ORACLE-GUIDED SPARSE PREFILL: SEPARATING ORACLE, INDEXER, AND RUNTIME GAPS IN LONG-CONTEXT GQA

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-context prefill remains expensive because global attention layers, including those using grouped-query attention (GQA), must compute pairwise query–key scores across the entire history, incurring quadratic cost in sequence length even in hybrid models that combine global attention with local, linear, or recurrent components. We study when this computation can be sparsified without materially changing task performance. We use an attention-mass top-k oracle that selects a head-averaged token support from dense attention. This non-deployable reference separates sparse-budget feasibility from indexer error and runtime realization effects. We evaluate a head-collapsed indexer trained by KL distillation with the backbone frozen. On Qwen3.5-27B RULER samples from 32K to 128K, matched per-query and 32-query-shared oracle and indexer configurations remain within one observed macro-accuracy point of dense. At 128K, the 32-query-shared indexer scores 0.61 points below dense with a 1.46× single-request TTFT speedup. On sampled tail queries, the indexer recovers 94–98% of the attention mass retained by the oracle, yet task-level failures coexist with this high aggregate fidelity. Key-block experiments show that oracle-feasible support structures can still fail under learned selection. These observations motivate configuration-aware support selection and establish a diagnostic study of sparse-prefill quality–latency tradeoffs.

1 INTRODUCTION

Long-context prefill is a major cost in serving language models because the prompt must be processed before the first generated token. Modern long-context architectures increasingly combine local, sparse, linear, recurrent, and full-attention components, as in Gemma-style hybrids, Qwen3-Next, Jamba, and related deployments (Gemma Team, 2024; 2025; Qwen Team, 2025; Lieber et al., 2024; Jiang et al., 2023). Full or grouped-query attention (GQA) layers often remain the places where long-range evidence is retrieved and fused. GQA reduces KV-cache size by sharing key-value heads (Ainslie et al., 2023), but it does not remove the dense search over historical tokens inside full-attention layers. This raises a diagnostic question for sparse prefill: *under a fixed support granularity and top-k budget, how much of this dense attention search is actually needed to preserve task-level behavior?*

Sparse prefill attends to selected KV positions, but its quality losses can have distinct origins: an insufficient budget, inaccurate support prediction, coarse support sharing, or runtime realization effects. A controlled support-selection reference helps distinguish these cases.

We take an oracle-first view: an *attention-mass top-k oracle* computes dense attention, selects head-averaged support at a specified granularity, and recomputes sparse attention. This non-deployable reference tests support feasibility before evaluating the learned selector and its runtime realization.

MInference, FlexPrefill, Prism, and NSA pursue sparse-attention efficiency (Jiang et al., 2024; Lai et al., 2025; Wang et al., 2026; Yuan et al., 2025). Learned selection and frozen-backbone distillation are also established: DSA uses frozen-backbone KL warm-up before joint sparse training (DeepSeek-AI, 2025); SeerAttention learns pooled block gates (Gao et al., 2024); VSPrefill learns vertical–slash prefill selection (Chen, 2026). SpotAttention uses a related KL-distilled selector, but its task evaluations retain dense prefill and sparsify decode (Ahmad and Yun, 2026). The Sparse Frontier studies training-free sparsity tradeoffs across granularities and tasks (Nawrot et al., 2026).

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.

Building on this established distillation approach, our contribution is a matched diagnostic study of oracle feasibility, learned selection, and runtime realization on frozen GQA backbones (Section 2). We instantiate and evaluate a head-collapsed scorer with a training objective closely related to DSA's warm-up, keeping the backbone frozen and the inference budget adjustable. We do not establish superiority over existing systems.

On Qwen3.5-27B at 128K, sharing one support across 32 query positions yields macro-accuracy 0.61 points below dense with a 1.46× single-request TTFT speedup against its paired dense baseline. Key-block and cross-model results reveal configuration-dependent operating points.

Our contributions are:

- We define an attention-mass top-k oracle that selects head-averaged token support from dense attention and use it to organize evidence around oracle gap, indexer gap, and realization gap.
- We instantiate and evaluate a frozen-backbone, head-collapsed, KL-distilled indexer with an adjustable inference budget.
- We establish a matched Qwen3.5-27B dense–oracle–indexer quality–latency frontier at context lengths ranging from 32K to 128K.
- We identify two limits: key-block oracle feasibility does not ensure indexer quality, and the RULER operating point does not transfer uniformly across the evaluated models. Task-wise and non-retrieval results further delimit these findings.

2 METHOD

2.1 SUPPORT GRANULARITY AND TOP-K BUDGET

Consider a frozen causal language model with a full-attention GQA layer ℓ. Let H_q be the number of query heads, H_{kv} the number of KV heads, d_h the head dimension, and $g : \{1, \dots, H_q\} \to \{1, \dots, H_{kv}\}$ the GQA group mapping from query head to KV head. For query position t and key position $s \le t$, dense GQA attention computes

$$A_{\ell,t,h,s} = \mathrm{softmax}_{s' \le t}\left(\frac{\langle q_{\ell,t,h}, k_{\ell,s',g(h)}\rangle}{\sqrt{d_h}}\right)_s, \qquad o_{\ell,t,h} = \sum_{s \le t} A_{\ell,t,h,s} v_{\ell,s,g(h)}.$$

Sparse prefill replaces the full historical sequence with a support set. In this paper, support granularity has two distinct axes:

- **Head axis:** support may be selected separately per query head $\mathcal{S}_{\ell,t,h}$, shared within each KV group $\mathcal{S}_{\ell,t,r}$ for query heads with $g(h) = r$, or collapsed into one head-averaged token support $\mathcal{S}_{\ell,t}$ shared by all query heads at position t.
- **Query-position axis:** support may be selected per query position (`query_share_size` = 1), shared by a selection block of query positions (`query_share_size` > 1), or selected at a coarser token/block level depending on the runtime.

The reported experiments use head-averaged token support; per-head and per-KV-group supports are not evaluated. We denote the number of consecutive query positions sharing one selected support by `query_share_size`, the *selection block size*. When `query_share_size` = 1, we call the realization *per-query*; when `query_share_size` > 1, we call it *selection-block-shared*. This notation separates query-position support sharing from the GQA head mapping $g(h)$.

For a head-averaged token support $\mathcal{S}_{\ell,t}$ with $|\mathcal{S}_{\ell,t}| \le k$, sparse attention recomputes the original attention operation over selected keys:

$$A^{\mathcal{S}}_{\ell,t,h,s} = \mathrm{softmax}_{s' \in \mathcal{S}_{\ell,t}}\left(\frac{\langle q_{\ell,t,h}, k_{\ell,s',g(h)}\rangle}{\sqrt{d_h}}\right)_s, \qquad o^{\mathcal{S}}_{\ell,t,h} = \sum_{s \in \mathcal{S}_{\ell,t}} A^{\mathcal{S}}_{\ell,t,h,s} v_{\ell,s,g(h)}.$$

2.2 Attention-Mass Oracle and Three Gaps

The oracle aggregates dense attention into a head-averaged token-importance distribution:

$$P_{\ell,t}(s) = \frac{1}{H_q} \sum_{h=1}^{H_q} A_{\ell,t,h,s}.$$

Because each head-wise causal attention row sums to one over valid keys, their arithmetic mean $P_{\ell,t}$ is itself a normalized distribution. The oracle support is

$$\mathcal{S}^{\star}_{\ell,t} = \mathrm{TopK}_s(P_{\ell,t}(s), k).$$

Sparse attention is then recomputed over $\mathcal{S}^{\star}_{\ell,t}$ using the original model's queries, keys, and values. The oracle therefore uses dense attention only to choose the support, not to replace the sparse attention output.

In the full-model oracle evaluation, patched full-attention layers are executed recursively along the sparse trajectory. At each patched layer, the oracle computes dense attention scores from the current hidden states to choose the top-k support, but the layer returns the recomputed sparse attention output to the model. Consequently, later oracle decisions are based on hidden states already affected by earlier sparse replacements rather than on a separate dense-teacher trajectory.

This oracle is intentionally narrower than a global sparse optimum. It evaluates whether head-averaged token support selected by dense attention mass is sufficient at a given budget.

For a selection block B of consecutive query positions, the reported selection-block-shared implementation aggregates valid per-query token scores with a max rule,

$$P_{\ell,B}(s) = \max_{t \in B:\, s \le t} P_{\ell,t}(s), \qquad \mathcal{S}^{\star}_{\ell,B} = \mathrm{TopK}_s(P_{\ell,B}(s), k).$$

Each query contributes only causally valid probabilities to the max aggregate. The selected support is shared within B, but each query masks entries with $s > t$ during sparse recomputation. Selection can nevertheless depend on later queries within the same block. This is a block-level approximation for full-prompt prefill, not a guarantee of prefix-invariant streaming execution.

We use three controlled comparisons, rather than independent additive error terms:

- **Oracle gap:** dense versus attention-mass oracle sparse attention at a specified budget and support granularity.
- **Indexer gap:** indexer versus oracle support at matched budget, granularity, and evaluation settings.
- **Realization gap:** comparisons varying support sharing or runtime choices while holding the selector and budget fixed. Sharing already enters the oracle gap at the chosen granularity; its effect is not counted again as an independent error.

2.3 Head-Collapsed Indexer and KL Distillation

The original model is kept frozen. The sparse module is an auxiliary indexer trained after pretraining. It consumes hidden states from the frozen model and predicts token importance for sparse top-k selection.

To predict the head-averaged target $P_{\ell,t}(s)$, we choose a single auxiliary scoring head:

$$q_{\ell,t} = \bar{W}_Q x_{\ell,t}, \qquad k^{\mathrm{idx}}_{\ell,s} = \mathrm{LN}(\bar{W}_K x_{\ell,s}),$$

followed by positional encoding and scaled similarity:

$$z_{\ell,t,s} = \mathrm{ReLU}\left(\frac{\left\langle \mathrm{RoPE}(q_{\ell,t}), \mathrm{RoPE}(k^{\mathrm{idx}}_{\ell,s}) \right\rangle}{\sqrt{d_{\mathrm{idx}}}} \right).$$

This reduces indexer-side projection activations and removes the need to carry an explicit auxiliary head dimension through scoring and top-k selection. Importantly, this is not backbone head pruning.

