THE MODEL KNOWS WHERE TO LOOK: DISTILLING ATTENTION SELECTORS FOR LONG-CONTEXT INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Language models now process whole documents, conversations and codebases, but generating each token requires reading a growing key-value cache. Attention often concentrates on a small fraction of that context, allowing sparse attention to reduce these reads. Selecting the relevant keys, however, introduces its own cost: the selector must inspect the cache or score a compressed representation of it. We present Highlight, a lightweight selector that skims the full context and *highlights* the relevant blocks for the model to attend to. It scores the context with a few narrow attention heads over its own small key cache, and selects 16-token blocks. It learns from the frozen model's own attention, through KL divergence on the tokens it selects. On five full-attention and hybrid models from 4B to 32B, accuracy stays close to dense attention on long-context benchmarks up to 128K tokens. On Qwen3-8B, every speedup we report comes from a configuration that stays within two points of dense or above it on each benchmark. At KV-cache saturation, selecting a quarter of the context yields $1.4\times$ to $1.8\times$ dense throughput. Adaptive top-p selection sets a budget for each query and layer, retaining 12 to 19 percent of the context and reaching up to $1.9\times$ throughput. Quantizing the selector's key cache to four bits raises throughput up to $2.5\times$ dense. Highlight also selects blocks up to 8 times faster than Twilight, an adaptive top-p selector, while both stay within two points of dense on held-out examples.

1 INTRODUCTION

Language models now take whole documents, long conversations and codebases as context. As the context grows, so does the key-value (KV) cache, which the model reads again for every token it generates. Reading this cache from GPU memory can limit decoding speed. Attention, however, is sparse: it concentrates on a small fraction of the context (Zhang et al., 2023; Tang et al., 2024), so most of what the model reads contributes little to the next token. To make the model generate faster, the key problem is deciding which tokens to read and which to skip. The decision itself is costly: the scores that identify relevant context come from reading it. A selector must therefore estimate each token's relevance from something cheaper to read.

Existing selectors take different routes. Quest and Twilight build the estimate by hand from the keys (Tang et al., 2024; Lin et al., 2025), so the relevance they measure is fixed by design rather than learned from the model. DeepSeek Sparse Attention (DSA) learns a token indexer but adapts the pretrained model to sparse attention through continued training (DeepSeek-AI, 2025). NSA and MoBA go further and train sparse attention into the model itself (Yuan et al., 2025; Lu et al., 2025). SeerAttention-R distills a gate for a frozen model, targets long reasoning generations and reports a sparse-decoding kernel speedup (Gao et al., 2025).

In this paper, we present Highlight, a block selector distilled from a frozen model's own attention. It skims the full context and *highlights* the relevant blocks for the model to attend to. The model already knows where it looks: its attention is the relevance signal a selector needs, and the model computes that signal on any text. Highlight learns to predict it with a few narrow heads over a small key cache of its own. At each decoding step, it selects the KV-cache blocks that each layer attends to. The selector trains by KL divergence against the model's attention on general pretraining text.

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.

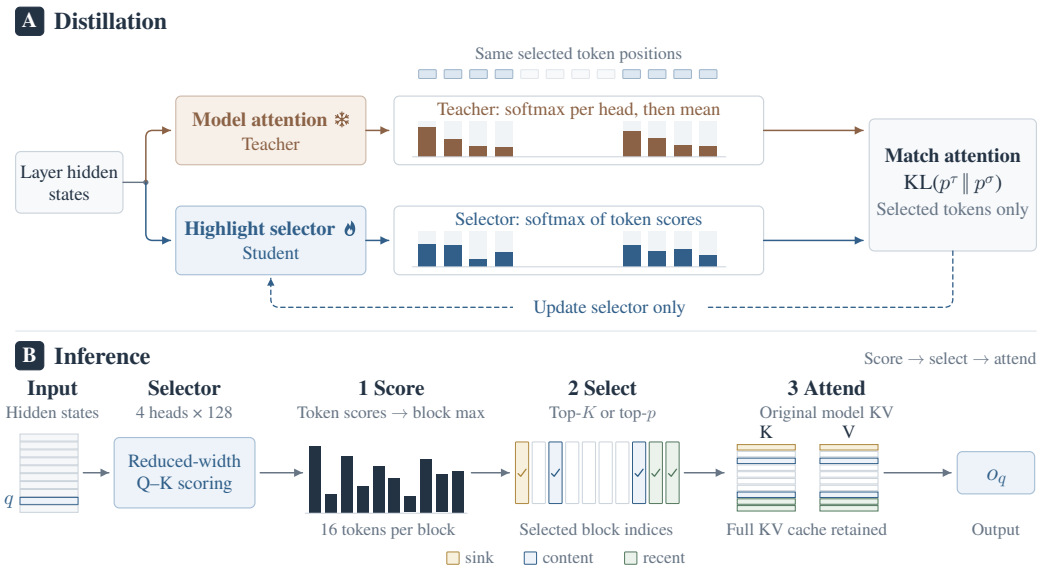

Figure 1: **Highlight learns where the model looks, then highlights the blocks worth reading.** **(A)** Training: the teacher and the selector take the same hidden states. On the tokens the selector picks, its outputs are compared with the teacher's, and KL updates only the selector's weights. **(B)** Inference: at each decoding step, the selector scores every cached token and selects blocks under a top-K or dual top-p budget. Attention then reads the original keys and values of those blocks only.

Highlight follows the design of the DSA lightning indexer and adapts it to a frozen pretrained model, selecting blocks from a selector key cache kept in BF16 or quantized formats. We show that models keep their accuracy close to dense attention. We implement Highlight in vLLM (Kwon et al., 2023) and measure its speed across models and sequence lengths. On Qwen3-8B at KV-cache saturation, it reaches up to $2.5\times$ dense throughput, and at settings that pass a held-out accuracy check, its selector runs up to 8 times faster than Twilight's top-p selector.

Contributions.

- **Distill.** We distill a frozen model's attention into Highlight, a block selector with a simple architecture, using general pretraining text. The selector keeps accuracy close to dense attention on five Qwen3 and Qwen3.5 models from 4B to 32B. On the two Qwen3.5 models, selectors trained at 16K tokens track dense accuracy up to 128K without retraining.
- **Decide.** Highlight decides which blocks to read under a fixed top-K budget or an adaptive dual top-p budget. At $p = 0.8$ on Qwen3-8B, the adaptive budget reads 12 to 19% of the context.
- **Deploy.** On Qwen3-8B at KV-cache saturation, adaptive selection with four-bit selector keys reaches up to $2.5\times$ dense throughput. The selector call is up to 8 times faster than Twilight's, and from 8K to 255.5K tokens it is 1.4 to 3.9 times faster than Quest's.

2 Distilling the Selector

We attach a small selector to every full-attention layer of a frozen model and distill that layer's attention into it, as shown in Figure 1A. During training, the layer's attention supervises the selector on the tokens it chooses. Dense attention is needed only for this supervision. At inference, the layer attends only to the blocks the selector picks. Appendix A gives the full algorithm and how each model family integrates the selector.

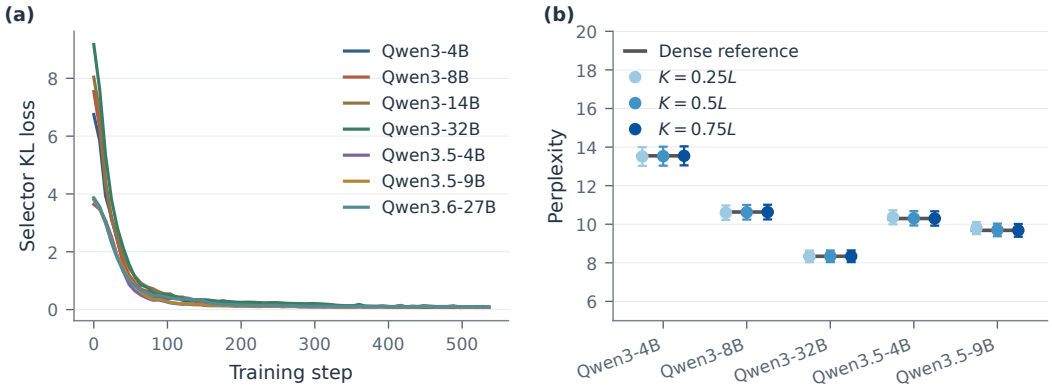

Figure 2: **The selector converges quickly, and sparse perplexity tracks dense.** **(a)** The loss per training step for the seven models. **(b)** Perplexity of the five models of the accuracy study at $K \in \{0.25, 0.5, 0.75\}L$ against dense attention. Error bars show ± 1 standard error over the five 16K windows.

2.1 Selector architecture

The selector estimates attention with a few narrow heads. From the current hidden state, it computes $H_{\text{idx}} = 4$ query heads $\mathbf{q}^I_{q,h}$ of width $d_{\text{idx}} = 128$ and one mixing weight $w_{q,h}$ per head. Each past token gets one key $\mathbf{k}^I_{h,t}$ per head. The key is computed once from the token's hidden state and stored in a selector key cache, separate from the model's KV cache. The score of cached token t for query q is

$$s_{q,t} = \sum_{h=1}^{H_{\text{idx}}} w_{q,h}\, \mathrm{ReLU}\big(\mathbf{q}^I_{q,h} \cdot \mathbf{k}^I_{h,t}\big), \tag{1}$$

where the mixing weights carry the scale $H_{\text{idx}}^{-1/2} d_{\text{idx}}^{-1/2}$. The selector thus learns a cheaper representation for choosing context, and attention still reads the original keys and values.

The selector chooses blocks of 16 tokens so that attention runs efficiently on tensor cores. Our vLLM deployment also stores the KV cache in pages of 16 tokens, so each block is one page. Blocks decide only which keys are read: the selector's scores, the model's attention and the loss stay per token.

During training, each tile Q_i of 16 consecutive queries shares one block set, so the training kernel loads each selected block once for all 16 queries. A block's score averages the token scores over the tile and takes the maximum within the block, and the highest-scoring blocks form the tile's support. During decoding, each query selects its own blocks, so training selects per tile while decoding selects per query. Appendix K.1 shows that training with per-query selection instead makes no measurable difference.

2.2 KL over selected tokens

The selector learns by matching the model's attention on the tokens it selects. For a query q, let $\mathcal{T}_q$ be the positions in its tile's selected blocks that q can see. The teacher takes the model's attention logits $a_{h,q,t}$ for each of its H_{attn} heads, normalizes each head over $\mathcal{T}_q$ and averages the heads. The student normalizes the selector's token scores over the same positions:

$$p^{\tau}_{q,t} = \frac{1}{H_{\text{attn}}} \sum_{h=1}^{H_{\text{attn}}} \mathrm{softmax}_{t \in \mathcal{T}_q}(a_{h,q,t}), \qquad p^{\sigma}_{q,t} = \mathrm{softmax}_{t \in \mathcal{T}_q}(s_{q,t}). \tag{2}$$

Selection uses the tile means and block maxima, but the loss uses each query's own token scores. For each layer, the loss is the KL divergence between the two, averaged over all valid query positions in the batch:

$$\mathcal{L}_{\text{KL}} = \mathbb{E}_q\, \mathrm{KL}\big(p^{\tau}_q \,\|\, p^{\sigma}_q\big) = \mathbb{E}_q \sum_{t \in \mathcal{T}_q} p^{\tau}_{q,t} \log \frac{p^{\tau}_{q,t}}{p^{\sigma}_{q,t}}. \tag{3}$$

Model	Dense	Highlight top-K			Highlight top-p		
		$0.25L$	$0.5L$	$0.75L$	0.7	0.8	0.9
Qwen3-4B	54.3 (1.9)	51.7 (1.9)	52.4 (1.9)	52.8 (1.8)	50.5 (2.0)	51.9 (1.9)	52.3 (1.9)
Qwen3-8B	53.7 (1.9)	54.3 (1.9)	54.4 (1.9)	54.4 (1.9)	52.5 (2.0)	54.0 (2.0)	54.9 (2.0)
Qwen3-32B	59.9 (2.0)	57.5 (2.0)	58.1 (2.0)	57.7 (1.9)	51.6 (2.2)	54.1 (2.1)	57.2 (2.0)
Qwen3.5-4B	56.1 (1.4)	56.4 (1.4)	56.9 (1.4)	55.7 (1.4)	56.3 (1.4)	57.0 (1.4)	57.2 (1.4)
Qwen3.5-9B	55.8 (1.5)	56.4 (1.3)	56.6 (1.4)	56.6 (1.3)	57.3 (1.3)	57.9 (1.4)	56.6 (1.3)

Table 1: **Highlight stays close to dense accuracy.** Percent accuracy, with the bootstrap standard error in parentheses, averaged equally over RULER, BABILong and LongBench-v2. InfiniteBench runs only on Qwen3.5 and stays out of the mean. Inputs reach 32K tokens on Qwen3 and 128K on Qwen3.5. Appendix F gives each benchmark and the paired differences.

We average this loss over layers and update only the selector. Appendix K.2 compares it with distilling the full attention distribution.

2.3 Training

We train selectors for seven models from three families: Qwen3 (Yang et al., 2025) at 4B, 8B, 14B and 32B, Qwen3.5 at 4B and 9B, and Qwen3.6 at 27B. Qwen3 uses full attention in every layer. Qwen3.5 and Qwen3.6 are hybrids whose linear-attention layers keep a fixed-size state instead of a KV cache, so only their full-attention layers receive a selector. Every selector trains for 540 AdamW steps (Loshchilov & Hutter, 2019) on 142M tokens of FineWeb-Edu (Penedo et al., 2024). Training sequences are $L = 16$K tokens long, and the training budget is 8K selected tokens. Appendix B lists the full recipe and the selector sizes.

Figure 2a shows that the selector converges quickly. On every model, the loss falls by more than 95% within the first 200 steps and ends between 0.07 and 0.10 nats. Selection also preserves perplexity, as shown in Figure 2b. On 16K windows of FineWeb-Edu, drawn with a different shuffle from training, sparse perplexity stays within 0.4% of dense at every budget, except $K = 0.25L$ on the two Qwen3.5 models, which add 0.6% and 1.2%. The model is frozen, so the gap comes from selection alone. Appendix C shows that the fixed budgets also cost no measurable perplexity on Python code, which the selectors never saw.

3 Selecting what to read

At each decoding step, every layer decides which blocks of the context to read. Prefill stays dense, and selection runs only while decoding. A block's score is the maximum of its token scores, $\bar{s}_{q,b} = \max_{t \in b} s_{q,t}$, and attention reads the original keys and values of the chosen blocks. We use two rules, which differ in what they hold fixed: the number of blocks, or the attention mass the blocks must cover.

Fixed top-K. A budget of K tokens keeps the $K/16$ highest-scoring blocks. We set K as a fraction of the input length L, so every query and layer reads the same share of the context, whether its scores are concentrated or spread.

Dual top-p. Top-p fixes a mass instead of a count: it keeps the fewest blocks whose estimated attention mass reaches p. A concentrated estimate reaches p with a few blocks and a spread one needs more, so the budget adapts to each query and layer. Sink and recent tokens, however, often draw much of the attention (Xiao et al., 2024), and a single threshold can be spent on them before other content is selected. Dual top-p therefore selects in two parts. The first part is fixed: it always keeps the first 128 tokens and the 256 most recent tokens. The second part is adaptive: over the remaining blocks $\mathcal{B}_{\text{res}}$, it turns the block scores into an estimated attention distribution,

$$\widetilde{p}_{q,b} = \operatorname*{softmax}_{b \in \mathcal{B}_{\text{res}}} \bar{s}_{q,b}, \tag{4}$$

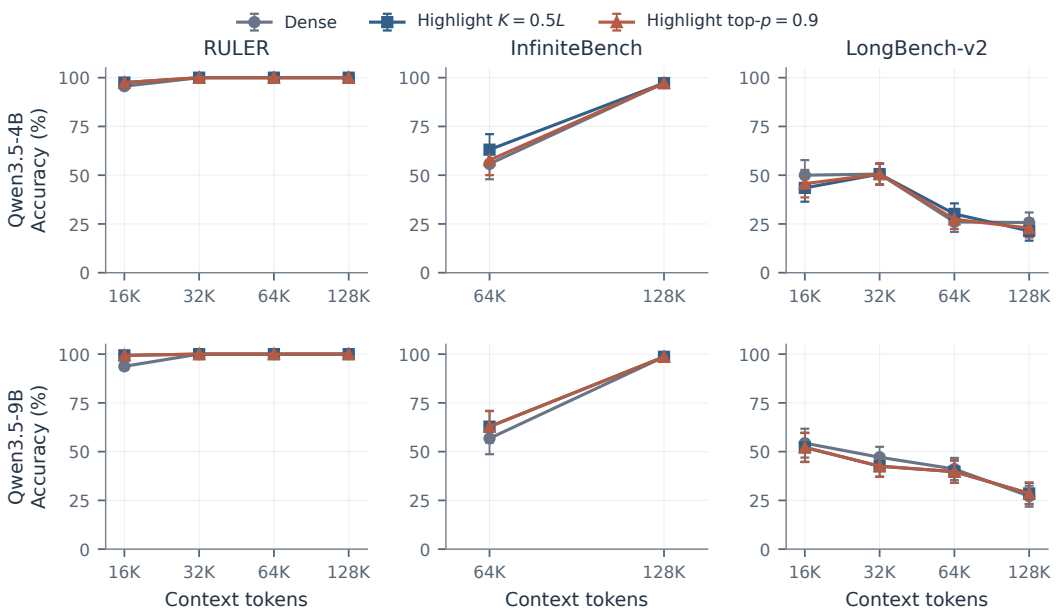

Figure 3: **Selectors distilled on 16K pretraining text track dense accuracy up to 128K without retraining.** Percent accuracy per length bin on the two Qwen3.5 models for dense attention and Highlight at $K = 0.5L$ and top-$p = 0.9$, with bootstrap standard errors. Beyond 16K, the bins hold 9 to 14 RULER examples, 29 and 72 InfiniteBench examples, and 70 to 87 LongBench-v2 examples. The InfiniteBench bins differ in task: 64K holds book QA and book choice, and 128K is mostly passkey retrieval.

and applies nucleus selection (Holtzman et al., 2020): it adds blocks in descending order of $\widetilde{p}_{q,b}$ until their mass reaches p. If the selected blocks still hold fewer than $K_{\min} = 512$ tokens, selection continues in score order until they do.

