PARSEE: PAGING-AWARE KV CACHE COMPRESSION WITH SHARED SEMANTIC SUPPORT

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-context language-model serving requires large key–value (KV) caches, consuming GPU memory and limiting request concurrency. Although token eviction reduces logical cache size, heterogeneous per-head demands and page granularity can prevent these reductions from translating into reusable physical capacity. To address this challenge, we propose PARSEE, a paging-aware KV cache compression framework with shared semantic support. PARSEE is based on the observation that unequal retained counts among members sharing a page group inflate storage costs, and that shared semantic scores can coordinate these counts while preserving independent token identities. It selects shared semantic and head-specific question evidence through disjoint quotas, then allocates indivisible pages by marginal utility under a physical budget. Furthermore, a paged execution backend packs selected entries, transfers page ownership, and reclaims dense blocks, enabling batched attention over request–head groups with different retained lengths. Extensive experiments on five representative models from the Qwen and Llama families, covering dense, grouped-query, hybrid, and mixture-of-experts architectures, show that PARSEE retains near-lossless SCBench performance on Qwen3.8-27B while compressing 90% of prefill-context KV. Its Qwen3-8B backend achieves up to 5.28× the throughput of uncompressed vLLM inference.

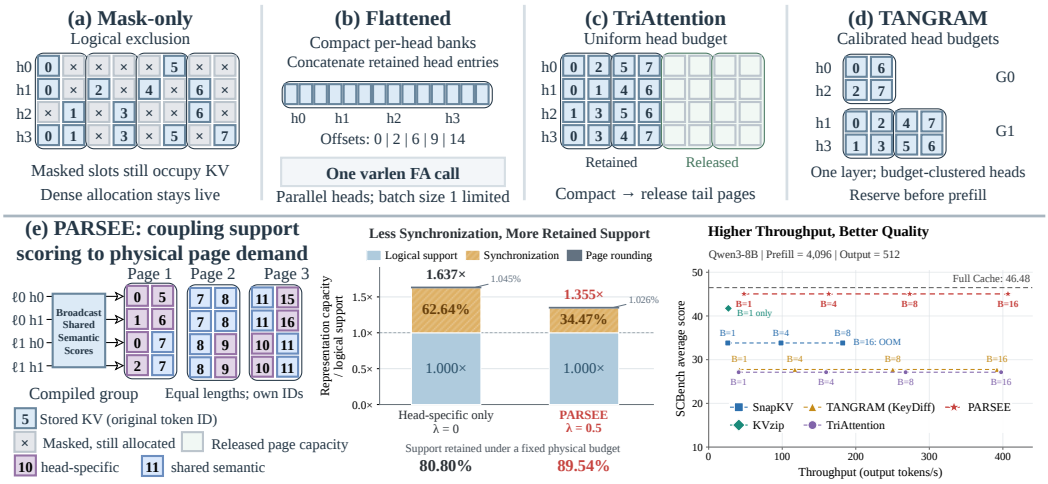


Figure 1: Illustrative comparison of KV cache compression methods across different physical memory layouts. PARSEE coordinates demand with shared semantic scores while retaining independent member IDs. **Middle**: PARSEE structural results at a 90% compression ratio. **Right**: Accuracy vs. throughput for Qwen3-8B evaluated on SCBench at batch sizes 1/4/8/16.

1 INTRODUCTION

Long-context language models enable increasingly demanding applications, but their key–value (KV) caches grow with context length, increasing memory consumption and decoding traffic and ultimately

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.

limiting serving concurrency. Although existing systems optimizations like paged allocation and fused attention improve how KV states are stored and accessed (Kwon et al., 2023; Dao et al., 2022; Dao, 2023), they do not reduce the amount of context that must remain resident. KV cache compression addresses this pressure by retaining only a subset of context entries, with prior methods using attention, adaptive head budgets, or reconstruction signals to identify useful states (Zhang et al., 2023; Li et al., 2024b; Feng et al., 2025; Fu et al., 2025; Kim et al., 2025; 2026b). However, these methods are primarily formulated around a *logical* token budget. In paged serving, logical sparsity does not directly determine the physical capacity needed to realize a selection, nor whether the resulting free space can actually be reclaimed by the serving allocator.

This gap becomes concrete when heterogeneous selections meet grouped paged storage. Members sharing an allocated length must provision for their largest retained count and round capacity to whole pages, even when their token identities differ. Consequently, two selections with the same number of entries can require different amounts of physical storage. TriAttention compacts selected KV with a reported uniform head budget, while TANGRAM groups heads using calibrated budgets to reduce mismatch (Mao et al., 2026b; Kim et al., 2026a; Mao et al., 2026a). These systems motivate a complementary question: how can token selection itself coordinate input-dependent demand with the page layout that will execute it?

In this paper, we identify **member-demand disparity**, the difference in retained counts within a page group, as the link between token selection and physical cost (§ 2). Our motivating experiments examine both sides of this link: regrouping a fixed selection changes its representation cost (§ 2.1), while sharing semantic scores changes demand within a fixed layout (§ 2.2). Together, these observations show that physical efficiency depends not only on *which* entries are retained, but also on how the resulting demand is distributed across members sharing page capacity. They expose a missing interface between token importance and KV memory management: selection determines not only semantic support, but also the shape of the demand that the allocator must realize. This reframes KV compression as a joint selection–representation problem rather than a token-ranking problem alone. We therefore seek to coordinate member demand before page allocation without forcing members to share identical token supports.

Building on these observations, we introduce PARSEE (**P**aging-**A**wa**R**e KV Cache Compression with **S**har**E**d s**E**mantic Support), a framework that coordinates token selection with physical page allocation (Figure 4). PARSEE combines a shared semantic reference with head-specific question evidence through separate quotas, coordinating member demand while preserving independent token identities (§ 3.2). The resulting support is compiled under an explicit physical budget: indivisible pages are allocated by marginal utility, with synchronization and page rounding charged directly to the budget (§ 3.3). A paged execution backend then packs the selected KV, commits sparse ownership, and reclaims dense blocks for reuse while supporting batched decoding over groups with heterogeneous retained lengths (§ 4).

Our evaluation spans five models from the Qwen and Llama families with different scales and attention or expert architectures, combining structural measurements, long-context benchmarks, decoding-intensive math and coding benchmarks, and backend throughput tests. At 90% prefill-context KV compression, PARSEE achieves near-lossless SCBench performance on Qwen3.8-27B and retains 95.6%–96.9% of Full KV scores on Qwen3-4B and Qwen3-8B. The Qwen3-8B backend achieves up to $5.28\times$ the throughput of uncompressed vLLM inference. Figure 1 summarizes how coordinated selection and physical execution translate into support retention and serving benefits.

Our contributions are summarized as follows:

- We connect shared scoring structure to physical KV representation cost through a conditional semantic count bound and a support-completion decomposition (§§ 3.1, 3.3).
- We introduce PARSEE, which couples shared-reference and head-specific selection with allocation under a physical page budget, preserving independent token identities across members (§ 3).
- We develop a paged execution backend that batches heterogeneous request–head groups and reclaims dense pages for reuse by the serving allocator (§ 4).
- We evaluate support geometry, task quality, and execution efficiency, demonstrating near-lossless SCBench performance on Qwen3.8-27B at 90% prefill-context KV compression and up to $5.28\times$ backend throughput over uncompressed vLLM on Qwen3-8B (§ 5).

2 MOTIVATING ANALYSES

These analyses explain why support selection should account for its eventual page representation. Section 2.1 examines how member demand and grouping turn a logical selection into physical storage cost. Section 2.2 then examines whether shared scores can coordinate that demand before allocation, motivating shared semantic support.

2.1 LOGICAL SPARSITY AND PHYSICAL CAPACITY

Token-level compression specifies how many entries to retain, but shared pages charge for how those entries are distributed across members. We examine this mismatch at equal logical size, then regroup an unchanged support to isolate grouping effects from token selection.

A tile contains m layer–head members, each allocated pages of b positions. Members share a length but can retain different token positions. For our two-layer, two-head layout, $m = 4$ and $b = 16$. Figure 2a contrasts two 32-entry supports: counts $(20, 4, 4, 4)$ require 128 physical entries, while $(8, 8, 8, 8)$ require 64. For arbitrary member counts e_h,

$$B_{\text{repr}} = mb \sum_{g} \left\lceil \frac{\max_{h \in g} e_h}{b} \right\rceil . \tag{1}$$

We next fix each captured support and its member counts, changing only the grouping (Appendix B). Regrouping the fixed support under structurally constrained layouts leaves substantial representation overhead (1.35–1.50× logical size), whereas unconstrained count sorting approaches the logical minimum (1.02×; Figure 2b). This gap motivates coordinating member counts before physical allocation.

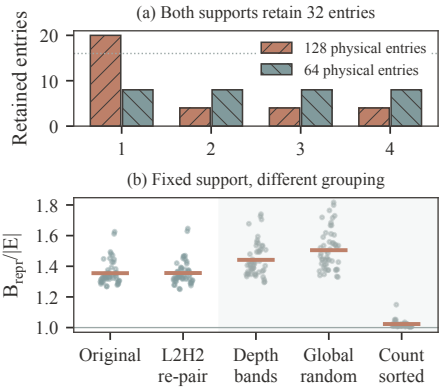


Figure 2: **The cost of realizing a support.** (a) Equal logical counts, different page costs. (b) $E(0.5)$ under different groupings; marks are task means. Shading denotes relaxed layouts; sorting excludes reorganization costs.

Observation 1. Logical sparsity alone does not determine representation capacity; the interaction between member counts and grouping matters. The fixed-support grouping study measures representation capacity. PARSEE therefore charges the pages required by its actual grouping when compiling a selection (Section 3.3).

2.2 SHARED SCORES AND SUPPORT GEOMETRY

Equation 1 charges the largest member demand in each group. For a logical support E before compilation, define

$$D_g(E) = m \max_{h \in g} e_h - \sum_{h \in g} e_h, \qquad e_h = |E_h|. \tag{2}$$

This extra capacity synchronizes member counts before page rounding. Equal-size supports can incur different costs through $D_g(E)$; different token identities alone incur no such penalty.

A shared reference with bounded member corrections can limit semantic count mismatch under a common cutoff; Section 3.1 formalizes this connection.

