target p, so the budget adapts to each head and decoding step. Existing top-p methods, however, use top-p only to *prune*: a fixed-budget selector first proposes candidate tokens, and token-level score estimation then removes some of them. Such pipelines cannot recover attention mass that the candidate set misses, and their estimation cost grows with context length. We present **Double-P**, which uses top-p to *select*. Double-P estimates the attention mass of every KV cluster from size-weighted centroids and applies top-p over the entire context, so the attended set is sized by the attention distribution rather than by a budget. A second top-p threshold on the same estimates allocates precision: high-mass clusters receive exact token-level attention, and the remaining selected clusters are approximated by their centroids instead of being discarded. Across long-context benchmarks, Double-P consistently achieves near-zero accuracy drop, reducing attention computation overhead by up to 1.8× and delivers up to 1.3× end-to-end decoding speedup over state-of-the-art fixed-budget sparse attention methods.

1 INTRODUCTION

Large language models (LLMs) now support contexts of tens to hundreds of thousands of tokens, enabling retrieval-augmented generation (Fan et al., 2024; Jin et al., 2024), document understanding (Appalaraju et al., 2021; Luo et al., 2024), and long-horizon reasoning (Grattafiori et al., 2024; Yang et al., 2025b; Team et al., 2023). During autoregressive decoding, every new token attends to the entire key-value (KV) cache, so attention cost grows with context length and dominates decoding latency at long contexts.

Sparse attention reduces this cost by attending only to the tokens that matter most for the current query. Most methods (Tang et al., 2024b; Liu et al., 2024; Zhang et al., 2025a; Liu et al., 2025a; Chen et al., 2025) follow a *top-k* formulation that selects a fixed budget of tokens, pages, or clusters. No single budget fits all heads, layers, and decoding steps: the number of tokens needed to retain 95% of the attention mass changes substantially from step to step (Figure 1), and any fixed budget, even the mean top-p budget (k=256), misses this target in more than 20% of heads at difficult steps (Figure 2). *Top-p sparse attention* (Lin et al., 2025) removes the fixed budget by keeping the smallest set of tokens whose cumulative attention mass reaches a target p.

Realizing top-p efficiently is difficult because the cutoff depends on attention scores that are unknown until attention is computed. Twilight (Lin et al., 2025) adopts a *select-then-prune* design: an existing sparse attention method first selects candidates under a conservative fixed budget (e.g., 25% of the context), and a pruner then estimates token-level scores from a 4-bit copy of the keys and keeps the top-p subset of the candidates. Top-p is thus a post-selection refinement, with three structural consequences. **(i) A recall ceiling.** Pruning can only remove candidates, so the retained mass never exceeds that of the initial candidate set, which falls below $p = 0.95$ for 91.9% of LongBench samples at Quest's recommended budget (§2). **(ii) A context-dependent estimation cost.** The pruner scores every candidate token, whose number grows with the context, and lifting the ceiling requires even more candidates, up to the full context. **(iii) Extra memory.** The quantized key copy adds 26.6% to the KV cache even with ideal 4-bit packing. Cluster-based methods (Chen et al., 2025; Liu et al., 2025a) avoid token-level estimation but fix how many clusters are selected and computed exactly.

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.

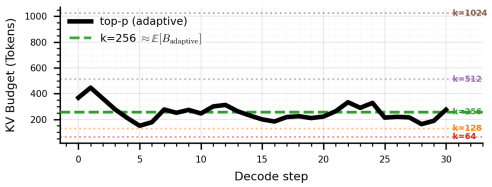


Figure 1: **Adaptive vs. fixed budgets.** Tokens needed to retain attention mass $p = 0.95$ at each decoding step (black).

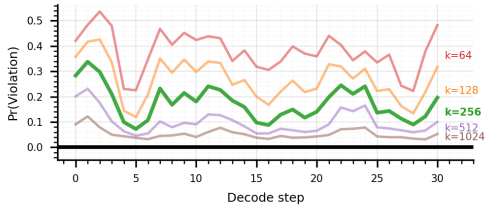


Figure 2: **Violation rates.** Fraction of heads whose fixed budget retains less than $p = 0.95$ of the attention mass.

Table 1: **Where each design places its two boundaries.** The *selection boundary* bounds the attended set; the *precision boundary* decides which attended entries are computed exactly. Double-P is the only design in which both are set by estimated attention mass over the entire context.

Design	Selection boundary	Precision boundary	Mass estimated over	Unit
Top-k selection	fixed budget	all exact	—	page / cluster
RetroInfer	fixed budget	fixed ratio	—	cluster
Twilight	fixed budget, then top-p	all exact	candidates only	token
Double-P (ours)	**top-p (p_1)**	**top-p (p_2)**	**entire context**	**cluster**

Top-p should decide *which part of the context is attended at all*, not only which pre-selected tokens to drop, and the same mass estimates should decide *how precisely* each attended part is computed. This requires estimating attention mass over the entire context far more cheaply than token-level scoring. Clustered KV caches make this possible: a cluster's centroid logit plus the logarithm of its size estimates its total attention mass (§3.2), so one pass over the $N/\bar{s}$ centroids (context length N, average cluster size $\bar{s} = 16$) yields a mass distribution over the whole context. We build on this idea with **Double-P**. **Stage 1** applies top-p with threshold p_1 to the estimated mass of *all* clusters, so the candidate set expands or shrinks with the attention distribution and no fixed budget bounds it. **Stage 2** applies a second threshold to the same estimates to split the selected clusters by precision: a high-mass prefix receives exact token-level attention, while the remaining selected clusters contribute through centroid approximations that are normalized jointly with the exact tokens, so no selected mass is discarded. Both boundaries are set by attention mass, per KV head and decoding step, without token-level estimation (Table 1).

We further introduce GPU efficient kernels that minimize top-p selection overhead and maximize data locality for sparse attention computation. Extensive evaluation shows that Double-P achieves **near-zero accuracy drop** and up to **2.27×** self-attention speedup and **1.26×** end-to-end decoding speedup.

2 Background and Motivation

Top-p sparse attention. For a decoding query $\mathbf{q}$ and cached keys $\{\mathbf{k}_j\}_{j=1}^{N}$, attention assigns token j the weight $a_j = \exp(x_j)/\sum_{l=1}^{N}\exp(x_l)$ with logit $x_j = \mathbf{q}\mathbf{k}_j^{\top}/\sqrt{d}$. We call $m(S) = \sum_{j\in S} a_j$ the *attention mass* of a token subset S. Top-k attention keeps a fixed number of tokens; top-p attention keeps the smallest S with $m(S) \geq p$, whose size follows the attention distribution. Because the weights are unknown before attention is computed, a top-p method must estimate them, and *where* and *at what granularity* it does so determine both its accuracy and its cost.

Select-then-prune. Twilight (Lin et al., 2025) estimates the weights only inside a candidate set. A base method $\mathcal{B}$, such as Quest (Tang et al., 2024b), first selects a candidate set $S_{\mathcal{B}}$ under a fixed budget. The pruner then estimates the candidates' weights with a sparse GEMV (SpGEMV) over 4-bit quantized keys and keeps the top-p subset $S \subseteq S_{\mathcal{B}}$. Since pruning can only remove candidates,

$$m(S) \;\leq\; m(S_{\mathcal{B}}) \qquad \text{for every pruner.} \tag{1}$$

Whenever $m(S_{\mathcal{B}}) < p$, the target is unreachable, however accurate the pruner is.

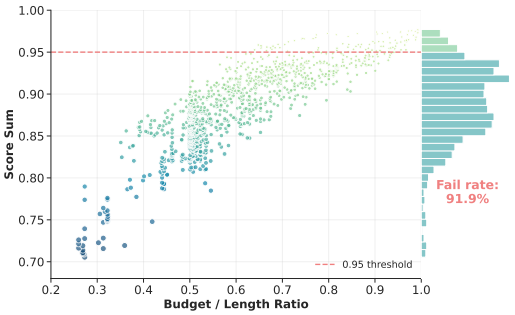


Figure 3: **The candidate set caps the retained mass.** Mass captured by Quest's fixed-budget candidates, one point per LongBench sample; 91.9% fall below $p = 0.95$ before any pruning.

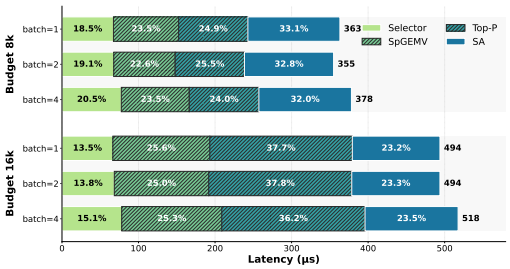


Figure 4: **Latency of token-level top-p attention.** Estimation (SpGEMV) and top-p selection (shaded) take more than half of the attention latency.