4 Accuracy

We measure the accuracy of Highlight on five models: the full-attention Qwen3-4B, 8B and 32B, and the hybrid Qwen3.5-4B and 9B. We evaluate them on RULER (Hsieh et al., 2024), BABILong tasks qa1 to qa3 (Kuratov et al., 2024), InfiniteBench (Zhang et al., 2024) and LongBench-v2 (Bai et al., 2025), with inputs up to 32K tokens on Qwen3 and 128K on Qwen3.5. InfiniteBench inputs exceed the Qwen3 cap, so only Qwen3.5 runs it. Every run generates in non-thinking mode. We test three settings of each rule: top-K at $K \in \{0.25, 0.5, 0.75\}L$ and dual top-p at $p \in \{0.7, 0.8, 0.9\}$. Appendix E gives sample counts, length bins, decoding settings, prompts and scoring.

Table 1 shows that Highlight stays close to dense accuracy, and that models differ in how they respond. Qwen3-8B and both Qwen3.5 models stay within half a point of dense or above it at every fixed budget, and all three gain at $p = 0.9$. The gains reach 1.2 points on Qwen3-8B and 1.1 on Qwen3.5-4B at $p = 0.9$, and 2.1 on Qwen3.5-9B at $p = 0.8$. Qwen3-4B and Qwen3-32B lose 1.5 to 2.7 points at the fixed budgets and at $p = 0.9$. All these gains and losses stay within one and a half standard errors of the dense benchmark mean, and reading three times more of the context recovers at most 1.1 of the lost points.

At low thresholds, the adaptive budget can become very small. At $p = 0.7$, selection stops once the kept blocks hold 70% of the estimated mass. On Qwen3, the estimate is often concentrated enough that a few blocks reach this mass: Appendix J.2 shows that at 16K, 14 to 25% of calls fall to the 512-token floor, which is 3% of the context, against 3 to 5% on Qwen3.5. Retrieval suffers most, and RULER drops 5 to 16 points on all three Qwen3 models. At $p = 0.8$, every model except Qwen3-32B is close to its fixed-budget accuracy or above it, and both Qwen3.5 models gain accuracy even at

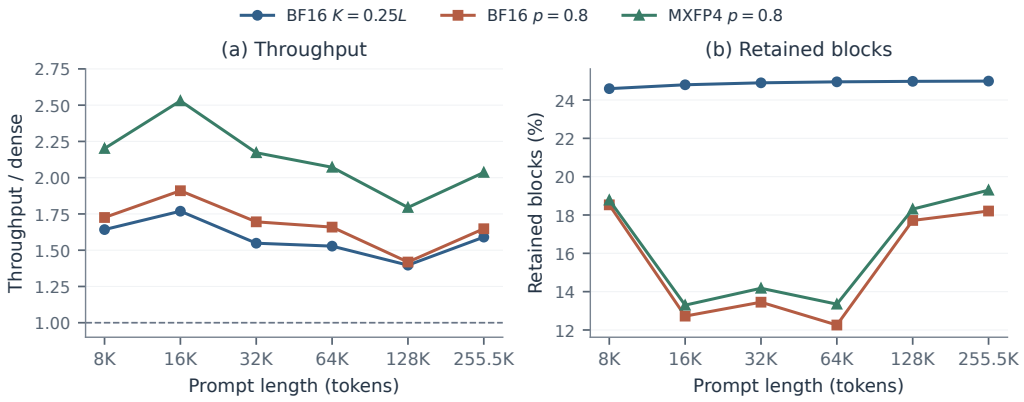

Figure 4: **Reading less raises throughput at every length.** Qwen3-8B on one B200, 256 generated tokens per request, one measurement per cell. Each configuration fills its own KV pool. **(a)** Aggregate throughput relative to dense attention, which the dashed line marks. **(b)** Share of blocks retained, over decode queries and sparse layers.

the low thresholds. Appendix F gives each benchmark and the paired difference from dense, and Section 5.3 compares Highlight with Twilight on the same decode path.

Highlight is trained only to estimate the model's attention. Its loss is the KL divergence between the model's attention and the selector's estimate, with no language-modeling loss, so the selector learns where the model attends rather than what it predicts. We train it only on 16K windows of FineWeb-Edu. Figure 3 shows that the estimate carries over to longer inputs and other tasks. On the two Qwen3.5 models, whose evaluation reaches 128K, the selectors track dense accuracy without retraining. RULER is unchanged in every bin beyond 16K. InfiniteBench gains 2.0 to 7.3 points at 64K and is unchanged at 128K, and LongBench-v2 moves by -4.6 to $+4.1$ points beyond 16K, with no trend in length. Appendix C shows the same transfer on Python code, which the selectors never saw. Appendix D plots the model's attention and the selector's estimate as heat maps on a 16K RULER prompt, at six layers of each model. The estimate reproduces the model's attention across layers and query positions, and at the query that produces the answer, the selector ranks a block of the answer needle first on six of the seven models.

5 Decoding speed

To measure the speed of Highlight, we implement it as a plugin for the vLLM engine (Kwon et al., 2023). The plugin runs the selector in every layer at each decoding step and passes stock FlashAttention (Dao et al., 2022) a compacted table of the selected blocks. For comparison, dense attention runs on the same checkpoint and backend with the selector turned off. Speed matters only where accuracy holds. Every Qwen3-8B configuration we report therefore passes an accuracy check on the serving engine itself: on 160 RULER, 300 BABILong and 89 LongBench-v2 examples, it stays within two points of dense or above it on each benchmark. Each configuration fills its own KV pool with requests of distinct real-text prompts, so the selector's key cache takes memory from the requests in flight. Every request is prefilled densely and generates 256 tokens on one NVIDIA B200 GPU. We test six prompt lengths: 8K, 16K, 32K, 64K, 128K and 255.5K tokens. To separate prefill from decoding, we add a barrier that holds decoding until the last request in the batch has prefilled. The measurement window runs from the moment every request has its first token to the moment the first request finishes, so prefill and the final drain are excluded. Throughput is the number of output tokens in the window divided by its length, and the step time is the number of requests in flight divided by throughput. Appendix G gives engine limits, position scaling and acceptance rules.

5.1 Throughput at KV saturation

At KV saturation, every configuration serves as many requests as its KV memory holds. Figure 4 shows that reading less raises throughput at every length. On Qwen3-8B with BF16 selector keys,

Policy	Held-out accuracy (%) RULER	BABILong	LB-v2	Largest drop	Selector call ms	Twilight / Highlight
Dense	98.41	33.33	23.19	0.00		
Twilight, pool 8192, $p = 0.85$	96.83	32.50	23.19	1.59	0.2294	
Highlight $K = 0.25L$, BF16	97.62	34.17	26.09	0.79	0.0284	8.04
Highlight $K = 0.5L$, BF16	97.62	32.50	26.09	0.83	0.0284	8.04
Highlight $p = 0.9$, BF16	97.62	34.17	26.09	0.79	0.0329	6.90
Highlight $p = 0.9$, MXFP4	97.62	34.58	26.09	0.79	0.0351	6.51

Table 2: **Every timed selector passes the held-out accuracy check, and Twilight's call costs 6.5 to 8.0 times Highlight's.** Qwen3-8B. Accuracy is in percent on the held-out split, $n = 126$, 240 and 69. The largest drop is the largest loss against dense, and the check allows two points. The selector call is the median time of one layer call over 30 decode queries at 15K to 28K tokens, on one B200 at batch 1. The ratio is the median of the per-query ratios. Table 22 gives the paired 95% intervals and the range of the per-query ratios.

reading a quarter of the context gives $1.40\times$ to $1.77\times$ dense throughput across the six lengths. Adaptive selection reads less and gains more: top-$p = 0.8$ keeps 12.3 to 18.5% of the blocks and reaches $1.42\times$ to $1.91\times$. The selector's cache takes part of the gain back. At 8K it displaces about a fifth of the requests, 89 in flight instead of 112, so the decode step shortens more than throughput rises: at $K = 0.25L$ the step falls from 32.5 to 15.7 ms, a factor of 2.1, while throughput rises by a factor of 1.64. Appendix G lists requests, step times and retained fractions for every cell, and times three more models.

5.2 Quantized selector keys

The selector scans its whole key cache at every step, so a smaller cache means fewer bytes read and more memory for requests. We quantize only the selector keys. The model's weights and KV cache stay in BF16. The selector still trains in BF16, and at serving time its keys are stored in FP8, INT8 or block-scaled MXFP4 and scored with native low-precision matrix multiplication on B200. Including scale metadata, a key row of 512 elements takes 1024 bytes in BF16, 528 in FP8 or INT8, and 272 in MXFP4, a $3.76\times$ reduction.

Accuracy. Ten of the twelve quantized configurations pass the check. FP8 and INT8 at top-$p = 0.8$ lose four of 160 RULER examples, 2.5 points, while MXFP4 at the same threshold passes. Appendix H gives every setting.

Throughput. Figure 4 shows that four-bit keys raise throughput further at every length. With MXFP4 keys, reading a quarter of the context gives $1.82\times$ to $2.29\times$ dense throughput, and top-$p = 0.8$ gives $1.79\times$ to $2.53\times$. At $K = 0.25L$, FP8 and INT8 fall between BF16 and MXFP4. MXFP4 helps in two ways: it admits more requests and shortens the step. At 16K, top-$p = 0.8$ admits 54 requests with MXFP4 instead of 46 with BF16, and the decode step still shortens from 14.1 to 12.5 ms. From 8K to 128K at top-$p = 0.8$, the MXFP4 step is 5 to 11% shorter than the BF16 step while serving more requests. At 255.5K, MXFP4 serves three requests instead of two, so its step is 21% longer, and throughput still rises from $1.65\times$ to $2.04\times$ dense.

5.3 Selector cost

Running a selector adds a cost to every decoding step. Throughput masks this cost within the whole step, so we time the selector on its own and compare it with existing selectors. We time one selector call per layer at batch size one. The call includes cache maintenance, any method-specific projection or query conversion, scoring, selection and compaction. The model's projections and KV writes, attention and the rest of the model fall outside the timer. On the same Qwen3-8B model, we compare with Quest for a fixed budget and with Twilight for an adaptive top-p budget, and run their released kernels with the correctness repairs of Appendix I.1. Quest keeps a fixed pool of 8192 candidate

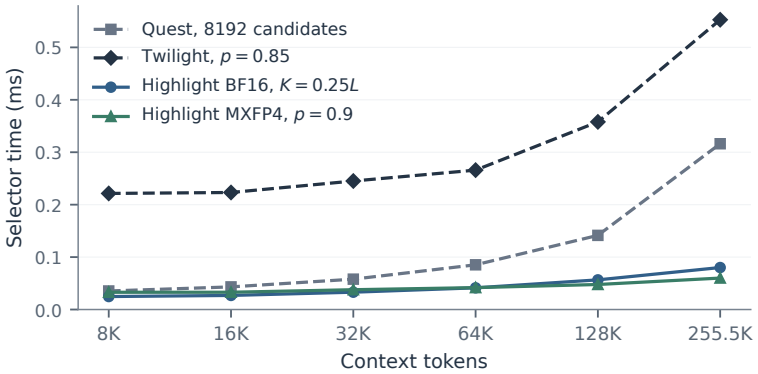

Figure 5: **Selector cost grows with context, and Highlight adds the least time.** Milliseconds per layer call on Qwen3-8B, one B200, batch 1, CUDA graphs, replaying the final-prefill query of one prompt per length. Each point is the mean of three independent runs, with standard deviations below 0.0002 ms. Quest keeps 8192 candidates, Twilight prunes that pool at $p = 0.85$, and Highlight runs top-$K = 0.25L$ in BF16 and top-$p = 0.9$ in MXFP4.

tokens, the setting of Twilight's own cost breakdown, and Twilight prunes that pool at its released top-$p = 0.85$.

Held-out accuracy. We compare Twilight and Highlight only at settings that pass a second accuracy check with the same two-point rule. On a held-out split of RULER, BABILong and LongBench-v2, 435 examples of 12K to 32K tokens, Twilight and four Highlight settings run the same decode path against dense attention on the same examples. Twilight publishes no Qwen3 setting, so its pool and threshold were chosen on 114 separate development examples and fixed before the held-out split ran. Table 2 shows that every sparse setting stays within two points of dense or above it on every benchmark: Twilight loses at most 1.59 points, and Highlight at most 0.83.

Real decode queries. We then time Twilight and Highlight on 30 decode queries generated by the dense model from six prompts, at layers 2, 18 and 35 and 15K to 28K tokens of context. The same table shows that Twilight's call costs $8.0\times$ Highlight's at top-K, $6.9\times$ at BF16 top-p and $6.5\times$ at MXFP4 top-p, and the ratio never falls below $5.5\times$ on any single query.

Out to 255.5K. Figure 5 shows that the margin holds as the context grows. Replaying one prompt per length from 8K to 255.5K tokens, three runs each, every call grows with the context it scores: Quest from 0.035 to 0.316 ms, Twilight from 0.221 to 0.553 ms, and Highlight from 0.025 to 0.080 ms at top-$K = 0.25L$ and from 0.033 to 0.060 ms at MXFP4 top-p. Highlight adds the least time. Quest costs $1.4\times$ to $3.9\times$ its top-K call, and Twilight costs $6.3\times$ to $9.2\times$ its MXFP4 top-p call. Twilight's call is Quest's plus a pruning stage that stays between 0.18 and 0.24 ms at every length.

5.4 What sets the cost

Highlight changes two reads in each decoding step. The selector scans its own key cache, and attention reads the selected blocks. Different things set the cost of each.

The selector's scan. The context length and the key precision set the selector's cost. The size of the selection barely changes it. Doubling Highlight's budget from a quarter to half of the context changes its call by under 1%. On Qwen3-8B, Quest at a quarter of the context, eight times its fixed pool at 255.5K, costs at most 9% more. Quantized keys cut the bytes read but add a query conversion that does not grow with the context. They pay off once the read dominates: MXFP4 top-p costs about 7% more than BF16 on the decode queries at 15K to 28K, breaks even near 32K and costs 30% less at 255.5K. Under load, the saving already appears at 8K, as Section 5.2 shows. Appendix I adds Qwen3-4B and Qwen3-32B, whose calls stay within 15% of Qwen3-8B's, and what each selector retains and stores.

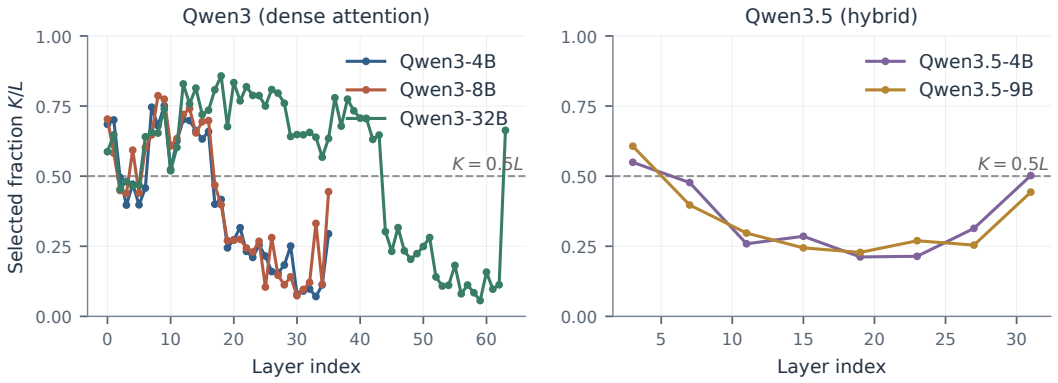

Figure 6: **Layers differ in how much they read.** Mean selected tokens per layer as a fraction of 16384, on 16K RULER inputs at top-$p = 0.9$. Left: every layer of Qwen3-4B, 8B and 32B. Right: the eight full-attention layers of each Qwen3.5 model. The dashed line marks a fixed budget of half the context. The fractions show how much each layer reads, not how much each layer matters.

Attention's read. The budget sets how much attention reads. A fixed budget gives every layer the same share of the context, while top-p lets each layer read what its selector needs. Figure 6 shows where this adaptive budget goes on 16K RULER inputs at $p = 0.9$. On Qwen3, the early and middle layers read 60 to 69% of the context on average, up to 86% in one layer, and the deep layers read about a fifth. On Qwen3.5, the first and last full-attention layers read the most, and the five middle layers read 21 to 31%. Each step reads the sum over layers, so on Qwen3 the deep layers carry most of the saving: 69, 70 and 54% of the skipped blocks on Qwen3-4B, 8B and 32B. Figure 4b reports the mean share over layers in the serving runs. Appendix J.1 shows the same profiles on BABILong and LongBench-v2 and at other thresholds, and Appendix J.2 follows the budget across context lengths.

6 Conclusion

A pretrained model already knows where it looks. We introduce Highlight, a selector that skims the context, estimates that attention and uses the estimate to read less. Highlight trains on general text with a KL loss between the frozen model's attention and its own estimate, on the tokens it selects. At each decoding step, it selects 16-token blocks under a fixed or adaptive budget, and the adaptive budget lets each layer read what it needs. The model's weights never change, and the selectors add at most 2.4% of its parameters.

Accuracy stays close to dense on five models from 4B to 32B, and on Qwen3.5, selectors trained at 16K tokens track dense accuracy up to 128K. On Qwen3-8B, every speedup we report comes from a configuration that stays within two points of dense or above it. At KV saturation, reading a quarter of the context gives $1.4\times$ to $1.8\times$ dense throughput, and adaptive selection with four-bit selector keys reaches up to $2.5\times$. At settings that pass a held-out accuracy check, the selector call is up to 8 times faster than Twilight's. From 8K to 255.5K tokens, it is also 1.4 to 3.9 times faster than Quest's.

Limitations

This paper distills a small selector that estimates the model's attention. Future work can use these estimates to select fewer tokens without losing accuracy, and to remove distracting tokens through sparsity. The same recipe trains every selector, but we test it only on Qwen-family models, plus Gemma-4-E4B in Appendix K.4, and only on English benchmarks. We measure accuracy inside each model's native context window. Latency tests extend Qwen3 past its window to 255.5K tokens with YaRN. We serve every model on one GPU with dense prefill. Tensor-parallel serving, which larger models need, and sparse prefill remain open. We keep the model frozen. Adapting it to sparse attention, as DSA does, is out of scope. All selection uses 16-token blocks, and larger blocks are left for future work.

Reproducibility statement

Appendix B gives the full selector recipe: architecture, optimizer, schedule and batch arithmetic. Appendix A describes the selection algorithm and its integration with each model. Appendix E gives the evaluation protocol: data and length bins, decoding settings, prompts, answer extraction, scoring and the bootstrap. Appendix C gives the perplexity protocol. Appendices G and I give the serving and selector-timing protocols, including engine settings, acceptance rules and the baseline repairs. Appendices F, G and H list every accuracy and serving result behind the summary numbers. Appendix K reports the controls and ablations. We will release the training code and the vLLM serving plugin.