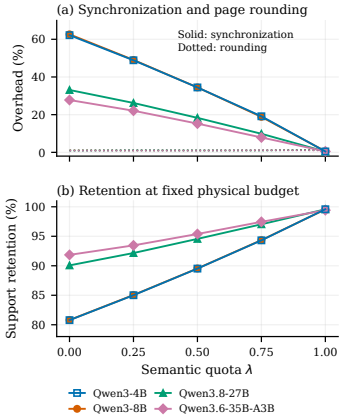


Figure 3: **Shared scores improve support geometry.** (a) Synchronization and rounding; (b) fixed-budget support retention. Task means across models.

Observation 2. In Figure 3, increasing the shared quota from attention-only to an equal quota lowers synchronization overhead and improves support retention within each model; rounding remains small. Each λ defines a different support, so Figure 3 measures capacity utilization rather than relevance quality.

Relevance is evaluated with matched-reference and shuffled-position controls in Section 5.3.

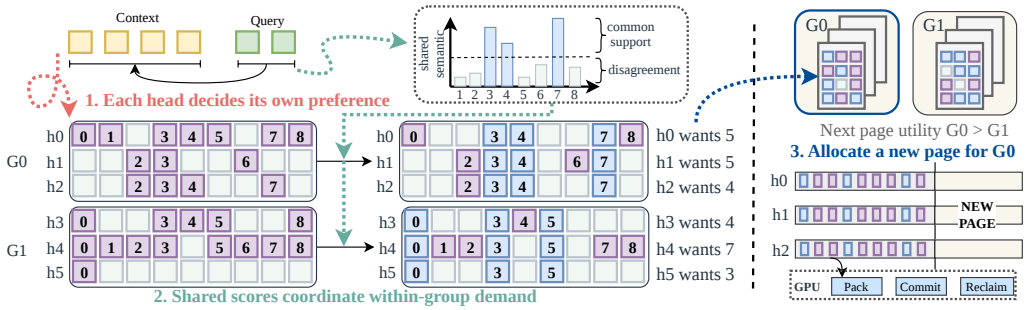


Figure 4: PARSEE workflow. (1) Head-specific question attention identifies useful context entries. (2) Shared semantic scores coordinate within-group demand while preserving member-specific evidence. (3) The compiler compares next-page marginal utility per charged entry and allocates within the physical budget. The runtime packs selected entries, commits sparse ownership, and reclaims dense pages. Group sizes, page capacities, and token IDs are illustrative.

3 PARSEE: Selection Under a Physical Budget

PARSEE maps context x and question q to a physically budgeted support through shared-score coordination, complementary relevance signals, and page allocation. We bound semantic member-count mismatch and characterize support completion. We illustrated the workflow in Figure 4.

3.1 Shared-Score Coordination

Coordination target. At a fixed logical support size, unequal member counts can increase the capacity required by grouped pages (Equations 1 and 2):

$$B_{\text{repr}}(E) = |E| + \sum_{g} D_g(E) + \text{Round}.$$

Here Round charges whole-page rounding. The goal is to control member-demand disparity while preserving independent token identities.

Shared-score structure. For a member h in tile g, a shared reference H_g and bounded member correction define

$$C_h(t) = \text{softmax}_t(H_g(t) + \delta_h(t)), \qquad 0 \le \delta_h(t) \le \eta. \tag{3}$$

Softmax includes all raw-context positions. Let A_0 denote common protected positions and F the remaining layer–head–position entries. The semantic support S_C consists of the global top-r entries in F under C; protection is excluded only at selection.

Theorem 1 (Semantic cutoff synchronization)**.** *In exact real arithmetic, consider a complete tile with common protected positions and scores from Equation 3, normalized over the same raw-context domain. Let F be the free entries and τ their global top-r cutoff, with $0 < r < |F|$. Let w_g count free positions whose shared score $\bar{C}_g(t) = \text{softmax}(H_g)(t)$ lies in the closed interval $[e^{-\eta}\tau, e^{\eta}\tau]$. Then the semantic counts $s_h = |(S_C)_h|$ satisfy*

$$D_g(S_C) := m \max_{h \in g} s_h - \sum_{h \in g} s_h \le (m-1) w_g. \tag{4}$$

Design implication. Under these conditions, semantic selection can differ across members only within the cutoff neighborhood. This bounds count mismatch while allowing different token identities, without explicitly imposing equal semantic counts. The bound depends on shared logits and bounded corrections, independently of the relevance estimator. The bound is instance-dependent because cutoff plateaus can enlarge w_g; proofs are deferred to Appendices A.3 and D.2.

This conditional count bound does not establish task relevance or quality. We next choose a shared reference for question-relevant evidence and use a separate attention quota to preserve member-specific evidence.

3.2 Shared Semantics with Head-Specific Evidence

Complementary relevance signals. PARSEE combines a shared semantic reference with head-specific question attention using disjoint quotas rather than additive score mixing. Let $P_h(t)$ denote the maximum question attention assigned to context position t for KV head h.

Residual reference. We instantiate H_g using question-conditioned similarity in contextualized residual space. The reference is broadcast through the representative-layer map f, whose shared units can span multiple tiles. For representative-question indices I_q and centered, normalized question states $\widehat{q}_{\ell,j}$ and context states $\widehat{x}_{\ell,t}$, we compute

$$z_\ell(t) = \max_{j \in I_q} \widehat{x}_{\ell,t}^{\mathsf{T}} \widehat{q}_{\ell,j}, \qquad H_g(t) = \max_{\substack{1 \le u \le T \\ |u-t| \le R}} z_{f(g)}(u). \tag{5}$$

Equation 5 aggregates representative-question similarity and applies local expansion with radius R.

Bounded member correction. Native PARSEE uses $\delta_h(t) = \eta\sigma(P_h(t))$, with sigmoid σ; numerical sensitivity is reported in Appendix F.4.

Disjoint support quotas. For T raw-context positions across L layers and H KV heads, compression ratio $\alpha \in [0, 1]$ sets the logical entry budget

$$K = LHT - \lfloor \alpha LHT \rfloor. \tag{6}$$

The protected set A replicates sink positions and bounded identifier anchors across all layer–head members. From the remaining $k = K - |A|$ entries, the top $r = \mathrm{round}(\lambda k)$ under C form S_C, and the top $k - r$ remaining entries under P form S_P. Thus

$$E = A \,\dot\cup\, S_C \,\dot\cup\, S_P. \tag{7}$$

Here $\lambda \in [0, 1]$ controls the pre-compilation semantic quota; default hyperparameters are specified in Appendix C.3.

3.3 Page Allocation and Support Completion

Every allocated page, including padding, is charged to a physical budget B. The budget definition and treatment of infeasible inputs are detailed in Appendix C.2. Question KV and subsequent decode growth are retained separately; model weights are outside this context budget.

Compiler utilities assign disjoint intervals to protected, semantic, attention, and unselected entries, ensuring selected entries precede unselected ones within each member; exact ranges are given in Appendix C.3.

Each member first orders protected entries before the remaining entries by utility. A tile starts with enough pages to hold every member's protected prefix. The compiler repeatedly allocates the next page with the greatest marginal utility per charged entry, until another page would exceed B. Members share the tile's allocated length while retaining their own token indices. Allocated padding counts against the budget even when it carries no selected entry.

The compiler reports infeasibility when the mandatory page floor exceeds B. Otherwise, allocation respects the physical budget but may stop before all of E is covered.

Support completion. Under equal page cost $c = mb$ and the stated member ordering, support completion is characterized as follows.

Theorem 2 (Support completion budget)**.** *For this compiler and the support in Equation 7, fix member orders and finite heap priorities. At a feasible budget B, the compiled support S_B contains E exactly when $B \ge B_{\mathrm{complete}}(E)$, where*

$$\begin{aligned} B_{\mathrm{repr}}(E) &= |E| + \mathsf{Sync} + \mathsf{Round}, \\ B_{\mathrm{complete}}(E) &= B_{\mathrm{repr}}(E) + \mathsf{Comp}. \end{aligned} \tag{8}$$

Every overhead term is nonnegative.

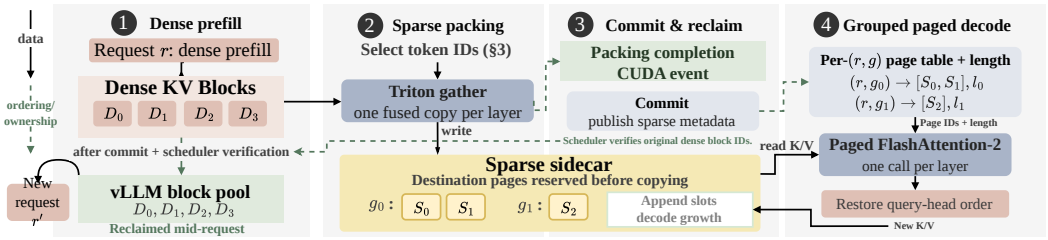


Figure 5: Physical KV execution path. Selected token IDs are packed into reserved sidecar pages. After packing completes, the scheduler reclaims the dense blocks; group-specific page tables and lengths then support paged decode and KV growth.

Sync $= \sum_g D_g(E)$ charges unequal pre-compilation member counts (Equation 2). Round charges whole-page rounding, and Comp charges nonrequired pages allocated before completion. Fitting the layout alone does not guarantee retention by the allocator. Appendix A.4 gives the rank-based decomposition and proof.

4 Physical KV Infrastructure

The backend realizes the page allocation of Section 3.3 through the pack–commit–reclaim path in Figure 5. We describe the grouped page layout, then the packing and reclamation sequence, and finally grouped decode over the resulting page tables.

Shared shape, independent identity. Each L2H2B16 page groups two layers and two KV heads, with 16 positions per member. Members share a page ID and effective length but retain their own selected token IDs. Packing sorts those IDs into original token order; keys keep their original positional encoding, and subsequent queries use their logical sequence positions. Physical capacity includes page padding rather than only the number of selected entries.

Transactional packing and reclamation. The worker reserves destination pages in a sparse sidecar and uses a fused gather to copy the selected KV. A completion event orders subsequent readers after packing. Only then does commit publish sparse metadata: the scheduler verifies the original dense block IDs before returning those blocks to the shared allocator. Sidecar capacity is reserved before admission, so reclamation cannot double-count memory. Completion or cancellation releases the sidecar pages.