Observation 1: the candidate set is usually the bottleneck. Figure 3 plots, for each LongBench sample, the attention mass captured by the candidate set against the budget normalized by context length. Under the commonly recommended budget of 8,192 tokens for Quest, **91.9% of samples** fall below $p = 0.95$, and the same budget recovers almost all of the mass for some samples while missing much of it for others. By Eq. (1), this mass is lost before top-p is ever applied. The only remedy within select-then-prune is a larger budget, up to the full context, which Twilight supports with a trivial selector that keeps all tokens.

Observation 2: token-level estimation is costly and grows with context. The pruner scores every candidate token and selects the top-p subset among them, so its cost grows with the candidate budget and hence with context length. Figure 4 shows that SpGEMV estimation and top-p selection already take more than half of the attention latency. Select-then-prune therefore faces a dilemma: a small budget caps the retained mass (Observation 1), and a large budget makes the pruner score a large fraction of the context at token granularity. The pruner also stores a quantized copy of the keys, which adds 26.6% to the KV cache with ideal 4-bit packing (Appendix C).

Observation 3: cluster-based methods are cheap but budget-driven. Cluster-based methods (Liu et al., 2025a; Chen et al., 2025) rank k-means key clusters by query–centroid similarity, far more cheaply than scoring tokens. RetroInfer (Chen et al., 2025) computes the top-ranked clusters exactly and approximates the next ones from centroids and value sums, but in fixed fractions (1.8% and 23.2% of clusters). Fixed fractions do not control the retained mass, and the number of clusters that need exact computation varies widely (§3.3).

3 DOUBLE-P

Double-P (Figure 5) keeps a compact summary of each KV cluster (§3.1) and, at every decoding step, makes two top-p decisions on the same cluster-level mass estimates: Stage 1 decides which clusters are attended (§3.2), and Stage 2 decides which of them are computed exactly (§3.3). With clusters sorted by estimated mass, the two thresholds split the context into an exactly computed high-mass prefix, an approximated middle region, and a skipped low-mass tail. RetroInfer fixes the sizes of these regions, and Twilight applies its mass cutoff only inside a fixed-budget candidate set; in Double-P, both boundaries follow the attention distribution of each KV head at each step. Following common practice (Xiao et al., 2023), the 4 initial (sink) tokens and the 64 most recent tokens are always attended exactly, and the rest of the context is clustered.

3.1 CLUSTER SUMMARIES

During prefill, Double-P partitions the keys of each KV head into clusters with k-means (Hartigan & Wong, 1979), using an average cluster size of $\bar{s} = 16$. For cluster i with token set $\mathcal{T}_i$, it stores the

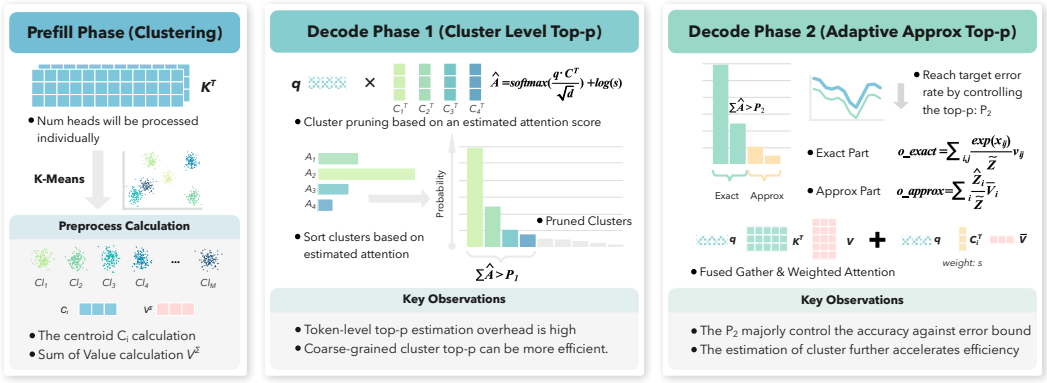


Figure 5: **Double-P overview.** Prefill clusters the keys. At each decoding step, Stage 1 applies top-p (p_1) to size-weighted centroid scores of all clusters, and Stage 2 applies a second threshold to the same scores to split the selected clusters into exact and centroid-approximated groups.

centroid $\mathbf{C}_i$, the size $s_i = |\mathcal{T}_i|$, and the value sum and mean

$$\mathbf{V}_i^{\Sigma} = \sum_{j \in \mathcal{T}_i} \mathbf{v}_{ij}, \qquad \bar{\mathbf{V}}_i = \mathbf{V}_i^{\Sigma} / s_i, \tag{2}$$

where $\mathbf{v}_{ij}$ is the j-th value vector of cluster i. These summaries are the only storage Double-P adds: 9.8% over the BF16 KV cache for Llama-3.1-8B, including a gather workspace, compared with 26.6% for Twilight's quantized key copy alone (Appendix C).

3.2 Stage 1: Global Top-p Cluster Selection

Consider a query $\mathbf{q} \in \mathbb{R}^{1\times d}$ and a cluster i with keys $\mathbf{k}_{ij}$. Its exact token logits, centroid logit, and exact unnormalized mass are

$$x_{ij} = \frac{\mathbf{q}\mathbf{k}_{ij}^{\top}}{\sqrt{d}}, \qquad \bar{x}_i = \frac{\mathbf{q}\mathbf{C}_i^{\top}}{\sqrt{d}}, \qquad Z_i = \sum_{j \in \mathcal{T}_i} \exp(x_{ij}). \tag{3}$$

Computing Z_i requires every token logit. Double-P instead treats all tokens of a cluster as if they had the centroid logit:

$$\widehat{Z}_i = s_i \exp(\bar{x}_i), \qquad \log \widehat{Z}_i = \bar{x}_i + \log s_i. \tag{4}$$

Normalizing over all K clusters, with centroid matrix $\mathbf{C} \in \mathbb{R}^{K\times d}$ and size vector $\mathbf{s} \in \mathbb{R}^{K}$, gives an estimated distribution of attention mass over clusters,

$$\widehat{\mathbf{A}} = \frac{\widehat{\mathbf{Z}}}{\sum_{\ell} \widehat{Z}_{\ell}} = \mathrm{softmax}\left(\frac{\mathbf{q}\mathbf{C}^{\top}}{\sqrt{d}} + \log \mathbf{s}\right). \tag{5}$$

Let π order the clusters by $\widehat{A}_i$ in descending order. Stage 1 selects the shortest prefix whose estimated mass reaches p_1:

$$\mathcal{C}_{p_1} = \{\pi_1, \ldots, \pi_{n_1}\}, \qquad n_1 = \min\left\{n : \textstyle\sum_{r=1}^{n} \widehat{A}_{\pi_r} \geq p_1\right\}. \tag{6}$$

This selection has three desirable properties. First, the $\log s_i$ term makes $\widehat{A}_i$ an estimate of each cluster's total attention mass rather than its per-token relevance, so clusters are ranked by their contribution to the attention output and the cumulative sum in Eq. (6) estimates the mass the selection retains. Second, the selection is global and budget-free: top-p runs over all K clusters, so $|\mathcal{C}_{p_1}|$ follows the attention distribution of each KV head at each step, growing when attention is diffuse and shrinking when it is concentrated, and no fixed candidate set caps the retained mass as in select-then-prune (Eq. 1). Third, it is cheap: one dense GEMV over the $K = N/\bar{s}$ centroids, a sort of K scores, and a prefix scan, with contiguous memory reads and no quantized key copy. The centroid estimate is also reliable where it matters most: on RULER needle retrieval, Stage 1 keeps the needle's cluster in over 99% of the heads that give the needle an attention weight above 5×10^{-3} (Appendix E).

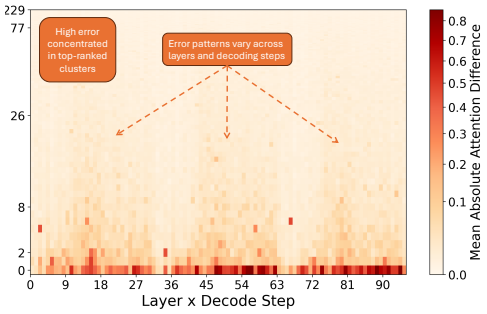


Figure 6: **Centroid approximation error by cluster rank.** The error concentrates in a few top-ranked clusters, and the pattern varies across layers and steps.