AI use statement

We used LLM-based AI assistants to draft and revise text, to inspect, implement and debug code, and to check result summaries and visualizations. This code includes the selector kernels and the vLLM serving plugin. The ideas, method and experiment design, narrative, interpretation and conclusions are the authors' own work. The authors reviewed all AI-assisted text and code. Reported measurements come from the recorded experimental runs.

References

Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3639–3664, 2025. URL https://aclanthology.org/2025.acl-long.183/.

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:2603.12201*, 2026. URL https://arxiv.org/abs/2603.12201.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/67d57c32e20fd0a7a302cb81d36e40d5-Abstract-Conference.html.

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

Yizhao Gao, Zhichen Zeng, Dayou Du, Shijie Cao, Peiyuan Zhou, Jiaxing Qi, Junjie Lai, Hayden Kwok-Hay So, Ting Cao, Fan Yang, and Mao Yang. SeerAttention: Learning intrinsic sparse attention in your LLMs. *arXiv:2410.13276*, 2024. URL https://arxiv.org/abs/2410.13276.

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

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *International Conference on Learning Representations (ICLR)*, 2020. URL https://openreview.net/forum?id=rygGQyrFvH.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. KVQuant: Towards 10 million context length LLM inference with KV cache quantization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. URL https://papers.nips.cc/paper_files/paper/2024/hash/028fcbcf85435d39a40c4d61b42c99a4-Abstract-Conference.html.

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? In *Conference on Language Modeling (COLM)*, 2024. URL https://openreview.net/forum?id=kIoBbc76Sy.

Yuri Kuratov, Aydar Bulatov, Petr Anokhin, Ivan Rodkin, Dmitry Sorokin, Artyom Sorokin, and Mikhail Burtsev. BABILong: Testing the limits of LLMs with long context reasoning-in-a-haystack. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, 2024. URL https://arxiv.org/abs/2406.10149.

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. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*, 2023. URL https://dl.acm.org/doi/10.1145/3600006.3613165.

Zhiwei Li, Lei Zhu, Hao Gu, Xiang Hu, Yan Wang, Haitao Mi, Sirui Han, Leo Liang, and Zhijiang Guo. SAS: Simple attention sparsification via end-to-end optimization of context ranking. *arXiv:2609.13141*, 2026. URL https://arxiv.org/abs/2609.13141.

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. URL https://openreview.net/forum?id=Ve693NkzcU.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024. URL https://proceedings.mlr.press/v235/liu24bz.html.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

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 (NeurIPS)*, 2025. URL https://openreview.net/forum?id=RlqYCpTu1P.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben Allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The FineWeb datasets: Decanting the web for the finest text data at scale. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=n6SCkn2QaG.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. YaRN: Efficient context window extension of large language models. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/874a4d89f2d04b4bcf9a2c19545cf040-Abstract-Conference.html.

Luka Ribar, Ivan Chelombiev, Luke Hudlass-Galley, Charlie Blake, Carlo Luschi, and Douglas Orr. SparQ attention: Bandwidth-efficient LLM inference. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, pp. 42558–42583, 2024. URL https://proceedings.mlr.press/v235/ribar24a.html.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context LLM inference. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024. URL https://proceedings.mlr.press/v235/tang24l.html.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=NG7sS51zVF.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Yuxing 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 *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 23078–23097, 2025. URL https://aclanthology.org/2025.acl-long.1126/.

Xinrong Zhang, Yingfa Chen, Shengding Hu, Zihang Xu, Junhao Chen, Moo Khai Hao, Xu Han, Zhen Leng Thai, Shuo Wang, Zhiyuan Liu, and Maosong Sun. ∞Bench: Extending long context evaluation beyond 100K tokens. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15262–15277, 2024. URL https://aclanthology.org/2024.acl-long.814/.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/6ceefa7b15572587b78ecfcebb2827f8-Abstract-Conference.html.

Ruijie Zhou, Fanxu Meng, Yufei Xu, Tongxuan Liu, Guangming Lu, Muhan Zhang, and Wenjie Pei. MISA: Mixture of indexer sparse attention for long-context LLM inference. *arXiv:2605.07363*, 2026. URL https://arxiv.org/abs/2605.07363.

APPENDIX CONTENTS

A Algorithm and model integration **14**

B Training recipe and selector size **14**

C Perplexity diagnostics **16**

D Attention maps **17**

E Evaluation protocol **20**

E.1 Data configuration 20

E.2 Decoding 20

E.3 Prompts and answer extraction 20

E.4 Scoring 21

E.5 Bootstrap protocol 21

F Full accuracy results **22**

F.1 Accuracy grids 22

F.2 Which settings stay within two points of dense 23

G Throughput at KV saturation **25**

H Quantized selector keys **30**

I Selector cost **31**

I.1 Timing boundary and operating points 31

I.2 Accuracy qualification 32

I.3 Timing sweeps 33

I.4 Retained tokens and selector caches 34

I.5 Selector cost by cache format 35

J Adaptive budgets **35**

J.1 Per-layer budget across datasets and thresholds 35

J.2 Budget across context lengths and the floor 36

J.3 Kept share across models and lengths 37

K Controls and ablations **38**

K.1 Per-query selection during training 38

K.2 KL over the full distribution 39

K.3 Small fixed budgets 39

K.4 A model outside the Qwen families 40

L Related work **40**

A ALGORITHM AND MODEL INTEGRATION

Selector inputs and scores. The selector reads the layer's input hidden states with gradients detached, so no gradient flows into the model. It projects each state into four query heads and four key heads of width 128 and predicts one mixing weight per query head. Keys pass through a LayerNorm with a learned bias, and rotary position encoding acts on the first 64 dimensions of each head. The mixing weights carry the scale $H_{\mathrm{idx}}^{-1/2} d_{\mathrm{idx}}^{-1/2}$. Only these selector parameters train, and the model's parameters stay frozen. Each token stores one selector key of 512 elements, separate from its keys and values in the KV cache.

Training support and gradients. For each query tile, the selector computes per-query token scores, averages them over the tile and takes the maximum within each key block. The highest-scoring blocks under the training budget form the tile's shared block set, but each query keeps its own causal support $\mathcal{T}_{b,q}$. The model's attention and the selector's distribution are compared on those positions: the teacher normalizes each attention head before averaging the heads, and the student uses each query's own token scores, not the block scores. Block selection is discrete and receives no gradient. Gradients reach only the selector's scores on the selected support. Future positions and padding enter neither distribution, and the loss is averaged over the trained layers.

Decoding. Prefill uses dense attention and writes the selector keys of the prompt. At each decoding step, the selector projects the new hidden state, normalizes and position-encodes its key, and appends it to the selector key cache. It then scores every cached selector key and reduces the token scores to block scores. The physical identifiers of the chosen KV-cache blocks are compacted into a table that FlashAttention reads, so attention reads the original keys and values of those blocks only. Selection never evicts a token: a block skipped at one step can be selected at the next. The last, partly filled block keeps its true number of valid tokens.

Architecture coverage. Qwen3 receives a selector at every attention layer. Qwen3.5 and Qwen3.6 keep their linear-attention layers and attach selectors to their full-attention layers, whose attention gates stay in place. Gemma-4 mixes sliding-window layers, full-attention layers and layers that reuse an earlier layer's KV cache. Selectors attach only to the full-attention layers that own a cache, and the other layers keep standard attention. Appendix K.4 reports this model. All serving results use one GPU, a BF16 KV cache and dense prefill. The hybrid cache layout holds BF16 selector keys but not quantized ones, so the hybrids have no quantized serving results. Appendix H explains why.

Selection semantics. Top-K selects whole blocks: the token budget must be divisible by 16 and is clipped to the available context. Fractional budgets in the serving grid are set from the initial prompt length. Top-p first reserves the 128-token prefix and the 256-token recent window, then renormalizes the remaining block scores and keeps the fewest highest-scoring blocks whose mass reaches the threshold. The 512-token minimum includes the reserved blocks and is capped by the available context, and overlapping reserved regions count once. The threshold applies to the selector's estimated block mass, not to the model's attention. Appendix D measures how much of the model's attention the selected blocks hold on one prompt.

B TRAINING RECIPE AND SELECTOR SIZE

Table 3 lists the full recipe. All seven runs share it except the per-device batch and gradient accumulation, which vary with model size so that every run keeps an effective batch of 16 sequences. Qwen3-4B, Qwen3.5-4B/9B and Qwen3.6-27B use a per-device batch of 4 without accumulation, Qwen3-8B/14B a batch of 2 with two accumulation steps, and Qwen3-32B a batch of 1 with four.

Convergence. Table 4 lists the selected-token KL behind Figure 2a at three points of each run. The Qwen3 selectors start at 6.7 to 9.2 nats, and those of the hybrids at 3.6 to 3.9. Every run loses 95.8 to 97.6% of its initial KL within 200 steps and ends between 0.0745 and 0.0996 nats.

Selector size. Table 5 counts the selector parameters of each checkpoint and the layers that carry a selector. Every selector layer has the same shape: two projections from the hidden size to 512 for

Hyperparameter	Value
Selector architecture	
heads H_{idx}	4
head dim d_{idx}	128
RoPE dims per head	64 of 128
block size	16 tokens
score	$\sum_h w_h \,\text{ReLU}(\mathbf{q}_h^I \cdot \mathbf{k}_h^I)$
Training data	
corpus	FineWeb-Edu `sample-10BT`
packing	streaming, EOS-delimited
sequence length L	16,384
training tokens	141,557,760
seed	42
Selector training	
trainable	selector only (model frozen)
loss	KL over selected tokens (Eq. 3)
training budget K_{train}	8,192 tokens ($0.5L$)
precision	bfloat16
Optimizer & schedule	
optimizer	AdamW
(β_1, β_2)	$(0.9, 0.95)$
ϵ	10^{-8}
weight decay	0.1
peak LR	1×10^{-3}
min LR	5×10^{-5}
warmup steps	100
constant-phase fraction	0.30
decay phase	cosine to min LR
gradient clip (global ℓ_2)	1.0
Batch arithmetic	
per-device batch / accumulation	$(4, 1)$, $(2, 2)$ or $(1, 4)$ by model
processes (DDP)	4 GPUs
effective batch	16 sequences (262,144 tokens)
optimizer steps	$141{,}557{,}760/262{,}144 = 540$
Inference (selection modes)	
top-K fractions	$K \in \{0.25,\ 0.5,\ 0.75\}\, L$
dual top-p masses	$p \in \{0.7,\ 0.8,\ 0.9\}$
sink prefix	128 tokens
recent window	256 tokens
$K_{\min}$ floor	512 tokens (32 blocks)
softmax temperature	1.0

Table 3: **One recipe trains every selector.** The effective batch is the same for all seven models, and the per-device batch varies with model size. The inference rows list the settings of the accuracy study. The serving grid uses the subset in Appendix G.

Model	Step 0	Step 200	Drop by 200 (%)	Step 536
Qwen3-4B	6.75	0.194	97.1	0.0745
Qwen3-8B	7.54	0.241	96.8	0.0777
Qwen3-14B	8.04	0.197	97.6	0.0813
Qwen3-32B	9.17	0.252	97.3	0.0820
Qwen3.5-4B	3.64	0.121	96.7	0.0753
Qwen3.5-9B	3.80	0.129	96.6	0.0822
Qwen3.6-27B	3.87	0.161	95.8	0.0996

Table 4: **Every selector converges within 200 steps.** Selected-token KL in nats, as logged by the first process at the given optimizer step (one micro-batch), for the runs of Figure 2a. Training runs 540 steps, and the last logged step is 536.

the four 128-dimensional query and key heads, a projection to the four mixing weights, and a key LayerNorm with bias. The total therefore follows the hidden size and the number of layers with a selector. The selectors add 2.36% of the parameters on Qwen3-4B, where all 36 layers carry one, and 0.31% on Qwen3.6-27B and 0.14% on Gemma-4-E4B, where only the full-attention layers do. Unlike the weights, the selector's key cache grows with context: it stores 512 elements per token and selector layer, in the formats of Appendix H.

Model	Layers	Selector layers	Selector (M)	Model (B)	Share (%)
Qwen3-4B	36	36	94.7	4.02	2.36
Qwen3-8B	36	36	151.6	8.19	1.85
Qwen3-14B	40	40	210.5	14.77	1.43
Qwen3-32B	64	64	336.9	32.76	1.03
Qwen3.5-4B	32	8	21.1	4.21	0.50
Qwen3.5-9B	32	8	33.7	8.95	0.38
Qwen3.6-27B	64	16	84.2	26.90	0.31
Gemma-4-E4B	42	4	10.5	7.46	0.14

Table 5: **The selectors add at most 2.4% of the model's parameters.** Decoder layers, layers with a selector, and the parameters of the selectors and of the frozen model, read from the tensor shapes of each checkpoint. Hybrid models attach selectors to their full-attention layers only. Gemma-4-E4B is the model of Appendix K.4.

C PERPLEXITY DIAGNOSTICS

Protocol. We score FineWeb-Edu `sample-10BT` with the training packing loader and shuffle seed 987654321. The seed differs from training, but we did not check that the windows avoid training documents, so the result is a diagnostic on the training distribution. Each reported cell scores five 16384-token windows, or 81915 next-token predictions. Dense and sparse settings score the same windows.

Each window runs in one causal forward pass without a KV cache. Positions run from 0 to $L-1$ across the packed window and do not reset at document boundaries. The sparse path uses the training attention implementation at $K = 0.25L$, $0.5L$ and $0.75L$, or 4096, 8192 and 12288 tokens, and the dense reference turns selection off. The sparse path therefore selects per 16-query tile at every position, whereas the task evaluation prefills densely and selects per query only while decoding.

Perplexity is $\exp(\overline{\mathrm{CE}})$. We estimate the standard error of mean cross-entropy across the five windows and propagate it with the delta method: $\mathrm{SE}(\mathrm{PPL}) = \mathrm{PPL}\,\mathrm{SE}(\overline{\mathrm{CE}})$.

FineWeb-Edu. Table 6 lists the values behind Figure 2b. Every fixed budget stays within 0.4% of dense perplexity, except $K = 0.25L$ on the two Qwen3.5 models, which add 0.60% and 1.22%. The spread across windows is larger, 3.4 to 3.7% of the dense value, but every setting scores the same

windows, so the paired differences are much tighter. Paired, the two Qwen3.5 increases lie 2.9 and 4.0 standard errors from zero, and on Qwen3 most budgets lower perplexity slightly.

		Change from dense (%)		
Model	Dense	$K = 0.25L$	$K = 0.5L$	$K = 0.75L$
Qwen3-4B	13.55 (0.49)	-0.25	-0.13	-0.01
Qwen3-8B	10.63 (0.38)	-0.32	-0.12	-0.03
Qwen3-32B	8.34 (0.30)	-0.07	-0.01	$+0.02$
Qwen3.5-4B	10.30 (0.37)	$+0.60$	$+0.10$	$+0.02$
Qwen3.5-9B	9.68 (0.33)	$+1.22$	$+0.22$	-0.01

Table 6: **Sparse selection preserves FineWeb-Edu perplexity.** Dense perplexity with its delta-method standard error in parentheses, and the relative change of each fixed budget, on the five 16K windows of Appendix C. The same windows score every setting.

Out-of-domain code. Selector training sees FineWeb-Edu only. Table 7 repeats the diagnostic on deduplicated Python from `codeparrot-clean-valid` for Qwen3-4B, Qwen3.5-4B and Gemma-4-E4B, with the same five-window protocol and the same windows for every setting. The standard errors in the table are dominated by the spread across windows. The paired per-window differences are far tighter. On Qwen3-4B the fixed budgets change the mean cross-entropy by at most $+0.006$ nats, at $K = 0.25L$ with a standard error of 0.005, and on Qwen3.5-4B by at most $+0.001$ nats. Top-$p = 0.9$ adds 0.014 ± 0.005 nats on Qwen3-4B, a 1.4% increase in perplexity, and removes 0.003 ± 0.002 nats on Qwen3.5-4B. Gemma-4-E4B's windows vary far more, and its sparse settings sit 0.01 to 0.11 nats below dense, within two standard errors of zero. The selectors transfer to code at the fixed budgets without a measurable penalty on the two Qwen models. Top-$p = 0.9$ on Qwen3-4B is the one setting with a measurable cost, and it is small.

Model	Dense	$K = 0.25L$	$K = 0.5L$	$K = 0.75L$	top-$p = 0.9$
Qwen3-4B	3.107 (0.31)	3.125 (0.32)	3.107 (0.31)	3.106 (0.31)	3.151 (0.33)
Qwen3.5-4B	2.558 (0.18)	2.562 (0.19)	2.560 (0.18)	2.558 (0.18)	2.550 (0.18)
Gemma-4-E4B	9.430 (3.88)	8.428 (2.89)	9.039 (3.52)	9.339 (3.79)	–

Table 7: **Fixed budgets transfer to code, a domain absent from selector training.** Deduplicated Python from `codeparrot-clean-valid`, scored under the protocol of Appendix C on five windows shared by every setting. Parentheses give the delta-method standard error, which is dominated by the spread across the five windows. The paired per-window differences are given in the text. Gemma-4-E4B has no top-p cell.

D ATTENTION MAPS

Figure 7 compares where each model attends with where its selector expects it to attend, on one 16K RULER prompt. The prompt holds four needle sentences in essay text, and one of them answers the question. It uses the evaluation template of Appendix E.3. We run a dense prefill and take the last query of every 16-token tile at every full-attention layer. The model's attention is the softmax per head, averaged over heads, and the selector's estimate is the softmax of its token scores from Equation 1. Both are summed into 64-token bins.