Grouped decode with stable metadata. At each layer, attention flattens (request, KV-head group) into rows with separate page tables and retained lengths. One paged FlashAttention-2 call batches these rows and restores query-head order afterward. New K/V enters sidecar append slots as generation proceeds, while persistent metadata updates lengths and page crossings. The quality evaluator uses variable-length head banks; Section 5.4 reports paged-backend throughput and specifies the timing protocol. Appendix C.5 gives the accounting, rollback, mixed-batch, and runtime details.

5 Experiments

5.1 Experimental Settings

Models, tasks, and baselines. We evaluate five long-context models, encompassing three conventional GQA models (Qwen3-8B and Qwen3-4B (Yang et al., 2025), and Llama-3.1-8B-Instruct (Grattafiori et al., 2024)) and two GDN-based counterparts (the dense Qwen3.8-27B (Qwen Team, 2026b) and MoE Qwen3.6-35B-A3B (Qwen Team, 2026a)), on 16 general tasks from LongBench (Bai et al., 2024), nine KV-cache-intensive tasks from SCBench (Li et al., 2024a), as well as AIME and LiveCodeBench (Jain et al., 2025) for decode-intensive math and coding evaluation, respectively. The baselines are SnapKV, KVzip, TANGRAM with KeyDiff or SnapKV scoring, TriAttention, and Fast KVzip (Li et al., 2024b; Kim et al., 2025; 2026a; Mao et al., 2026b; Kim et al., 2026b). We additionally compare NestedKV, ChunkKV, and AdaKV (Chen et al., 2026; Liu et al., 2025; Feng et al., 2025) on Qwen3-8B SCBench in Appendix Table 9.

Table 1: SCBench results at 90% compression ratio. Pre./Var./Mix./Chc./MShot/Sum./RQA denote prefix retrieval, variable tracking, mixed retrieval, choice, many-shot, summarization, and RepoQA.

Model	Method	KV	Pre.	Var.	Mix.	QA	Chc.	MShot	Sum.	RQA	Avg.
Qwen3-8B	Full KV	64.80	43.00	31.91	38.50	39.90	72.22	43.70	35.39	48.86	46.48
	SnapKV	15.20	16.40	32.31	29.33	38.71	72.22	**43.33**	35.54	21.36	33.82↓12.66
	TriAttention	0.00	0.00	30.62	31.17	30.11	**73.61**	**43.33**	31.45	3.86	27.13↓19.35
	TANGRAM+KD	0.10	0.00	31.64	29.67	37.28	72.22	**43.33**	33.05	2.50	27.76↓18.72
	KVzip	53.70	24.80	31.87	**37.00**	35.28	72.22	42.59	**35.72**	**42.50**	41.74↓4.74
	PARSEE	**77.60**	**46.60**	**32.89**	27.17	**39.41**	**73.61**	42.59	35.55	30.00	**45.05**↓1.43
Qwen3-4B	Full KV	49.80	33.40	37.29	22.83	41.91	66.67	36.67	36.60	23.64	38.76
	SnapKV	9.40	11.80	38.36	17.50	**38.35**	**66.67**	36.30	**36.85**	10.00	29.47↓9.29
	TriAttention	0.00	0.60	38.00	17.33	27.78	**66.67**	**37.41**	32.26	2.27	24.70↓14.06
	TANGRAM+KD	0.00	0.00	34.44	19.17	29.82	63.89	36.67	34.22	0.45	24.30↓14.46
	KVzip	32.40	11.00	**38.67**	**23.67**	34.15	**66.67**	37.04	35.63	**20.00**	33.25↓5.51
	PARSEE	**59.10**	**30.80**	37.16	16.17	37.10	**66.67**	37.04	36.62	12.73	**37.04**↓1.72
Qwen3.6-35B-A3B	Full KV	99.50	99.40	94.27	58.00	58.91	90.28	55.56	36.98	73.18	74.01
	SnapKV	19.10	29.40	**94.76**	**58.00**	57.66	90.28	55.19	27.57	10.23	49.13↓24.88
	TriAttention	0.00	0.00	94.31	55.67	52.39	90.28	55.19	30.59	12.27	43.41↓30.60
	TANGRAM+KD	0.00	0.00	85.51	**58.00**	56.26	90.28	55.19	28.93	3.86	42.00↓32.01
	KVzip	14.10	1.40	93.87	56.00	52.79	90.28	**55.56**	29.58	16.14	45.52↓28.49
	PARSEE	**99.40**	**94.80**	94.13	57.67	**59.98**	**91.67**	**55.56**	**37.21**	**67.73**	**73.13**↓0.88
Qwen3.8-27B	Full KV	99.90	100.00	99.64	79.00	59.33	86.11	61.85	39.63	75.23	77.85
	SnapKV	16.90	80.00	99.73	**78.83**	58.71	83.33	**62.22**	39.73	70.45	65.54↓12.31
	TriAttention	0.30	1.20	99.73	76.67	51.04	81.94	**62.22**	33.11	17.50	47.08↓30.77
	TANGRAM+KD	0.10	0.20	94.18	75.50	57.11	**84.72**	**62.22**	31.63	6.36	45.78↓32.07
	KVzip	80.10	70.00	**99.87**	73.17	55.46	80.56	61.85	34.11	62.50	68.62↓9.23
	PARSEE	**99.80**	**99.80**	99.60	77.17	**59.02**	**84.72**	61.85	**39.96**	**73.41**	**77.26**↓0.59
Llama-3.1-8B-Instruct	Full KV	95.50	93.60	25.02	18.33	44.72	80.56	34.44	39.02	42.73	52.66
	SnapKV	41.30	74.60	23.87	**19.17**	**48.01**	**80.56**	34.07	**39.27**	19.77	42.29↓10.37
	TriAttention	0.30	1.00	25.42	**19.17**	41.01	**80.56**	33.33	30.94	5.91	26.40↓26.26
	TANGRAM+KD	0.50	0.00	19.07	**19.17**	43.90	79.17	33.70	34.42	6.36	26.25↓26.41
	KVzip	92.70	**94.20**	**26.36**	18.33	44.84	**80.56**	**34.44**	37.27	23.18	50.21↓2.45
	PARSEE	**95.40**	93.00	25.64	18.33	46.30	**80.56**	**34.44**	38.56	**26.14**	**50.93**↓1.73

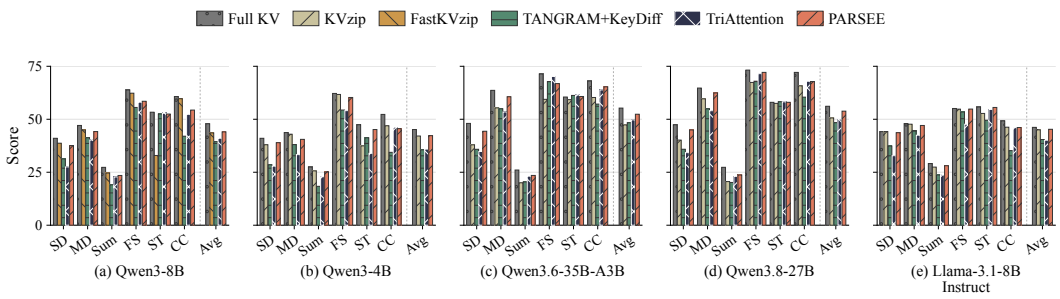


Figure 6: LongBench category scores at 90% compression ratio. Category bars average task scores: SD, Single-Doc QA (3 tasks); MD, Multi-Doc QA (3); Sum, Summarize (3); FS, Few-shot Learning (3); ST, Synthetic Task (2); and CC, Code Completion (2). Avg. is the macro over all 16 tasks.

Budget and inference protocol. Our primary evaluations use a node with two A100 GPUs. The long-context quality runs use a 90% compression ratio under the raw-context capacity budget (R), including page padding. They share dense context/question prefill and the first output token, then compress context only. The main-table comparisons share tasks and budgets. Detailed evaluation settings and memory accounting are provided in Appendix C.

5.2 Main Results

KV-cache-intensive long-context tasks. On SCBench, PARSEE stays close to Full KV across the five evaluated models. Table 1 shows that it also has the highest task macro among the displayed compressed methods for each model. On Qwen3.8-27B, PARSEE scores 77.26 versus 65.54 for SnapKV, a gain of 11.72 points; on Qwen3.6-35B-A3B, it scores 73.13 versus 49.13, a gain of 24.00 points. The largest task-level gains are on KV and prefix retrieval. SnapKV scores higher on variable tracking and mixed retrieval in both models, so the macro advantage does not imply uniform gains on every task.

General long-context tasks. On LongBench, PARSEE leads the displayed compressed methods on the task macro, with model-dependent margins. Figure 6 summarizes the 16 tasks by category. Its advantage over KVzip is modest on Qwen3-4B and Llama-3.1-8B-Instruct, and it remains below Full KV across the models.

Table 2: Decode-only KV compression on AIME and LiveCodeBench at the reported KV budgets.

Model	Method	Budget	AIME	LiveCodeBench
Qwen3-8B	Full KV	–	46.6	48.1
	R-KV	2048	33.3	43.5
	TriAttention	2048	36.7	45.0
	PARSEE	2048	**40.0**	**47.3**
	R-KV	1024	26.7	38.9
	TriAttention	1024	30.0	40.5
	PARSEE	1024	**33.3**	**43.5**
Qwen3.8-27B	Full KV	–	90.0	87.8
	R-KV	2048	76.7	80.9
	TriAttention	2048	83.3	83.2
	PARSEE	2048	**83.3**	**84.7**
	R-KV	1024	70.0	75.6
	TriAttention	1024	76.6	80.2
	PARSEE	1024	**80.0**	**80.9**

Decode-intensive math and coding tasks. For generated-KV compression, PARSEE generally outperforms R-KV (Cai et al., 2025) and TriAttention on AIME and LiveCodeBench. Table 2 shows this pattern for both models at 2,048 and 1,024 retained-KV budgets; the exception is a tie with TriAttention on Qwen3.8-27B AIME at the larger budget.

5.3 Shared-Reference and Component Ablations

Relevance beyond coordination. Table 3 compares matched semantic and attention references and tests shuffled semantic positions. Semantic references improve quality, while shuffling degrades it, supporting positional relevance beyond shared coordination. The core statistical evidence comes from the paired shuffle controls in Appendix Table 13.