The original model and its KV cache remain unchanged; the compression applies only to the auxiliary support predictor.

The implementation uses a single sparse index head. For Qwen3.5, the reported rows use $d_{\text{idx}} = 256$, which equals the backbone attention head dimension rather than being smaller. In the deployed Qwen3.5-27B path, each of the 16 full-attention layers contains a bias-free $5120 \to 2 \times 256$ Q/K projection and a 256-dimensional affine key normalization, for 41,951,232 auxiliary parameters in total.

The head-collapsed scorer is a systems-oriented design choice, not an ablation-proven optimum: multi-head, per-KV-group, and indexer-dimension alternatives are not evaluated.

The student indexer produces a score $z_{\ell,t}(s)$ for each candidate key position. It is trained to match the dense teacher token-importance distribution $P_{\ell,t}(s)$ through KL divergence:

$$\mathcal{L}_{\text{KL}} = \frac{1}{|\mathcal{Q}|} \sum_{(\ell,t)\in\mathcal{Q}} \text{KL}\left(P_{\ell,t} \,\|\, \text{softmax}(z_{\ell,t})\right),$$

where the softmax is over valid causal key positions $s \le t$, and $\mathcal{Q}$ denotes a streamed set of supervised layer/query rows drawn from the internal distillation corpus. Chunked training avoids materializing all attention rows across all layers at once. All distilled-indexer results reported use this KL distillation objective; we do not mix in additional hit-rate or rank losses.

At inference, the block indexer first normalizes each query over valid causal keys, $\widehat{P}_{\ell,t} = \text{softmax}(z_{\ell,t})$, then selects top-k from $\max_{t\in B:\, s\le t} \widehat{P}_{\ell,t}(s)$. Thus sharing aggregates per-query probabilities, not raw ReLU scores; the budget remains adjustable.

2.4 Cost Model

For sequence length S, indexer scoring remains $O(S^2 d_{\text{idx}})$ per layer: every query-key pair is scored before block aggregation. At $d_{\text{idx}} = d_h$ and $H_q = 24$, indexer QK arithmetic is $1/24$ of dense QK arithmetic, excluding normalization and selection; this is not a runtime speedup bound. With $g_q = \texttt{query_share_size}$, sharing reduces selection rows to $O(S/g_q)$ and amortizes top-k work, but does not divide QK arithmetic by g_q. Sparse value aggregation costs $O(Skd_h)$ per head. Memory traffic, normalization, top-k, and kernel fusion also affect runtime, motivating end-to-end TTFT measurements.

In the reported chunked path, full K/V tensors may be reconstructed before sparse attention; the implementation is not limited to reconstructing only the selected KV blocks.

Detailed algorithms, scorer choices, and training-metadata limitations are provided in Appendix A.

3 Qwen3.5-27B Quality-Latency Frontier

We compare RULER (Hsieh et al., 2024) accuracy and end-to-end time to first token (TTFT) across matched Qwen3.5-27B configurations, using the oracle to assess sparse-support feasibility and the distilled indexer to evaluate the realized quality–latency tradeoff.

3.1 RULER Accuracy

We compare dense, per-query and block oracle, and per-query and block indexer accuracy at 32K / 64K / 128K, with 50 fixed RULER samples in each of 13 equally weighted task groups; the 256K extension is oracle-only. The one-point boundary is a pre-specified engineering criterion, not a statistical-equivalence test. Block rows use `query_share_size` = 32, selected on these same samples in Section 4. For binary-scored tasks, one sample changes macro-accuracy by $2/13 \approx 0.154$ points; partial-credit tasks have finer increments. Results are descriptive point estimates, with no independent confirmation of the selected operating point. Appendix B gives the protocol.

All evaluated sparse configurations stay within one observed macro-accuracy point of dense (Table 1), but this does not imply per-task preservation. At 128K, dense saturates 11 of 13 task groups; the block indexer reduces niah_multikey_3 from 100 to 92 (four additional failures out of 50), despite a

Table 1: Qwen3.5-27B RULER accuracy across dense, per-query, and selection-block-shared oracle and indexer configurations.

Context	Top-k	Dense	Per-query oracle	Block oracle	Per-query indexer	Block indexer
32K	1024	97.54	97.38	97.38	97.38	97.38
64K	2048	97.69	97.33	97.54	97.54	97.13
128K	2048	96.46	96.46	96.46	96.62	95.85
256K	2048	96.86	96.40	95.97	–	–

macro gap of only -0.61. The block oracle matches dense, indicating feasibility at this granularity on these samples. The per-query indexer is closer to dense than the block indexer at 64K and 128K; these observed orderings are not statistically established rankings.

Accuracy gaps are reported in Appendix C, Table 9. To assess which task groups account for the quality differences, we report per-task accuracy in Tables 7 and 8. Appendix E presents task-wise failures at 64K and 128K, comparing the model's prediction with the expected answer.

The 256K extension supports sparse feasibility under the oracle only. Indexer accuracy and TTFT require further evaluation to establish the matched frontier at this context length.

3.2 END-TO-END TTFT

We measure single-request TTFT through the serving stack, including non-attention computation, KV-cache movement, top-k selection, and execution overheads. Appendix A describes the implementation path.

In distilled-indexer quality runs, the backbone is fully frozen: indexer-selected sparse attention is used during prefill, decode remains dense, and only the auxiliary indexer weights are loaded. The method therefore does not reduce decode attention work, decode-time KV-cache traffic, or KV-cache storage. Dense decode consumes prefix states and cached activations produced by sparse prefill, so the quality tables evaluate the full sparse-prefill followed by dense-decode generation path.

We use 3 warmup requests and 10 timed requests per prompt. Each request contains a synthetic prompt of the target token length and generates one token with temperature zero. Prefix caching is disabled. Each iteration is measured independently with GPU events and followed by GPU synchronization; model loading and one-time kernel compilation are outside the timed region. With a context sweep of 32K / 64K / 128K, we report paired mean TTFT, standard deviation, and speedup; full statistics and memory measurements are provided in Appendix D.

The Qwen3.5-27B TTFT frontier was measured on one 80GB PCIe GPU with tensor parallelism disabled (TP = 1). All experiments below use BF16, eager execution, a 4K chunked-prefill budget, $d_{\text{idx}} = 256$, top-k = 1024 at 32K, and top-k = 2048 at 64K / 128K. The runner sets `max_model_len` to the target context plus a 16-token safety margin and otherwise uses vLLM's default GPU-memory utilization. The canonical logs record PyTorch 2.11.0, Transformers 5.7.0, vLLM 0.20.1, and implementation commit `7284816`. Dense rows use vLLM's FlashAttention-2 (Dao, 2023) backend, while sparse rows use the PropheticSparseAttn custom attention backend for the modified full-attention prefill path. The pq_pk pipeline refers to per-query, per-token-key support (`query_share_size` = 1), while bq_pk refers to query-block, per-token-key support (`query_share_size` = 32); the later bq_bk notation instead uses contiguous key blocks. Both sparse rows load the same distilled Qwen3.5-27B indexer checkpoint, and each speedup is calculated against the dense run paired with that sparse cell.

Table 2 shows that measured speedup increases with context length and is higher for the bq_pk pipeline. This is consistent with fixed overheads accounting for a larger fraction of TTFT at shorter contexts and attention-related computation becoming more important at longer contexts. Full paired timings are reported in Appendix D, Table 10.

The independently launched dense cells differ by 2.4% at 32K, 0.12% at 64K, and 0.21% at 128K. We therefore calculate every speedup against the dense run paired with that sparse cell rather than

Table 2: End-to-end TTFT of Qwen3.5-27B per-query and selection-block-shared sparse indexer relative to dense baseline. Paired means and standard deviations (std) are in milliseconds; speedups use paired mean timings.

Context	Pipeline	Dense mean	Sparse mean	Dense std	Sparse std	Speedup
32K	pq_pk	9958.2	9778.1	93.7	59.9	1.02x
32K	bq_pk	9719.1	8905.8	30.9	27.9	1.09x
64K	pq_pk	22941.2	21624.3	79.4	91.0	1.06x
64K	bq_pk	22968.0	18978.2	59.3	62.9	1.21x
128K	pq_pk	58095.5	44640.4	56.0	41.5	1.30x
128K	bq_pk	58217.3	39807.5	223.3	78.1	1.46x

reusing a single baseline. The 32K pq_pk reduction is only 1.81% and should be interpreted as approximately parity; the stronger 64K / 128K reductions and the bq_pk trend remain visible under the paired comparisons.

3.3 Quality-Latency Tradeoff

Figure 1: Configuration-level quality–latency tradeoffs for Qwen3.5-27B. Solid lines connect lengths within each pipeline; dotted lines pair pipelines at the same length. Different lengths are distinct workloads, not a single Pareto comparison.

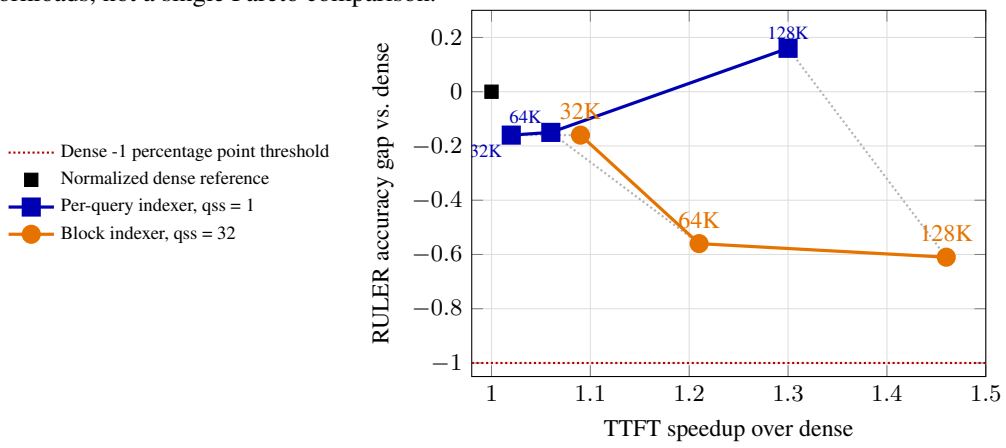


Figure 1 summarizes the measured tradeoff. At 128K, the block indexer achieves a 1.46× single-request TTFT speedup with a 0.61-point macro-accuracy loss, within our reporting criterion. The largest measured speedup occurs at the longest evaluated context.