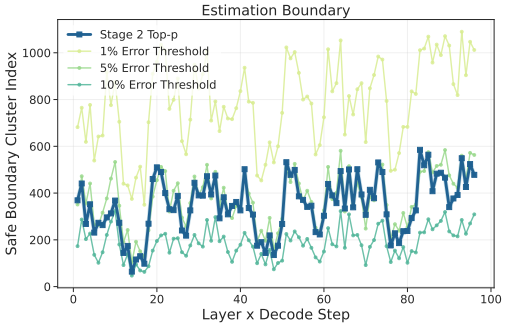


Figure 7: **Exact clusters needed for a bounded error.** Thin: minimum per error threshold. Bold: the number Stage 2 selects, which tracks the requirement.

3.3 Stage 2: Mass-Guided Precision Allocation

Stage 1 typically selects many clusters, and computing exact token-level attention for all of them is unnecessary. Figure 6 shows the absolute error of replacing each cluster's exact token attention with its centroid approximation, by cluster rank, across layers and decoding steps. The error is long-tailed: a few top-ranked clusters account for most of it, and the distribution changes across layers and steps. Figure 7 shows the minimum number of clusters that must be computed exactly to keep the attention error below a given threshold. This number fluctuates widely, so any fixed exact budget either wastes computation or violates the error bound.

Stage 2 therefore reuses the Stage-1 order and estimates to decide precision. It computes exactly the shortest prefix whose estimated mass reaches a fraction $p_2 \in (0, 1]$ of the Stage-1 target, and approximates the other selected clusters:

$$\mathcal{C}_{\text{exact}} = \{\pi_1, \ldots, \pi_{n_2}\}, \quad n_2 = \min\left\{n : \textstyle\sum_{r=1}^{n} \widehat{A}_{\pi_r} \geq p_2\, p_1\right\}, \qquad \mathcal{C}_{\text{approx}} = \mathcal{C}_{p_1} \setminus \mathcal{C}_{\text{exact}}. \quad (7)$$

The bold curve in Figure 7 is n_2, which closely tracks the minimum number of exact clusters required for a bounded error. Clusters in $\mathcal{C}_{\text{approx}}$ are not discarded: their estimated masses and the exact token masses share one normalizer,

$$\widetilde{Z} = \sum_{i \in \mathcal{C}_{\text{exact}}} \sum_{j \in \mathcal{T}_i} \exp(x_{ij}) + \sum_{i \in \mathcal{C}_{\text{approx}}} \widehat{Z}_i, \quad (8)$$

and the output mixes both contributions (sink and recent tokens are included in the exact term):

$$\mathbf{o}(\mathbf{q}) = \sum_{i \in \mathcal{C}_{\text{exact}}} \sum_{j \in \mathcal{T}_i} \frac{\exp(x_{ij})}{\widetilde{Z}}\, \mathbf{v}_{ij} \; + \sum_{i \in \mathcal{C}_{\text{approx}}} \frac{\widehat{Z}_i}{\widetilde{Z}}\, \bar{\mathbf{v}}_i. \quad (9)$$

Stage 2 therefore keeps every selected cluster in the output while spending exact computation only where it matters. Clusters outside $\mathcal{C}_{\text{exact}}$ still contribute through their summaries, so their only error comes from approximating a low-mass cluster by its centroid, which is small (Figure 6). The exact set adapts to each KV head and decoding step and closely tracks the minimum needed for a bounded error (Figure 7), and both cutoffs come from the same sorted estimates, so the split adds no estimation cost. In §4.5, Stage 2 matches the accuracy of computing all selected clusters exactly with about $5\times$ less exact computation, and preserves accuracy on aggregation-heavy tasks where a fixed exact fraction falls short.

3.4 Efficient Implementation

Top-p on sorted scores. Both stages operate on the same sorted cluster masses, so a custom kernel computes both cutoffs n_1 and n_2 in a single prefix-sum pass with early stopping, avoiding the iterative threshold search of token-level pruners. **Fused gathering.** One kernel gathers the tokens of $\mathcal{C}_{\text{exact}}$

Table 2: **RULER accuracy on Llama-3.1-8B.** Average over 13 tasks at each context length. Best sparse method per column in bold.

	Method	16K	32K	64K	128K	Avg.
	Full attention	93.25	90.00	85.36	75.66	86.07
Fixed	Quest	89.24	85.95	83.61	70.79	82.40
Fixed	ClusterKV	92.05	85.21	78.61	69.89	81.44
Fixed	LServe	91.78	87.49	83.02	70.77	83.27
Fixed	RetroInfer	91.56	88.12	83.77	75.05	84.63
Top-p	Quest+Twilight	86.73	86.50	81.87	71.21	81.58
Top-p	RetroInfer+Twilight	92.83	89.95	**85.21**	**75.72**	**85.93**
Top-p	**Double-P (ours)**	**92.87**	**90.31**	84.55	75.13	85.72

and the summaries of $\mathcal{C}_{\text{approx}}$ into a contiguous buffer, so that attention reads memory in a coalesced manner. **Weighted attention.** Eq. (9) is computed by the weighted FlashAttention kernel (Chen et al., 2025), which weights each cluster summary by its size, so exact tokens and summaries are processed in one kernel. Selection is performed per KV head, aggregating the estimated masses over the query heads of a GQA group.

4 Evaluation

We evaluate Double-P's accuracy (§4.2), compare it directly with select-then-prune top-p (§4.3), measure its efficiency (§4.4), and isolate the contribution of each stage (§4.5).

4.1 Experimental Setup

Models and hardware. Our main models are Llama-3.1-8B-Instruct (Grattafiori et al., 2024) (32 layers, 8 KV heads) and Qwen3-8B (Yang et al., 2025a); Appendix A adds the mixture-of-experts Qwen3-30B-A3B. We use one NVIDIA H100 PCIe GPU (80 GB), CUDA 12.8, and PyTorch 2.8.

Baselines. All methods attend exactly to 4 sink and 64 recent tokens (Xiao et al., 2023) and sparsify the remaining tokens. *Fixed-budget:* Quest (Tang et al., 2024b) (page size 16, 25% of tokens); ClusterKV (Liu et al., 2025a) (2K tokens at 16K–64K) and LServe (Yang et al., 2025c) (25% of tokens at 16K–64K), both with 8K tokens at 128K; and RetroInfer (Chen et al., 2025) with its default configuration (1.8% exact and 23.2% approximated clusters). *Top-p:* Twilight (Lin et al., 2025) applied to Quest (Quest+Twilight, with the recommended 25% candidate budget) and to RetroInfer (RetroInfer+Twilight, which gathers the top 25% of clusters, and prunes their tokens with Twilight). Appendix A adds StreamingLLM (Xiao et al., 2023) and PQCache (Zhang et al., 2025a), and Appendix B adds MoBA (Lu et al., 2025).

Double-P configuration. We use an average cluster size of 16 and 10 k-means iterations for all models. The thresholds are calibrated once per model on PG-19 continuation perplexity (§4.5), a workload disjoint from all evaluation benchmarks, and then fixed: $(p_1, p_2) = (0.95, 0.7)$ for Llama-3.1-8B and $(0.99, 0.8)$ for Qwen3-8B and Qwen3-30B-A3B.

Benchmarks. RULER (Hsieh et al., 2024) has 13 synthetic tasks covering retrieval, multi-hop tracing, aggregation, and question answering; we evaluate contexts from 16K to 128K tokens. LongBench (Bai et al., 2024) contains realistic long-context tasks; we use its 13 English and code tasks (Table 11), whose inputs are mostly 5K–15K tokens long.

4.2 Accuracy

Near-zero accuracy drop on RULER. Table 2 reports RULER accuracy on Llama-3.1-8B. Double-P achieves **near-zero accuracy drop** compared with full attention: it stays within 0.81 points at every context length up to 128K, **0.35 points on average**, and even exceeds full attention at 32K. It outperforms every fixed-budget baseline at every context length, by up to **+2.19** points over the strongest one at 32K, and matches the accuracy of RetroInfer+Twilight while decoding up to 2.3×

Table 3: **Accuracy on Qwen3-8B RULER and on LongBench.** RULER and LongBench averages over 13 tasks each. Best sparse method per column in bold. Results on Qwen3-30B-A3B are in Appendix A.