Table 3: Reference controls on Qwen3-8B and Llama-3.1-8B-Instruct (LongBench / SCBench).

Reference	Qwen	Llama
Matched attention	42.35 / 43.12	43.75 / 49.10
Matched semantics	43.82 / 44.88	45.10 / 50.72
Native	44.05 / 45.05	45.27 / 50.93
Shuffled	36.80 / 35.15	38.10 / 40.20

Component ablations. Table 4 shows empirical score trends when components are removed or modified on a diagnostic SCBench subset. These comparisons do not establish that every component is necessary; none is significant against the logical reference after Holm adjustment. Appendix F.1 details the configurations and uncertainty estimates.

Table 4: SCBench diagnostic component ablations. Δ: difference from Full KV; CIs and Holm p: versus the logical reference. No MS: excluding many-shot. Details in Appendix F.1.

Configuration	Score	Δ	95% CI	Holm p	No MS
Full KV	45.79	-	-	-	-
PARSEE	42.28	-3.51	[-0.15, +8.46]	0.8784	42.41
w/o physical projection	41.80	-3.99	—	—	42.65
Anchor bonus	40.97	-4.82	[-2.50, +0.00]	1.0000	41.71
Attention-only quota	39.09	-6.70	[-5.88, +0.24]	0.8742	38.98
Semantic-only quota	34.04	-11.75	[-13.09, -2.56]	0.1296	33.92
w/o identifier anchors	34.30	-11.49	[-8.89, -6.11]	0.8742	34.21
Hard protection + L2H2B16	41.47	-4.32	[-3.17, +2.30]	1.0000	42.28
Anchor bonus + L2H2B16	41.47	-4.32	[-3.17, +2.30]	1.0000	42.28

5.4 Throughput and Cost

Figure 7(a–c) compares Qwen3-8B BF16 backends on the two-A100 node at 4,096 prefill tokens and 128, 512 or 2,048 output tokens. Its clock includes prefill, packing or eviction, generation and cache release, but excludes precomputed PARSEE/HF scoring and selection; TANGRAM and TriAttention retain native inline work. Under this backend timing boundary, PARSEE exceeds TANGRAM+KeyDiff and TriAttention in all 11 settings ($1.036\times$ geometric mean over the faster one). KVzip and SnapKV complete three and nine HF settings; their lower rates also reflect different runtimes.

Figure 7(d) shows the separate common-runtime 32K raw-context bucket with batch size four and 512 output tokens per request. With online scoring, selection and packing included, PARSEE takes 56.87 seconds, 3.6% less than TANGRAM+SnapKV (58.99 seconds) and 25.0% less than Full KV (75.79 seconds). Relative to TANGRAM+SnapKV, shorter context prefill and decode offset the 1.699-second scoring stage (Table 21). Dense source KV remains allocated, so this panel does not

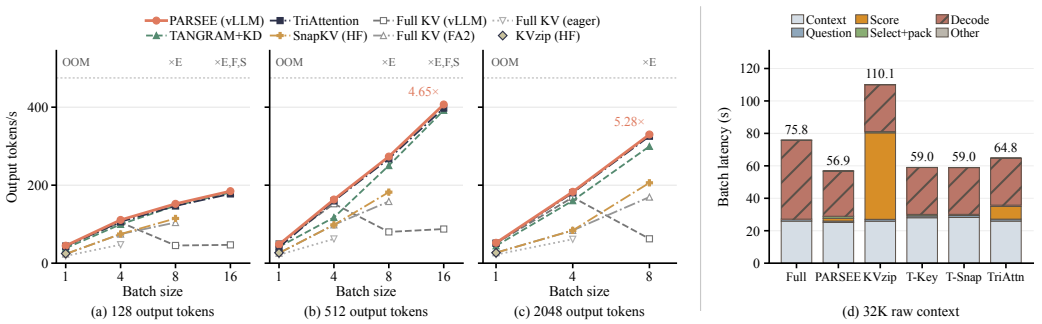


Figure 7: Backend throughput and complete batch latency on Qwen3-8B. (a–c) Fixed-selection backend throughput for three output lengths; (d) scorer-inclusive 32K raw-context latency with batch size four and 512 output tokens per request. Phase means average two inputs, three repeats and two worker runs (Table 21); Select+pack combines two columns. The timing protocols differ; see Section 5.4 and Appendices C.6 and F.7.2.

measure native V1 online serving throughput. Appendices C.6 and F.7 give timing, capacity and storage details; Appendix F.6 separately pairs SCBench quality with fixed-length online serving measurements at concurrency one and eight.

6 Related Work

KV selection and adaptive budgets. Attention-based eviction protects heavy hitters and recent tokens (H_2O) or aggregates a prompt-end window to select per-head positions (SnapKV) (Zhang et al., 2023; Li et al., 2024b). Reconstruction methods rank entries by context reconstruction before a future query is known (KVzip) or learn sink-attention gates for the same objective (Fast KVzip) (Kim et al., 2025; 2026b). HeadKV and Ada-KV adapt budgets to retrieval/reasoning importance and attention-output error, respectively (Fu et al., 2025; Feng et al., 2025). ChunkKV reuses semantic chunks across layers, while NestedKV combines multi-scale key scores with adaptive head budgets (Liu et al., 2025; Chen et al., 2026). PARSEE combines question-conditioned shared residual scores with head-specific attention. Disjoint quotas preserve per-head token identities, while shared scores coordinate member counts under a physical page budget.

Physical storage and serving. PagedAttention reuses physical KV blocks and FlashAttention tiles attention to limit memory traffic (Kwon et al., 2023; Dao et al., 2022; Dao, 2023). Reclaiming blocks after eviction requires compaction (Mao et al., 2026a). TriAttention uses pre-RoPE geometry for key scoring and compacts survivors under a uniform head budget, while TANGRAM groups offline-calibrated head budgets in ragged page tables (Mao et al., 2026b; Kim et al., 2026a). Post-hoc grouping cannot change selected head counts; PARSEE coordinates them during selection, then allocates under an executable physical-support budget that charges whole pages.

7 Conclusion

In this work, we addressed the gap between logical KV-cache compression and reusable physical capacity in LLM serving. We observed that unequal retained counts within shared page groups inflate storage costs, even under identical logical budgets. Based on this observation, we introduce PARSEE, a paging-aware compression framework. It first coordinates shared semantic and head-specific evidence through disjoint selection quotas. A physical compiler then allocates pages under a capacity budget, while a paged backend packs survivors and reclaims dense blocks for reuse. Experiments across five models show that PARSEE preserves SCBench quality close to Full KV at 90% prefill-context compression. Backend measurements also demonstrate throughput gains, supporting efficient long-context serving.

Reproducibility Statement

The appendix provides complete proofs, scoring and layout definitions, evaluation protocols, and detailed results. Comparisons specify their budgets, input populations, aggregation, and timing boundaries. The manuscript package includes the LaTeX sources, figures, tables, and bibliography. Code is available on https://anonymous.4open.science/r/parsee-kv-465F/.

AI Use Statement

Generative AI was employed in this work for manuscript polishing, experimental code implementation, and debugging. They also assisted with schematic preparation, source compilation and finding related works.

References

Changwoo Baek, Seungjun Shin, and Kyeongbo Kong. RestoreKV: Recovering Full-Cache Behavior Under Aggressive Query-Agnostic KV Cache Eviction. *arXiv preprint arXiv:2608.01247*, 2026. URL https://arxiv.org/abs/2608.01247v1.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3119–3137, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.172. URL https://aclanthology.org/2024.acl-long.172/.

Zefan Cai, Wen Xiao, Hanshi Sun, Cheng Luo, Yikai Zhang, Ke Wan, Yucheng Li, Yeyang Zhou, Li-Wen Chang, Jiuxiang Gu, Zhen Dong, Animashree Anandkumar, Abedelkadir Asi, and Junjie Hu. R-KV: Redundancy-aware KV Cache Compression for Reasoning Models. In *Advances in Neural Information Processing Systems*, volume 38, pp. 60980–61005. Curran Associates, Inc., 2025. doi: 10.52202/085713-2038. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/57e0358aed314784cad14a26e1eba642-Paper-Conference.pdf.

Hong Chen, Xiang Liu, Yubo Gao, Yuxuan Fan, Bo Wang, Yuanlin Chu, Yuanguo Lin, and Xuming Hu. NestedKV: Nested Memory Routing for Long-Context KV Cache Compression. *arXiv preprint arXiv:2605.26678*, 2026. URL https://arxiv.org/abs/2605.26678.

Tri Dao. FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning. *arXiv preprint arXiv:2307.08691*, 2023. URL https://arxiv.org/abs/2307.08691.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. *arXiv preprint arXiv:2205.14135*, 2022. URL https://arxiv.org/abs/2205.14135.

Chao Fei, Guozhong Li, Chenxi Liu, and Panos Kalnis. CHESS: Context-aware Hierarchical Efficient Semantic Selection for Long-Context LLM Inference. *arXiv preprint arXiv:2602.20732*, 2026. URL https://arxiv.org/abs/2602.20732v1.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S. Kevin Zhou. Ada-KV: Optimizing KV Cache Eviction by Adaptive Budget Allocation for Efficient LLM Inference. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://arxiv.org/abs/2407.11550.

Yu Fu, Zefan Cai, Abedelkadir Asi, Wayne Xiong, Yue Dong, and Wen Xiao. Not All Heads Matter: A Head-Level KV Cache Compression Method with Integrated Retrieval and Reasoning. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/f649556471416b35e60ae0de7c1e3619-Paper-Conference.pdf.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, et al. The Llama 3 Herd of Models. *arXiv preprint arXiv:2407.21783*, 2024. URL https://arxiv.org/abs/2407.21783.

Naman Jain, Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and Contamination Free Evaluation of Large Language Models for Code. In *International Conference on Learning Representations*, pp. 58791–58831, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/94074dd5a072d28ff75a76dabed43767-Paper-Conference.pdf.

Hyungmin Kim, Minsoo Kim, Hongseok Kim, and Jungwook Choi. Tangram: Unlocking Non-Uniform KV Cache Compression for Efficient Multi-turn LLM Serving. *arXiv preprint arXiv:2606.06302*, 2026a. URL https://arxiv.org/abs/2606.06302v2.