Accuracy is measured on RULER prompts and latency on length-matched synthetic prompts, so this is a configuration-level comparison rather than a per-prompt paired measurement.

4 Support Granularity Ablation

We compare `query_share_size` (qss) = 1 / 8 / 16 / 32 / 64 to assess how sharing support across consecutive query positions affects accuracy and TTFT. Per-query selection (qss = 1) provides the finest-grained adaptation, while query-block sharing amortizes support-selection cost. We evaluate Qwen3.5-27B at 64K / 128K using the same RULER protocol as above, seeking an operating point within one percentage point of dense accuracy.

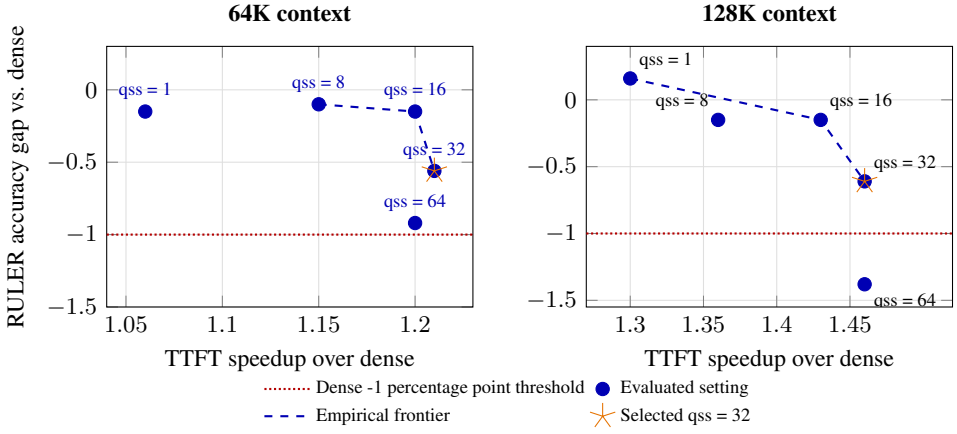
The attention-mass oracle is a controlled support-selection reference, not a theoretical upper bound on task accuracy, because it selects support according to dense attention mass rather than task correctness.

Table 3: Qwen3.5-27B RULER accuracy and sparse-indexer TTFT speedup relative to dense across query-support sharing sizes (qss). Dense accuracy is 97.69 at 64K and 96.46 at 128K.

Context	qss	Oracle accuracy	Indexer accuracy	TTFT speedup
64K	1	97.33	97.54	1.06x
64K	8	97.49	97.59	1.15x
64K	16	97.49	97.54	1.20x
64K	32	97.54	97.13	1.21x
64K	64	97.49	96.77	1.20x
128K	1	96.46	96.62	1.30x
128K	8	96.46	96.31	1.36x
128K	16	96.46	96.31	1.43x
128K	32	96.46	95.85	1.46x
128K	64	96.31	95.08	1.46x

Table 3 shows that the evaluated indexer configurations with qss ≤ 32 remain within one percentage point of dense at both context lengths. Coarser sharing at qss = 64 reduces indexer accuracy further and exceeds our threshold at 128K, while the oracle remains close to dense. Task-wise results show quality losses on retrieval-heavy tasks, notably niah_multikey_3 at 64K and niah_multikey_2 at 128K (Appendix F, Table 17 for niah_multikey_3).

Figure 2: Observed quality–latency tradeoffs across support-sharing settings. Higher values are better. Dashed lines connect empirically non-dominated measured points, without statistical ranking guarantees; the red line marks the one-point engineering criterion.



We select qss = 32 as the highest-sharing evaluated configuration within the one-point threshold at both lengths, with the largest measured speedup under that constraint. On these samples, qss = 64 offers no additional speedup and scores lower. Figure 2 shows the empirical frontier of measured points, not a statistically resolved optimum. Selection and reporting use the same samples; qss = 32 is therefore a configuration-specific operating point, not an independently validated recommendation.

At 128K, the block oracle with qss = 32 scores 0.59 points below dense on Gemma-4-26B-A4B-it but 11.32 points below dense on Qwen3-32B under its evaluated context-extension setting (Appendix J). These oracle-only results do not establish cross-model indexer quality or TTFT. On the evaluated InfiniteBench and LongBench-v2 subsets, qss = 64 yields smaller aggregate gaps than qss = 32 for both oracle and indexer (Appendix H). Thus, the RULER operating point does not transfer uniformly across these benchmarks, models, and context-extension settings.

5 INDEXER GAP ABLATION

To characterize the indexer gap, we distinguish two groups: Group A consisting of dense / per-query oracle / per-query indexer and Group B consisting of dense / block oracle / block indexer (`query_share_size` = 32). We use the Qwen3.5-27B model with the corresponding indexer checkpoint, testing on 50 pre-prepared RULER samples per task group. Consequently, each pipeline / context length pair evaluates 650 prompts. For each prompt, we generate a single token (`max_tokens` = 1), since generation serves only to execute the inference request and full predictions are not required by this ablation.

We report oracle top-k overlap (support intersection), attention mass recall (AMR; dense attention mass retained), relative AMR (indexer/oracle AMR), and validation KL. Table 4 averages these metrics over the 16 full-attention layers, using 256 tail queries per included prompt. Only prompts reaching the measurement checkpoint are included. Full metric definitions, measurement protocol, sample-exclusion counts, and layer-wise fidelity results are provided in Appendix G.

Table 4: Qwen3.5-27B layer-average oracle top-k overlap, attention mass recall (AMR), relative AMR, AMR gap, and validation KL across RULER tasks for Group A: dense / per-query oracle / per-query indexer and Group B: dense / block oracle / block indexer. All cases use a measurement position one 4K chunk below the context length: 28K, 60K, and 124K respectively. Overlap and AMR quantities are fractions; KL is reported in nats.

Context	Sharing	Top-k overlap	Oracle AMR	Indexer AMR	Relative AMR	AMR gap	KL
32K	Per-query	0.74	0.75	0.72	0.96	0.03	0.22
32K	Block	0.69	0.62	0.61	0.98	0.01	0.22
64K	Per-query	0.71	0.76	0.73	0.95	0.03	0.27
64K	Block	0.66	0.65	0.63	0.97	0.02	0.27
128K	Per-query	0.65	0.71	0.67	0.94	0.04	0.35
128K	Block	0.62	0.61	0.59	0.96	0.02	0.35

Table 4 shows lower oracle top-k overlap with block-shared selection than with per-query selection at all evaluated context lengths.

The indexer retains 94–98% of its oracle reference's attention mass, yet task-level failures coexist with this high aggregate relative AMR. The denominators differ across granularities: at 128K, block indexer absolute AMR is 0.59 versus 0.67 per-query, despite higher relative AMR. These aggregate measurements do not isolate support identity or particular layers as causal sources of task errors.

Validation KL increases with context length, suggesting a greater discrepancy between dense and indexer distributions at longer contexts.

6 KEY-BLOCK REALIZATION ABLATION

We evaluate query-block key-block support (bq_bk) with $b_k \in \{64, 128, 256\}$ at 64K and 128K, using Qwen3.5-27B, 50 RULER samples per task, and qss = 32. For both selectors, each key block is scored by summing its causally valid token probabilities per query, then taking the maximum across the query block. The fixed 2048-token budget selects 32/16/8 key blocks for $b_k = 64/128/256$, respectively.

Table 5: Qwen3.5-27B RULER accuracy and end-to-end sparse-indexer TTFT speedup relative to dense for query-block versus key-block support across $b_k \in \{64, 128, 256\}$.

Context	Metric	Query block	Key block $b_k = 64$	Key block $b_k = 128$	Key block $b_k = 256$
64K	Oracle accuracy	97.54	97.49	97.43	95.38
64K	Indexer accuracy	97.13	90.92	87.74	79.49
64K	Indexer TTFT speedup	1.21x	1.20x	1.18x	1.21x
128K	Oracle accuracy	96.46	96.60	96.60	93.67
128K	Indexer accuracy	95.85	86.48	82.88	74.89
128K	Indexer TTFT speedup	1.46x	1.45x	1.46x	1.45x

Table 5 shows that key-block oracle accuracy remains close to the query-block reference for $b_k \in \{64, 128\}$, but drops at $b_k = 256$ at both lengths. The key-block indexer loses 6.21–20.96 points relative to the query-block indexer, failing our quality criterion at every evaluated block size and degrading as b_k increases (Table 5). Failures concentrate in niah_multikey_2, niah_multikey_3, and cwe. Oracle-level feasibility at $b_k \in \{64, 128\}$ therefore does not translate into quality preservation for the current key-block indexer; the mechanism remains unresolved.

Table 5 shows no additional TTFT gain over the query-block indexer in the current implementation and evaluated configurations, alongside lower accuracy. This is not a general limitation of key-block sparse kernels; profiling is needed to explain the observed runtime behavior.

7 Limitations

The matched quality–latency conclusions apply to Qwen3.5-27B on the evaluated 32K–128K RULER samples, not to arbitrary GQA backbones. Only its 16 full-attention layers are modified; 48 GDN layers remain unchanged. Preliminary non-retrieval, scaling, and cross-model studies (Appendices H, I, and J) do not establish a matched distilled-indexer frontier across models or isolate the cause of Qwen3-32B's 128K oracle loss.

TTFT uses single requests (batch size 1) in eager execution; batched throughput and per-stage profiling are not evaluated. With ten timed requests, the observed maximum is a descriptive tail proxy, not a p99 estimate. Matched comparisons with external sparse-prefill methods and simple selectors (random top-k, sink plus window, or untrained indexers) remain future work. Thus, the present comparisons do not establish an advantage over these alternatives.

8 Conclusion

This paper studies GQA sparse prefill through an attention-mass top-k oracle that tests support feasibility, defines the distillation target, and organizes oracle / indexer / realization gap analysis. We provide matched Qwen3.5-27B quality–latency measurements from 32K to 128K. At 128K, the selection-block-shared indexer scores 0.61 macro-accuracy points below dense with a 1.46× single-request TTFT speedup.