Every model attends to the first tokens, which act as attention sinks (Xiao et al., 2024), and to recent tokens along the diagonal. On average, the first 64 tokens hold 27 to 45% of the attention in the Qwen3 models and only 4 to 6% in the hybrids. The hybrids instead give about half of their attention to the query's own 64-token bin and the three before it. Dual top-p reserves both regions before it applies its threshold, as described in Section 3. The selector reproduces both patterns. Its share on the first bin and on the recent bins is within one point of the model's for all seven models, and it follows the model's share from layer to layer with a correlation of at least 0.97. It also reproduces the finer

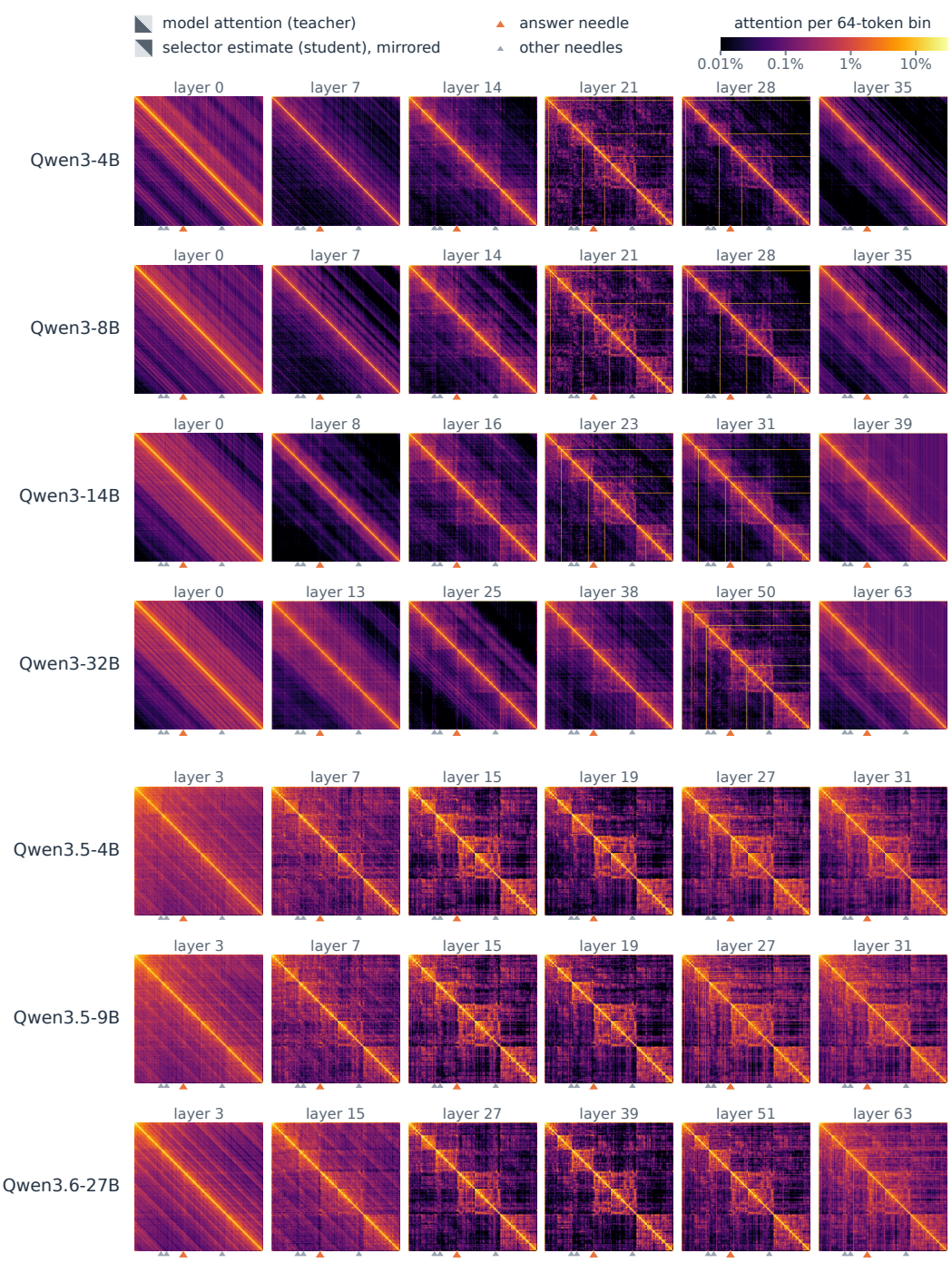

Figure 7: **The selector reproduces where each layer attends.** One 16K RULER prompt, a multi-key needle retrieval in essay text, and six full-attention layers per model at evenly spaced depths, counted from 0. The hybrids have full attention in every fourth layer. Below each diagonal is the model's attention, the teacher, with queries as rows and keys as columns, taken as a softmax per head and averaged over heads. Above it is the selector's estimate, the student, mirrored. Where the selector matches the model, each square is symmetric about its diagonal. Colors give attention per 64-token bin on one shared log scale, and triangles mark the needle sentences. The selector is supervised only on the tokens it selects, and the rest of its estimate is unsupervised.

structure: thin lines where a token inside the context acts as a further sink, and squares along the diagonal where a span attends mostly within itself.

Figure 8 follows the query that produces the answer, the last prompt token, at the layer where the model attends most to the answer needle. The model's attention alone picks this layer, so the selector plays no part in the choice. There the model gives the needle's two blocks 16 to 61% of its attention, and the selector's estimate gives them 18 to 51%. The selector ranks one of the two blocks first on six of the seven models and second on Qwen3-4B. Both the model and the selector give the three other needles together less than the answer needle. At this step, top-$p = 0.8$ reads only 3.1 to 6.6% of the context, where 3.1% is its 512-token floor, and keeps 91.6 to 96.4% of the model's attention.

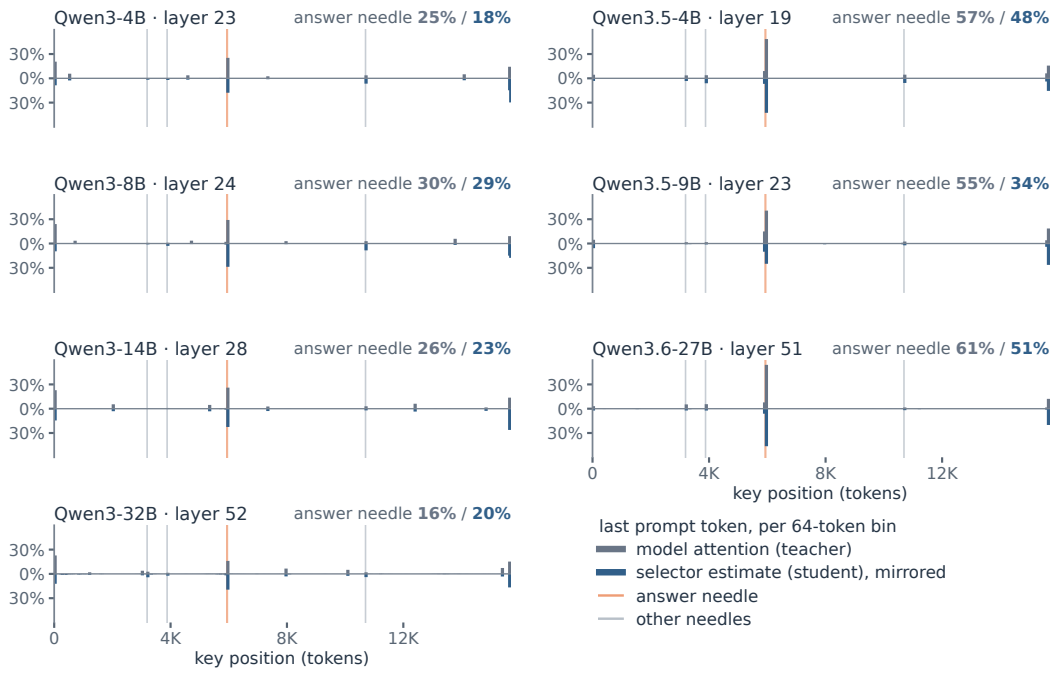

Figure 8: **At the answer, the selector points where the model looks.** Attention of the last prompt token over the prompt of Figure 7, per 64-token bin, at the layer where each model attends most to the answer needle. The model's attention rises above the axis and the selector's estimate is mirrored below it. Vertical lines mark the four needles, and the numbers give the model's and the selector's shares on the answer needle's blocks.

Table 8 measures what sparse decoding keeps. Each query selects blocks as it would at a decoding step, under the rules of Section 3, and we record the share of the model's attention inside them. At $K = 0.25L$ the kept blocks hold 87.2 to 92.7% of the attention on average, and no layer's average falls below 68.8%. Top-$p = 0.8$ keeps 87.6 to 91.6% while reading 20.8 to 30.2% of the context.

		Top-K kept (%)			$p = 0.9$ (%)		$p = 0.8$ (%)	
Model	Selector layers	$0.25L$	worst layer	$0.5L$	kept	read	kept	read
Qwen3-4B	36	92.7	76.3	96.6	92.7	30.2	89.6	20.8
Qwen3-8B	36	91.1	77.3	95.8	91.9	32.0	88.1	21.5
Qwen3-14B	40	90.3	73.0	95.2	92.2	34.3	88.9	23.7
Qwen3-32B	64	92.2	70.3	96.3	94.2	42.3	91.6	30.2
Qwen3.5-4B	8	87.6	68.8	95.0	93.6	41.0	88.3	28.1
Qwen3.5-9B	8	87.2	69.9	94.8	93.3	40.8	87.6	27.6
Qwen3.6-27B	16	88.7	74.4	95.3	94.0	42.0	89.0	27.9

Table 8: **The selected blocks hold most of the model's attention.** Share of the model's attention, averaged over heads, that falls inside the blocks each reading budget keeps, on the prompt of Figure 7. Queries are the last token of every 16-token tile with at least 2048 tokens of context, and each selects as at a decoding step. Values average over those queries and over the full-attention layers. Worst layer is the lowest layer average at $K = 0.25L$. Read is the share of the context that top-p keeps.

E Evaluation protocol

E.1 Data configuration

The accuracy grid requested 192 RULER examples across three single-key and three multi-key needle-retrieval tasks, and 300 BABILong examples, 100 for each of qa1 to qa3, both drawn across the 16K, 32K, 64K and 128K bins. It also requested every LongBench-v2 example of 16K to 128K tokens, and 64 InfiniteBench examples for each of four tasks: passkey, key–value retrieval, book multiple-choice and book QA. Sampling and each model's context then set the evaluated counts in Table 9. Qwen3 has a 40960-token window, and reserving 8192 output tokens leaves a 32K input cap. Qwen3.5 reaches the largest bin, 128K. BABILong contributes only 16K examples in these runs, and no InfiniteBench key–value example remains after filtering.

	Examples per model		Realized length bins	
Benchmark	Qwen3	Qwen3.5	Qwen3	Qwen3.5
RULER	160	192	16K	16/32/64/128K
BABILong (qa1–3)	300	300	16K	16K
InfiniteBench	NA	101	NA	64/128K
LongBench-v2	89	276	16/32K	16/32/64/128K

Table 9: **Qwen3 is evaluated up to 32K and Qwen3.5 up to 128K.** Counts after sampling and context filtering. K is 1024 tokens, and a bin names a dataset length, not an exact prompt length.

E.2 Decoding

The accuracy study generates in non-thinking mode with temperature 0.7, sampling top-$p = 0.8$, top-$k = 20$, min-$p = 0$ and seed 42. The sampling top-p acts on output tokens and is unrelated to the dual top-p of selection. Every benchmark reserves up to 8192 output tokens, and the prompt-length filter counts this reserve.

E.3 Prompts and answer extraction

Prompt templates. All four benchmarks share one system prompt and two user templates, one for free-text and one for multiple-choice answers. Both templates ask the model to wrap its final answer in `<answer>...</answer>` so a regex can recover it. The system prompt is

```
You are a helpful assistant for question-answering
tasks. Always answer based solely on the provided
context.
```

The free-text template serves RULER, BABILong and the InfiniteBench tasks `passkey`, `kv_retrieval` and `longbook_qa_eng`:

```
# Context
{context}
# Question
{question}
# Instructions
Answer the question based on the context. Provide
your final answer wrapped in <answer> tags, e.g.
<answer>X</answer>.
```

The multiple-choice template, for LongBench-v2 and InfiniteBench `longbook_choice_eng`, reuses the context and question blocks, adds the four options `(A)` to `(D)`, and asks for the letter inside `<answer>...</answer>`. Both templates pass through `tokenizer.apply_chat_template(..., add_generation_prompt=True)`, so the chat formatting is the model's own.

Answer extraction. We extract the last `<answer>...</answer>` payload from the completion. For RULER, passkey, key–value and the multiple-choice tasks, we append the opening tag to the prompt, and the model answers right after it. The extractor puts the tag back before applying the regex. On BABILong and book QA, the model writes the tag itself. When the closing tag is missing, BABILong, multiple-choice and F1 answers score 0. RULER, passkey and key–value answers instead fall back to matching the whole completion, because their gold values are random strings that rarely match by accident.

E.4 Scoring

Each example is scored by one of three rules:

- **Substring**, for RULER needles, BABILong and InfiniteBench `passkey` and `kv_retrieval`: the score is the fraction of gold answers that appear as case-insensitive substrings of the extracted text. With a single gold this collapses to a 0/1 indicator.
- **SQuAD-style F1**, for InfiniteBench `longbook_qa_eng`: token F1 between extracted and gold after lowercasing, stripping articles and punctuation, and whitespace tokenization, taking the maximum across gold candidates.
- **Multiple-choice letter**, for LongBench-v2 and InfiniteBench `longbook_choice_eng`: the first A–D token in the extracted text, scored 1 if it matches the gold letter and 0 otherwise.

Per-example scores enter the bootstrap unchanged.

E.5 Bootstrap protocol

The accuracy tables report means with bootstrap standard errors. Each cell, one model, setting and benchmark, is estimated on its own. We resample the per-example scores: 0/1 for single-answer and multiple-choice scores, and values in $[0, 1]$ for substring scores with several gold answers and for token F1. Each cell draws $B = 1000$ resamples with replacement, and we report the cell mean with the standard deviation of the resample means. The seed is fixed, so reruns are bit-identical.

Within a benchmark, the score is the mean over its examples. Across benchmarks, the benchmark mean of Table 1 weights each benchmark equally, and its standard error is $\mathrm{SE}_{\mathrm{avg}} = \sqrt{\sum_{d=1}^{D} \mathrm{SE}_d^2}/D$ over D benchmarks, assuming they are sampled independently.

Paired comparisons resample matched examples, so the dense and sparse scores of an example always enter a draw together. The paired tables of Appendix F.2 and the serving accuracy checks in Tables 15 and 21 report percentage-point differences with intervals from 10000 such resamples.

F FULL ACCURACY RESULTS

F.1 ACCURACY GRIDS

Qwen3 family (4B, 8B, 32B). Table 10 gives every setting per benchmark. The Quest and Twilight rows ran in the same harness and on the same examples, at budgets of $0.25L$, $0.5L$ and $0.75L$. The timed comparison of Section 5.3 uses other settings, which Table 2 checks on a held-out split.

Model	Method	RULER	BABILong	LongBench-v2
Qwen3-4B	Dense	98.1 (1.1)	33.3 (2.7)	31.5 (4.9)
	Quest (Tang et al., 2024) K=$0.25L$	98.1 (1.1)	31.7 (2.6)	28.1 (4.7)
	Quest K=$0.5L$	98.1 (1.1)	33.7 (2.7)	29.2 (4.8)
	Quest K=$0.75L$	98.1 (1.1)	33.0 (2.8)	28.1 (4.8)
	Twilight (Lin et al., 2025) K=$0.25L$	91.9 (2.2)	31.7 (2.6)	28.1 (4.7)
	Twilight K=$0.5L$	97.5 (1.2)	30.7 (2.7)	28.1 (4.6)
	Twilight K=$0.75L$	98.1 (1.1)	31.7 (2.6)	27.0 (4.5)
	Highlight K=$0.25L$	96.2 (1.5)	32.0 (2.7)	27.0 (4.7)
	Highlight K=$0.5L$	96.9 (1.4)	32.3 (2.7)	28.1 (4.8)
	Highlight K=$0.75L$	98.1 (1.1)	32.3 (2.7)	28.1 (4.6)
	Highlight top-p=0.7	93.1 (2.0)	31.3 (2.7)	27.0 (4.8)
	Highlight top-p=0.8	95.6 (1.6)	33.0 (2.8)	27.0 (4.7)
	Highlight top-p=0.9	97.5 (1.3)	32.3 (2.8)	27.0 (4.7)
Qwen3-8B	Dense	98.8 (0.9)	33.0 (2.7)	29.2 (4.8)
	Quest (Tang et al., 2024) K=$0.25L$	99.4 (0.6)	33.3 (2.7)	31.5 (4.8)
	Quest K=$0.5L$	98.8 (0.9)	32.3 (2.6)	32.6 (5.1)
	Quest K=$0.75L$	98.8 (0.9)	33.3 (2.8)	32.6 (4.9)
	Twilight (Lin et al., 2025) K=$0.25L$	93.8 (1.9)	32.0 (2.7)	31.5 (4.9)
	Twilight K=$0.5L$	97.5 (1.2)	30.3 (2.6)	32.6 (5.1)
	Twilight K=$0.75L$	98.8 (0.9)	32.3 (2.8)	32.6 (5.0)
	Highlight K=$0.25L$	98.1 (1.1)	33.3 (2.7)	31.5 (5.0)
	Highlight K=$0.5L$	98.8 (0.9)	33.0 (2.8)	31.5 (4.9)
	Highlight K=$0.75L$	98.8 (0.9)	32.0 (2.6)	32.6 (5.1)
	Highlight top-p=0.7	93.1 (2.1)	33.0 (2.8)	31.5 (5.0)
	Highlight top-p=0.8	96.2 (1.5)	34.3 (2.8)	31.5 (4.9)
	Highlight top-p=0.9	98.1 (1.1)	35.0 (2.8)	31.5 (5.1)
Qwen3-32B	Dense	100.0 (0.0)	40.3 (2.9)	39.3 (5.4)
	Quest (Tang et al., 2024) K=$0.25L$	100.0 (0.0)	39.0 (2.8)	38.2 (5.0)
	Quest K=$0.5L$	100.0 (0.0)	40.3 (2.8)	39.3 (5.3)
	Quest K=$0.75L$	100.0 (0.0)	40.7 (2.8)	38.2 (5.0)
	Twilight (Lin et al., 2025) K=$0.25L$	96.2 (1.5)	36.3 (2.8)	37.1 (5.0)
	Twilight K=$0.5L$	99.4 (0.6)	38.3 (2.8)	40.4 (5.0)
	Twilight K=$0.75L$	100.0 (0.0)	40.3 (2.8)	39.3 (5.1)
	Highlight K=$0.25L$	100.0 (0.0)	37.7 (2.8)	34.8 (5.2)
	Highlight K=$0.5L$	100.0 (0.0)	38.3 (2.9)	36.0 (5.2)
	Highlight K=$0.75L$	100.0 (0.0)	38.3 (2.9)	34.8 (5.0)
	Highlight top-p=0.7	83.8 (2.9)	35.0 (2.7)	36.0 (5.2)
	Highlight top-p=0.8	90.0 (2.5)	35.3 (2.7)	37.1 (5.0)
	Highlight top-p=0.9	99.4 (0.6)	36.3 (2.9)	36.0 (5.1)

Table 10: **On Qwen3, Highlight keeps RULER within two points of dense at every fixed budget and at $p = 0.9$.** Percent accuracy over all length bins, with the bootstrap standard error in parentheses. The 32K input cap excludes InfiniteBench, whose bins start at 64K. Table 2 checks the timed settings.