	Qwen3-8B RULER				LongBench	
Method	16K	32K	64K	Avg.	Llama-3.1-8B	Qwen3-8B
Full attention	91.25	90.59	80.40	87.41	39.33	35.82
Quest	77.12	79.80	60.12	72.35	38.76	35.52
RetroInfer	**91.16**	**90.65**	**80.09**	**87.30**	39.03	35.63
Quest+Twilight	76.69	79.50	58.59	71.59	38.96	35.26
Double-P (ours)	91.02	90.36	79.26	86.88	**39.06**	**35.78**

Table 4: **Double-P vs. RetroInfer+Twilight** on Llama-3.1-8B. LongBench is the average over 13 tasks (Table 11). Latency per output token at batch size 1 (measurement details in Appendix B); memory relative to the BF16 KV cache.

	RULER accuracy				Long-	Decode latency (ms)			Memory
Method	16K	32K	64K	128K	Bench	16K	32K	64K	overhead
Full attention	93.25	90.00	85.36	75.66	39.33	20.52	24.61	33.15	0%
RetroInfer	91.56	88.12	83.77	75.05	39.03	22.97	23.74	23.85	+9.8%
RetroInfer+Twilight	92.83	89.95	85.21	75.72	38.59	29.36	34.81	47.78	+36.4%
Double-P (ours)	92.87	90.31	84.55	75.13	39.06	**18.61**	**19.30**	**20.43**	+9.8%

faster (§4.3). The advantage holds at long contexts: at 128K, Quest, ClusterKV, and LServe lose 4.9–5.8 points relative to full attention, whereas Double-P loses only 0.53. Post-selection pruning does not close this gap: Quest+Twilight never exceeds Quest by more than 0.55 points and falls below it at 16K and 64K, since pruning can only remove candidates (Eq. (1)). These results show that global top-p selection preserves attention mass more reliably than fixed budgets or post-selection pruning.

Consistent accuracy across models. Double-P shows the same near-zero accuracy drop on Qwen3-8B (Table 3): its average RULER accuracy is within 0.53 points of full attention (86.88 vs. 87.41) and on par with RetroInfer, while the page-based Quest and Quest+Twilight lose 15–16 points. On the mixture-of-experts Qwen3-30B-A3B, Double-P slightly exceeds full attention (Appendix A).

Accuracy on LongBench. On realistic long-context tasks, Double-P stays within **0.27 and 0.04 points** of full attention on Llama-3.1-8B and Qwen3-8B, the smallest gap among the sparse attention methods in Table 3. Double-P thus preserves attention quality beyond synthetic retrieval benchmarks.

4.3 TOP-p AS SELECTION VS. TOP-p AS PRUNING

RetroInfer+Twilight is the most direct test of our central claim: it shares Double-P's cluster index and Twilight's top-p objective, and differs only in *where* top-p is applied, at token level after a fixed-budget cluster retrieval rather than over all clusters before anything is retrieved.

Accuracy is on par. Table 4 shows that the two designs are within 0.7 points of each other at every RULER context length, and Double-P is higher at 16K and 32K. On LongBench, Double-P is 0.47 points higher (39.06 vs. 38.59), most visibly on passage retrieval (96.22 vs. 90.08; Table 11). Adding Twilight recovers most of the RULER accuracy that RetroInfer's fixed budget loses, confirming the value of the top-p objective; Double-P attains the same accuracy without any token-level estimation.

Cost diverges with context length. The attention kernels of RetroInfer+Twilight take 16.6 ms per token at 16K and 34.7 ms at 64K (2.1×), because retrieval, gathering, SpGEMV, and top-p selection all scale with its candidate set, while its final sparse attention stays nearly constant (Appendix B). Double-P's end-to-end decode latency grows only from 18.6 to 20.4 ms (1.1×), because Stage 1 scores $N/\bar{s}$ centroids with one dense GEMV instead of gathering and scoring candidate tokens. In end-to-end latency, Double-P decodes 1.6× faster than RetroInfer+Twilight at 16K and 2.3× faster at 64K.

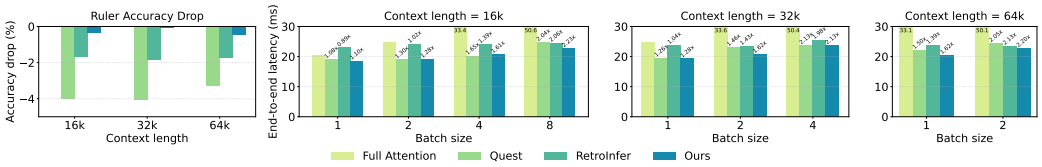


Figure 8: **Accuracy drop and end-to-end decoding latency** on RULER with Llama-3.1-8B across context lengths and batch sizes. Double-P has near-zero accuracy drop while consistently decoding faster.

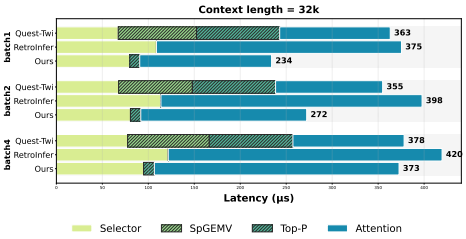


Figure 9: **Attention latency breakdown** at 32K context. Shaded bars are top-p estimation overhead, which Double-P nearly eliminates.

Table 5: **Stage-2 isolation** on RULER-32K (Llama-3.1-8B, 13 tasks), with Stage 1 fixed at $p_1 = 0.95$. "Exact" is the fraction of Stage-1 clusters computed exactly. All rows come from one evaluation run.

Stage-2 policy	Exact	Mean	Retrieval	CWE
Adaptive (ours)	adapt.	90.61	98.17	**58.5**
All exact	1.000	90.80	99.14	53.9
Fixed (RetroInfer)	0.072	89.60	97.79	46.7

Memory. Twilight's quantized key copy, added to RetroInfer's index, raises the footprint to 36.4% above the KV cache, whereas Double-P needs only the index (9.8%; Appendix C). Double-P therefore reaches the accuracy of select-then-prune top-p at the cost of a fixed-budget method.

4.4 Efficiency

We measure latency on Llama-3.1-8B with RULER needle-in-a-haystack inputs across context lengths and batch sizes.

Attention latency. Figure 9 breaks down attention latency. Token-level estimation takes over 60% of the attention time of Quest+Twilight. Double-P replaces it with cluster-level estimation, and Stage 2 further shortens the sparse attention step. Overall, Double-P is up to 1.74× faster than Quest+Twilight and 1.78× faster than RetroInfer at the attention level.

End-to-end decoding. Figure 8 reports per-token decode latency together with the accuracy drop from full attention. Double-P is up to 1.11× faster than Quest, 1.26× faster than RetroInfer, and 2.23× faster than full attention.

Prefill overhead. Clustering adds 4.3–4.8%, 2.1–2.6%, and 0.6–2.1% to prefill latency at 16K, 32K, and 64K, which is recovered after 4–27 output tokens; at 512 output tokens, end-to-end latency is up to 1.6× lower than full attention (Appendix B).

Long generation. During generation, new tokens are first attended exactly in a dense window. Every 1,024 generated tokens, Double-P clusters the oldest 1,024 of them into 64 new clusters and appends them to the index without re-clustering existing clusters. Each update therefore costs the same regardless of how many tokens have been generated, and updates take 0.15–0.18% of decoding time. On AIME 2024 with Qwen3-8B in thinking mode, whose reasoning traces reach 38,912 tokens, Double-P matches full attention (76.7% pass@1; Appendix D).

4.5 Ablation Studies

Both stages are needed. Table 5 isolates Stage 2 while holding Stage 1 fixed. Computing every Stage-1 cluster exactly raises the mean by only 0.19 points while requiring about 5× more exact-cluster computation. Replacing the adaptive split with RetroInfer's fixed exact fraction lowers the mean by

1.01 points: retrieval barely changes (-0.38), but the aggregation-heavy CWE task drops from 58.5 to 46.7, since a fixed exact budget cannot follow heads whose mass spans many clusters. Stage 1, in turn, separates Double-P from fixed-budget methods in accuracy (§4.2) and from RetroInfer+Twilight in cost (§4.3).

Calibration and sensitivity. We calibrate (p_1, p_2) on PG-19 continuation perplexity (32K-token prefix, 4K teacher-forced tokens), separate from all benchmarks: p_1 is the smallest value in the perplexity saturation region and p_2 the knee of the error–computation curve, giving $(0.95, 0.7)$ for Llama-3.1-8B and $(0.99, 0.8)$ for Qwen3-8B. Accuracy changes smoothly around these points, and cluster sizes from 8 to 32 tokens change RULER-32K accuracy by less than one point and decode latency by less than 1 ms (Appendix F).