Jang-Hyun Kim, Jinuk Kim, Sangwoo Kwon, Jae W. Lee, Sangdoo Yun, and Hyun Oh Song. KVzip: Query-Agnostic KV Cache Compression with Context Reconstruction. In *Advances in Neural Information Processing Systems*, volume 38, pp. 167563–167591. Curran Associates, Inc., 2025. doi: 10.52202/085713-5585. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/f4eaa4b8f2d08edb3f0af990d56134ea-Paper-Conference.pdf.

Jang-Hyun Kim, Dongyoon Han, and Sangdoo Yun. Fast KVzip: Efficient and Accurate LLM Inference with Gated KV Eviction. *arXiv preprint arXiv:2601.17668*, 2026b. URL https://arxiv.org/abs/2601.17668v2.

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. URL https://arxiv.org/abs/2309.06180.

Yucheng Li, Huiqiang Jiang, Qianhui Wu, Xufang Luo, Surin Ahn, Chengruidong Zhang, Amir H. Abdi, Dongsheng Li, Jianfeng Gao, Yuqing Yang, and Lili Qiu. SCBench: A KV Cache-Centric Analysis of Long-Context Methods. *arXiv preprint arXiv:2412.10319*, 2024a. URL https://arxiv.org/abs/2412.10319.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. SnapKV: LLM Knows What You are Looking for Before Generation. *arXiv preprint arXiv:2404.14469*, 2024b. URL https://arxiv.org/abs/2404.14469.

Xiang Liu, Zhenheng Tang, Peijie Dong, Zeyu Li, Yue Liu, Bo Li, Xuming Hu, and Xiaowen Chu. ChunkKV: Semantic-Preserving KV Cache Compression for Efficient Long-Context LLM Inference. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://arxiv.org/abs/2502.00299v4.

Weian Mao, Yukang Chen, Wei Huang, Shuai Yang, Luozhou Wang, and Song Han. KV Cache Compression and Its Infra Problems. NVIDIA Research, June 2026a. URL https://research.nvidia.com/labs/eai/blogs/kv-cache-compression-and-its-infra-problems/.

Weian Mao, Xi Lin, Wei Huang, Yuxin Xie, Tianfu Fu, Bohan Zhuang, Song Han, and Yukang Chen. TriAttention: Efficient Long Reasoning with Trigonometric KV Compression. *arXiv preprint arXiv:2604.04921*, 2026b. URL https://arxiv.org/abs/2604.04921.

Qwen Team. Qwen3.6-35B-A3B model card, 2026a. URL https://huggingface.co/Qwen/Qwen3.6-35B-A3B.

Qwen Team. Qwen3.8-27B model card, 2026b. URL https://huggingface.co/Qwen/Qwen3.8-27B.

Haolin Tian, Yuzhe Liu, and Tonghan Wang. Every Cache Entry Earns Its Place: Global Allocation of Resolution and Coverage for KV Cache Compression. *arXiv preprint arXiv:2608.07001*, 2026. URL https://arxiv.org/abs/2608.07001v1.

An Yang, Anfeng Li, Baosong Yang, et al. Qwen3 Technical Report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

Yanqi Zhang, Yuwei Hu, Runyuan Zhao, John C. S. Lui, and Haibo Chen. DiffKV: Differentiated Memory Management for Large Language Models with Parallel KV Compaction. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles*, pp. 431–445. ACM, 2025. doi: 10.1145/3731569.3764810. URL https://doi.org/10.1145/3731569.3764810.

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. H_2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models. *arXiv preprint arXiv:2306.14048*, 2023. URL https://arxiv.org/abs/2306.14048.

APPENDIX CONTENTS

A Complete Proofs . . . 14

B Settings for the Motivating Experiments . . . 17

B.1 Inputs and Interventions . . . 17

B.2 Geometry and Numerical Diagnostics . . . 17

C Main Evaluation Settings . . . 18

C.1 Models, Tasks, and Inference . . . 18

C.2 Budgets and Baselines . . . 18

C.3 Scoring and Protection . . . 19

C.4 Decode-Only Adaptation . . . 20

C.5 Physical Backend . . . 20

C.6 Throughput Workload and Timing . . . 20

D Results of the Motivating Experiments . . . 21

D.1 Grouping and Shared-Quota Geometry . . . 21

D.2 Semantic Cutoff Bound . . . 21

E Detailed Main Results . . . 22

E.1 Full Task Scores . . . 22

E.2 Backend Throughput . . . 22

F Supplementary Experiments . . . 22

F.1 Component Ablations . . . 22

F.2 Shared-Reference and Position Controls . . . 24

F.3 Scoring Sensitivity . . . 25

F.4 Perturbation Diagnostics . . . 28

F.5 Semantic Quota . . . 28

F.6 Quality and Online Serving . . . 28

F.7 Additional Systems Measurements . . . 29

G Additional Related Work and Implementation Discussion . . . 33

H Discussion, Limitations, and Future Work . . . 34

H.1 Discussion: Successful Retrieval Cases . . . 34

H.2 Discussion: Failure Cases . . . 35

H.3 Discussion: Scope and Design Choices . . . 37

H.4 Limitations . . . 37

H.5 Future Work . . . 38

A Complete Proofs

This appendix gives the complete support-retention argument. Budgets are measured in layer–KV-head–token entries. At BF16 with head dimension 128, one entry's K+V payload is 512 bytes. The budget fixes the raw-context payload; weights, question KV, metadata, and a retained dense source are separate quantities.

A.1 Model and Edge Cases

Fix $T \geq 1$, a tile partition $\mathcal{G}$, m members per slab, and page length b. The slab cost is $c = mb$. Partial edge tiles are padded with virtual members whose counts, ranks, and utilities are zero. Each actual member h has a protected set A_h and a fixed permutation π_h: all protected positions precede the remaining positions, which are ordered by finite utility with fixed tie-breaking. The declared support E_h contains A_h. This support is fixed throughout a budget sweep.

Let $N = \lceil T/b \rceil$. Allocating $n_g \in \{0, \dots, N\}$ pages to tile g retains the first $\min(bn_g, T)$ positions in every actual member's order. Protection requires $a_g = \lceil \max_{h \in g} |A_h|/b \rceil$ pages, with total floor $B_0 = c \sum_g a_g$. Budgets below the floor are infeasible. Write $e_h = |E_h|$, $r_h = \max\{\pi_h^{-1}(t) : t \in E_h\}$, $d_g = \max_{h \in g} r_h$, and $j_g = \lceil d_g/b \rceil$. Ranks are one-based and empty supports have rank zero. Since E contains protection, $j_g \geq a_g$.

Proposition 1 (Representation and ordered representation)**.** *With arbitrary token identities, the minimum slab capacity containing E is*

$$B_{\mathrm{repr}}(E) = c \sum_g \left\lceil \max_{h \in g} e_h/b \right\rceil .$$

With fixed member orders and prefix retention, the corresponding minimum is

$$B_{\mathrm{rank}}(E) = c \sum_g \left\lceil \max_{h \in g} r_h/b \right\rceil = c \sum_g j_g .$$

Proof. Every member needs at least e_h slots to contain its support, so $bn_g \geq \max_{h \in g} e_h$ is necessary. Rounding and summing gives the first lower bound. It is attainable by placing each E_h into these slots, filling unused positions with other entries or padding. For a fixed order, E_h is contained in the retained prefix exactly when $r_h \leq bn_g$. Thus $n_g \geq j_g$ is both necessary and sufficient, proving the second formula. □

The formulas depend on member counts or ranks. They do not require contiguous source positions or identical token identities across members.

A.2 Equal-Cost Allocation

The utility of page j in tile g is

$$v_{gj} = \sum_{h \in g} \sum_{i=(j-1)b+1}^{\min(jb,T)} u_{h,\pi_h(i)} .$$

The optional pages have $a_g < j \leq N$. Beginning at the protected floors, the heap pops the greatest exposed priority and exposes the next page of the same tile. Its deterministic tie rule uses tile and page indices. Let $\rho(g, j)$ be the one-based order in this complete heap execution without a budget cutoff. This execution order is defined for finite computed priorities, even if arithmetic rounding affects their monotonicity.

Lemma 1 (Discrete concave allocation)**.** *If optional page utilities are nonnegative and nonincreasing along each tile chain, the heap selects the first M pages in the global order $(-v_{gj}/c, g, j)$, where*

$$M = \min \left\{ \lfloor (B - B_0)/c \rfloor , \sum_g (N - a_g) \right\} .$$

These pages maximize total utility under the capacity and prefix constraints. In exact arithmetic, nonnegative descending entry utilities after protection imply the required page monotonicity.

Proof. The protection floor contains all protected entries. Each remaining member suffix is nonnegative and descending. Padding its last page with zeros preserves descent, so sums of successive b entries are nonincreasing; summing members preserves this property. In the global page order, every predecessor appears before its successor, including tied utilities because the smaller page index comes first. Hence the first M pages form legal tile prefixes. At any step, the globally first unselected page must be exposed: otherwise an unselected predecessor would occur earlier in the global order. Induction therefore identifies the heap sequence with that global order. Any feasible allocation chooses at most M optional pages, so its utility cannot exceed the top-M sum. The top-M allocation is feasible and attains this bound. Nonnegativity allows all affordable pages to be used; the protected-page contribution is constant. □

This is the equal-cost discrete concave allocation property. Applied to computed priorities, its optimality statement concerns those priorities and requires their chain monotonicity. The completion threshold below instead uses the actual execution order directly.