High aggregate relative AMR coexists with task-level failures; identifying their causes requires further controlled analysis. Key-block oracles with $b_k = 64$ and $b_k = 128$ remain close to the query-block reference, whereas key-block indexers lose accuracy without additional TTFT gains in this implementation. The selected RULER operating point does not transfer uniformly to the evaluated non-retrieval samples or across models and context-extension settings (Appendices H and J).

AI USE STATEMENT

Generative AI tools were used for language editing and formulation, improving the structure and readability of the manuscript, and reviewing the presentation of the work to provide feedback. They were also used for limited code adaptation and visualization assistance, and provided review comments and feedback on the methodology and the authors' analysis and interpretation of results.

The authors conducted the analyses, checked the reasoning, validated the results, and reviewed all AI-assisted content. They take full responsibility for the final manuscript and its conclusions.

REPRODUCIBILITY STATEMENT

The method formulation is provided in Section 2, with oracle execution and indexer implementation details in Appendix A. Appendix B documents the experimental setup and evaluation protocol. Additional appendices provide metric definitions and measurement procedures (Appendix G), non-retrieval subset construction (Appendix H), and detailed results. Some training metadata, including the sampling distribution and checkpoint-selection rule, are not publicly available.

REFERENCES

Huzama Ahmad and Se-Young Yun. SpotAttention: Plug-in block-sparse routing for pretrained long-context transformers. *arXiv preprint arXiv:2606.22874*, 2026.

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*, 2023.

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. *arXiv preprint arXiv:2412.15204*, 2024.

Guanzhong Chen. VSPrefill: Vertical-slash sparse attention with lightweight indexing for long-context prefilling. *arXiv preprint arXiv:2603.04460*, 2026.

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

DeepSeek-AI. DeepSeek-V3.2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.

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 preprint arXiv:2410.13276*, 2024.

Gemma Team. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

Gemma Team. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

Gemma Team. Gemma 4 technical report. *arXiv preprint arXiv:2607.02770*, 2026.

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

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lelio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothee Lacroix, and William El Sayed. Mistral 7B. *arXiv preprint arXiv:2310.06825*, 2023.

Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, et al. MInference 1.0: Accelerating pre-filling for long-context LLMs via dynamic sparse attention. *arXiv preprint arXiv:2407.02490*, 2024.

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

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

Opher Lieber et al. Jamba: A hybrid Transformer-Mamba language model. *arXiv preprint arXiv:2403.19887*, 2024.

Piotr Nawrot, Jianing Li, Renjie Huang, Sebastian Ruder, Kelly Marchisio, and Edoardo Ponti. The Sparse Frontier: Sparse attention trade-offs in Transformer LLMs. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 38667–38701, 2026.

Qwen Team. Qwen3-Next-80B-A3B-Instruct model card. Official model card, 2025. `https://huggingface.co/Qwen/Qwen3-Next-80B-A3B-Instruct`. See also Qwen3 technical report, *arXiv preprint arXiv:2505.09388*, 2025.

Qwen Team. Qwen3.5-27B model card. Official model card, 2026. https://huggingface.co/Qwen/Qwen3.5-27B.

Xinghao Wang, Pengyu Wang, Xiaoran Liu, Fangxu Liu, Jason Chu, Kai Song, and Xipeng Qiu. Prism: Spectral-aware block-sparse attention. *arXiv preprint arXiv:2602.08426*, 2026.

Jingyang Yuan, Huazuo Gao, Damai Dai, et al. Native sparse attention: Hardware-aligned and natively trainable sparse attention. *arXiv preprint arXiv:2502.11089*, 2025.

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. *arXiv preprint arXiv:2402.13718*, 2024.

A Oracle and Indexer Details

A.1 Oracle Assumption and Execution

The oracle relies on a specific empirical assumption: for the evaluated support family, task-relevant evidence is sufficiently reflected in the head-averaged dense attention-mass distribution. The oracle tables test this assumption for the reported model and task slices, but they do not make it a theorem or rule out better per-head, per-KV-group, or task-optimized sparse supports.

Algorithm 1: Attention-mass top-k oracle for one patched layer.

1. Given current sparse-trajectory hidden states, form the layer's original GQA Q, K, V tensors.
2. Compute dense causal attention scores and softmax normalizers tile by tile.
3. Reconstruct dense attention probabilities tile by tile and average them over query heads to obtain $P_{\ell,t}(s)$.
4. Select top-k token indices from $P_{\ell,t}$ for per-query support, or from $P_{\ell,B}$ for block-shared support, as defined in Section 2.
5. Broadcast the selected token indices across KV heads, pad invalid causal slots with a sentinel, and recompute the original GQA attention only on the selected support.
6. Return the recomputed sparse attention output to the model, so later oracle decisions are made on the sparse trajectory.

A.2 Indexer Implementation and Training Details

The ReLU clipping and key-side normalization are implementation choices for stabilizing the scorer and making top-k selection depend on positively aligned keys. The oracle-guided claim does not require this exact scoring function; what matters is whether the distilled ranker recovers the teacher support under a serving-oriented cost.

The exact sampling distribution over layers and positions, the number of supervised rows, and the checkpoint-selection rule are not exposed as public reproducibility metadata.

The efficiency argument therefore does not come only from a lower projection dimension; it also comes from removing the auxiliary head dimension, sharing support where the runtime permits it, fusing scoring with top-k, and reading only selected values inside sparse attention. This is consistent with the oracle argument: the auxiliary module should predict support rankings, not reproduce full multi-head attention computation. This paper does not perform a controlled ablation over d_{idx}.

The serving path executes the sparse full-attention layer in five stages: project hidden states into the indexer space, score candidate keys for every query and aggregate normalized scores for shared support, select top-k indices, reconstruct K/V from the paged KV cache when the chunked path requires contiguous tensors, and run sparse attention that gathers the selected support inside the attention kernel.

Algorithm 2: Oracle-distilled indexer sparse prefill path.

1. Train only the auxiliary indexer against the oracle distribution $P_{\ell,t}$ using the KL objective in Section 2.3; keep the backbone frozen.
2. At prefill time, detach hidden states and project them with a single head-collapsed indexer projection to obtain $Q_{\text{idx}}, K_{\text{idx}} \in \mathbb{R}^{B\times S\times d_{\text{idx}}}$.
3. Apply key-side normalization, RoPE, and the query scale used by the indexer scorer.
4. Score candidates for every query with $Q_{\text{idx}}K_{\text{idx}}^{\top}$, apply ReLU, and normalize each row by softmax over valid causal keys to obtain $\widehat{P}_{\ell,t}$. Max-aggregate these normalized probabilities over each `query_share_size` query block.
5. Select top-k token indices from the aggregate, replicate the shared support across KV heads, and run sparse GQA prefill on the original Q, K, V tensors with each query's causal mask.
6. Keep decode dense in the reported quality runs.

B Experimental Setup and Evaluation Protocol

B.1 Experimental Setup

Gemma-4 (Gemma Team, 2026) model evaluations use the Hugging Face backend, while all other experiments use vLLM (Kwon et al., 2023).

The primary checkpoint is the public Hugging Face model `Qwen/Qwen3.5-27B` (Qwen Team, 2026). The run metadata records `revision=None`, so an immutable model revision was not pinned; the identifier resolves to the repository default downloaded for those runs. The loaded language backbone has 64 layers: 16 full softmax-GQA layers at indices 3, 7, ..., 63, interleaved with 48 Gated DeltaNet (GDN) layers. Each full-attention layer uses $H_q = 24$ query heads, $H_{kv} = 4$ KV heads, head dimension $d_h = 256$, and hidden size 5120. Our sparse modification is applied only to the 16 full-attention layers; GDN layers remain unchanged.

The reported Qwen3.5-27B TTFT frontier was measured in BF16 on one 80GB PCIe GPU with tensor parallelism disabled (TP = 1), on a four-GPU host. The Gemma-4 scope-validation runs instead use the Hugging Face path with pipeline parallelism across two GPUs; their accuracy is compared only within that matched backend.

Based on a preliminary analysis of response lengths, `max_new_tokens` was set to 170 for all needle-in-a-haystack tasks, 180 for vt, 240 for cwe, 50 for fwe, and 32 for the question-answering tasks. These RULER task-specific limits reduce premature truncation of the predictions while avoiding unnecessary generation.

B.2 Evaluation Protocol

The Qwen3.5-27B TTFT frontier uses a single 80GB GPU (TP = 1), BF16, eager execution, disabled prefix caching, one generated token, and independent GPU-event timing with synchronization. Each cell uses 3 warmups and 10 timed requests. The 32K–128K runs use the default vLLM GPU-memory utilization and `max_model_len` equal to the target context plus 16 tokens. The unreported 256K indexer stress runs use `gpu_memory_utilization`=0.93 and `max_model_len`=262144; the 256K TTFT attempt remains out of memory.

Table 6: Evaluation protocol. Preliminary random-initialized indexer TTFT rows provide runtime diagnostics only.