Qwen3.5 family (4B, 9B). Table 11 gives every setting per benchmark, up to 128K.

Subtasks. Table 12 gives subtask accuracy on Qwen3-4B at $K = 0.5L$ and top-$p = 0.9$. The benchmark rows of the family tables are example-weighted means over these subtasks.

Model	Method	RULER	BABILong	InfiniteBench	LongBench-v2
Qwen3.5-4B	Dense	96.4 (1.4)	34.3 (2.8)	85.3 (3.3)	37.7 (3.0)
	Highlight K=$0.25L$	97.9 (1.1)	34.7 (2.7)	87.2 (3.1)	36.6 (3.0)
	Highlight K=$0.5L$	97.9 (1.0)	36.3 (2.7)	87.4 (3.0)	36.6 (2.8)
	Highlight K=$0.75L$	97.9 (1.0)	34.0 (2.8)	84.4 (3.5)	35.1 (2.9)
	Highlight top-p=0.7	97.9 (1.0)	33.7 (2.7)	86.0 (3.3)	37.3 (3.0)
	Highlight top-p=0.8	97.9 (1.0)	35.0 (2.8)	87.3 (3.1)	38.0 (2.9)
	Highlight top-p=0.9	97.9 (1.1)	37.0 (2.9)	85.9 (3.3)	36.6 (2.8)
Qwen3.5-9B	Dense	94.8 (1.6)	31.0 (2.7)	86.6 (3.2)	41.7 (3.0)
	Highlight K=$0.25L$	99.5 (0.5)	30.3 (2.7)	88.3 (2.9)	39.5 (2.9)
	Highlight K=$0.5L$	99.5 (0.5)	30.3 (2.7)	88.3 (3.0)	39.9 (3.1)
	Highlight K=$0.75L$	99.5 (0.5)	30.3 (2.7)	88.3 (3.0)	39.9 (2.9)
	Highlight top-p=0.7	99.5 (0.5)	33.7 (2.7)	86.6 (3.3)	38.8 (2.9)
	Highlight top-p=0.8	99.0 (0.8)	35.7 (2.8)	89.4 (2.8)	39.1 (3.0)
	Highlight top-p=0.9	99.5 (0.6)	30.3 (2.6)	88.3 (2.9)	39.9 (2.9)

Table 11: **On Qwen3.5, Highlight stays close to dense on all four benchmarks up to 128K.** Percent accuracy over all length bins, with the bootstrap standard error in parentheses. Only dense attention and Highlight run on this family.

Task	n	Dense	K=$0.5L$	top-p=0.9
RULER				
Single-key 1	26	100.0 (0.0)	100.0 (0.0)	100.0 (0.0)
Single-key 2	24	100.0 (0.0)	100.0 (0.0)	100.0 (0.0)
Single-key 3	33	93.9 (4.1)	93.9 (4.2)	93.9 (4.0)
Multi-key 1	24	100.0 (0.0)	100.0 (0.0)	100.0 (0.0)
Multi-key 2	30	100.0 (0.0)	100.0 (0.0)	100.0 (0.0)
Multi-key 3	23	95.7 (4.3)	87.0 (7.0)	91.3 (6.0)
BABILong				
qa1	100	68.0 (4.7)	71.0 (4.5)	70.0 (4.6)
qa2	100	17.0 (3.7)	18.0 (4.0)	18.0 (3.9)
qa3	100	15.0 (3.7)	8.0 (2.8)	9.0 (2.9)
LongBench-v2				
Code repository	3	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)
In-context learning	5	20.0 (17.3)	20.0 (17.6)	20.0 (18.1)
Structured data	1	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)
Dialogue history	12	25.0 (12.2)	16.7 (10.6)	8.3 (7.6)
Multi-document QA	19	36.8 (11.3)	31.6 (10.7)	31.6 (10.7)
Single-document QA	49	34.7 (6.5)	32.7 (6.8)	32.7 (6.9)

Table 12: **On Qwen3-4B, Highlight matches or exceeds dense on most subtasks.** Percent accuracy at $K = 0.5L$ and top-$p = 0.9$ over all length bins, with the bootstrap standard error in parentheses. The n column counts the examples of each subtask.

F.2 Which settings stay within two points of dense

This subsection tests, for each setting, whether it stays within two points of dense or above it, and shows which benchmark any drop comes from. Every setting in Tables 10 and 11 scores the same examples as dense attention, so we compare them example by example. This paired difference is much tighter than two means with separate standard errors. Tables 13 and 14 report the mean difference per benchmark and pooled over the three benchmarks both families share, with a one-sided 95% lower bound on the pooled difference from 10000 resamples of matched examples. A bound above -2 means that, at 95% confidence, the setting stays within two points of dense or above it. The pooled difference weights every example equally, so BABILong, with 300 examples, counts most. It therefore differs from the equal-weight benchmark mean of Table 1. Each bound holds for its own row, with no correction for the number of rows.

The bounds separate the models. At $K = 0.5L$ and $p = 0.9$, the settings of Figure 3, the bound lies above -2 points on Qwen3-8B, Qwen3.5-4B and Qwen3.5-9B, and below it on Qwen3-4B and Qwen3-32B, where the pooled differences are -1.5 to -2.9 points. On Qwen3-4B at these settings, LongBench-v2 drops most, 3.4 and 4.5 points, and every Highlight setting loses three to four of its 89 examples there. Quest and Twilight lose 2.2 to 4.5 points there as well. On Qwen3-32B, the loss sits on BABILong and LongBench-v2, while RULER moves by at most 0.6 points. Quest at $0.5L$ has a bound above -2 on all three Qwen3 models, and Twilight at $0.5L$ on none. Section 5.3 shows that Quest's selector call costs 1.4 to 3.9 times Highlight's top-K call. The low thresholds move RULER most on Qwen3: $p = 0.7$ costs 5 to 16 points there across the family, and $p = 0.8$ costs 10 points on Qwen3-32B. On Qwen3.5, both thresholds gain on RULER.

Model	Method	RULER	BABILong	LongBench-v2	Pooled	Lower bound
Qwen3-4B	Quest K=$0.25L$	+0.0	−1.7	−3.4	−1.5	−2.9
	Quest K=$0.5L$	+0.0	+0.3	−2.2	−0.2	−1.5
	Quest K=$0.75L$	+0.0	−0.3	−3.4	−0.7	−2.2
	Twilight K=$0.25L$	−6.2	−1.7	−3.4	−3.3	−5.5
	Twilight K=$0.5L$	−0.6	−2.7	−3.4	−2.2	−4.0
	Twilight K=$0.75L$	+0.0	−1.7	−4.5	−1.6	−3.3
	Highlight K=$0.25L$	−1.9	−1.3	−4.5	−2.0	−3.6
	Highlight K=$0.5L$	−1.2	−1.0	−3.4	−1.5	−2.7
	Highlight K=$0.75L$	+0.0	−1.0	−3.4	−1.1	−2.4
	Highlight top-p=0.7	−5.0	−2.0	−4.5	−3.3	−5.3
	Highlight top-p=0.8	−2.5	−0.3	−4.5	−1.6	−3.5
	Highlight top-p=0.9	−0.6	−1.0	−4.5	−1.5	−3.1
Qwen3-8B	Quest K=$0.25L$	+0.6	+0.3	+2.2	+0.7	−0.4
	Quest K=$0.5L$	+0.0	−0.7	+3.4	+0.2	−0.5
	Quest K=$0.75L$	+0.0	+0.3	+3.4	+0.7	+0.0
	Twilight K=$0.25L$	−5.0	−1.0	+2.2	−1.6	−3.5
	Twilight K=$0.5L$	−1.2	−2.7	+3.4	−1.3	−2.7
	Twilight K=$0.75L$	+0.0	−0.7	+3.4	+0.2	−1.1
	Highlight K=$0.25L$	−0.6	+0.3	+2.2	+0.4	−0.9
	Highlight K=$0.5L$	+0.0	+0.0	+2.2	+0.4	−0.7
	Highlight K=$0.75L$	+0.0	−1.0	+3.4	+0.0	−0.9
	Highlight top-p=0.7	−5.6	+0.0	+2.2	−1.3	−3.5
	Highlight top-p=0.8	−2.5	+1.3	+2.2	+0.4	−1.5
	Highlight top-p=0.9	−0.6	+2.0	+2.2	+1.3	+0.0
Qwen3-32B	Quest K=$0.25L$	+0.0	−1.3	−1.1	−0.9	−2.0
	Quest K=$0.5L$	+0.0	+0.0	+0.0	+0.0	−1.1
	Quest K=$0.75L$	+0.0	+0.3	−1.1	+0.0	−1.1
	Twilight K=$0.25L$	−3.8	−4.0	−2.2	−3.6	−5.5
	Twilight K=$0.5L$	−0.6	−2.0	+1.1	−1.1	−2.7
	Twilight K=$0.75L$	+0.0	+0.0	+0.0	+0.0	−1.1
	Highlight K=$0.25L$	+0.0	−2.7	−4.5	−2.2	−3.8
	Highlight K=$0.5L$	+0.0	−2.0	−3.4	−1.6	−3.1
	Highlight K=$0.75L$	+0.0	−2.0	−4.5	−1.8	−2.9
	Highlight top-p=0.7	−16.2	−5.3	−3.4	−8.2	−10.7
	Highlight top-p=0.8	−10.0	−5.0	−2.2	−6.0	−8.4
	Highlight top-p=0.9	−0.6	−4.0	−3.4	−2.9	−4.9

Table 13: **On Qwen3-8B, every setting above $p = 0.7$ stays within two points of dense.** Setting minus dense, in percentage points, on the same examples. Pooled covers all 549 examples: 160 RULER, 300 BABILong and 89 LongBench-v2. Lower bound is the one-sided 95% bootstrap bound of the pooled difference. A bound above -2 means the setting loses less than two points at 95% confidence. Rows match Table 10.

Model	Method	RULER	BABILong	LongBench-v2	Pooled	Lower bound
Qwen3.5-4B	Highlight K=$0.25L$	$+1.6$	$+0.3$	-1.1	$+0.1$	-1.8
	Highlight K=$0.5L$	$+1.6$	$+2.0$	-1.1	$+0.8$	-1.0
	Highlight K=$0.75L$	$+1.6$	-0.3	-2.5	-0.7	-2.6
	Highlight top-p=0.7	$+1.6$	-0.7	-0.4	$+0.0$	-2.1
	Highlight top-p=0.8	$+1.6$	$+0.7$	$+0.4$	$+0.8$	-1.2
	Highlight top-p=0.9	$+1.6$	$+2.7$	-1.1	$+1.0$	-0.8
Qwen3.5-9B	Highlight K=$0.25L$	$+4.7$	-0.7	-2.2	$+0.1$	-1.7
	Highlight K=$0.5L$	$+4.7$	-0.7	-1.8	$+0.3$	-1.6
	Highlight K=$0.75L$	$+4.7$	-0.7	-1.8	$+0.3$	-1.6
	Highlight top-p=0.7	$+4.7$	$+2.7$	-2.9	$+1.2$	-0.9
	Highlight top-p=0.8	$+4.2$	$+4.7$	-2.5	$+2.0$	$+0.0$
	Highlight top-p=0.9	$+4.7$	-0.7	-1.8	$+0.3$	-1.6

Table 14: **On Qwen3.5, 10 of the 12 settings stay within two points of dense.** Setting minus dense, in percentage points, on the same examples. Pooled covers all 768 examples: 192 RULER, 300 BABILong and 276 LongBench-v2. Lower bound is the one-sided 95% bootstrap bound of the pooled difference. A bound above -2 means the setting loses less than two points at 95% confidence. Rows match Table 11.

G THROUGHPUT AT KV SATURATION

Workload and controls. Each cell runs on one B200 and generates 256 tokens per request after dense prefill. Distinct FineWeb-Edu text spans provide prompts of the specified token length without chat templating. Each configuration admits as many requests as its own KV memory holds, and the memory utilization setting is the same for every configuration. The dense control disables the selector on the same checkpoint. The sparse configurations keep the model weights, BF16 KV precision and the FlashAttention backend. The selector key cache takes memory from the model's KV cache, so matching the dense batch size would omit part of the selector's deployment cost.

Measurement window. A barrier holds decoding until every request has prefilled. Let t_{start} be the last arrival of a request's first output token and t_{end} the first arrival of a request's final output token. If N_W output tokens arrive inside that common window, throughput is $N_W/(t_{\mathrm{end}} - t_{\mathrm{start}})$. For admitted concurrency C, effective step time is $1000C/\mathrm{throughput}$ milliseconds. The window is a client-observed decode measurement, not a CUDA-kernel timer. Prefill and the batch drain are excluded.

Grid and acceptance. The six prompt lengths are 8192, 16384, 32768, 65536, 131072 and 261632 tokens. The last leaves room for 256 outputs and a 256-token margin inside the engine's 262144-token limit. Qwen3 uses YaRN (Peng et al., 2024) with factors 2, 4 and 7 for the 64K, 128K and 255.5K lengths, which exceed its 40960-token checkpoint window. Qwen3.6-27B uses its native window. Each model's plan holds one dense cell and sixteen sparse cells per length: top-K at 25% and 50% and top-p at 0.9 and 0.8, each with BF16, FP8, INT8 and MXFP4 selector keys. Each accepted cell is one measurement. Qwen3-8B has 90 accepted cells after the twelve that fail the accuracy check are removed. Qwen3-4B and Qwen3-32B have 102 timing cells each, and Qwen3.6-27B has 30, dense and BF16 only, because its hybrid cache layout admits no quantized selector keys, as Appendix H explains. Tables 17 to 20 give every accepted cell.

Accuracy eligibility. A Qwen3-8B configuration is served only if it passes a two-point accuracy check on the serving engine itself: on 160 RULER, 300 BABILong and 89 LongBench-v2 examples, it must stay within two points of dense or above it on each benchmark. Table 15 gives the check of the four BF16 settings, and Appendix H the check of the quantized settings. The multi-model study of Appendix F runs on a separate evaluation harness and does not decide eligibility. On Qwen3-8B at top-$p = 0.8$, Table 13 records a RULER loss of 2.5 points on that harness, four examples of 160, where the serving-engine check records 1.88 points, three examples. Qwen3-4B, Qwen3-32B and Qwen3.6-27B have no accuracy check and are timed only.

Other models. The selector is one term of the decoding step, and the model sets the rest. At 8K with a quarter of the context and BF16 selector keys, Qwen3-4B and Qwen3-8B reach $1.70\times$ and $1.64\times$ dense throughput, while Qwen3.6-27B reaches $1.18\times$ and Qwen3-32B $1.12\times$. Table 16 shows that at every length, Qwen3-4B and Qwen3-8B stay ahead of Qwen3-32B. Selection shortens only the attention read, so its gain depends on that read's share of the step. Qwen3-32B reads about 64 GB of weights at every step, and the hybrid Qwen3.6-27B keeps a KV cache in only 16 of its 64 layers, so the same budget saves a smaller share of their steps.

Selection	Cache	RULER Δ	BABILong Δ	LB-v2 Δ	Eligible
$K = 0.25L$	BF16	-1.25 $[-3.12, +0.00]$	$+1.67$ $[-0.67, +4.33]$	$+1.12$ $[+0.00, +3.37]$	Yes
$K = 0.5L$	BF16	-0.62 $[-1.88, +0.00]$	$+1.00$ $[-0.33, +2.67]$	$+1.12$ $[+0.00, +3.37]$	Yes
$p = 0.9$	BF16	-1.25 $[-3.12, +0.00]$	$+2.67$ $[+0.00, +5.33]$	$+0.00$ $[+0.00, +0.00]$	Yes
$p = 0.8$	BF16	-1.88 $[-4.38, +0.00]$	$+2.67$ $[-0.67, +6.00]$	$+0.00$ $[+0.00, +0.00]$	Yes

Table 15: **BF16 selector keys pass the same check.** Qwen3-8B on the serving engine, same examples and rule as Table 21. Deltas and paired 95% bootstrap intervals (brackets) are in percentage points against the dense reference of this check: 99.375% RULER, 31.667% BABILong, and 38.202% LongBench-v2 ($n = 160, 300, 89$). The two dense runs differ by one example on RULER and one on BABILong.

Model	BF16 $K = 0.25L$	BF16 $p = 0.8$	MXFP4 $p = 0.8$	Accuracy check
Qwen3-8B	1.40–1.77	1.42–1.91	1.79–2.53	Yes
Qwen3-4B	1.61–1.83	1.63–2.05	2.04–2.57	No
Qwen3-32B	1.08–1.32	1.07–1.27	1.14–1.49	No
Qwen3.6-27B	1.17–1.57	1.17–1.60	NA	No

Table 16: **The same budget gives different gains across models.** Throughput relative to dense at KV saturation, as a range over the six prompt lengths on one B200. Each configuration fills its own KV pool. Only Qwen3-8B has a matching accuracy check, and the other models are timed only. Quantized selector caches do not run on the hybrid Qwen3.6-27B, marked NA.