A.3 Proof of Theorem 1

All members in a complete tile share H_g and protected positions A_0. For $0 \le \delta_h(t) \le \eta$, set $C_h(t) = \mathrm{softmax}(H_g(t) + \delta_h(t))$ and $\bar{C}_g(t) = \mathrm{softmax}(H_g(t))$. Normalization includes all raw-context positions; the global top-r selection excludes protected positions. Let F denote all free entries and τ their global top-r cutoff, with $0 < r < |F|$. The closed cutoff neighborhood and semantic member counts are

$$\mathcal{B}_g = \{t \notin A_0 : e^{-\eta}\tau \le \bar{C}_g(t) \le e^{\eta}\tau\}, \qquad w_g = |\mathcal{B}_g|, \qquad s_h = |(S_C)_h|.$$

Proof. Dividing the two softmax expressions yields

$$\frac{C_h(t)}{\bar{C}_g(t)} = \frac{e^{\delta_h(t)}}{\sum_s \bar{C}_g(s) e^{\delta_h(s)}} \in [e^{-\eta}, e^{\eta}].$$

If a free position satisfies $\bar{C}_g(t) > e^{\eta}\tau$, every member's score is strictly above τ, so every member selects it. If $\bar{C}_g(t) < e^{-\eta}\tau$, every member's score is below τ, so none selects it. Only positions in the closed neighborhood $\mathcal{B}_g$ can differ. Let z_g count the positions necessarily selected by all members. Each member count is $s_h = z_g + x_h$ for $0 \le x_h \le w_g$. At least one member attains the maximum, and every other count differs from it by at most w_g. Consequently,

$$D_g(S_C) = \sum_{h \in g} (\max_j s_j - s_h) \le (m-1)(\max_h s_h - \min_h s_h) \le (m-1) w_g.$$

□

When $r = 0$ or all free entries are selected, all member counts agree and $D_g(S_C) = 0$; define $w_g = 0$ at these endpoints. Ties at the cutoff belong to the closed neighborhood, including when $\eta = 0$. Max pooling can create tied score plateaus, so neighborhood width is an instance-dependent measurement.

Computed-score extension. If $|\log \widehat{C}_h(t) - \log C_h(t)| \le \epsilon$, let $\widehat{\tau}$ be the actual top-r cutoff of $\widehat{C}$. The same argument holds for the closed neighborhood

$$[e^{-(\eta+\epsilon)}\widehat{\tau},\ e^{\eta+\epsilon}\widehat{\tau}]$$

around $\bar{C}_g$. Both the cutoff and radius change. The real-arithmetic theorem does not presume an unmeasured machine-error bound; the structural experiment records numerical comparisons explicitly.

A.4 Proof of Theorem 2

The required optional set is $\mathcal{R}_E = \{(g, j) : a_g < j \le j_g\}$. Let $q_E = |\mathcal{R}_E|$ and $M_E = \max_{\mathcal{R}_E} \rho(g, j)$, with $M_E = 0$ if the set is empty. For arbitrary fixed member orders, finite heap

priorities, and a support containing protection, the complete decomposition is

$$\begin{aligned} B_{\text{complete}}(E) &= B_0 + cM_E \\ &= |E| + \mathsf{Rank} + \mathsf{Sync} + \mathsf{Round} + \mathsf{Comp}, \end{aligned} \tag{9}$$

$$\mathsf{Rank} = \sum_h (r_h - e_h), \qquad \mathsf{Sync} = \sum_g \left(md_g - \sum_{h \in g} r_h \right),$$

$$\mathsf{Round} = m \sum_g (bj_g - d_g), \qquad \mathsf{Comp} = c(M_E - q_E).$$

The terms charge higher-ranked filler, unequal member demand, page rounding, and optional pages allocated before completion beyond those required for E.

Proof. All optional pages have the same cost. Whenever remaining capacity is at least c, every exposed page is affordable; if remaining capacity is smaller, no page is affordable. Thus a budgeted run is exactly a prefix of the untruncated heap run, of length $\min\{\lfloor (B - B_0)/c \rfloor, \sum_g (N - a_g)\}$. By Proposition 1, it retains all support exactly when it has selected every page in $\mathcal{R}_E$. This happens exactly when it has selected at least M_E optional pages, or

$$B \geq B_0 + cM_E.$$

For nonnegativity, $r_h \geq e_h$ because e_h distinct entries lie within the first r_h positions. Also $r_h \leq d_g$ and $d_g \leq bj_g$. The q_E required pages have distinct positive execution ranks; their maximum is at least q_E. These facts prove that all four overheads in Equation 9 are nonnegative. Finally,

$$\begin{aligned} |E| + \mathsf{Rank} + \mathsf{Sync} + \mathsf{Round} &= \sum_h e_h + \sum_h (r_h - e_h) \\ &\quad + \sum_g \left(md_g - \sum_{h \in g} r_h \right) + m \sum_g (bj_g - d_g) \\ &= c \sum_g j_g. \end{aligned}$$

Since $q_E = \sum_g (j_g - a_g)$, adding $c(M_E - q_E)$ gives $B_0 + cM_E$. □

For PARSEE's separated utility intervals, selected entries precede unselected entries, so $r_h = e_h$ and $\mathsf{Rank} = 0$. Hence $B_{\text{rank}} = B_{\text{repr}}$, and Equation 9 specializes to Theorem 2. The threshold depends on the actual execution order and remains valid for finite computed priorities without requiring floating-page sums to preserve monotonicity. The same formulas include virtual members, partial last pages, empty supports, and protection-only supports. An empty E requires empty protection and then every term is zero. The capacity variable is the final effective compiler budget; if an API caps the admissible range below the threshold, the support is unreachable within that range. Changing compression ratio and recomputing scores or E defines a different support-completion problem.

A.5 Hybrid Support and Equal-Budget Preservation

The support $E = A \dot{\cup} S_C \dot{\cup} S_P$ has common protection, r semantic entries, and $k - r$ attention entries. The four utility intervals are strictly separated, so selected entries precede unselected entries in each member. Thus $r_h = e_h$ and $\mathsf{Rank} = 0$. Writing $s_h = |(S_C)_h|$ and $p_h = |(S_P)_h|$,

$$m \max_h (s_h + p_h) - \sum_h (s_h + p_h) \leq \left(m \max_h s_h - \sum_h s_h \right) + \left(m \max_h p_h - \sum_h p_h \right).$$

Common protection adds equal counts and does not change this difference. Theorem 1 controls the first summand, and the second is at most $(m-1) \sum_h p_h$. Writing $D(E) = \sum_g (m \max_{h \in g} e_h - \sum_{h \in g} e_h)$ and summing over complete tiles yields

$$B_{\text{repr}}(E) = |E| + D(E) + \mathsf{Round}, \qquad D(E) \leq (m-1) \left(\sum_g w_g + k - r \right). \tag{10}$$

Integer rounding additionally gives $0 \le \mathsf{Round} \le m(b-1)|\mathcal{G}|$. The shared semantic component controls a source of physical capacity overhead while the attention component retains member-specific evidence; the quota chooses a quality–representation tradeoff.

Corollary 1 (Equal-budget preservation)**.** *For complete tiles and the stated utility intervals, the actual compiler retains every entry of E at $B = |E|$ if and only if each tile's member counts are equal multiples of b.*

Proof. The condition is equivalent to $D(E) + \mathsf{Round} = 0$. If this sum is positive, Proposition 1 gives $B_{\mathrm{repr}} > |E|$, so no admissible representation can contain all of E. Conversely, if the sum is zero, each required page is a complete page of selected entries, with utility at least $2c$, while every other page has utility at most c. All protection fits within the required pages. Every still-needed tile has an exposed required page until completion; the heap therefore selects all required pages before any other page. There is no competition overhead and $B_{\mathrm{complete}} = |E|$. □

This criterion concerns the declared support and specified compiler. The experiments separately measure generated-answer quality and support retention.

A.6 Finite Verification

The independent CPU verification suite contains 134,528 assertions, including 262 compiler instances, 51,116 threshold checks, 7,588 decomposition checks, exhaustive allocation comparisons on 1,650 small instances, 100 Hybrid cases, and cutoff/tie and numerical-perturbation tests. These enumerate small finite cases and boundary constructions; the arguments above provide the general results.

B Settings for the Motivating Experiments

B.1 Inputs and Interventions

The motivating analyses use a 90% compression ratio under the raw-context budget, including page padding. We select two development inputs from each of 25 tasks, using the first actual question of each case. The 50 inputs contain 48 distinct raw contexts and span 564–124,539 context tokens. Results average inputs within each task and then weight tasks equally.

For Figure 2, we fix the support $E(0.5)$ and every member's retained count, varying only the grouping. The reference L2H2B16 layout groups two layers and two KV heads with 16 positions per page. Alternatives re-pair heads within the same two layers, form random groups within depth bands or globally, or sort counts into unconstrained groups. Random-grouping points average 100 mappings; relaxed groupings exclude reorganization cost.

Figure 3 evaluates Qwen3-8B, Qwen3-4B, Qwen3.8-27B, and Qwen3.6-35B-A3B on the same inputs. It sweeps $\lambda \in \{0, 0.25, 0.5, 0.75, 1\}$ with fixed protection, layout, and physical budget. Each quota defines its own support $E(\lambda)$. Hybrid models pair adjacent full-attention layers in their original order and retain recurrent state separately. Models use native tokenizers and templates; prefill chunks are 16K for Qwen3-8B and 2K for the other models. Comparisons are within model.

B.2 Geometry and Numerical Diagnostics

The Qwen3-8B probe captures FP32 attention and semantic signals, protection masks, and representative-question indices once per input, then applies the selector and compiler at each quota. Synchronization, rounding, and representation capacity are normalized by $|E(\lambda)|$; compiled recall is $|E \cap S_B|/|E|$. Thus representation capacity is $100\% + \mathrm{Sync} + \mathrm{Round}$, while recall measures retention of each configuration's declared support.

The cutoff diagnostic compares native scores with a shared FP64 log-softmax base. At the actual native cutoff $\widehat{\tau}$, the numerical log-radius a_g^{num} is the larger of η plus the measured log-score error and the measured deviation from the shared FP64 base, with a 10^{-12} comparison guard. It counts

$$w_g^{\mathrm{num}} = \left|\left\{t \notin A_0 : \left|\log \bar{C}_g^{64}(t) - \log \widehat{\tau}\right| \le a_g^{\mathrm{num}}\right\}\right|, \qquad U_g^{\mathrm{num}} = (m-1)w_g^{\mathrm{num}}. \tag{11}$$

For $n_g > 0$ free positions, Shell is $100w_g^{\mathrm{num}}/n_g$; tied scores are included. Synchronization and upper-bound columns are $100\sum_g \widehat{D}_g/|E|$ and $100\sum_g U_g^{\mathrm{num}}/|E|$, averaged by task. We also report

$$\rho_{\mathrm{agg}} = \frac{\sum_g \widehat{D}_g}{\sum_g U_g^{\mathrm{num}}}, \qquad \sum_g U_g^{\mathrm{num}} > 0, \tag{12}$$

averaged over defined input-level ratios within tasks. Empty tiles and zero bounds are handled separately; endpoint quotas follow Appendix A.3. FP64 comparison is a numerical diagnostic, not a certified real-arithmetic error bound. These measurements concern representation cost; allocation competition remains a separate completion cost.