Experiment	Model(s)	Data / metric	Setting
27B Accuracy Frontier	Qwen3.5-27B	RULER accuracy; 50 samples per task group	32K / 64K / 128K: all five configurations; 256K: dense and both oracles only; top-k = 1024 at 32K and 2048 otherwise; block rows use `query_share_size` = 32
27B TTFT Frontier	Qwen3.5-27B	End-to-end mean, p50, observed maximum, standard deviation, speedup, and peak GPU memory	32K / 64K / 128K; dense versus per-query and block indexer; 4K chunk; $d_{\text{idx}} = 256$; top-k = 1024 / 2048; `query_share_size` = 1 / 32; paired dense cells
Support Granularity Ablation	Qwen3.5-27B	RULER accuracy and end-to-end TTFT	64K / 128K; `query_share_size` $\in \{1, 8, 16, 32, 64\}$; dense, per-query oracle, block oracle, per-query indexer, and block indexer
Task-wise failure analysis	Qwen3.5-27B	RULER task-group scores and selected prediction failures	64K / 128K; same configurations as 27B Accuracy Frontier
Indexer Gap Ablation	Qwen3.5-27B	oracle top-k overlap, AMR, relative AMR, validation KL	32K / 64K / 128K; group A: dense / per-query oracle / per-query indexer, group B: dense / block oracle / block indexer
Key-block Realization Ablation	Qwen3.5-27B	RULER accuracy and end-to-end TTFT	64K / 128K; `query_share_size` = 32; `key_block_size` (b_k) $\in \{64, 128, 256\}$
Non-retrieval Benchmark	Qwen3.5-27B	InfiniteBench (Zhang et al., 2024) and LongBench-v2 (Bai et al., 2024) accuracy	64K / 128K; dense, per-query oracle, block oracle, and block indexer; `query_share_size` $\in \{32, 64\}$ for block rows
Model Scaling	Qwen3.5-0.8B, 9B, 27B, and 122B-A10B	RULER accuracy and preliminary TTFT diagnostics	64K / 128K; dense versus block oracle for quality; TTFT uses random-initialized indexers for 0.8B and 9B; 122B-A10B TTFT is excluded because it requires tensor parallelism of 4
Scope Validation	Qwen3-32B; Gemma-4-26B-A4B-it	RULER accuracy; 50 samples per task group	64K / 128K; dense versus block oracle

C 27B ACCURACY FRONTIER

Tables 7 and 8 provide the per-task results; Table 9 reports the corresponding accuracy differences.

Table 7: Qwen3.5-27B RULER accuracy on needle-in-a-haystack tasks across dense, per-query, and selection-block-shared oracle and indexer configurations. [2]

Context	Configuration	ns1	ns2	ns3	nmk1
32K	Dense	100.0	100.0	100.0	100.0
32K	Per-Query Oracle	100.0	100.0	100.0	100.0
32K	Block Oracle	100.0	100.0	100.0	100.0
32K	Per-Query Indexer	100.0	100.0	100.0	100.0
32K	Block Indexer	100.0	100.0	100.0	100.0
64K	Dense	100.0	100.0	100.0	100.0
64K	Per-Query Oracle	100.0	100.0	100.0	100.0
64K	Block Oracle	100.0	100.0	100.0	100.0
64K	Per-Query Indexer	100.0	100.0	100.0	100.0
64K	Block Indexer	100.0	100.0	100.0	100.0
128K	Dense	100.0	100.0	100.0	100.0
128K	Per-Query Oracle	100.0	100.0	100.0	100.0
128K	Block Oracle	100.0	100.0	100.0	100.0
128K	Per-Query Indexer	100.0	100.0	100.0	100.0
128K	Block Indexer	100.0	100.0	100.0	100.0
256K	Dense	100.0	100.0	100.0	100.0
256K	Per-Query Oracle	100.0	100.0	100.0	100.0
256K	Block Oracle	100.0	100.0	100.0	100.0

Context	Configuration	nmk2	nmk3	nmval	nmq
32K	Dense	100.0	100.0	100.0	100.0
32K	Per-Query Oracle	100.0	100.0	100.0	100.0
32K	Block Oracle	100.0	100.0	100.0	100.0
32K	Per-Query Indexer	100.0	100.0	100.0	100.0
32K	Block Indexer	100.0	100.0	100.0	100.0
64K	Dense	100.0	100.0	100.0	100.0
64K	Per-Query Oracle	100.0	100.0	100.0	100.0
64K	Block Oracle	100.0	100.0	100.0	100.0
64K	Per-Query Indexer	100.0	100.0	100.0	100.0
64K	Block Indexer	100.0	96.0	100.0	100.0
128K	Dense	100.0	100.0	100.0	100.0
128K	Per-Query Oracle	100.0	100.0	100.0	100.0
128K	Block Oracle	100.0	100.0	100.0	100.0
128K	Per-Query Indexer	100.0	100.0	100.0	100.0
128K	Block Indexer	100.0	92.0	100.0	100.0
256K	Dense	100.0	100.0	100.0	100.0
256K	Per-Query Oracle	100.0	100.0	100.0	100.0
256K	Block Oracle	100.0	96.0	100.0	100.0

[2]The task names were abbreviated as follows in Table 7: ns1 for niah_single_1, ns2 for niah_single_2, ns3 for niah_single_3, nmk1 for niah_multikey_1, nmk2 for niah_multikey_2, nmk3 for niah_multikey_3, nmval for niah_multivalue, and nmq for niah_multiquery.

Table 8: Qwen3.5-27B RULER accuracy on multi-hop tracing, aggregation, and question-answering tasks across dense, per-query, and selection-block-shared oracle and indexer configurations.

Context	Configuration	vt	cwe	fwe	qa_1	qa_2
32K	Dense	100.0	100.0	100.0	94.0	74.0
32K	Per-Query Oracle	100.0	100.0	100.0	94.0	72.0
32K	Block Oracle	100.0	100.0	100.0	94.0	72.0
32K	Per-Query Indexer	100.0	100.0	100.0	94.0	72.0
32K	Block Indexer	100.0	100.0	100.0	92.0	74.0
64K	Dense	100.0	100.0	100.0	94.0	76.0
64K	Per-Query Oracle	100.0	100.0	99.33	92.0	74.0
64K	Block Oracle	100.0	100.0	100.0	94.0	74.0
64K	Per-Query Indexer	100.0	100.0	100.0	94.0	74.0
64K	Block Indexer	100.0	100.0	98.67	94.0	74.0
128K	Dense	100.0	100.0	100.0	82.0	72.0
128K	Per-Query Oracle	100.0	100.0	100.0	84.0	70.0
128K	Block Oracle	100.0	100.0	100.0	82.0	72.0
128K	Per-Query Indexer	100.0	100.0	100.0	84.0	72.0
128K	Block Indexer	100.0	100.0	100.0	84.0	70.0
256K	Dense	100.0	99.8	99.33	88.0	72.0
256K	Per-Query Oracle	100.0	99.2	100.0	86.0	68.0
256K	Block Oracle	99.6	98.0	100.0	86.0	68.0

Table 9: Qwen3.5-27B RULER accuracy gaps in percentage points, corresponding to Table 1.

	Oracle vs dense		Indexer vs oracle		Indexer vs dense	
Context	Per-query	Block	Per-query	Block	Per-query	Block
32K	-0.16	-0.16	+0.00	+0.00	-0.16	-0.16
64K	-0.36	-0.15	+0.21	-0.41	-0.15	-0.56
128K	+0.00	+0.00	+0.16	-0.61	+0.16	-0.61
256K	-0.46	-0.89	–	–	–	–

D Additional TTFT Statistics and Memory

Table 10 gives the full paired timing statistics underlying Table 2, including medians and observed maxima. The maximum is a descriptive tail proxy for these ten-request samples, not a p99 estimate. The following subsections report reductions, variability, and observed system-level GPU memory from a GPU memory monitoring utility.

D.1 Mean TTFT

Table 10: End-to-end TTFT of Qwen3.5-27B per-query and selection-block-shared sparse indexer relative to dense baseline. All latency values are in milliseconds.

Context	Pipeline	Dense mean	Sparse mean	Dense p50	Sparse p50
32K	pq_pk	9958.2	9778.1	9998.2	9782.4
32K	bq_pk	9719.1	8905.8	9732.7	8919.4
64K	pq_pk	22941.2	21624.3	22977.2	21618.4
64K	bq_pk	22968.0	18978.2	22961.6	19005.2
128K	pq_pk	58095.5	44640.4	58124.6	44649.9
128K	bq_pk	58217.3	39807.5	58242.8	39831.4

Context	Pipeline	Dense max	Sparse max	Dense std	Sparse std	Speedup
32K	pq_pk	10070.8	9856.2	93.7	59.9	1.02x
32K	bq_pk	9757.2	8940.4	30.9	27.9	1.09x
64K	pq_pk	23014.1	21759.7	79.4	91.0	1.06x
64K	bq_pk	23054.7	19078.3	59.3	62.9	1.21x
128K	pq_pk	58145.0	44695.0	56.0	41.5	1.30x
128K	bq_pk	58699.7	39927.0	223.3	78.1	1.46x

Table 11: Mean end-to-end TTFT reduction of Qwen3.5-27B per-query and selection-block-shared sparse indexer relative to dense baseline.

Context	Pipeline	Absolute reduction	Relative reduction	Speedup
32K	pq_pk	180.1 ms	1.81%	1.02×
32K	bq_pk	813.3 ms	8.37%	1.09×
64K	pq_pk	1316.9 ms	5.74%	1.06×
64K	bq_pk	3989.8 ms	17.37%	1.21×
128K	pq_pk	13455.1 ms	23.16%	1.30×
128K	bq_pk	18409.8 ms	31.62%	1.46×

D.2 P50 TTFT

The median TTFT reductions (Table 12) follow the same pattern as the mean reductions (Table 11), suggesting that the speedups reflect typical runs rather than outliers in these ten-request samples.

Table 12: p50 end-to-end TTFT reduction of Qwen3.5-27B per-query and selection-block-shared sparse indexer relative to dense baseline.

Context	Pipeline	p50 reduction	Relative reduction
32K	pq_pk	215.8 ms	2.16%
32K	bq_pk	813.3 ms	8.36%
64K	pq_pk	1358.8 ms	5.91%
64K	bq_pk	3956.4 ms	17.23%
128K	pq_pk	13474.7 ms	23.18%
128K	bq_pk	18411.4 ms	31.61%

D.3 OBSERVED MAXIMUM TTFT

With ten timed requests per configuration, the percentile calculation used by the benchmark selects the largest observation. We therefore report this value as the observed maximum rather than as an

estimated p99. Table 13 reports its absolute and relative reduction. In all available cells, the observed maximum TTFT is lower for sparse execution than for its paired dense run. A statistically meaningful p99 estimate requires substantially more requests and confidence-interval reporting.

Table 13: Observed maximum TTFT reduction of Qwen3.5-27B per-query and selection-block-shared sparse indexer relative to dense baseline over ten timed requests per configuration.