Cache	Selection	8K	16K	32K	64K	128K	255.5K
Throughput: dense tokens/s, then relative to dense (×)							
Dense		3445	1708	892	454	224	106
BF16	$K = 0.25L$	1.64	1.77	1.55	1.53	1.40	1.59
	$K = 0.5L$	1.26	1.29	1.14	1.12	1.02	1.29
	$p = 0.9$	1.56	1.75	1.50	1.41	1.18	1.48
	$p = 0.8$	1.73	1.91	1.70	1.66	1.42	1.65
FP8	$K = 0.25L$	1.83	2.03	1.78	1.90	1.77	1.90
	$K = 0.5L$	1.35	1.46	1.29	1.35	1.24	1.34
	$p = 0.9$	1.77	1.98	1.69	1.67	1.43	1.59
INT8	$K = 0.25L$	1.89	2.08	1.85	1.95	1.79	1.91
	$K = 0.5L$	1.38	1.49	1.33	1.37	1.27	1.37
	$p = 0.9$	1.81	2.08	1.78	1.71	1.48	1.63
MXFP4	$K = 0.25L$	2.02	2.29	2.05	1.96	1.82	1.96
	$K = 0.5L$	1.45	1.61	1.43	1.39	1.29	1.40
	$p = 0.9$	1.88	2.19	1.87	1.69	1.46	1.63
	$p = 0.8$	2.20	2.53	2.17	2.07	1.79	2.04
Requests in flight							
Dense		112	57	29	14	7	3
BF16		89	46	23	11	5	2
FP8		99	51	25	13	6	3
INT8		99	51	25	13	6	3
MXFP4		105	54	27	13	6	3
Decode step (ms)							
Dense		32.5	33.4	32.5	30.8	31.2	28.3
BF16	$K = 0.25L$	15.7	15.2	16.6	15.8	16.0	11.8
	$K = 0.5L$	20.4	20.8	22.6	21.6	21.8	14.6
	$p = 0.9$	16.5	15.4	17.2	17.2	18.9	12.8
	$p = 0.8$	15.0	14.1	15.2	14.6	15.7	11.4
FP8	$K = 0.25L$	15.7	14.7	15.7	15.1	15.1	14.8
	$K = 0.5L$	21.3	20.4	21.8	21.2	21.5	21.1
	$p = 0.9$	16.2	15.1	16.6	17.1	18.7	17.7
INT8	$K = 0.25L$	15.2	14.4	15.2	14.7	15.0	14.8
	$K = 0.5L$	20.9	20.0	21.0	20.9	21.1	20.7
	$p = 0.9$	15.8	14.3	15.7	16.7	18.1	17.4
MXFP4	$K = 0.25L$	15.1	13.8	14.7	14.6	14.7	14.4
	$K = 0.5L$	21.0	19.6	21.1	20.6	20.8	20.3
	$p = 0.9$	16.2	14.4	16.2	16.9	18.3	17.4
	$p = 0.8$	13.8	12.5	13.9	13.8	14.9	13.9
Retained blocks under top-p (%)							
BF16	$p = 0.9$	26.8	20.2	22.4	21.3	29.8	29.5
	$p = 0.8$	18.5	12.7	13.5	12.3	17.7	18.2
FP8	$p = 0.9$	26.8	20.1	22.4	21.7	29.3	30.1
INT8	$p = 0.9$	26.6	19.9	22.3	21.8	29.3	29.9
MXFP4	$p = 0.9$	27.6	21.0	23.4	22.5	30.6	31.6
	$p = 0.8$	18.8	13.3	14.2	13.3	18.3	19.3

Table 17: **On Qwen3-8B, every accuracy-checked configuration beats dense at every length.** One measurement per cell on one B200, 256 generated tokens per request. Each configuration fills its own KV pool, and every selection with one cache format admits the same requests. Step time is requests in flight divided by throughput. Retained blocks average over decode queries and sparse layers. Top-K keeps its fraction of the prompt, 24.6 to 25.0% and 49.2 to 50.0% of the blocks. Every cell passed the accuracy check, which excludes top-$p = 0.8$ in FP8 and INT8.

Cache	Selection	8K	16K	32K	64K	128K	255.5K
Throughput: dense tokens/s, then relative to dense (×)							
Dense		3724	1842	974	488	235	111
BF16	$K = 0.25L$	1.70	1.83	1.61	1.61	1.69	1.82
	$K = 0.5L$	1.25	1.30	1.16	1.18	1.22	1.32
	$p = 0.9$	1.65	1.83	1.52	1.43	1.39	1.52
	$p = 0.8$	1.82	2.05	1.74	1.74	1.63	1.78
FP8	$K = 0.25L$	1.97	2.17	1.88	1.94	1.82	2.02
	$K = 0.5L$	1.40	1.48	1.31	1.32	1.29	1.41
	$p = 0.9$	1.87	2.10	1.73	1.63	1.51	1.61
	$p = 0.8$	2.11	2.37	2.02	1.99	1.85	2.03
INT8	$K = 0.25L$	2.02	2.19	1.93	1.94	1.84	2.00
	$K = 0.5L$	1.41	1.52	1.33	1.34	1.32	1.44
	$p = 0.9$	1.88	2.17	1.79	1.66	1.51	1.71
	$p = 0.8$	2.13	2.44	2.08	2.05	1.85	2.07
MXFP4	$K = 0.25L$	2.11	2.13	2.12	2.18	2.23	2.12
	$K = 0.5L$	1.49	1.46	1.46	1.47	1.51	1.44
	$p = 0.9$	1.94	2.24	1.94	1.81	1.73	1.62
	$p = 0.8$	2.22	2.57	2.31	2.23	2.08	2.04
Requests in flight							
Dense		118	61	31	15	7	3
BF16		95	49	24	12	6	3
FP8		105	54	27	13	6	3
INT8		105	54	27	13	6	3
MXFP4		111	57	29	14	7	3
Decode step (ms)							
Dense		31.7	33.1	31.8	30.7	29.8	27.1
BF16	$K = 0.25L$	15.0	14.6	15.3	15.3	15.1	14.9
	$K = 0.5L$	20.4	20.4	21.3	20.8	21.0	20.6
	$p = 0.9$	15.4	14.6	16.2	17.2	18.4	17.8
	$p = 0.8$	14.0	13.0	14.2	14.1	15.7	15.2
FP8	$K = 0.25L$	14.3	13.5	14.7	13.7	14.1	13.5
	$K = 0.5L$	20.1	19.8	21.2	20.1	19.9	19.2
	$p = 0.9$	15.1	13.9	16.1	16.3	17.0	16.9
	$p = 0.8$	13.3	12.4	13.7	13.4	13.8	13.3
INT8	$K = 0.25L$	14.0	13.4	14.3	13.7	13.9	13.5
	$K = 0.5L$	20.0	19.3	20.8	19.8	19.4	18.8
	$p = 0.9$	15.0	13.5	15.5	16.0	16.9	15.9
	$p = 0.8$	13.2	12.0	13.3	13.0	13.8	13.1
MXFP4	$K = 0.25L$	14.1	14.5	14.1	13.1	13.4	12.8
	$K = 0.5L$	20.0	21.3	20.5	19.5	19.7	18.9
	$p = 0.9$	15.4	13.8	15.3	15.8	17.2	16.8
	$p = 0.8$	13.4	12.0	12.9	12.9	14.4	13.3
Retained blocks under top-p (%)							
BF16	$p = 0.9$	26.2	20.1	22.1	22.8	29.8	31.5
	$p = 0.8$	18.1	13.0	13.5	13.1	18.3	19.7
FP8	$p = 0.9$	26.2	20.4	22.3	22.8	30.0	32.9
	$p = 0.8$	18.1	13.1	13.6	13.4	18.3	19.5
INT8	$p = 0.9$	26.2	20.0	22.0	22.4	29.8	30.8
	$p = 0.8$	18.1	13.0	13.6	13.1	18.3	19.7
MXFP4	$p = 0.9$	27.7	21.4	23.4	24.2	31.4	33.7
	$p = 0.8$	18.9	13.7	14.4	14.0	19.4	21.5

Table 18: **On Qwen3-4B, every configuration beats dense at every length.** Layout as Table 17. Timing only: no accuracy check covers this model.

Cache	Selection	8K	16K	32K	64K	128K	255.5K
Throughput: dense tokens/s, then relative to dense (×)							
Dense		1271	569	299	149	85	44
BF16	$K = 0.25L$	1.12	1.32	1.25	1.23	1.08	1.13
	$K = 0.5L$	1.00	1.17	1.11	1.10	0.97	0.98
	$p = 0.9$	1.04	1.20	1.08	1.17	1.02	1.08
	$p = 0.8$	1.07	1.27	1.14	1.24	1.08	1.11
FP8	$K = 0.25L$	1.31	1.49	1.25	1.27	1.12	1.18
	$K = 0.5L$	1.16	1.32	1.03	1.13	1.00	1.02
	$p = 0.9$	1.17	1.36	1.04	1.25	1.09	1.12
	$p = 0.8$	1.24	1.45	1.14	1.28	1.12	1.19
INT8	$K = 0.25L$	1.33	1.52	1.28	1.28	1.12	1.19
	$K = 0.5L$	1.17	1.34	1.05	1.17	1.01	1.05
	$p = 0.9$	1.18	1.38	1.06	1.22	1.07	1.14
	$p = 0.8$	1.26	1.47	1.16	1.29	1.13	1.21
MXFP4	$K = 0.25L$	1.33	1.43	1.28	1.41	1.14	1.21
	$K = 0.5L$	1.12	1.17	1.05	1.15	1.02	1.07
	$p = 0.9$	1.22	1.36	1.05	1.27	1.08	1.12
	$p = 0.8$	1.31	1.49	1.16	1.42	1.14	1.19
Requests in flight							
Dense		41	20	10	5	2	1
BF16		32	16	8	4	2	1
FP8		36	18	9	4	2	1
INT8		36	18	9	4	2	1
MXFP4		38	19	9	5	2	1
Decode step (ms)							
Dense		32.2	35.2	33.5	33.6	23.5	22.8
BF16	$K = 0.25L$	22.4	21.3	21.4	21.9	21.7	20.3
	$K = 0.5L$	25.0	24.0	24.0	24.5	24.2	23.3
	$p = 0.9$	24.2	23.5	24.8	23.0	23.0	21.2
	$p = 0.8$	23.4	22.2	23.4	21.6	21.7	20.7
FP8	$K = 0.25L$	21.7	21.2	24.0	21.2	21.0	19.4
	$K = 0.5L$	24.4	24.0	29.2	23.9	23.5	22.5
	$p = 0.9$	24.1	23.3	29.1	21.6	21.6	20.4
	$p = 0.8$	22.8	21.9	26.3	21.0	21.0	19.2
INT8	$K = 0.25L$	21.3	20.8	23.6	21.1	20.9	19.2
	$K = 0.5L$	24.2	23.6	28.7	23.0	23.4	21.7
	$p = 0.9$	23.9	22.9	28.5	22.0	22.0	20.1
	$p = 0.8$	22.5	21.6	25.9	20.8	20.7	18.9
MXFP4	$K = 0.25L$	22.5	23.3	23.5	23.8	20.5	18.9
	$K = 0.5L$	26.7	28.6	28.7	29.1	22.9	21.4
	$p = 0.9$	24.6	24.6	28.8	26.5	21.8	20.5
	$p = 0.8$	22.8	22.4	26.0	23.7	20.6	19.3
Retained blocks under top-p (%)							
BF16	$p = 0.9$	38.3	28.7	29.5	25.5	31.2	30.4
	$p = 0.8$	26.9	19.1	18.7	15.1	19.2	18.5
FP8	$p = 0.9$	38.0	29.0	29.3	25.5	31.1	30.4
	$p = 0.8$	27.1	19.5	18.6	15.2	18.8	19.0
INT8	$p = 0.9$	38.1	28.9	29.3	25.1	31.0	29.8
	$p = 0.8$	27.1	19.3	18.8	15.3	19.1	18.3
MXFP4	$p = 0.9$	38.9	29.6	30.3	27.6	32.2	30.9
	$p = 0.8$	27.7	19.9	19.6	16.7	20.4	18.8

Table 19: **On Qwen3-32B, gains are smaller, up to** $1.52\times$**.** Layout as Table 17. Timing only: no accuracy check covers this model.

Cache	Selection	8K	16K	32K	64K	128K	255.5K
Throughput: dense tokens/s, then relative to dense (×)							
Dense		3699	2017	1286	699	273	152
BF16	$K = 0.25L$	1.18	1.40	1.21	1.17	1.57	1.42
	$K = 0.5L$	1.03	1.20	1.02	0.96	1.29	1.19
	$p = 0.9$	1.11	1.17	1.15	1.09	1.47	1.42
	$p = 0.8$	1.17	1.32	1.28	1.23	1.60	1.53
Requests in flight							
Dense		136	81	44	23	11	6
BF16		113	65	35	18	9	4
Decode step (ms)							
Dense		36.8	40.2	34.2	32.9	40.3	39.4
BF16	$K = 0.25L$	26.0	23.0	22.5	22.0	21.0	18.5
	$K = 0.5L$	29.7	26.8	26.8	26.7	25.5	22.1
	$p = 0.9$	27.6	27.5	23.7	23.7	22.5	18.5
	$p = 0.8$	26.0	24.4	21.2	20.8	20.6	17.2
Retained blocks under top-p (%)							
BF16	$p = 0.9$	32.0	30.0	28.7	25.4	24.4	19.4
	$p = 0.8$	20.4	17.4	16.4	14.1	13.8	11.9

Table 20: **On Qwen3.6-27B, gains are largest at long context.** Layout as Table 17. Timing only: no accuracy check covers this model. Appendix H explains why quantized selector caches do not run on this hybrid model.

H QUANTIZED SELECTOR KEYS

Storage and execution. The deployment study compares BF16 selector keys with FP8, INT8 and block-scaled MXFP4. A key row holds four 128-dimensional heads, and Section 5.2 gives its size in each format. On B200, the native scoring paths use FP8 matrix multiplication, INT8 matrix multiplication, or block-scaled E2M1 matrix multiplication, respectively. Query conversion, cache maintenance, reductions and selection are included in selector-call timings.

Accuracy check. The Qwen3-8B native accuracy check applies the rule of Appendix G to four selections in each of three native formats, against a fresh dense reference. Table 21 gives every setting, and a setting is eligible when its unrounded difference from dense is at least -2 points on every benchmark. Ten settings pass. FP8 and INT8 at top-$p = 0.8$ fail on RULER and are left out of the serving tables.

Quantized keys on hybrid models. On Qwen3, the selector key cache shares the model's KV group, block identifiers and prefix caching. The hybrid cache manager of Qwen3.5 and Qwen3.6 places the selector key cache in its own group and unifies page sizes across groups. A page of 16 quantized rows of 528 or 272 bytes divides no page of the model's KV cache, so the hybrid serving grid in Table 20 contains BF16 rows only.

Selection	Cache	RULER Δ	BABILong Δ	LB-v2 Δ	Eligible
$K = 0.25L$	FP8	-1.25	$+1.67$	$+1.12$	Yes
		$[-3.12, +0.00]$	$[-1.00, +4.33]$	$[+0.00, +3.37]$	
$K = 0.25L$	INT8	-1.25	$+1.33$	$+0.00$	Yes
		$[-3.12, +0.00]$	$[-1.00, +3.67]$	$[+0.00, +0.00]$	
$K = 0.25L$	MXFP4	-1.25	$+2.67$	$+0.00$	Yes
		$[-3.12, +0.00]$	$[+0.00, +5.33]$	$[+0.00, +0.00]$	
$K = 0.5L$	FP8	-0.63	$+1.67$	$+1.12$	Yes
		$[-1.88, +0.00]$	$[+0.00, +3.33]$	$[+0.00, +3.37]$	
$K = 0.5L$	INT8	-0.63	$+0.33$	$+0.00$	Yes
		$[-1.88, +0.00]$	$[-1.00, +1.67]$	$[+0.00, +0.00]$	
$K = 0.5L$	MXFP4	-0.63	$+0.67$	$+1.12$	Yes
		$[-1.88, +0.00]$	$[-1.33, +2.67]$	$[+0.00, +3.37]$	
$p = 0.8$	FP8	-2.50	$+4.00$	$+0.00$	No
		$[-5.00, -0.62]$	$[+0.33, +7.67]$	$[+0.00, +0.00]$	
$p = 0.8$	INT8	-2.50	$+3.00$	$+0.00$	No
		$[-5.00, -0.62]$	$[-0.33, +6.33]$	$[+0.00, +0.00]$	
$p = 0.8$	MXFP4	-1.88	$+3.33$	$+0.00$	Yes
		$[-4.38, +0.00]$	$[-0.33, +7.00]$	$[+0.00, +0.00]$	
$p = 0.9$	FP8	-1.25	$+2.67$	$+0.00$	Yes
		$[-3.12, +0.00]$	$[+0.00, +5.33]$	$[+0.00, +0.00]$	
$p = 0.9$	INT8	-1.25	$+3.67$	$+0.00$	Yes
		$[-3.12, +0.00]$	$[+1.00, +6.33]$	$[+0.00, +0.00]$	
$p = 0.9$	MXFP4	-1.25	$+1.67$	$+0.00$	Yes
		$[-3.12, +0.00]$	$[-1.00, +4.67]$	$[+0.00, +0.00]$	

Table 21: **Ten of the twelve quantized settings pass the two-point accuracy check.** Qwen3-8B on the serving engine. Deltas and paired 95% bootstrap intervals (brackets) are in percentage points against the dense reference of the same check: 98.750% RULER, 32.000% BABILong, and 38.202% LongBench-v2 ($n = 160, 300, 89$). A setting passes when every unrounded delta is at least -2 points. Failed configurations are excluded from the accepted serving tables.

I SELECTOR COST

This appendix documents the selector comparison of Section 5.3: what a timed call contains, how the accuracy check was run, the timing sweeps behind the figures, and what each selector retains and stores.

I.1 TIMING BOUNDARY AND OPERATING POINTS

Timing boundary. Each measurement times one complete selector call on one layer at batch size one: the method's own projection or query conversion, maintenance of the selector's cache, scoring, selection and output compaction, with the union over the query heads of a KV group where a method needs it. The model's shared projections, its KV writes, attention and feed-forward computation are outside the timer. Every selector is checked against its numerical reference before CUDA-graph timing.

Baseline operating points. Quest runs its released kernels at a pinned version, with a fixed budget of 8192 candidate tokens per query head, the setting of Twilight's 32K cost breakdown and released harness. Twilight applies its released top-$p = 0.85$ to that pool. We also time Quest at a quarter of the context to test its budget: at 255.5K that fraction is 65408 candidates per query head, eight times the fixed pool, and costs at most 9% more.

Baseline repairs. Quest and Twilight run their released CUDA kernels with no added fusion across stages. Four repairs make Twilight compute what its paper describes. Its INT4 scoring kernel applies the per-token scale and offset, its dequantization returns the decoded values, and its host code indexes the packed INT4 keys with the correct strides. Its top-p search starts from the row maximum, as in its Algorithm 1, instead of the first unsorted probability.