5 Related Work

Retrieval-based sparse attention. These methods select part of the KV cache before computing attention, using page-level summaries (Tang et al., 2024b; Xiao et al., 2024; Yang et al., 2025c), key clusters (Liu et al., 2025a; Chen et al., 2025; Zhu et al., 2025; Liu et al., 2025c), or approximate nearest-neighbor search over tokens (Liu et al., 2024; Zhang et al., 2025a; Chen et al., 2024), mostly under fixed budgets. Double-P reuses RetroInfer's cluster index and weighted attention kernel (Chen et al., 2025) but ranks clusters by estimated mass and sets both boundaries by mass (§3).

Top-p and adaptive budgets. Twilight (Lin et al., 2025) introduced top-p sparse attention as a pruner on top of existing methods, so its retained mass is bounded by the base method's candidates (§2). Tactic (Zhu et al., 2025) also targets a cumulative attention fraction: it ranks key clusters by centroid similarity, fits the score distribution to estimate how many tokens are needed, and computes all selected tokens exactly. Double-P instead estimates cluster mass from size-weighted centroids and uses a second threshold to approximate, rather than compute token by token, the selected clusters outside the high-mass prefix. Other work allocates KV budgets unevenly across layers or heads based on estimated importance (Feng et al., 2024; Tang et al., 2024a; Xiao et al., 2024); Double-P adapts per KV head at every decoding step.

Complementary directions. Token eviction (Xiao et al., 2023; Zhang et al., 2023; Li et al., 2024), sparse prefill (Jiang et al., 2024; Xu et al., 2025; Gao et al., 2024; Zhang et al., 2025b; Jones et al., 2026), trainable sparse attention (Lu et al., 2025; Yuan et al., 2025), quantization (Frantar et al., 2022; Lin et al., 2024; Tseng et al., 2024; Van Baalen et al., 2024; Liu et al., 2025b; Hooper et al., 2024; Yue et al., 2024; Kang et al., 2024), and linear attention (Katharopoulos et al., 2020; Gu & Dao, 2024) reduce cost in ways orthogonal to Double-P, which targets decoding over a full-precision KV cache.

6 Conclusion

This paper presents **Double-P**, a hierarchical top-p sparse attention framework for efficient long-context LLM inference. Instead of pruning a fixed-budget candidate set, Double-P uses top-p to select: it estimates the attention mass of every cluster from size-weighted centroids and selects clusters by top-p over the entire context, then applies a second top-p stage that allocates exact token-level attention only to high-mass clusters and approximates the rest. This design eliminates fixed-budget constraints and token-level estimation, and adapts both the attended set and the exact computation to each head and decoding step. As a result, Double-P preserves near-full-attention accuracy across models and benchmarks, with near-zero accuracy drop relative to full attention, and matches the accuracy of RetroInfer+Twilight while decoding up to $2.3\times$ faster. It also achieves up to $1.78\times$ attention-level and $1.26\times$ end-to-end decoding speedups over strong sparse attention baselines, and decodes up to $2.23\times$ faster than full attention. These findings demonstrate that effective top-p sparse attention requires probability-driven selection and computation across multiple levels of granularity, and position Double-P as a principled and scalable foundation for future sparse attention designs in long-context language models.

AI USE STATEMENT

In this work, we used generative AI tools to implement evaluation and profiling code for baseline methods, to help design and run latency measurements, to process and reformat experimental data, to help state the candidate-set bound in Eq. 1, and to help interpret results while drafting the evaluation. We did not use generative AI tools to generate synthetic data sets, develop theoretical models, prove mathematical claims, propose research hypotheses, assist with translation, or support qualitative analysis. Additionally, we used generative AI tools to create and edit software code, create figures, search for and summarize related literature, format references, suggest a structure for the paper, and draft and edit parts of the paper to improve readability. The authors reviewed all AI-assisted work: AI-assisted code was tested for correctness, and reported numbers were checked against the experiment logs. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work improves the efficiency of long-context inference in existing LLMs through algorithmic and system-level changes to attention computation. It does not introduce new model capabilities, datasets, or human-subject studies. We do not foresee negative societal consequences specific to this work beyond those already associated with LLMs.

REPRODUCIBILITY STATEMENT

Section 3 specifies the algorithm, including the mass estimator (Eq. 5), both selection rules (Eqs. 6 and 7), and the combined output (Eq. 9). Section 4.1 lists models, baselines, hardware, and hyperparameters, and Section 4.5 describes the threshold calibration procedure. Appendix A reports per-length and per-task results, Appendix C the memory accounting, and Appendix D the long-generation update policy. Anonymized code is provided in the supplementary material.

REFERENCES

Srikar Appalaraju, Bhavan Jasani, Bhargava Urala Kota, Yusheng Xie, and R Manmatha. Docformer: End-to-end transformer for document understanding. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 993–1003, 2021.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 3119–3137, 2024.

Yaoqi Chen, Jinkai Zhang, Baotong Lu, Qianxi Zhang, Chengruidong Zhang, Jingjia Luo, Di Liu, Huiqiang Jiang, Qi Chen, Jing Liu, et al. Retroinfer: A vector-storage approach for scalable long-context llm inference. *arXiv preprint arXiv:2505.02922*, 2025.

Zhuoming Chen, Ranajoy Sadhukhan, Zihao Ye, Yang Zhou, Jianyu Zhang, Niklas Nolte, Yuandong Tian, Matthijs Douze, Leon Bottou, Zhihao Jia, et al. Magicpig: Lsh sampling for efficient llm generation. *arXiv preprint arXiv:2410.16179*, 2024.

Wenqi Fan, Yujuan Ding, Liangbo Ning, Shijie Wang, Hengyun Li, Dawei Yin, Tat-Seng Chua, and Qing Li. A survey on rag meeting llms: Towards retrieval-augmented large language models. In *Proceedings of the 30th ACM SIGKDD conference on knowledge discovery and data mining*, pp. 6491–6501, 2024.

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. *arXiv preprint arXiv:2407.11550*, 2024.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

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.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *First conference on language modeling*, 2024.

John A Hartigan and Manchek A Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the royal statistical society. series c (applied statistics)*, 28(1):100–108, 1979.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Yakun S Shao, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *Advances in Neural Information Processing Systems*, 37:1270–1303, 2024.

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

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.

Bowen Jin, Jinsung Yoon, Jiawei Han, and Sercan O Arik. Long-context llms meet rag: Overcoming challenges for long inputs in rag. *arXiv preprint arXiv:2410.05983*, 2024.

Dalton Jones, Junyoung Park, Matthew J. Morse, Mingu Lee, Matthew Harper Langston, and Christopher Lott. Quoka: Query-oriented kv selection for efficient llm prefill. *ICLR 2026*, 2026.

Hao Kang, Qingru Zhang, Souvik Kundu, Geonhwa Jeong, Zaoxing Liu, Tushar Krishna, and Tuo Zhao. Gear: An efficient kv cache compression recipe for near-lossless generative inference of llm. *arXiv preprint arXiv:2403.05527*, 2024.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pp. 5156–5165. PMLR, 2020.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. Snapkv: Llm knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37:22947–22970, 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. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of machine learning and systems*, 6: 87–100, 2024.

Di Liu, Meng Chen, Baotong Lu, Huiqiang Jiang, Zhenhua Han, Qianxi Zhang, Qi Chen, Chengruidong Zhang, Bailu Ding, Kai Zhang, et al. Retrievalattention: Accelerating long-context llm inference via vector retrieval. *arXiv preprint arXiv:2409.10516*, 2024.

Guangda Liu, Chengwei Li, Jieru Zhao, Chenqi Zhang, and Minyi Guo. Clusterkv: Manipulating llm kv cache in semantic space for recallable compression. In *2025 62nd ACM/IEEE Design Automation Conference (DAC)*, pp. 1–7. IEEE, 2025a.

Zihan Liu, Xinhao Luo, Junxian Guo, Wentao Ni, Yangjie Zhou, Yue Guan, Cong Guo, Weihao Cui, Yu Feng, Minyi Guo, et al. Vq-llm: High-performance code generation for vector quantization augmented llm inference. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 1496–1509. IEEE, 2025b.

Zihan Liu, Wentao Ni, Jingwen Leng, Yu Feng, Cong Guo, Quan Chen, Chao Li, Minyi Guo, Yufei Ma, Feng Zhang, et al. Juno++: Optimizing anns and enabling efficient sparse attention in llm via ray tracing core. *ACM Transactions on Architecture and Code Optimization*, 22(4):1–25, 2025c.

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. *arXiv preprint arXiv:2502.13189*, 2025.