Context	Pipeline	Maximum reduction	Relative reduction
32K	pq_pk	214.6 ms	2.13%
32K	bq_pk	816.8 ms	8.37%
64K	pq_pk	1254.4 ms	5.45%
64K	bq_pk	3976.4 ms	17.25%
128K	pq_pk	13450.0 ms	23.13%
128K	bq_pk	18772.7 ms	31.98%

D.4 TTFT STANDARD DEVIATION

Across the ten requests collected per case, the observed standard deviation is below 1% of the corresponding mean. Table 14 reports all standard deviations and their ratios to the mean. Sparse execution has a lower standard deviation at 32K and 128K and a slight increase at 64K, reaching at most 0.42% of mean TTFT. The standard-deviation-to-mean ratio is usually smaller at longer context lengths.

Table 14: Observed TTFT variability of Qwen3.5-27B per-query and selection-block-shared sparse indexer relative to dense baseline over ten timed requests per configuration.

Context	Pipeline	Dense		Sparse	
		std (ms)	std / mean	std (ms)	std / mean
32K	pq_pk	93.7	0.941%	59.9	0.613%
32K	bq_pk	30.9	0.318%	27.9	0.313%
64K	pq_pk	79.4	0.346%	91.0	0.421%
64K	bq_pk	59.3	0.258%	62.9	0.331%
128K	pq_pk	56.0	0.096%	41.5	0.093%
128K	bq_pk	223.3	0.384%	78.1	0.196%

D.5 OBSERVED SYSTEM-LEVEL GPU MEMORY

Table 15 reports the observed peak GPU memory. The dense peak was 74959 MiB in all runs, while the sparse peak varied. The sparse pq_pk pipeline used an additional 3406 MiB at 64K. At 128K, the overhead was higher for both pipelines: +6046 MiB for pq_pk and +6040 MiB for bq_pk.

Table 15: Observed peak GPU memory of Qwen3.5-27B per-query and selection-block-shared sparse indexer relative to dense baseline.

Context	Pipeline	Dense Peak (MiB)	Sparse Peak (MiB)	Additional (MiB)	Overhead (%)
32K	pq_pk	74959	75453	494	0.66%
32K	bq_pk	74959	75455	496	0.66%
64K	pq_pk	74959	78365	3406	4.54%
64K	bq_pk	74959	75455	496	0.66%
128K	pq_pk	74959	81005	6046	8.07%
128K	bq_pk	74959	80999	6040	8.06%

E TASK-WISE FAILURES

The task-wise failures underlying Section 3.1 cover dense, per-query oracle, block oracle, per-query indexer, and block indexer configurations at 64K and 128K, with additional dense-baseline examples at 256K. The 128K examples illustrate long-context failures; the 256K examples expose further evaluator sensitivities. The block oracle uses `query_share_size` = 32.

The examples include retrieval failures with suggestions of similar items or repeated searches, answers containing a different sentence fragment than required, incomplete specifications, and evaluator sensitivity to formatting or punctuation.

E.1 64K

Sometimes, the prediction contains part of the answer but omits a required specification, which causes it to be considered inaccurate. For example, at 64K, the block oracle and block indexer predict "Dyrrachium was located on the southern shores of Albania.", while the correct answer is "Dyrrachium was located on the southern shores of Albania, specifically on the coast of the Adriatic Sea.".

In a 64K qa_2 sample, the question is "Kaiser Ventures corporation was founded by an American industrialist who became known as the father of modern American shipbuilding?", with the expected answer "Henry J. Kaiser". The question asks for the person's name, but due to the ambiguous formulation, the block oracle and block indexer both interpret it as a yes / no question, and predict "Yes".

E.1.1 BLOCK INDEXER

The block indexer loses accuracy on the retrieval-heavy niah_multikey_3 task (Table 7). In the following example, it fails to find the requested UUID, suggests a similar UUID, and again fails to find it:

> "Based on the text provided, there is **no mention** of the UUID
>
> `49ff024b-8359-48ea-83bc-0dc1a43928d7`.
>
> It appears you may have misremembered the UUID or combined parts of different lines. The text does contain a very similar UUID:
>
> **Input:**
>
> `49ff024b-8359-48ea-83bc-0dc1a43928d7` (Not found)
>
> **Similar Input in text:**
>
> `49ff024b-8359-48ea-83bc-0dc1a43928d7` is not present, but "4"."

On the fwe task, the block indexer scores 98.67 due to two predictions which are only partially correct. In one of these cases, the model predicts "uielop, qemwis, yexnot" instead of "uiegop, qemwis, yexnot". That is, one of the three words differs from the expected answer by a single letter.

E.2 128K

E.2.1 DENSE BASELINE

The aggregate quality decrease from 64K to 128K is concentrated in the qa tasks. The examples in Table 16 illustrate two ways in which a long context decreases the quality: the model either identifies the main idea but fails to predict the full answer, or is unable to retrieve the answer altogether.

On a qa_2 sample, the answer "1969 to 1974" is considered incorrect (correct answer: "1969 until 1974") because of the preposition difference. Hence, sometimes small accuracy drops can also come from minor variations that the verifier does not recognize as semantically equivalent.

Table 16: Selected Qwen3.5-27B dense attention qa_1 task failures at 128K, compared with the accurate predictions on the corresponding 64K samples.

ID	Prediction 128K (incorrect)	Prediction 64K (correct)
6	The Duchy of Normandy was ruled by the Normans, a people descended from Norse raiders who, under their leader Rollo, agreed to swear fe	Rollo, Richard I of Normandy, and William II of Normandy.
23	The Normans teamed up with the Turks in Anatolia.	Turkish forces
31	The Normans attacked Dyrrachium in 1107 and again in 1081.	The Normans attacked Dyrrachium in 1081, 1185, and 1107.
33	Dyrrachium was located on the southern shores of Albania.	Dyrrachium was located on the southern shores of Albania, specifically on the coast of the Adriatic Sea.
32	Naval Submarine Base Kings Bay	Dyrrachium
49	Malcolm III of Scotland	Edgar Atheling

E.2.2 BLOCK ORACLE

The specificity is again missing in some qa_1 predictions: the block oracle responded only "Norseman", whereas the dense baseline predicted "The word Norman comes from "Norseman" or "Viking", derived from Old Norse Norðmaðr or Old Low Franconian". However, there are also cases in which the block oracle finds the answer when the dense baseline does not. For example, on a qa_2 sample, the block oracle accurately identifies "Mumbai", whereas the dense model had erroneously predicted "Kochi".

E.2.3 BLOCK INDEXER

As at 64K, accuracy drops on niah_multikey_3. At 128K, the model does not find the required UUID but claims that there exists a very similar one, then loops back and re-examines the text:

> "Based on the text provided, there is no mention of the UUID
> `5710fa0a-f727-4fe7-af28-f0e9d715d7f0`.
> However, there is a very similar UUID listed in the text:
> `5710fa0a-f727-4fe7-af28-f0e9d715d7f0` is **not** in the list.
> The closest match in the text is:
> `5710fa0a-f727-4fe7-af28-f0e9d715d7f0` → **Not found.**
> Let's re-examine the text for `5710fa0a...`:"

E.3 256K

E.3.1 DENSE BASELINE

The slight cwe and fwe accuracy decreases for the dense 256K baseline (Table 8) include evaluator-sensitive cases in which the only discrepancy is a space. For cwe, sample 41, 9 of the 10 words are

correct, except for "inter talk" which is supposed to be "intertalk". Similarly, for fwe, sample 4, 2 of the 3 words are correct, except for "vs aixk" which is supposed to be "vsaixk".

The qa_1 accuracy increases by 6% compared to dense 128K. However, for question ID 2, the answer "Denmark, Iceland and Norway" is considered correct for dense 256K, but not for dense 32K, 64K, or 128K ("Denmark, Iceland, and Norway"), because of only a comma.

F Support Granularity

Table 17: Qwen3.5-27B niah_multikey_3 sparse indexer accuracy across query-support sharing granularities. Each cell uses 50 binary-scored samples; one correct/incorrect flip changes the task score by 2 points.

Context	qss=1	qss=8	qss=16	qss=32	qss=64
64K	100.0	100.0	98.0	96.0	92.0
128K	100.0	96.0	98.0	92.0	84.0

G Indexer Fidelity Details

G.1 Metrics and Measurement Protocol

The four metrics are oracle top-k overlap, AMR, relative AMR, and validation KL. Let $\mathcal{Q}_\ell$ be the measured active query rows at layer ℓ, let $\mathcal{S}^\star_{\ell,t}$ and $\widehat{\mathcal{S}}_{\ell,t}$ denote the causally valid oracle and indexer supports after expansion to token IDs, and let $\widehat{P}_{\ell,t} = \mathrm{softmax}(z_{\ell,t})$. The per-layer metrics are

$$\mathrm{Overlap}_\ell = \frac{1}{|\mathcal{Q}_\ell|} \sum_{t \in \mathcal{Q}_\ell} \frac{|\mathcal{S}^\star_{\ell,t} \cap \widehat{\mathcal{S}}_{\ell,t}|}{|\mathcal{S}^\star_{\ell,t}|},$$

$$\mathrm{AMR}^\star_\ell = \frac{1}{|\mathcal{Q}_\ell|} \sum_{t \in \mathcal{Q}_\ell} \sum_{s \in \mathcal{S}^\star_{\ell,t}} P_{\ell,t}(s), \qquad \widehat{\mathrm{AMR}}_\ell = \frac{1}{|\mathcal{Q}_\ell|} \sum_{t \in \mathcal{Q}_\ell} \sum_{s \in \widehat{\mathcal{S}}_{\ell,t}} P_{\ell,t}(s),$$