I.2 Accuracy qualification

Held-out accuracy. The comparison in Section 5.3 uses settings checked on a fixed split of RULER, BABILong and LongBench-v2: 114 development examples, 34, 60 and 20 per benchmark, and 435 held-out examples, 126, 240 and 69, with held-out inputs of 12256 to 31873 tokens. Every policy runs dense prefill and the same masked decode attention over the original BF16 keys and values, in a separate evaluation adapter rather than the serving engine, against a fresh dense control on the same examples. Selecting every token reproduces the dense logits exactly. Twilight keeps its first two layers dense, as in its paper. We checked six Twilight configurations on the development examples, with pools of 8192 and 16384 candidates and thresholds from 0.85 to 0.99. A rule fixed in advance, the smallest passing pool at its smallest passing threshold, froze 8192 and $p = 0.85$ before any held-out example ran. Table 2 reports the held-out scores: every sparse policy passes the two-point rule on every benchmark. Table 22 gives the paired 95% intervals of the differences from dense.

Two runtimes. The adapter and the serving engine both sample at temperature 0.7, but they draw different samples from the same seed. On the 89 LongBench-v2 examples of both splits, dense attention answers 26 correctly in the adapter, matching the 29.2% of Table 10, and 34 in the serving engine of Table 21. Each difference from dense compares two runs of the same runtime.

Policy	RULER Δ	BABILong Δ	LB-v2 Δ	Twilight / Highlight range
Twilight, pool 8192, $p = 0.85$	-1.59 $[-3.97, +0.00]$	-0.83 $[-4.17, +2.50]$	$+0.00$ $[+0.00, +0.00]$	
Highlight $K = 0.25L$, BF16	-0.79 $[-3.17, +1.59]$	$+0.83$ $[-1.25, +3.33]$	$+2.90$ $[+0.00, +7.25]$	7.44–8.69
Highlight $K = 0.5L$, BF16	-0.79 $[-2.38, +0.00]$	-0.83 $[-2.92, +1.25]$	$+2.90$ $[+0.00, +7.25]$	7.44–8.69
Highlight $p = 0.9$, BF16	-0.79 $[-2.38, +0.00]$	$+0.83$ $[-2.08, +3.75]$	$+2.90$ $[+0.00, +7.25]$	5.88–7.46
Highlight $p = 0.9$, MXFP4	-0.79 $[-2.38, +0.00]$	$+1.25$ $[-1.25, +4.17]$	$+2.90$ $[+0.00, +7.25]$	5.60–6.87

Table 22: **Twilight's call costs at least 5.5 times Highlight's on every query.** Qwen3-8B on the held-out split of Table 2, $n = 126$, 240 and 69. Each Δ is the setting minus dense, in percentage points, with the paired 95% bootstrap interval below it, from 10000 resamples of the examples. The last column gives the smallest and largest of the 30 per-query ratios whose medians Table 2 reports.

Development split. Table 23 reports the development split behind the frozen Twilight setting. All six Twilight configurations pass the two-point rule on the 114 development examples. The table also shows why the development split is not evidence on its own. Every Highlight setting scores 97.06% on the 34 development RULER examples, one example below dense, a loss of 2.94 points that fails the rule at that sample size. The same settings pass on the 126 held-out RULER examples, with losses of at most 0.83 points on any benchmark. The Highlight settings come from the serving grid and the earlier timing sweep and were not chosen on the development examples. The held-out split of Table 2 is the evidence for every policy, Twilight included.

Policy	RULER (%)	BABILong (%)	LB-v2 (%)	Largest drop	Passes
Dense	100.00	30.00	50.00	0.00	Yes
Twilight, pool 8192, $p = 0.85$ (frozen)	100.00	31.67	50.00	0.00	Yes
Twilight, pool 8192, $p = 0.9$	100.00	31.67	50.00	0.00	Yes
Twilight, pool 8192, $p = 0.95$	100.00	33.33	50.00	0.00	Yes
Twilight, pool 8192, $p = 0.99$	100.00	33.33	50.00	0.00	Yes
Twilight, pool 16384, $p = 0.9$	100.00	31.67	50.00	0.00	Yes
Twilight, pool 16384, $p = 0.95$	100.00	31.67	50.00	0.00	Yes
Highlight $K = 0.25L$, BF16	97.06	35.00	50.00	2.94	No
Highlight $K = 0.5L$, BF16	97.06	35.00	50.00	2.94	No
Highlight $p = 0.9$, BF16	97.06	35.00	50.00	2.94	No
Highlight $p = 0.9$, MXFP4	97.06	33.33	50.00	2.94	No

Table 23: **All six Twilight settings pass on the development split.** Percent accuracy on the 34, 60 and 20 development examples, the largest decrease against dense in percentage points, and whether the two-point rule holds on every benchmark. The rule froze the smallest passing Twilight pool at its smallest passing threshold before any held-out example ran. The Highlight settings were fixed by the serving grid and were not chosen on this split. Each misses one RULER example here and passes on the held-out split of Table 2.

I.3 Timing sweeps

Generated-decode traces. Timing inputs are 30 selector calls captured from dense greedy decoding of six development prompts, two per benchmark, at layers 2, 18 and 35 and decode positions 0 and 7. Two LongBench-v2 prompts end before position 7, which leaves 30 calls. Context at the captured calls spans 15124 to 27605 tokens. Each policy replays every trace in three independent processes, with eager and CUDA-graph timings kept separately, and Table 2 reports the CUDA-graph medians. The largest repeat standard deviation is 0.00024 ms for Highlight and 0.00014 ms for Twilight.

Fixed-budget repeats. Table 24 and Figure 5 rerun the six Qwen3-8B traces in three independent processes for four Highlight settings: top-$K = 0.25L$ and $0.5L$, and top-$p = 0.9$ in BF16 and in MXFP4. The 72 records each carry their own Quest and Twilight timing on the same trace. No standard deviation across the three processes exceeds 0.0002 ms.

Three-model sweep. Table 25 times Qwen3-4B, Qwen3-8B and Qwen3-32B at six lengths on one real-text prompt and one middle layer per model and length, replaying the final-prefill query in one session of 20 samples of 20 calls. The largest within-session relative standard deviation is 1.25%. The 4B and 32B calls sit within 15% of the 8B at every length and setting.

Prompt	Quest	Twilight	BF16 $K = 0.25L$	BF16 $K = 0.5L$	BF16 $p = 0.9$	MXFP4 $p = 0.9$
8K	0.0354	0.2214	0.0247	0.0247	0.0287	0.0329
16K	0.0432	0.2232	0.0269	0.0268	0.0309	0.0330
32K	0.0579	0.2451	0.0330	0.0330	0.0392	0.0378
64K	0.0853	0.2659	0.0414	0.0414	0.0459	0.0418
128K	0.1414	0.3579	0.0566	0.0567	0.0618	0.0477
255.5K	0.3162	0.5527	0.0801	0.0801	0.0854	0.0600

Table 24: **Highlight's call stays below 0.09 ms up to 255.5K.** Milliseconds per layer call on Qwen3-8B, B200, batch 1, layer 18, one final-prefill trace per length, CUDA graphs. Each entry is the mean over three independent processes, and no standard deviation exceeds 0.0002 ms. Quest keeps 8192 candidates per query head, Twilight prunes that pool at $p = 0.85$, and Highlight runs $K = 0.25L$, $K = 0.5L$ or $p = 0.9$. Timing only: Table 2 qualifies accuracy separately at 12K to 32K.

Model	Prompt	Quest	Twilight	BF16 top-K	BF16 top-p	MXFP4 top-p
Qwen3-4B	8K	0.037	0.224	0.023	0.027	0.031
Qwen3-4B	16K	0.043	0.232	0.027	0.031	0.034
Qwen3-4B	32K	0.058	0.245	0.033	0.039	0.037
Qwen3-4B	64K	0.085	0.271	0.040	0.046	0.041
Qwen3-4B	128K	0.141	0.361	0.056	0.061	0.047
Qwen3-4B	255.5K	0.318	0.558	0.080	0.085	0.060
Qwen3-8B	8K	0.035	0.218	0.025	0.029	0.031
Qwen3-8B	16K	0.043	0.225	0.027	0.031	0.033
Qwen3-8B	32K	0.056	0.241	0.035	0.039	0.039
Qwen3-8B	64K	0.085	0.265	0.041	0.046	0.042
Qwen3-8B	128K	0.140	0.351	0.058	0.062	0.048
Qwen3-8B	255.5K	0.318	0.556	0.080	0.086	0.060
Qwen3-32B	8K	0.039	0.226	0.027	0.031	0.035
Qwen3-32B	16K	0.045	0.234	0.029	0.033	0.035
Qwen3-32B	32K	0.060	0.249	0.035	0.041	0.041
Qwen3-32B	64K	0.087	0.279	0.042	0.050	0.044
Qwen3-32B	128K	0.143	0.386	0.059	0.064	0.050
Qwen3-32B	255.5K	0.332	0.575	0.081	0.087	0.062

Table 25: **Selector cost barely depends on the model.** Milliseconds per layer call on B200 with batch 1. Quest uses 8192 tokens, and Twilight prunes that pool at $p = 0.85$. Highlight uses $K = 0.25L$ or $p = 0.9$. One middle-layer final-prefill trace per length and model, CUDA graphs, 20 samples of 20 calls in one session. Budgets and accuracy are not matched.

I.4 Retained tokens and selector caches

Table 26 lists what each selector retains and stores on the Qwen3-8B traces of the three-model sweep. The budgets differ by construction, so the millisecond columns of the timing tables are not an equal-recall comparison. Quest at its fixed pool keeps 8192 to 15448 tokens per query, the union over the query heads of a KV group, which grows slowly with context. At a quarter of the context, it keeps 3014 to 99390. Twilight keeps 108 to 377 tokens: its nucleus at $p = 0.85$ inside a pool whose size never changes retains whatever reaches that mass, so the count follows how peaked each trace is rather than the context length. Highlight keeps exactly its budget at $K = 0.25L$ and 1392 to 100240 tokens at $p = 0.9$, between 14% and 38% of the context. The cache columns give each selector's own state for one layer and the prompt: Quest's page summaries take 2 to 64 MiB, Twilight's summaries and INT4 keys 6 to 200 MiB, and Highlight's selector keys 8 to 256 MiB in BF16 and 2 to 68 MiB in MXFP4. The MXFP4 column shows the $3.76\times$ reduction of Section 5.2 in absolute terms.

Prompt	Quest 8192	Quest $0.25L$	Twilight	$K = 0.25L$	$p = 0.9$	Quest	Twilight	BF16	MXFP4
8K	8192	3014	158	2048	1392	2.0	6.2	8.0	2.1
16K	10562	6112	260	4096	2240	4.0	12.5	16.0	4.2
32K	11996	12002	377	8192	7376	8.0	25.0	32.0	8.5
64K	13334	24272	108	16384	13264	16.0	50.0	64.0	17.0
128K	14220	48838	248	32768	43984	32.0	100.0	128.0	34.0
255.5K	15448	99390	273	65408	100240	63.9	199.6	255.5	67.9

Table 26: **The selectors keep very different numbers of tokens.** On the Qwen3-8B traces, columns two to six give tokens retained per query on the final-prefill trace of each length: Quest at its fixed 8192-token pool and at a quarter of the context (mean over KV groups of the union of the query heads' selected pages), Twilight (the same union of its nucleus selections at $p = 0.85$ inside the 8192-token pool) and Highlight at $K = 0.25L$ and $p = 0.9$ (tokens in 16-token blocks). Columns seven to ten give each selector's cache for one layer and the prompt in MiB: Quest's page summaries, Twilight's summaries plus its INT4 pool, and Highlight's selector keys in BF16 and MXFP4. The counts are properties of the trace and identical across the sixteen cells of a length.

I.5 SELECTOR COST BY CACHE FORMAT

Table 27 times the Highlight call at $K = 0.25L$ in each cache format on the three-model sweep traces. At 8K and 16K the quantized formats cost more than BF16: on Qwen3-8B, 0.029 ms in FP8 or MXFP4 against 0.025 ms in BF16 at 8K. The quantized query conversion is paid at every length, and the narrower cache read repays it only once the read dominates the call. The formats cross near 32K, and at 255.5K MXFP4 costs 0.055 ms against 0.080 ms in BF16 on the 8B, about 32% less, with FP8 and INT8 in between. The three models agree within 0.005 ms at every cell. Table 24 shows the same ordering at top-p in the fixed-budget repeats, so the effect of precision does not depend on the selection rule. Quest and Twilight keep their own precision and do not move with this axis.

Model	Cache	8K	16K	32K	64K	128K	255.5K
Qwen3-4B	BF16	0.023	0.027	0.033	0.040	0.056	0.080
Qwen3-4B	FP8	0.029	0.029	0.033	0.039	0.046	0.066
Qwen3-4B	INT8	0.027	0.027	0.031	0.037	0.045	0.064
Qwen3-4B	MXFP4	0.027	0.029	0.033	0.036	0.041	0.054
Qwen3-8B	BF16	0.025	0.027	0.035	0.041	0.058	0.080
Qwen3-8B	FP8	0.029	0.031	0.035	0.039	0.048	0.066
Qwen3-8B	INT8	0.027	0.029	0.033	0.038	0.046	0.064
Qwen3-8B	MXFP4	0.029	0.030	0.033	0.037	0.043	0.055
Qwen3-32B	BF16	0.027	0.029	0.035	0.042	0.059	0.081
Qwen3-32B	FP8	0.033	0.031	0.036	0.041	0.049	0.068
Qwen3-32B	INT8	0.031	0.030	0.033	0.039	0.048	0.066
Qwen3-32B	MXFP4	0.031	0.031	0.035	0.039	0.045	0.056

Table 27: **Quantized selector keys pay off at long context.** Milliseconds per Highlight call at $K = 0.25L$ on the three-model sweep traces: B200, batch 1, CUDA graphs, 20 samples of 20 calls in one session. Quantized formats add a query conversion and read a narrower cache. In this single-request call, the saving in the read outweighs the conversion only at long context.

J ADAPTIVE BUDGETS

J.1 PER-LAYER BUDGET ACROSS DATASETS AND THRESHOLDS

Figure 9 extends Figure 6 to three datasets and three thresholds. The layer profiles recur on RULER, BABILong and LongBench-v2, and a higher p raises the share in almost every layer. Appendix D shows how attention itself varies across layers on one prompt.

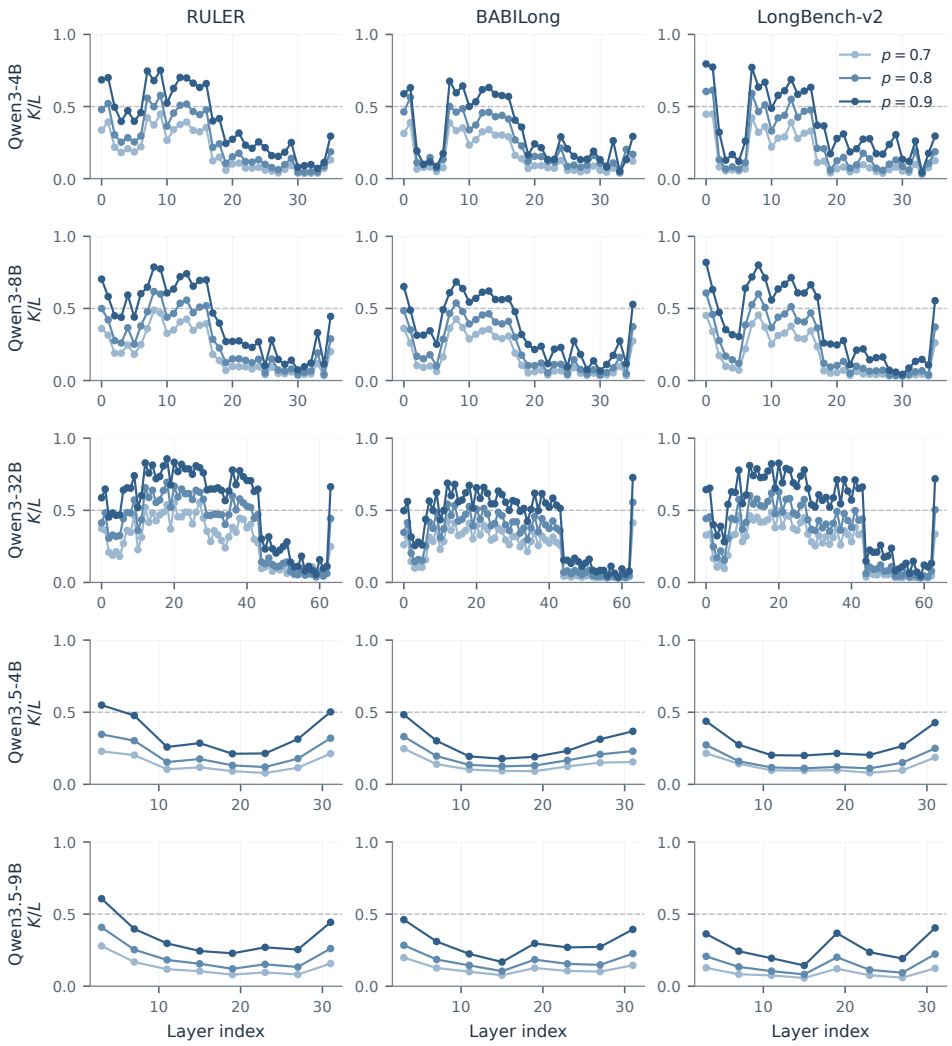

Figure 9: **The per-layer profiles hold across datasets and thresholds.** Mean selected tokens per layer as a fraction of 16384, on 16K inputs. Rows are models and columns are datasets. Within each panel, lines from light to dark show $p = 0.7$, 0.8 and 0.9. The dashed line marks a fixed budget of half the context.

J.2 Budget across context lengths and the floor

Top-p sets the budget from the selector's block distribution, so the selected fraction can change with the context length. Table 28 reports the mean selected fraction by length bin over every layer and decode step of the accuracy-study runs, together with the share of entries at the 512-token floor. Figure 10 plots both. On the Qwen3.5 models, whose runs reach 128K, the fraction at $p = 0.9$ falls from 0.30 and 0.32 at 16K to 0.18 and 0.22 at 64K and stays near 0.20 at 128K. The budget therefore grows more slowly than the context. The Qwen3 runs stop at 32K. Over that range, the fraction at $p = 0.9$ falls on Qwen3-8B and Qwen3-32B and stays flat on Qwen3-4B. The floor binds rarely at $p = 0.9$: under 1% of entries on the Qwen3.5 models and 7 to 9% at 16K on the Qwen3 models.