Chuwei Luo, Yufan Shen, Zhaoqing Zhu, Qi Zheng, Zhi Yu, and Cong Yao. Layoutllm: Layout instruction tuning with large language models for document understanding. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 15630–15640, 2024.

Hanlin Tang, Yang Lin, Jing Lin, Qingsen Han, Shikuan Hong, Yiwu Yao, and Gongyi Wang. Razorattention: Efficient kv cache compression through retrieval heads. *arXiv preprint arXiv:2407.15891*, 2024a.

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*, 2024b.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. Quip#: Even better llm quantization with hadamard incoherence and lattice codebooks. *arXiv preprint arXiv:2402.04396*, 2024.

Mart Van Baalen, Andrey Kuzmin, Ivan Koryakovskiy, Markus Nagel, Peter Couperus, Cedric Bastoul, Eric Mahurin, Tijmen Blankevoort, and Paul Whatmough. Gptvq: The blessing of dimensionality for llm quantization. *arXiv preprint arXiv:2402.15319*, 2024.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*, 2023.

Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. Duoattention: Efficient long-context llm inference with retrieval and streaming heads. *arXiv preprint arXiv:2410.10819*, 2024.

Ruyi Xu, Guangxuan Xiao, Haofeng Huang, Junxian Guo, and Song Han. Xattention: Block sparse attention with antidiagonal scoring. *arXiv preprint arXiv:2503.16428*, 2025.

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.

An Yang, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoyan Huang, Jiandong Jiang, Jianhong Tu, Jianwei Zhang, Jingren Zhou, et al. Qwen2. 5-1m technical report. *arXiv preprint arXiv:2501.15383*, 2025b.

Shang Yang, Junxian Guo, Haotian Tang, Qinghao Hu, Guangxuan Xiao, Jiaming Tang, Yujun Lin, Zhijian Liu, Yao Lu, and Song Han. LServe: Efficient long-sequence LLM serving with unified sparse attention. In *Proceedings of Machine Learning and Systems (MLSys)*, 2025c.

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.

Yuxuan Yue, Zhihang Yuan, Haojie Duanmu, Sifan Zhou, Jianlong Wu, and Liqiang Nie. Wkvquant: Quantizing weight and key/value cache for large language models gains more. *arXiv preprint arXiv:2402.12065*, 2024.

Hailin Zhang, Xiaodong Ji, Yilin Chen, Fangcheng Fu, Xupeng Miao, Xiaonan Nie, Weipeng Chen, and Bin Cui. Pqcache: Product quantization-based kvcache for long context llm inference. *Proceedings of the ACM on Management of Data*, 3(3):1–30, 2025a.

Jintao Zhang, Chendong Xiang, Haofeng Huang, Jia Wei, Haocheng Xi, Jun Zhu, and Jianfei Chen. Spargeattn: Accurate sparse attention accelerating any model inference. *arXiv preprint arXiv:2502.18137*, 2025b.

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, Yile Gu, Zhichen Zeng, Rohan Kadekodi, Liangyu Zhao, Ang Li, Arvind Krishnamurthy, and Baris Kasikci. Tactic: Adaptive sparse attention with clustering and distribution fitting for long-context llms. *arXiv preprint arXiv:2502.12216*, 2025.

A Additional Accuracy Results

Additional baselines. Table 6 adds two baselines on Llama-3.1-8B. StreamingLLM (Xiao et al., 2023) is a permanent-eviction method that keeps only sink and recent tokens; we give it the same KV budget as Quest (4 sink tokens plus 25% of the KV cache). PQCache (Zhang et al., 2025a) is a token-level method based on product quantization of the KV cache.

Table 6: RULER accuracy on Llama-3.1-8B, including eviction and product-quantization baselines.

Method	16K	32K	64K	Avg.
Full attention	93.25	90.00	85.36	89.54
StreamingLLM	34.91	32.54	30.87	32.77
PQCache	67.45	65.84	67.11	66.80
Quest	89.24	85.95	83.61	86.27
RetroInfer	91.56	88.12	83.77	87.82
Quest+Twilight	86.73	86.50	81.87	85.03
Double-P (ours)	92.87	90.31	84.55	89.24

Qwen3-30B-A3B. Table 7 reports RULER accuracy on the mixture-of-experts Qwen3-30B-A3B; Qwen3-8B results are in Table 3.

Table 7: RULER accuracy on Qwen3-30B-A3B, a mixture-of-experts model.

Method	16K	32K	64K	Avg.
Full attention	95.50	94.95	90.51	93.65
Quest	81.15	81.15	72.05	78.12
RetroInfer	95.41	94.84	91.72	93.99
Quest+Twilight	80.33	81.18	68.77	76.76
Double-P (ours)	95.38	94.89	91.29	93.85

Per-task results. Tables 8–10 give per-task RULER results, and Table 11 gives per-task LongBench results.

Table 8: RULER results across tasks for Llama-3.1-8B and Qwen3-8B, 16K context length.

Method	s1_niah	s2_niah	s3_niah	mk1_niah	mk2_niah	mk3_niah	mv_niah	mq_niah	vt	cwe	fwe	qa_1	qa_2	**Avg.**
Llama-3.1-8B (full)	100.00	100.00	100.00	100.00	100.00	100.00	96.50	100.00	99.60	93.85	91.33	76.50	54.50	93.25
Quest	99.00	100.00	87.00	100.00	96.00	70.00	95.25	97.75	98.60	86.50	88.00	84.00	58.00	89.24
RetroInfer	100.00	99.50	99.50	100.00	98.50	96.50	95.62	99.88	99.90	82.40	87.00	76.00	55.50	91.56
Quest+Twilight	99.00	100.00	80.50	96.00	95.50	69.50	94.62	96.50	98.80	81.70	86.33	74.50	54.50	86.73
Double-P	100.00	100.00	100.00	100.00	99.5	98.00	97.00	99.75	99.50	92.90	88.67	78.00	54.00	92.87
Qwen3-8B (full)	100.00	100.00	100.00	99.50	100.00	99.00	92.88	100.00	100.00	91.65	93.17	59.50	50.50	91.25
Quest	98.00	98.00	53.00	99.00	84.00	40.00	86.25	86.38	94.30	67.40	90.67	58.50	47.00	77.12
RetroInfer	100.00	100.00	100.00	99.50	100.00	99.00	93.88	100.00	100.00	92.20	92.50	59.00	49.00	91.16
Quest+Twilight	99.00	98.50	52.50	98.50	83.50	36.50	86.88	86.25	92.70	66.20	90.00	61.00	45.50	76.69
Double-P	100.00	100.00	99.50	99.00	100.00	99.00	92.88	99.88	99.70	92.60	92.67	59.00	49.00	91.02

Table 9: RULER results across tasks for Llama-3.1-8B and Qwen3-8B, 32K context length.

Method	s1_niah	s2_niah	s3_niah	mk1_niah	mk2_niah	mk3_niah	mv_niah	mq_niah	vt	cwe	fwe	qa_1	qa_2	**Avg.**
Llama-3.1-8B (full)	100.00	100.00	100.00	98.50	100.00	100.00	97.38	99.25	99.80	50.40	95.17	75.50	54.00	90.00
Quest	97.50	100.00	86.50	99.00	97.50	89.00	92.38	96.50	99.00	44.50	90.00	73.00	52.50	85.95
RetroInfer	100.00	100.00	100.00	98.00	99.50	98.50	95.38	98.75	99.20	37.20	90.50	76.00	52.50	88.12
Quest+Twilight	99.00	98.50	90.00	99.50	99.00	88.50	90.12	95.88	99.60	47.05	90.83	73.50	53.00	86.50
Double-P	100.00	100.00	100.00	99.50	99.50	97.50	96.75	99.50	99.20	62.55	91.00	75.50	53.00	90.31
Qwen3-8B (full)	100.00	100.00	100.00	99.00	97.50	96.50	99.00	99.50	99.80	80.65	95.17	60.50	50.00	90.59
Quest	100.00	100.00	72.50	99.50	89.00	43.50	93.12	86.62	92.60	60.85	90.17	62.00	47.50	79.80
RetroInfer	100.00	100.00	100.00	99.50	97.50	97.50	99.25	99.12	99.80	79.90	94.83	60.00	51.00	90.65
Quest+Twilight	100.00	99.00	72.00	99.00	90.50	42.50	92.88	84.00	91.00	61.35	89.33	62.50	49.50	79.50
Double-P	100.00	99.50	99.50	99.00	97.50	95.50	99.00	99.50	99.40	81.75	95.00	59.00	50.00	90.36

Table 10: RULER results across tasks for Llama-3.1-8B and Qwen3-8B, 64K context length.