C Main Evaluation Settings

C.1 Models, Tasks, and Inference

We evaluate the five Qwen and Llama models listed in Section 5 using BF16, FlashAttention-2, native tokenizers, and native chat templates. Dense Qwen3-8B/4B models use YaRN factor four and a maximum sequence length of 131,072. Hybrid models compress full-attention KV while retaining recurrent state. Systems experiments use Qwen3-8B on the two-A100 node.

Long-context quality evaluation covers 16 LongBench tasks and nine SCBench tasks, with greedy generation and native task-specific output limits. All methods share dense context/question prefill and the first output token, then compress context KV for subsequent generation. Question KV and generated KV remain retained. The quality evaluator keeps the dense source alongside compressed head banks; quality scores therefore do not measure allocator reclamation. The LongBench comparison uses a common input exclusion for a short context whose mandatory page floor exceeds the budget.

Scores use native benchmark metrics on a 0–100 scale, averaging questions within rows, rows within tasks, and tasks equally. LongBench combines token F1, ROUGE-L, class matching, numeric retrieval, and code similarity; SCBench uses task-specific answer inclusion, exact match, F1, ROUGE-L, and RepoQA matching. Gold answers enter metric calculation only. Figure 6 averages tasks within its six categories, while its final Avg. uses all 16 tasks. Main-task comparisons are descriptive point estimates; paired tests are specified with the supplementary experiments.

C.2 Budgets and Baselines

The primary quality comparisons use a 90% raw-context compression ratio with physical budget $B = K$ from Equation 6. All allocated padding is charged. For hybrid models, this cap applies to full-attention KV, with recurrent state accounted for separately. Protection and question conditioning follow the same rules on every input.

Full KV retains the cache. KVzip selects individual entries using reconstruction attention; the dense Qwen configurations use 2,048-token reconstruction chunks and an eight-token postfix. Qwen3-8B uses BF16 softmax, maximum reduction, and query-score batch 64; Qwen3-4B uses full-reconstruction-query BF16 recomputation. SnapKV uses the final 64 prompt queries, average pooling with kernel five, and mean aggregation across query heads. Fast KVzip uses the released Qwen3-8B gate, global layer-budget allocation, four sink priorities, and native tail priorities within the retained-entry budget. TANGRAM+KeyDiff and TANGRAM+SnapKV use their respective scorers with calibrated head grouping. Backend-specific timing boundaries are given in Appendix C.6.

TriAttention adaptation for prefill-cache compression. TriAttention was introduced for periodic KV eviction during long-reasoning decoding (Mao et al., 2026b). For LongBench and SCBench, we apply its scorer once to raw-context keys after dense context/question prefill, using fixed model-specific offline pre-RoPE query statistics and the full prompt length as the scoring-round origin. We retain its trigonometric and norm terms, geometric future offsets, per-query-head normalization, maximum reduction within each GQA group, and mean reduction across compressed layers. Its per-head selection policy assigns a uniform budget by dividing the common raw-context entry budget across compressed layers and KV heads, rounded down; each KV head shares its selected positions across layers. Four sink positions count toward this budget. Prefix, question, and generated KV

remain uncompressed, and the first output token uses the shared dense-prefill logits. The selected context set then stays fixed throughout decoding.

C.3 Scoring and Protection

Table 5: Shared-support hyperparameters for the dense-model evaluations.

Parameter	Default value
Nominal compression / layout	0.90 / 2 layers × 2 KV heads × 16 positions
Semantic coordinate dimension	512
Semantic representative layers (zero-based)	11, 23, 35
Diverse question representatives / pool radius	8 / 16
Semantic quota / head perturbation	0.5 / $10^{-5}\sigma(P)$
Sinks / identifier anchor cap / recent retention	4 / 144 / 0
Prefill chunk / maximum total sequence	16,384 / 131,072 tokens

Table 5 specifies the dense-model defaults. The protected support includes four sink positions and at most 144 identifier-anchor positions. A qualifying identifier has at least 12 characters, includes a digit and delimiter, and occurs 1–8 times in the context; anchor expansion uses bridge threshold 64, radius 56, and at most eight matches. Scoring uses the actual question. Named reference, dimension, pooling, and perturbation variants change only their stated factors.

Reference construction. Layer-input residual sketches are captured during prefill. A fixed coordinate permutation supplies 512 sorted coordinates, shared across context, question, and representative layers. Sketches are stored in BF16 and promoted to FP32. At the deepest representative layer, farthest-point sampling on normalized, uncentered question sketches starts from the final question token and selects up to eight representatives. Their indices are reused at every representative layer.

For each layer ℓ, subtract the context mean μ_ℓ from sampled context and question vectors and normalize with $N(v) = v/\max(\|v\|_2, 10^{-12})$. Centering follows representative selection. The shared reference takes the maximum cosine over representatives, then radius-16 local maximum pooling over context positions, with truncated boundary windows. This is coordinate subsampling and local relevance expansion; it does not impose contiguous support.

Sharing scope. For dense Qwen3-8B/4B, the nearest representative layer among 11/23/35 supplies the reference, with smaller index breaking distance ties. Thus layers 0–17 map to 11, 18–29 to 23, and 30–35 to 35. Scores are broadcast to all eight KV heads in each mapped layer. The actual shared unit is

$$\mathcal{B}(g) = \{h = (\ell, a) : f(\ell) = f(g),\ 0 \le a < H\}. \tag{13}$$

Other models retain their own representative-layer map; hybrid models preserve original full-attention layer indices. Reference replacements keep these broadcast units fixed.

Precision and ordering. Centered cosine, pooling, attention reduction, the bounded correction, and semantic softmax use FP32; question-attention QK products use FP16. Semantic softmax covers all raw-context positions, including protected ones. Both quota stages rank free layer–head–position entries with native `topk`, using round-to-even quota counts and no added lexicographic tie rule. Compiler utilities use stable descending order with protection first and token-position order for equal utilities; heap ties use tile and page indices. The $10^{-5}\sigma(P_h)$ correction does not guarantee unique scores or cross-device ordering. Sensitivity is measured in Appendices F.3 and F.4.

Attention and quota conventions. $P_h(t)$ takes the maximum attention to context position t over actual question queries and query heads associated with KV head h, retaining the model's positional encoding. Identifier anchors are a separate bounded protection policy outside the free-entry quotas; reference comparisons hold this policy fixed. Both quota selections are global across layers, heads, and positions, so normalization affects quota membership. The parameter λ is neither an additive mixing weight nor the semantic fraction after compilation. There is no mandatory recent-token window. For complete tiles, common protection adds equal member counts and $D_g(E) \le D_g(S_C) + D_g(S_P)$ (Appendix A.5); the attention quota can contribute disparity beyond the semantic count bound.

Compiler utility intervals. Utilities use $[6, 7]$ for protection, $[4, 5]$ for semantic selections, $[2, 3]$ for attention selections, and $[0, 1]$ for unselected entries; normalized scores order entries within intervals. These intervals encode a fixed allocation preference and place selected entries before unselected entries within each member. Their spacing also affects competition between page-utility sums. Appendix A.2 proves heap optimality for nonnegative, nonincreasing optional page utilities under equal page cost and prefix constraints.

C.4 Decode-Only Adaptation

For AIME and LiveCodeBench (Table 2), compression applies to generated KV. At each compression event, attention scores use the most recent 64 actual decode queries over prompt and surviving generated KV, taking the maximum across queries and GQA heads. Evicted positions cannot re-enter the candidate set.

The semantic branch retains 512-coordinate residual sketches at representative layers and selects eight representatives from the recent 64 generated tokens. It uses the same centered cosine, maximum aggregation, radius-16 expansion, and layer broadcasting as prompt scoring. The global semantic quota is 0.5, with attention filling the remaining logical quota before the unchanged L2H2B16 compiler. This fraction specifies pre-compilation support, not the final compiled mixture.

C.5 Physical Backend

L2H2B16 groups two layers and two KV heads with 16 positions per member per page. Members share page IDs and allocated lengths but retain independent token IDs, sorted into original position order before packing. Keys keep their positional encoding. For head dimension d, scalar size s, and tile page size c, context payload is $2ds\,c\sum_g n_g$, including padding; metadata and decode growth receive separate reserves.

Requests transition through Dense→Packing→Sparse. Destination pages are reserved before a fused per-layer gather, and a CUDA event orders consumers after packing. Commit publishes sparse metadata only after scheduler validation of the dense block IDs; completion or cancellation releases the sidecar pages. Reservations are deducted from the dense allocator before admission. Tensor-parallel admission uses an all-rank vote and rollback; the measured TP1 path uses local acknowledgement.

Attention batches request–KV-head-group rows with separate page tables and lengths in one paged FlashAttention-2 call per layer. New KV enters sidecar append slots; persistent metadata updates lengths and page crossings. Mixed batches use explicit row mappings. The evaluated runtime uses synchronous eager execution, chunked prefill, one decode token per request per step, and disabled prefix caching.

C.6 Throughput Workload and Timing

Backend measurements use Qwen3-8B BF16 on the two-A100 node, with TP1 and eight CPU cores per run. Prefill length is $P = 4096$; output lengths are $G \in \{128, 512\}$ at batch sizes $B \in \{1, 4, 8, 16\}$ and $G = 2048$ at $B \in \{1, 4, 8\}$. EOS stopping is disabled. Each configuration has a full-length warmup and three measured batches; throughput is $3BG$ divided by their total duration.

The vLLM backends use 16-token pages, a 2,048-token prefill scheduling budget, GPU-memory utilization 0.84, and eager execution. PARSEE and Full KV use vLLM V1 with PyTorch 2.9/CUDA 12.8; TANGRAM and TriAttention use their native integrations. PARSEE retains actual L2H2B16 selections at 90% context compression, all 32 question tokens, and generated KV. Full KV (FA2) and Full KV (eager) use the same Transformers 5.2.0 stack with different attention backends.