Lowering the threshold moves entries to the floor. At $p = 0.7$ on 16K inputs, 20%, 25% and 14% of the Qwen3-4B, 8B and 32B entries sit at 512 tokens, against 5% and 3% on Qwen3.5-4B and 9B. For those entries, the floor sets the budget instead of the threshold, and the call reads 3% of the context. The Qwen3 models also carry the RULER losses at $p = 0.7$ in Appendix F.2, 5 to 16 points, while the Qwen3.5 models lose nothing. The floor rate alone does not order those losses: Qwen3-32B has the lowest floor rate of the three Qwen3 models and the largest loss. The floor only guards against a

degenerate nucleus. At the floor, a retrieval query succeeds only when the needed block is among the few the selector ranks first.

Model	p	16K	32K	64K	128K
Qwen3-4B	0.7	0.17 (19.9)	0.14 (14.6)	–	–
	0.8	0.25 (10.6)	0.22 (7.8)	–	–
	0.9	0.35 (7.0)	0.35 (2.8)	–	–
Qwen3-8B	0.7	0.18 (25.5)	0.15 (11.6)	–	–
	0.8	0.25 (17.7)	0.23 (6.8)	–	–
	0.9	0.38 (7.9)	0.36 (1.8)	–	–
Qwen3-32B	0.7	0.27 (13.8)	0.18 (8.1)	–	–
	0.8	0.33 (15.1)	0.25 (4.5)	–	–
	0.9	0.45 (9.2)	0.40 (1.0)	–	–
Qwen3.5-4B	0.7	0.15 (4.7)	0.08 (6.4)	0.06 (4.3)	0.06 (4.4)
	0.8	0.20 (2.2)	0.14 (4.1)	0.10 (1.9)	0.10 (1.4)
	0.9	0.30 (0.6)	0.21 (0.7)	0.18 (0.4)	0.20 (0.6)
Qwen3.5-9B	0.7	0.13 (2.7)	0.08 (2.9)	0.06 (2.5)	0.07 (2.7)
	0.8	0.19 (0.9)	0.13 (1.2)	0.10 (0.6)	0.11 (0.8)
	0.9	0.32 (0.2)	0.23 (0.3)	0.22 (0.2)	0.20 (0.2)

Table 28: **Lower thresholds select less at every length, and on Qwen3.5 the budget grows more slowly than the context.** Mean selected fraction K/L over every layer and decode step of the accuracy-study runs in each length bin, on RULER, BABILong and LongBench-v2. Parentheses give the share of entries at the 512-token floor, in percent. The Qwen3 runs have no bins beyond 32K. An entry at the floor has a nucleus below 512 tokens, so the floor sets its budget instead of the threshold.

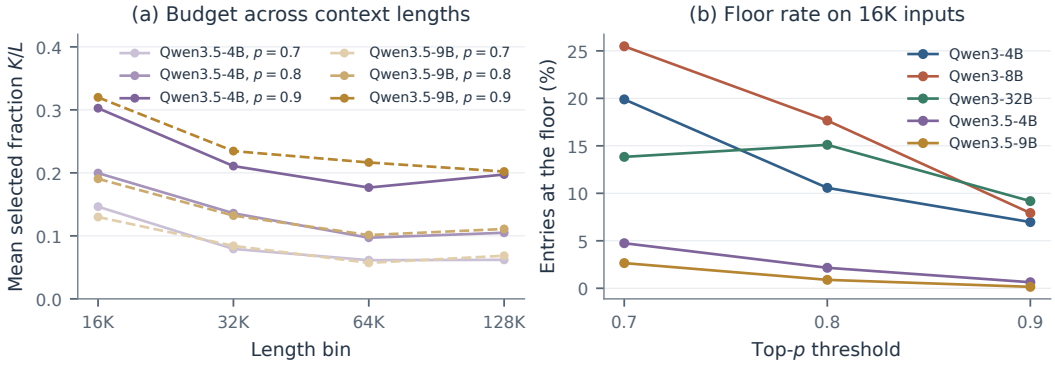

Figure 10: **On Qwen3.5 the budget grows more slowly than the context, and on Qwen3 low thresholds often hit the floor. (a)** Mean selected fraction K/L per length bin on the two Qwen3.5 models, over every layer and decode step of the accuracy-study runs on RULER, BABILong and LongBench-v2. Lines from light to dark show $p = 0.7$, 0.8 and 0.9. **(b)** Share of layer and decode-step entries at the 512-token floor on 16K inputs, by threshold, for the five models of the accuracy study.

J.3 Kept share across models and lengths

Table 29 follows top-p into the serving runs, where every prompt is real text of exactly the stated length and every decode query of every sparse layer is counted. At $p = 0.9$ the four models keep 19 to 38% of the blocks, and at $p = 0.8$ they keep 12 to 27%. The share depends on the model and the length. Qwen3.6-27B keeps less as the prompt grows, from 32% at 8K to 19% at 255.5K at $p = 0.9$. The Qwen3 models keep their smallest share between 16K and 64K and a larger one at 128K and 255.5K.

Model	Selection	8K	16K	32K	64K	128K	255.5K
Qwen3-4B	$p = 0.9$	26.2	20.1	22.1	22.8	29.8	31.5
	$p = 0.8$	18.1	13.0	13.5	13.1	18.3	19.7
Qwen3-8B	$p = 0.9$	26.8	20.2	22.4	21.3	29.8	29.5
	$p = 0.8$	18.5	12.7	13.5	12.3	17.7	18.2
Qwen3-32B	$p = 0.9$	38.3	28.7	29.5	25.5	31.2	30.4
	$p = 0.8$	26.9	19.1	18.7	15.1	19.2	18.5
Qwen3.6-27B	$p = 0.9$	32.0	30.0	28.7	25.4	24.4	19.4
	$p = 0.8$	20.4	17.4	16.4	14.1	13.8	11.9

Table 29: **Top-p keeps a different share of the context on each model and length.** Retained blocks in percent with BF16 selector keys, in the serving runs of Tables 17 to 20, counted by the selection kernels over every decode query and sparse layer. Qwen3 prompts longer than its 40960-token checkpoint window use YaRN, and Qwen3.6-27B runs in its native window.

K Controls and ablations

The runs in this appendix test four choices behind the main results: the shared tile support during training, the loss over selected tokens, the smallest budget, and the model family. Each compares against dense attention or our selector on identical examples, with the evaluation harness of Appendix E.

K.1 Per-query selection during training

Section 2.1 trains with one block set per tile of 16 queries, while decoding selects blocks for each query. Two controls on Qwen3-4B test whether this mismatch matters. The first keeps the paper's checkpoint and switches only the evaluation to per-query selection: every query selects its own blocks from its own scores at the same budgets. The second trains a selector from scratch with a tile of one query, using the recipe in Table 3, so selection and loss are per query throughout training.

Figure 11 overlays the two training curves. They track each other and meet at the last common logged step, 0.0773 nats at step 520. Table 30 reports perplexity and RULER at 32K with $K = 0.5L$. Per-query selection at evaluation moves perplexity by at most 0.006, and the per-query-trained selector moves it by 0.013 at $K = 0.5L$. Both shifts are far smaller than the 0.49 standard error across windows, and no sparse value exceeds dense by more than 0.002. On RULER, both selectors score 96.9% on the same 192 examples, no example changes outcome between them, and both answer two more examples than dense attention. Tile-shared selection therefore costs nothing measurable, and every result in the paper uses it.

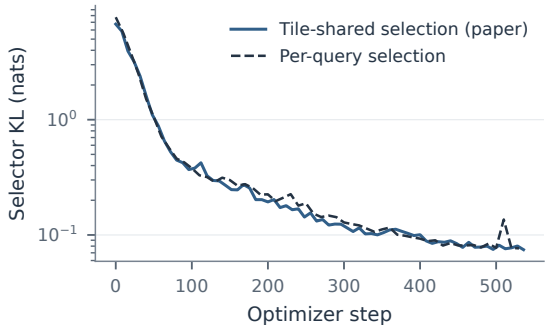

Figure 11: **Training with per-query selection follows the same curve.** Selector KL per logged optimizer step on Qwen3-4B for the paper's tile-shared selection, with one block set per 16 queries, and for per-query selection, with one block set per query, under the same recipe.

Training	Diagnostic	PPL $K = 0.25L$	PPL $K = 0.5L$	PPL $K = 0.75L$	RULER 32K (%)
Tile (paper)	tile	13.511 (0.48)	13.529 (0.49)	13.545 (0.49)	96.9
	query	13.517 (0.49)	13.535 (0.49)	13.547 (0.49)	
Query	query	–	13.542 (0.49)	–	96.9
Dense	–		13.546 (0.49)		95.8

Table 30: **Per-query selection during training changes nothing measurable.** Qwen3-4B. The Training and Diagnostic columns give the selection unit, a 16-query tile or a single query, during training and during the perplexity forward. Standard errors are in parentheses. RULER uses the 32K bin at $K = 0.5L$ ($n = 192$), and decoding selects per query for both selectors.

K.2 KL OVER THE FULL DISTRIBUTION

Section 2.2 normalizes the teacher and the selector over the selected tokens only. The alternative matches their full distributions, normalized over every token the query can see, so the selected blocks never enter the loss. Table 31 compares a Qwen3-4B selector trained with this full-distribution loss against our selector, on the same 192 examples of the 32K RULER bin at three fixed budgets. The full-distribution selector was trained at a budget of 2048 tokens, which its loss never sees, and ours at 8192. The full-distribution selector is less accurate at every budget. At 512 tokens it scores 68.2%, against 94.3% for our selector. At 2048 tokens it scores 87.0%, against 96.9% for our selector and 95.8% for dense attention. It misses 19 examples that dense attention answers and answers two that dense attention misses.

Perplexity does not show this gap. At $K = 0.5L$ the two selectors are within 0.004 of each other, both below dense attention. At 2048 tokens the full-distribution selector reaches 13.491, also below dense. Perplexity averages over every position, so it barely registers the few blocks that a retrieval query needs. Normalizing over the selected tokens therefore keeps retrieval close to dense attention at small budgets, and every result in the paper uses this loss.

	RULER 32K (%) at a fixed budget			
Selector objective	512 (1.6%)	1024 (3.1%)	2048 (6.3%)	PPL $K = 0.5L$
Full distribution	68.2	77.6	87.0	13.532 (0.49)
Selected tokens (paper)	94.3	96.9	96.9	13.529 (0.49)
Dense attention		95.8		13.546 (0.49)

Table 31: **KL over the full distribution loses retrieval accuracy at small budgets.** Qwen3-4B on the same 192 examples of the 32K RULER bin, at three fixed budgets given in tokens and as a share of the bin. PPL is the perplexity diagnostic at $K = 0.5L$, with the standard error in parentheses. The full-distribution selector was trained at $K_{\text{train}} = 2048$, a budget its loss never sees, and ours at $K_{\text{train}} = 8192$. Both selectors use the same architecture and pretrained model and differ in the loss.

K.3 SMALL FIXED BUDGETS

The accuracy study expresses budgets as fractions of the input, and the smallest is $0.25L$. A fixed budget of 2048 tokens tests the selectors far below their training budget of 8192 tokens: 6.3% of a 32K input and 1.6% of a 128K input. Table 32 pairs it with dense attention on RULER for the five models of the accuracy study, on the harness of Appendix E with 256 output tokens. These runs sample their own RULER examples, so their dense scores differ from those of Table 31. Three of the seven cells are unchanged to the example, one trades one example each way, two gain one example, and one loses three: Qwen3-32B at 32K, -1.5 points. The result is specific to retrieval. RULER's needles occupy a few blocks, and a selector that ranks those blocks first needs no more than this budget. Tasks that need broad context carry no such guarantee, and we did not evaluate the 2048-token budget on BABILong or LongBench-v2.

Model	Bin	Share (%)	n	Dense	$K = 2048$	Δ	Flips +/−
Qwen3-4B	32K	6.3	200	97.5	97.5	+0.0	0 / 0
Qwen3-8B	32K	6.3	200	93.0	93.5	+0.5	1 / 0
Qwen3-32B	32K	6.3	200	96.5	95.0	−1.5	0 / 3
Qwen3.5-4B	64K	3.1	211	95.3	95.3	+0.0	0 / 0
	128K	1.6	189	96.3	96.3	+0.0	0 / 0
Qwen3.5-9B	64K	3.1	211	98.1	98.6	+0.5	1 / 0
	128K	1.6	189	97.9	97.9	+0.0	1 / 1

Table 32: **A fixed budget of 2048 tokens stays within 1.5 points of dense or above it on RULER.** The selectors trained at $K_{\text{train}} = 8192$ run at 2048 selected tokens without retraining, paired with dense attention on the same examples of the evaluation harness, with 256 output tokens. Share is 2048 as a percentage of the bin length, and Δ is the accuracy difference in percentage points. Flips count examples correct only under selection, +, and only under dense attention, −.

K.4 A MODEL OUTSIDE THE QWEN FAMILIES

Gemma-4-E4B mixes sliding-window layers, full-attention layers and layers that reuse the KV cache of an earlier layer. We attach selectors to the four full-attention layers that own a cache, layers 5, 11, 17 and 23. They add 10.5M parameters, 0.14% of the model, and train with the recipe of Table 3 unchanged. Table 33 evaluates the checkpoint on the harness of Appendix E with dense prefill and sparse decode, against dense attention on the same examples. Decoding uses the checkpoint's own generation settings instead of the Qwen protocol of Appendix E.2, and prefill runs in chunks because a single 128K forward does not fit. RULER is 99.5% under every setting, with no example changing. The pooled differences from dense are $+0.7$ points at $K = 0.5L$ and 0.0 at $p = 0.9$, with one-sided bounds of -1.1 and -1.5. The equal-weight mean over the three benchmarks moves from 53.5 to 54.1 and 53.6. The Gemma result is one run on one checkpoint and serves as an accuracy control, not a deployment result. Gemma-4's 512-wide attention heads have no FlashAttention decode path on B200, so we do not serve it.

Setting	RULER	BABILong	LongBench-v2	Mean	Pooled Δ (bound)
Dense attention	99.5 (0.5)	27.7 (2.5)	33.5 (3.0)	53.5 (1.3)	–
Highlight K=0.5L	99.5 (0.5)	28.3 (2.5)	34.6 (3.0)	54.1 (1.3)	+0.7 (−1.1)
Highlight top-p=0.9	99.5 (0.5)	26.3 (2.5)	35.0 (3.0)	53.6 (1.3)	+0.0 (−1.5)

Table 33: **On Gemma-4-E4B, the unchanged recipe matches dense accuracy.** Percent accuracy with bootstrap standard errors on the evaluation harness of Appendix E, on 183, 300 and 254 examples, the equal-weight mean, and the pooled paired difference from dense in percentage points with its one-sided 95% lower bound in parentheses. Decoding uses the generation settings of the checkpoint, temperature 1.0, top-$p = 0.95$ and top-$k = 64$, with chunked prefill. The selector uses the recipe of Table 3 unchanged on the four full-attention layers that own a KV cache, layers 5, 11, 17 and 23.

L RELATED WORK

Learned sparse attention. NSA and MoBA train block-sparse attention into the model itself (Yuan et al., 2025; Lu et al., 2025). DSA introduces a learned lightning indexer and trains the model with it (DeepSeek-AI, 2025). A short warm-up distills the indexer from dense attention with the model frozen, and continued training then adapts the whole model to sparse attention. Highlight follows its scoring design and keeps the model frozen throughout. SeerAttention learns block-level gates from pooled queries and keys through self-distillation for sparse prefill (Gao et al., 2024). SeerAttention-R, the closest prior work, extends the gate to decoding (Gao et al., 2025). It distills max-pooled block attention of a frozen reasoning model into a plug-in gate over 64- or 128-token blocks, and reports kernel-level speedups while leaving end-to-end serving to future work. Highlight instead scores

tokens, uses 16-token blocks only to choose what to read, and matches token distributions on the selected tokens. We also measure whole-model throughput in a serving engine. SAS argues that distilled rankings misalign with prediction under a fixed budget, and trains the selector through the language-modeling loss (Li et al., 2026). Highlight stays with distillation and restricts the loss to the selected tokens. MISA routes a subset of an existing DSA indexer's heads, and IndexCache reuses one layer's selections in the layers that follow (Zhou et al., 2026; Bai et al., 2026). Both reduce the indexing cost of models trained with DSA, whereas Highlight adds a selector to models trained without one. Table 34 compares these selectors side by side.

Selection without selector training. Quest bounds each page's relevance to the query from the minimum and maximum of its keys (Tang et al., 2024), and Twilight prunes such a candidate set to an adaptive top-p budget (Lin et al., 2025). SparQ approximates attention scores from a subset of query and key dimensions (Ribar et al., 2024). Each estimate adds its own cost at every step, so Section 5.3 times the selector call apart from attention. Highlight learns its scoring representation from the model and attends to the original keys and values of the selected blocks.

KV-cache management and precision. PagedAttention manages KV memory for batched serving (Kwon et al., 2023). Eviction methods such as H2O drop cached tokens permanently (Zhang et al., 2023), and KVQuant and KIVI quantize the cached keys and values (Hooper et al., 2024; Liu et al., 2024). Highlight keeps the model's KV cache complete and in BF16, so a block skipped at one step can be read at the next. Its quantization applies only to the selector's own keys.

Method	Signal	Scoring unit	Selector training	Budget	Model
StreamingLLM	sink and recent positions	fixed positions	none	fixed window	frozen
Quest	page key bounds (min, max)	16-token pages	none	top-K pages	frozen
Twilight	a base selector's candidates, re-scored with INT4 keys	tokens	none	top-p within the candidates	frozen
DSA	learned lightning indexer	tokens	KL to dense attention, then on the selected tokens	top-K	trained in the sparse stage
SeerAttention-R	learned gate on pooled keys	64-token blocks	KL to block-pooled attention	top-K or threshold	frozen
SAS	the SeerAttention-R gate	64-token blocks	language-modeling loss	top-K	frozen
MISA	routed heads of a DSA indexer	tokens	none	top-K	frozen
IndexCache	a DSA indexer shared across layers	tokens	none, or KL to the attention of the layers served	top-K	frozen, or trained with DSA
Highlight	learned selector (Eq. 1)	tokens, selected in 16-token blocks	KL on the selected tokens	top-K or dual top-p	frozen

Table 34: **Selectors differ in the signal that decides what to read and in the rule that sets how much.** Each row follows the method's paper: StreamingLLM (Xiao et al., 2024), Quest (Tang et al., 2024), Twilight (Lin et al., 2025), DSA (DeepSeek-AI, 2025), SeerAttention-R (Gao et al., 2025), SAS (Li et al., 2026), MISA (Zhou et al., 2026) and IndexCache (Bai et al., 2026). Block sizes are the papers' defaults. For SAS, the model is frozen in its main post-training setting. For IndexCache, the two entries are its training-free and training-aware variants.