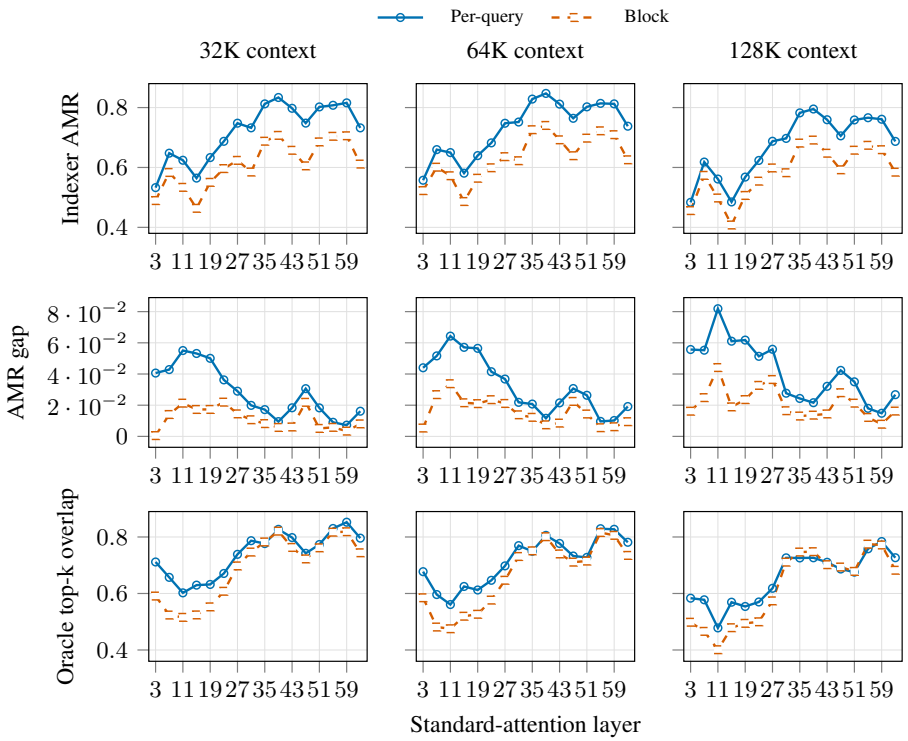
$$\mathrm{RelativeAMR}_\ell = \frac{\widehat{\mathrm{AMR}}_\ell}{\mathrm{AMR}^\star_\ell}, \qquad \mathrm{KL}_\ell = \frac{1}{|\mathcal{Q}_\ell|} \sum_{t \in \mathcal{Q}_\ell} \sum_{s} P_{\ell,t}(s) \log \frac{P_{\ell,t}(s)}{\widehat{P}_{\ell,t}(s)}.$$

Validation KL is computed before top-k truncation and is therefore identical across the per-query and block settings when the scorer and measured rows are unchanged. Active sparse queries exclude early causal rows whose valid-key count does not exceed the effective support budget. At each layer, raw sufficient statistics are summed over measured tail queries and all included prompts before normalization, producing a query-level micro-average; Table 4 then reports the arithmetic mean over the 16 measured layers. Relative AMR is the ratio of aggregate indexer and oracle AMR within each layer, not the mean of per-query ratios.

These metrics are computed on the 16 standard softmax-attention layers of Qwen3.5-27B, at layer indices 3, 7, ..., 63. The 48 GDN layers are excluded from this analysis. For each context length, we use a measurement position one 4K prefill chunk below the full context length. This helps avoid losing samples whose rendered prompts do not reach exactly 32768 / 65536 / 131072 tokens, corresponding to the 32K / 64K / 128K context lengths. We use a budget of top-k = 1024 at 32K and top-k = 2048 at 64K / 128K. At each measurement position, we measure a contiguous tail window of 256 queries, preferring tail queries because they feature the longest available histories.

Out of the 650 prompts evaluated in each condition, 638 reached the 28K token checkpoint at 32K context length, 637 reached the 60K token checkpoint at 64K context length, and 601 reached the 124K token checkpoint at 128K context length. The missing samples come from fwe prompts (12/50 samples at 32K, 13/50 samples at 64K, and 38/50 samples at 128K), 1 qa_1 sample and 10/50 qa_2 samples at 128K that remained below the corresponding checkpoint, while all remaining samples are included in the analysis. The reported aggregate measurements refer only to the prompts that reached the checkpoint.

Figure 3: Per-layer evolution of indexer AMR, AMR gap, and oracle top-k overlap across the per-query and block pipelines at 32K, 64K, and 128K. All cases use a checkpoint one 4K chunk below the context length: 28K, 60K, and 124K respectively.



G.2 Layer-wise Fidelity

Figure 3 shows the indexer AMR, AMR gap, and oracle top-k overlap per-layer evolution at 32K, 64K, and 128K context lengths, across the 16 standard softmax-attention layers of Qwen3.5-27B (layer indices 3, 7, ..., 63). The layer-wise results suggest that indexer fidelity is layer-dependent rather than continuously improving with depth. All three plotted metrics follow a structured trajectory, with similar peaks and troughs across 32K, 64K, and 128K contexts.

The per-query and block pipelines differ in top-k overlap up to around layer 35, after which their results are close. Middle-to-late layers have the strongest recorded fidelity around layers 35–43 and 55–59. Layers 11–19 and the dips around layers 47 and 63 show lower retained mass and larger AMR gaps on the measured queries; these observations do not establish which layers cause task-level failures.

H Non-retrieval Benchmark

The InfiniteBench and LongBench-v2 evaluation covers tasks beyond RULER, including code debug, code run, and code repository understanding. The comparison reports Qwen3.5-27B accuracy and gaps versus dense at 64K and 128K for dense, per-query oracle, block oracle, and block indexer configurations.

Both reported subsets were selected using exact Qwen-token counts with seed 20270617 and contain 150 examples. The `nonretrieval_64k_relaxed` name denotes the wider 50,000–85,000-token eligibility interval: task-stratified sampling selects 100 InfiniteBench and 50 LongBench-v2 examples. The `nonretrieval_128k_pm10` name denotes a $\pm 10\%$ interval around 128K, 117,964–144,179 tokens: all 44 eligible LongBench-v2 examples are retained and task-stratified InfiniteBench sampling fills the remaining 106 positions. Because the eligible tasks and selected

examples differ between these two subsets, all method comparisons below are made within a context-specific subset; we do not interpret cross-subset score differences as context-length effects.

H.1 CONTEXT LENGTH 64K: `nonretrieval_64k_relaxed` BENCHMARK

One additional incorrect prediction in the 50-example LongBench-v2 subset changes its accuracy by 2 points.

The comparison tests whether the RULER operating point, `query_share_size` = 32 (Section 4), transfers to these benchmarks.

Table 18: Qwen3.5-27B accuracy on the `nonretrieval_64k_relaxed` benchmark (IB: InfiniteBench; LB-v2: LongBench-v2; gaps are versus dense) across dense, oracle and indexer configurations.

		IB, N = 100		LB-v2, N = 50		Overall, N = 150	
Configuration	qss	Acc.	Gap	Acc.	Gap	Acc.	Gap
Dense	–	52.11	–	52.00	–	52.07	–
Per-query oracle	1	50.27	-1.84	48.00	-4.00	49.51	-2.56
Block oracle	32	51.47	-0.64	48.00	-4.00	50.31	-1.76
Block oracle	64	53.79	+1.68	52.00	0.00	53.19	+1.12
Block indexer	32	50.29	-1.82	48.00	-4.00	49.53	-2.54
Block indexer	64	51.49	-0.62	50.00	-2.00	50.99	-1.08

Sharing across 64 queries, evaluated after the 32-query results, yields higher observed aggregate scores for both oracle and indexer in this sample (Table 18). This descriptive ordering is not an independently validated advantage. The oracle scores above dense without establishing an improvement, while the indexer remains slightly below; its LongBench-v2 loss corresponds to one additional incorrect prediction.

Task performance varies substantially: code debugging and book-choice questions are strong, while code execution, book QA, and summarization remain difficult. Single-document QA outperforms multi-document QA at similar sample counts (Tables 19 and 20).

Table 19: Qwen3.5-27B per-task accuracy on the `nonretrieval_64k_relaxed` benchmark InfiniteBench samples. PQ: per-query; BO: block oracle; BI: block indexer; N: sample count.

Task	Dense	PQ oracle	BO qss 64	BO qss 32	BI qss 64	BI qss 32	N
`code_debug`	96.0	96.0	100.0	100.0	100.0	96.0	25
`code_run`	32.0	24.0	32.0	24.0	24.0	24.0	25
`longbook_choice_eng`	100.0	100.0	100.0	100.0	100.0	100.0	8
`longbook_qa_eng`	23.12	24.29	25.23	24.56	24.15	23.64	25
`longbook_sum_eng`	37.24	36.46	38.14	37.24	37.94	37.54	17

Table 20: Qwen3.5-27B per-task accuracy on the `nonretrieval_64k_relaxed` benchmark LongBench-v2 samples. PQ: per-query; BO: block oracle; BI: block indexer; N: sample count.

Task	Dense	PQ oracle	BO qss 64	BO qss 32	BI qss 64	BI qss 32	N
Code Repository Understanding	60.0	60.0	60.0	60.0	60.0	60.0	5
Long In-context Learning	66.67	66.67	66.67	66.67	66.67	66.67	3
Long Structured Data Understanding	50.0	50.0	50.0	50.0	50.0	50.0	2
Long-dialogue History Understanding	0.0	0.0	0.0	0.0	0.0	0.0	1
Multi-Document QA	35.0	30.0	35.0	30.0	30.0	30.0	20
Single-Document QA	68.42	63.16	68.42	63.16	68.42	63.16	19

H.2 Context length 128K: `nonretrieval_128k_pm10` benchmark

One additional incorrect prediction in the 44-example LongBench-v2 subset introduces a -2.27 accuracy gap.

Table 21: Qwen3.5-27B accuracy on the `nonretrieval_128k_pm10` benchmark (IB: InfiniteBench; LB-v2: LongBench-v2; gaps are versus dense) across dense, oracle and indexer configurations.

		IB, N = 106		LB-v2, N = 44		Overall, N = 150	
Configuration	qss	Acc.	Gap	Acc.	Gap	Acc.	Gap
Dense	–	66.02	–	47.73	–	60.65	–
Per-query oracle	1	65.22	-0.80	43.18	-4.55	58.76	-1.89
Block oracle	32	66.36	+0.34	36.36	-11.37	57.56	-3.09
Block oracle	64	66.50	+0.48	43.18	-4.55	59.66	-0.99
Block indexer	32	66.63	+0.61	40.91	-6.82	59.09	-1.56
Block indexer	64	67.48	+1.46	45.45	-2.28	61.02	+0.37

At 128K, sharing across 64 queries again gives higher observed aggregate scores than sharing across 32, whose losses are especially pronounced on LongBench-v2 (Table 21). The qss = 32 block oracle's 11.37-point LongBench-v2 loss corresponds to five fewer correct answers among 44 examples. These descriptive results favor 64-query sharing over the RULER operating point of 32 on the evaluated subsets, not as a general preference for coarser sharing.