Method	s1_niah	s2_niah	s3_niah	mk1_niah	mk2_niah	mk3_niah	mv_niah	mq_niah	vt	cwe	fwe	qa_1	qa_2	**Avg.**
Llama-3.1-8B (full)	100.00	100.00	100.00	100.00	99.00	96.50	98.38	99.75	98.70	9.50	86.33	72.50	49.00	85.36
Quest	100.00	99.50	93.50	96.50	98.00	90.50	98.39	98.62	95.75	9.95	85.17	73.00	48.00	83.61
RetroInfer	100.00	100.00	100.00	100.00	96.50	90.50	97.50	99.75	98.10	5.05	79.67	72.50	49.50	83.77
Quest+Twilight	100.00	99.50	93.00	96.00	95.50	71.50	99.25	96.25	96.70	11.10	85.00	72.50	48.00	81.87
Double-P	100.00	100.00	100.00	100.00	98.00	92.50	97.62	99.75	96.90	13.75	79.67	72.50	48.50	84.55
Qwen3-8B (full)	100.00	100.00	100.00	90.50	76.50	31.00	95.50	97.25	99.70	68.20	78.50	66.50	41.50	80.40
Quest	99.50	69.50	35.50	69.50	58.50	3.00	78.75	71.50	85.30	35.95	78.00	59.50	37.00	60.12
RetroInfer	100.00	99.50	100.00	90.00	75.00	29.50	94.88	97.12	99.70	68.75	77.67	67.50	41.50	80.09
Quest+Twilight	99.50	62.00	31.50	80.00	63.64	0.00	73.38	65.62	80.43	31.43	75.83	57.72	40.68	58.59
Double-P	100.00	98.50	96.50	89.00	72.00	28.00	94.50	95.25	97.00	70.50	78.67	69.00	41.50	79.26

Table 11: LongBench results across tasks for Llama-3.1-8B and Qwen3-8B. RetroInfer+Twilight is evaluated on Llama-3.1-8B only.

Method	2Wiki	Gov	Hotpot	Lcc	News	Field	Musi	Narr	Retriv	Qasper	Qmsum	Repo	Triv	**Avg.**
Llama-3.1-8B (full)	16.16	34.38	16.82	63.16	26.74	27.64	11.72	31.96	97.73	13.18	23.23	57.03	91.48	39.33
Quest	16.27	34.51	16.69	63.22	26.86	27.27	11.94	33.77	96.58	12.94	23.75	50	90.03	38.76
RetroInfer	16.1	34.37	17.49	61.72	25.61	27.4	11.75	33.08	96.74	13.02	22.48	56.08	91.49	39.03
Quest+Twilight	16.38	34.05	16.72	63.55	26.82	26.92	11.54	31.43	95.78	12.85	23.25	56.23	91.02	38.96
RetroInfer+Twilight	16.41	34.87	16.81	62.53	26.59	26.93	10.73	31.63	90.08	12.25	24.04	57.4	91.41	38.59
Double-P	16.18	34.93	17.1	62.85	26.78	27.36	11.47	32.07	96.22	12.67	22.66	56.17	91.32	39.06
Qwen3-8B (full)	12.68	29.87	12.86	67.52	23.04	26.38	7.5	3.74	95.46	11.44	21.12	63.79	90.21	35.82
Quest	13.24	30.32	12.44	67.42	23.05	25.8	7.76	3.46	94.5	11.39	22.18	61.73	88.53	35.52
RetroInfer	12.08	29.91	12.73	67.5	22.69	26.04	7.89	3.62	94.25	11.43	20.53	64.35	90.21	35.63
Quest+Twilight	13.05	30.91	11.82	67.53	23.56	24.95	7.54	3.33	92.5	11.53	21.77	61.35	88.56	35.26
Double-P	12.61	30.02	12.48	67.71	22.16	26.27	7.54	3.74	95.21	11.72	20.77	64.48	90.46	35.78

B Additional Efficiency Results

Latency of RetroInfer+Twilight. Table 12 decomposes the per-token attention time of RetroInfer+Twilight with the 25% candidate set used in the accuracy runs. Each component is timed in place with CUDA events and summed over the 32 layers (Llama-3.1-8B, batch size 1, 2 warm-up and 5 measured runs of 128 output tokens). Selection and gathering, SpGEMV, and top-p selection all grow with context length, while the final sparse attention stays nearly constant because top-p caps the number of attended tokens. Because Twilight's accuracy implementation computes attention with unoptimized operators, we obtain the end-to-end latency of RetroInfer+Twilight from its optimized kernels: we take Double-P's end-to-end latency (Table 4), subtract Double-P's attention time measured the same way (5.9, 7.1, and 7.3 ms at 16K, 32K, and 64K), and add the RetroInfer+Twilight attention time, since the two methods share all other operations (projections, MLP, normalization, and the generation loop).

Table 12: Per-token latency (ms) of RetroInfer+Twilight with a 25% candidate set: attention components summed over layers, and end-to-end latency.

Context	Candidates	Sel. + gather	SpGEMV	Top-p	Sparse attn.	Total	End-to-end
16K	255 / 1,024	7.54	1.69	2.11	5.30	16.64	29.36
32K	503 / 2,016	11.22	2.67	3.27	5.44	22.60	34.81
64K	1,007 / 4,032	18.60	4.73	5.59	5.75	34.67	47.78

Prefill and end-to-end latency. Figures 10 and 11 complement §4.4. The prefill overhead of clustering is small, and Double-P's end-to-end latency is already below that of full attention at 128 output tokens.

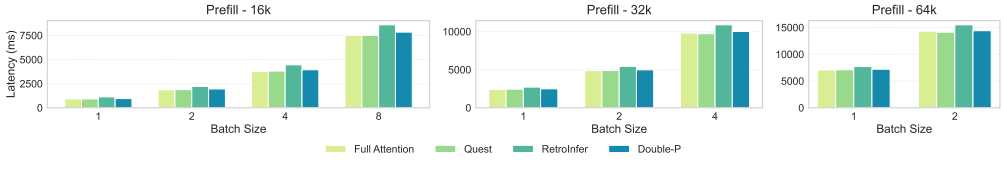


Figure 10: Prefill latency across methods, context lengths, and batch sizes. Double-P adds only a small clustering overhead, which lower decoding cost quickly offsets.

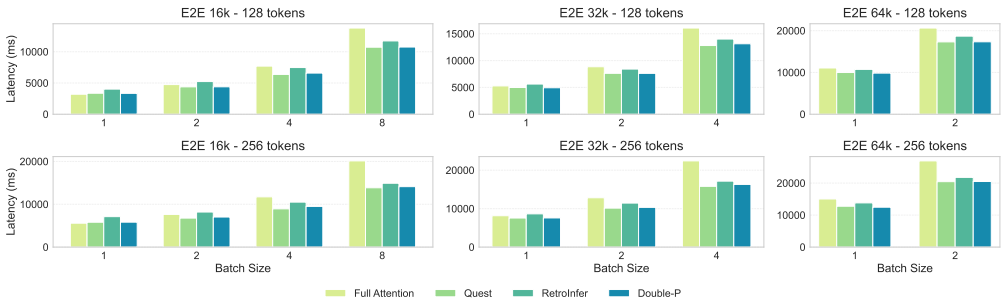


Figure 11: End-to-end latency across context lengths, batch sizes, and output lengths. Double-P outperforms full attention already at 128 output tokens.

Comparison with trainable sparse attention. Table 13 compares decoding latency with MoBA (Lu et al., 2025) at a 25% budget on Llama-3.1-8B. Double-P is faster in every setting.

Table 13: Decoding latency per output token (ms) of MoBA and Double-P.

Context	Batch	MoBA	Double-P
16K	1	26.60	18.61
	2	26.85	19.23
	4	26.88	20.76
	8	26.63	22.73
32K	1	26.15	19.30
	2	26.66	20.76
	4	26.86	23.69
64K	1	27.58	20.43
	2	OOM	22.80

Effect of the target mass. Table 14 varies the target mass on the RULER needle-in-a-haystack task at 32K context with 128 output tokens (Llama-3.1-8B). The effective sparsity is the number of attended entries (exact tokens plus approximated clusters) divided by the context length, since both contribute to kernel computation.

Table 14: Selected tokens and clusters, effective sparsity, and per-layer kernel time as the target mass varies.