The per-task 128K results are reported separately for InfiniteBench in Table 22 and LongBench-v2 in Table 23.

Table 22: Qwen3.5-27B per-task accuracy on the `nonretrieval_128k_pm10` benchmark InfiniteBench samples. PQ: per-query; BO: block oracle; BI: block indexer; N: sample count.

Task	Dense	PQ oracle	BO qss 64	BO qss 32	BI qss 64	BI qss 32	N
`code_debug`	62.96	62.96	70.37	66.67	66.67	62.96	27
`longbook_choice_eng`	100.0	100.0	100.0	100.0	100.0	100.0	27
`longbook_qa_eng`	45.81	49.24	49.67	49.0	50.27	50.0	27
`longbook_sum_eng`	35.05	36.52	38.0	38.23	37.24	38.31	16
`longdialogue_qa_eng`	88.89	66.67	55.56	66.67	77.78	77.78	9

Table 23: Qwen3.5-27B per-task accuracy on the `nonretrieval_128k_pm10` benchmark LongBench-v2 samples. PQ: per-query; BO: block oracle; BI: block indexer; N: sample count.

Task	Dense	PQ oracle	BO qss 64	BO qss 32	BI qss 64	BI qss 32	N
Long In-context Learning	70.0	70.0	80.0	60.0	70.0	70.0	10
Long Structured Data Understanding	50.0	0.0	0.0	0.0	0.0	0.0	2
Long-dialogue History Understanding	40.0	33.33	26.67	26.67	33.33	33.33	15
Multi-Document QA	60.0	60.0	60.0	60.0	60.0	60.0	5
Single-Document QA	33.33	33.33	33.33	25.0	41.67	25.0	12

I Model Scaling

The model-scaling evaluation compares Qwen3.5-0.8B, Qwen3.5-9B, Qwen3.5-27B, and Qwen3.5-122B-A10B, at context lengths 64K and 128K. For the accuracy comparison, we use the RULER benchmark (50 samples per task group), and compare the dense baseline with the block oracle. The TTFT runs use a random-weights indexer for Qwen3.5-0.8B, Qwen3.5-9B, and Qwen3.5-122B-A10B. The Qwen3.5-122B-A10B TTFT is not directly comparable to the other models, as this large model needs to be run with tensor-parallel-size 4 as opposed to 1 for the other models.

Block-indexer accuracy comparisons across model scales remain future work, pending the corresponding distilled checkpoints.

I.1 Accuracy

Table 24: RULER accuracy of dense and selection-block-shared attention-mass oracle across Qwen3.5 model scales: 0.8B, 9B, 27B, 122B-A10B.

Context	Model	Dense	Block Oracle	Gap vs dense
64K	Qwen3.5-0.8B	85.96	82.36	-3.60
64K	Qwen3.5-9B	96.71	96.34	-0.37
64K	Qwen3.5-27B	97.69	97.54	-0.15
64K	Qwen3.5-122B-A10B	97.18	97.10	-0.08
128K	Qwen3.5-0.8B	83.53	75.43	-8.10
128K	Qwen3.5-9B	96.09	94.70	-1.39
128K	Qwen3.5-27B	96.46	96.46	+0.00
128K	Qwen3.5-122B-A10B	97.03	96.74	-0.29

Qwen3.5-0.8B has lower dense accuracy and substantial oracle losses, making it less competitive in this RULER setting (Table 24). Its strong needle retrieval does not extend to tracing, aggregation, and QA (Table 25).

Qwen3.5-9B preserves quality better than 0.8B and is comparable to 27B at 64K, but exceeds the one-point oracle-gap threshold at 128K. Thus, a single sparse operating point does not transfer uniformly across model sizes.

Qwen3.5-122B-A10B scores slightly below 27B at 64K but achieves the highest dense and oracle accuracy at 128K, at the cost of four-GPU execution.

Table 25: Per-task RULER accuracy of dense Qwen3.5-0.8B at 64K and 128K.

Context	ns1	ns2	ns3	nmk1	nmk2	nmk3	nmval
64K	100.0	100.0	100.0	100.0	100.0	100.0	99.0
128K	100.0	100.0	100.0	100.0	98.0	98.0	97.5

Context	nmquery	vt	cwe	fwe	qa_1	qa_2
64K	100.0	83.6	65.6	73.33	50.0	46.0
128K	100.0	91.6	44.8	86.0	34.0	36.0

I.2 TTFT

Table 26 reports preliminary runtime diagnostics for the 0.8B, 9B, and 27B models. The 122B-A10B model is excluded because its tensor-parallel setting differs from the other models.

The reported TTFT increases with model size at both 64K and 128K, and speedups are higher at 128K.

The 0.8B model shows no speedup at 64K but benefits at 128K, consistent with overheads limiting gains at small scales; profiling is needed to confirm this explanation. Similar 9B and 27B speedups suggest potential for a less compute-heavy alternative, but comparable quality and TTFT require verification with distilled indexer weights.

Table 26: TTFT stress tests with untrained indexer weights for Qwen3.5-0.8B and Qwen3.5-9B, distilled indexer checkpoint for Qwen3.5-27B.

Context	Model	Speedup
64K	Qwen3.5-0.8B	0.99x
64K	Qwen3.5-9B	1.19x
64K	Qwen3.5-27B	1.21x
128K	Qwen3.5-0.8B	1.20x
128K	Qwen3.5-9B	1.43x
128K	Qwen3.5-27B	1.46x

J SCOPE VALIDATION

The cross-family evaluation compares the block oracle with dense attention on Qwen3-32B and Gemma-4-26B-A4B-it at 64K / 128K, extending the within-family comparison in Appendix I. Block-indexer evaluation with model-specific checkpoints and evaluation of MiniMax-M2.7 under the same settings remain future work.

J.1 MODEL CHOICE

Gemma-4-26B-A4B-it was selected because it followed the intended answer-generation format on the evaluated samples. The base checkpoint, Gemma-4-26B-A4B, sometimes continued the prompt instead of answering the question, confusing its role.

Table 27: RULER accuracy of dense and selection-block-shared attention-mass oracle across evaluated model families: Qwen3 and Gemma-4.

Context	Model	Dense	Block Oracle	Gap vs dense
64K	Qwen3-32B	88.48	88.46	-0.02
128K	Qwen3-32B	83.38	72.06	-11.32
64K	Gemma-4-26B-A4B-it	95.49	94.76	-0.73
128K	Gemma-4-26B-A4B-it	90.81	90.22	-0.59

J.2 QWEN3-32B

Initial testing showed a very low accuracy of the dense baseline at 128K. The reason was that Qwen3-32B is trained with a native context of 32K, which means that the YaRN (Yet another RoPE extensioN method) factor has to be increased to scale for longer inputs. Using a YaRN factor of 2 for 64K and 4 for 128K restores the accuracy.

The Qwen3-32B block oracle stays close to dense at 64K but loses 11.32 points at 128K (Table 27). Losses are concentrated in multikey retrieval and QA (Table 28), showing that the Qwen3.5-27B operating point does not transfer reliably to this model and context-extension setting.

Architecture and context extension change together in this comparison. Qwen3-32B has 64 full-attention layers,[3] all sparsified here, whereas Qwen3.5-27B sparsifies 16 full-attention layers and leaves its GDN layers unchanged. The Qwen3-32B 128K run also uses YaRN factor 4. These results do not isolate effects of context extension, architecture, or the number of sparsified layers; attributing the loss to any one factor requires controlled follow-up.

Table 28: Qwen3-32B per-task RULER accuracy of dense baseline versus block oracle with `query_share_size` = 32.

Context	Configuration	ns1	ns2	ns3	nmk1	nmk2	nmk3
64K	Dense	100.0	96.0	98.0	98.0	96.0	94.0
64K	Block Oracle	100.0	98.0	100.0	98.0	92.0	88.0
128K	Dense	100.0	96.0	100.0	98.0	86.0	72.0
128K	Block Oracle	100.0	96.0	100.0	94.0	42.0	14.0

Context	Configuration	nmval	nmquery	vt	cwe	fwe	qa_1	qa_2
64K	Dense	99.0	100.0	78.8	76.4	90.0	72.0	52.0
64K	Block Oracle	99.5	98.5	90.4	68.2	91.33	72.0	54.0
128K	Dense	96.0	96.0	88.8	47.8	85.33	70.0	48.0
128K	Block Oracle	91.5	90.5	90.0	45.4	81.33	56.0	36.0

[3] `https://huggingface.co/Qwen/Qwen3-32B/blob/main/config.json`

J.3 GEMMA-4-26B-A4B-IT

The Gemma-4 block oracle remains within one point of dense at both context lengths, supporting transfer of the oracle operating point in this setting (Table 27). This does not establish Gemma indexer quality or TTFT.

Task-level losses are concentrated in qa_1 at 64K and in aggregation, multivalue retrieval, and tracing at 128K, partly offset by higher QA scores at 128K. Dense accuracy also declines at the longer context, mainly on multikey and multivalue retrieval and QA (Table 29).

Table 29: Gemma-4-26B-A4B-it per-task RULER accuracy of dense baseline versus block oracle with `query_share_size` = 32.

Context	Configuration	ns1	ns2	ns3	nmk1	nmk2	nmk3
64K	Dense	100.0	100.0	100.0	100.0	100.0	96.0
64K	Block Oracle	100.0	100.0	100.0	100.0	100.0	98.0
128K	Dense	100.0	100.0	100.0	100.0	100.0	88.0
128K	Block Oracle	100.0	100.0	100.0	100.0	100.0	90.0

Context	Configuration	nmval	nmquery	vt	cwe	fwe	qa_1	qa_2
64K	Dense	98.0	100.0	100.0	100.0	99.33	82.0	66.0
64K	Block Oracle	98.0	100.0	98.0	98.6	99.33	72.0	68.0
128K	Dense	83.5	100.0	98.8	97.6	98.67	58.0	56.0
128K	Block Oracle	78.5	100.0	92.4	82.6	99.33	68.0	62.0

The contrasting Gemma-4 and Qwen3-32B outcomes motivate family-specific evaluation. Gemma-4 combines local and global attention (Gemma Team, 2026); its result does not establish transfer to an all-global-attention backbone.