Mass	Exact tokens	Approx. clusters	Effective sparsity	Gather (μs)	Sparse attention (μs)
0.60	1119.50	88.85	3.69%	45.19	81.44
0.70	1483.34	143.87	4.97%	48.25	83.82
0.80	1955.27	240.53	6.70%	52.02	86.95
0.90	2541.24	446.82	9.12%	59.24	91.75
0.95	2922.57	671.73	10.97%	63.45	92.91
0.99	3269.44	1141.12	13.46%	67.17	95.11

C Memory Footprint

Table 15 lists the per-token GPU memory footprint over all 32 layers of Llama-3.1-8B in BF16. Double-P stores the same index as RetroInfer: centroids, value sums, and cluster metadata take 6.5%, and a shared gather workspace that scales with context length takes 3.3%. Its two-stage selection keeps only per-head cutoff state. Twilight instead keeps the full-precision KV cache plus a quantized copy of the keys used only for pruning; even with ideal 4-bit packing, this copy takes about 32 KiB per token for the keys and 2 KiB per token for the per-token scales and zero points. Composed with RetroInfer, Twilight also needs RetroInfer's index, for at least 36.4% over full attention before its separate gather workspace.

Table 15: Per-token GPU memory footprint on Llama-3.1-8B (BF16, 32 layers).

Method	Footprint (KiB/token)	Overhead vs. full attention
Full attention	128.0	—
RetroInfer	140.6	+9.8%
Double-P (ours)	140.6	+9.8%
Twilight (KV cache + ideal 4-bit key copy)	162.0	+26.6%
RetroInfer+Twilight	$\geq$174.6	$\geq$+36.4%

D Long Generation

Update policy. Newly generated tokens are attended exactly within a dense window. Every 1,024 generated tokens, Double-P clusters the oldest 1,024 of them into 64 new centroids and appends these to the index; existing centroids and KV pages are not recomputed. Each update event therefore has a constant cost, and the number of events grows linearly with the generation length.

Accuracy. We evaluate Qwen3-8B in thinking mode on AIME 2024, whose reasoning traces contain up to 38,912 generated tokens. Full attention, RetroInfer, and Double-P all solve 23 of 30 problems (76.7% pass@1), so incremental clustering causes no observable accuracy loss in long reasoning traces.

Overhead. Table 16 reports the measured cost of index updates. The cost per event stays essentially constant as the index grows, and updates account for less than 0.2% of decoding time.

Table 16: Cost of incremental index updates during generation.

Model	Cost per event	Amortized cost	Fraction of decode time
Qwen3-8B (36 layers)	68.7 ms	0.067 ms/token	0.15%
Llama-3.1-8B (32 layers)	57.3 ms	0.056 ms/token	0.18%

E Cluster Failure Mode

Centroid-based mass estimates can underestimate an isolated informative token whose cluster is otherwise dissimilar to the query. To measure how often this happens, we instrument Double-P on 250 RULER needle-in-a-haystack samples at 32K context with Llama-3.1-8B, giving 64,000 layer and KV-head observations at the answer step. For each observation, we find the gold-answer token with the highest attention weight and record whether Stage 1 keeps its cluster and whether Stage 2 computes that cluster exactly or approximates it. Table 17 groups the observations by the needle's attention weight. The probability that the needle's cluster survives Stage 1 rises sharply with the attention the needle receives: Stage 1 almost never drops it when the needle is strongly attended, and the dropped cases concentrate in heads that give the needle little attention. An oracle ranking by exact cluster mass makes consistent decisions in these cases, suggesting that the centroid-based ranking rarely misses the needle's cluster in this evaluation.

Table 17: Fate of the needle's cluster, grouped by the needle's attention weight (RULER, 32K, Llama-3.1-8B).

Needle attention	Heads	Kept by Stage 1	Exact in Stage 2	Approx. in Stage 2	Dropped
$< 10^{-4}$	14,508	31.0%	2.9%	28.1%	69.0%
10^{-4}–10^{-3}	14,132	88.9%	38.2%	50.7%	11.1%
10^{-3}–5×10^{-3}	9,676	98.7%	77.4%	21.3%	1.31%
5×10^{-3}–10^{-2}	4,709	99.4%	89.8%	9.6%	0.55%
10^{-2}–5×10^{-2}	10,013	99.9%	94.4%	5.4%	0.13%
$> 5 \times 10^{-2}$	10,962	100.0%	97.7%	2.3%	0.03%

F SENSITIVITY ANALYSES

Clustering configuration. Table 18 varies the average cluster size and the number of k-means iterations on RULER at 32K context with 128 output tokens. Smaller clusters slightly improve accuracy because of their finer granularity, and larger clusters reduce prefill cost. The number of iterations barely affects accuracy or decode latency, so clustering quality saturates quickly. We use an average cluster size of 16 and 10 iterations for all models and benchmarks.

Table 18: Sensitivity to cluster size and k-means iterations (RULER, 32K, Llama-3.1-8B). Latencies in ms.

Setting	Accuracy	Prefill	Decode	End-to-end
Cluster size 8	90.47	2553.48	19.91	5129.08
Cluster size 16	90.31	2484.81	19.30	4932.20
Cluster size 32	89.49	2418.83	19.03	4880.84
5 iterations	90.30	2412.60	19.18	4867.64
10 iterations	90.31	2484.81	19.30	4955.21
20 iterations	90.30	2561.98	19.13	5010.62

Threshold calibration. Figure 12 shows the PG-19 calibration curves used to pick (p_1, p_2) for Llama-3.1-8B (§4.5).

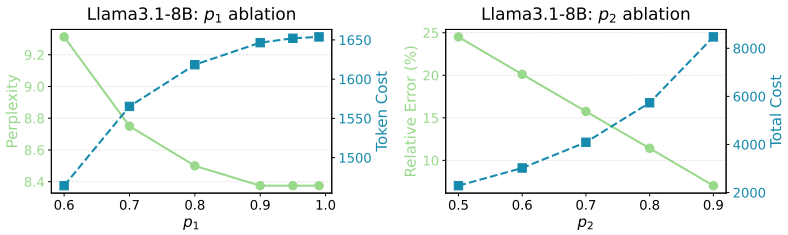


Figure 12: **Threshold calibration** on PG-19 with Llama-3.1-8B. Perplexity saturates as p_1 grows (left), and the error–computation curve of p_2 has a clear knee (right). We pick $p_1 = 0.95$ and $p_2 = 0.7$.

Thresholds and downstream accuracy. Figure 13 shows the accuracy–latency trade-off of different (p_1, p_2) configurations on the RULER CWE task at 32K context. The trade-off is smooth: increasing p_1 beyond the calibrated value gives little additional accuracy, and changing p_2 around the knee trades accuracy for computation gradually rather than causing abrupt failures.

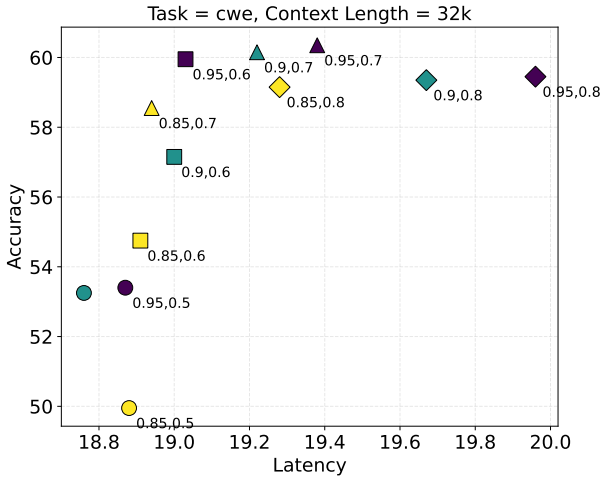


Figure 13: Accuracy and latency of different Double-P configurations on the RULER CWE task at 32K context.

G Limitations and Future Work

Isolated informative tokens. A cluster's centroid summarizes the average key direction, so a single token with a high query score can be underestimated if the other tokens in its cluster are dissimilar to the query. If Stage 1 skips that cluster, Stage 2 cannot recover the token. Appendix E shows that this is rare for strongly attended tokens, and the risk can be lowered further with smaller clusters or a larger p_1, at additional cost.

Per-model calibration. The thresholds (p_1, p_2) reflect a model's attention distribution and are calibrated once per model on held-out data; they are not tuned per task or benchmark. A calibration-free rule would simplify deployment.

Deeper hierarchies. Double-P uses a single cluster level. Multi-level indices in key space could apply coarse-to-fine top-p over several levels of summaries, further reducing estimation and sparse attention cost while preserving attention mass adaptively. Deeper hierarchies also add selection and indexing overhead, which makes the accuracy–efficiency trade-off more complex; we leave their exploration to future work.