KVzip uses Transformers 4.51.3, physical cache eviction, and variable-length FlashAttention-2; SnapKV uses Transformers 5.2.0 with window 64 and pooling kernel five. Both use PyTorch 2.6/CUDA 12.4 and FlashAttention 2.7.4. KVzip's compact metadata supports batch one; SnapKV supports batched decode. Unsupported configurations and out-of-memory cases have no throughput value.

The Backend clock includes prefill, packing or eviction, generation, and cache release, excluding loading, warmup, and precomputed scoring/selection for PARSEE and HF methods. TANGRAM and TriAttention retain native inline work. The separate HF Online clock includes scoring and selection, and each implementation has its own Full KV control. These clocks are measured independently. Scorer-inclusive common-runtime measurements are specified in Appendix F.7; they do not share the native-backend timing boundary.

D Results of the Motivating Experiments

D.1 Grouping and Shared-Quota Geometry

With support fixed, Figure 2 changes representation capacity from $1.355|E|$ under L2H2B16 to $1.023|E|$ under unconstrained count sorting, which excludes reorganization cost. This isolates the effect of grouping member counts.

Table 6 details the Qwen3-8B quota sweep under the settings in Appendix B. Moving from $\lambda = 0$ to 0.5 reduces synchronization from 62.64% to 34.47% and raises compiled recall from 80.80% to 89.54%. Figure 3 shows the same direction across the other evaluated models. Each quota declares a different support, so these geometry measurements do not rank answer quality. Page rounding remains separate from synchronization and is not minimized by the all-semantic endpoint.

Table 6: Support geometry across semantic quotas. Sync and Round denote synchronization and page-rounding overhead; Retained is compiled-support recall; Shell is the cutoff-neighborhood fraction.

Semantic quota	Sync (%)	Round (%)	B_{repr} (%)	Retained (%)	Shell (%)
0.00	62.641	1.045	163.686	80.805	0.000
0.25	49.050	1.041	150.091	85.041	0.161
0.50	34.468	**1.026**	135.494	89.540	0.120
0.75	<u>18.831</u>	<u>1.040</u>	<u>119.871</u>	<u>94.347</u>	0.170
1.00	**0.565**	1.297	**101.862**	**99.565**	0.152

Blue/orange: better/worse within a column; bold/underline: best/second distinct value. Recall bars: 0–100%. Shell is a diagnostic, without a preferred direction.

D.2 Semantic Cutoff Bound

For a complete tile, Theorem 1 bounds $D_g(S_C)$ by $(m-1)w_g$. Maximum pooling can create cutoff plateaus, so small η alone does not ensure small w_g. The Qwen3-8B five-quota probe satisfies the numerical count inequalities in all 18,000 tile–quota cases, with mean semantic synchronization 0.257% and mean shell fraction 0.120% at $\lambda = 0.5$.

Table 7 reports the multi-model diagnostic at $\alpha = 0.9, \lambda = 0.5$, using the definitions in Appendix B.2. P95 shell fractions are 0.24–0.42%; mean semantic synchronization is 0.17–0.31% of $|E|$, below the measured bounds of 0.52–0.88%. Mean aggregate tightness is 0.33–0.35, with no observed numerical violations. These checks concern semantic count mismatch, while mixed support also includes attention-driven disparity and completion includes rounding and competition.

Table 7: Semantic cutoff-bound diagnostics. Shell: neighborhood fraction (median/P95); U: upper bound; D: observed synchronization; Zero: zero-bound tiles; Viol.: numerical violations.

Model	Shell (%)	U (%)	D (%)	ρ_{agg}	Zero (%)	Viol.
Qwen3-8B	0.00 / 0.35	0.76	0.26	0.34	64.2	0
Qwen3-4B	0.00 / 0.38	0.79	0.27	0.34	62.8	0
Llama-3.1-8B-Instruct	0.00 / 0.42	0.88	0.31	0.35	59.5	0
Qwen3.8-27B	0.00 / 0.28	0.58	0.19	0.33	71.4	0
Qwen3.6-35B-A3B	0.00 / 0.24	0.52	0.17	0.33	74.0	0

Tint in U and D shows column-wise magnitude, not model quality.

E Detailed Main Results

E.1 Full Task Scores

Table 8 gives the full LongBench results underlying Figure 6. Table 9 extends the Qwen3-8B SCBench comparison to all evaluated baselines. Both follow the quality protocol and budgets in Appendix C.

Table 8: LongBench task-level scores (0–100). Avg. is the task macro.

Method	NQA	Qasper	MFQA	Hotpot	2Wiki	MuSiQue	GovRpt	QMSum	MultiNews	TREC	Trivia	SAMSum	Count	Retr.	LCC	Repo-P	Avg.
Qwen3-8B																	
Full KV	25.38	44.71	52.99	61.58	48.07	31.60	33.93	23.17	24.84	61.75	88.66	41.11	10.00	96.67	65.23	56.19	47.87
FastKVzip	23.78	**39.55**	**52.79**	**60.15**	**45.90**	28.81	**30.12**	**22.70**	**21.11**	**59.00**	88.91	39.01	9.50	56.33	62.08	57.33	43.57
TANGRAM+KD	**25.95**	29.22	38.74	55.23	43.06	25.55	22.08	21.26	13.81	39.75	88.75	38.24	9.50	95.50	25.59	**58.43**	39.42
TANGRAM+Snap	20.97	29.12	34.41	54.72	42.21	25.36	28.10	21.23	18.95	49.00	**89.16**	39.32	**10.00**	57.25	**62.83**	57.55	40.01
TriAttention	22.69	28.55	31.37	53.55	39.74	27.40	28.23	21.40	18.72	45.75	88.91	**39.50**	**10.00**	**95.92**	48.51	56.16	41.03
PARSEE	25.54	38.52	48.67	58.77	43.28	**30.50**	29.55	22.30	18.58	47.75	89.05	38.68	**10.00**	94.92	51.24	57.44	**44.05**
Qwen3-4B																	
Full KV	28.40	41.12	53.55	57.51	43.23	30.38	32.97	24.34	25.13	60.50	86.58	39.48	7.06	87.77	56.52	48.06	45.16
KVzip	24.88	38.51	**50.57**	**56.24**	**42.94**	**29.70**	**29.55**	**23.86**	23.51	**60.50**	**86.68**	37.95	6.11	68.78	**45.51**	48.37	42.10
TANGRAM+KD	24.79	25.37	35.40	50.68	38.83	24.61	19.67	21.39	14.01	39.25	86.12	37.70	6.61	75.92	21.77	47.01	35.57
TriAttention	24.79	28.49	30.34	44.63	35.63	19.69	26.48	21.73	20.51	38.00	86.22	38.01	5.23	62.62	45.02	47.46	35.93
PARSEE	**27.45**	**39.70**	49.70	54.78	38.65	28.13	27.29	23.58	**24.56**	55.00	86.58	**39.27**	**7.19**	**83.12**	42.71	**48.42**	**42.26**
Qwen3.6-35B-A3B																	
Full KV	33.23	51.29	59.62	68.41	71.46	51.07	31.03	22.74	24.39	79.50	92.57	42.11	21.00	100.00	70.30	65.82	55.28
KVzip	28.99	36.70	48.13	57.95	**66.72**	41.46	22.48	20.42	17.16	45.50	**93.03**	39.70	20.50	98.00	56.85	63.71	47.33
TANGRAM+KD	32.03	33.43	42.05	60.67	64.58	39.59	23.96	20.71	16.60	71.50	92.17	39.69	**23.00**	99.50	48.90	65.43	48.36
TriAttention	30.66	35.46	38.27	58.96	61.76	39.65	27.92	21.20	**20.30**	**76.00**	92.67	**41.93**	**23.00**	**100.00**	61.23	**66.51**	49.72
PARSEE	**32.52**	**43.68**	**56.84**	**66.20**	65.99	**49.69**	**28.95**	**22.66**	18.88	66.00	92.57	41.90	21.50	**100.00**	**65.60**	65.04	**52.38**
Qwen3.8-27B																	
Full KV	37.09	47.26	57.84	70.54	68.76	54.89	33.96	23.85	24.18	83.50	91.21	44.93	16.00	100.00	70.52	73.73	56.14
KVzip	34.17	37.04	49.33	65.37	65.01	48.58	23.37	21.74	16.67	67.50	**91.21**	43.42	16.00	99.00	59.66	71.91	50.62
TANGRAM+KD	35.35	33.41	38.81	60.70	60.20	43.90	23.38	21.36	15.81	70.00	91.05	42.83	**16.50**	**100.00**	48.80	72.09	48.39
TriAttention	34.52	32.26	35.97	58.53	57.38	46.22	27.34	21.95	**19.87**	79.00	91.05	44.55	**16.50**	**100.00**	62.62	**73.24**	50.06
PARSEE	**36.89**	**41.58**	**56.44**	**69.66**	**65.17**	**52.48**	**29.14**	**22.93**	19.05	**80.50**	**91.21**	**44.68**	16.00	**100.00**	**63.84**	71.76	**53.83**
Llama-3.1-8B-Instruct																	
Full KV	30.04	47.10	55.28	59.23	51.36	33.29	35.07	25.17	27.06	34.50	91.08	39.68	12.10	99.50	52.81	45.66	46.18
KVzip	**31.49**	45.19	**55.90**	**58.43**	**51.50**	**33.10**	31.48	**24.37**	26.44	**34.50**	91.06	38.80	**12.50**	93.00	**46.22**	**46.30**	45.02
TANGRAM+KD	31.43	33.02	47.74	56.26	47.35	30.23	26.40	23.49	21.26	29.00	**91.33**	**39.98**	10.59	88.50	25.19	45.19	40.43
TriAttention	29.79	31.55	37.13	54.86	46.50	26.08	25.74	22.87	22.99	14.00	91.11	37.90	10.79	98.50	46.18	45.94	40.12
PARSEE	30.26	**45.73**	55.00	58.20	50.42	32.60	**33.01**	24.29	**26.95**	**34.50**	91.26	38.69	12.15	**99.00**	46.13	46.09	**45.27**

Tint ranks compressed scores within each model/task; bold/underline: best/second (ties included). Gray: Full KV; cream: PARSEE. KD: KeyDiff; Snap: SnapKV.

E.2 Backend Throughput

Table 10 gives the full measurements for Figure 7(a–c) under Appendix C.6's Backend clock. PARSEE exceeds TANGRAM+KeyDiff and TriAttention in all 11 primary configurations, with a $1.036\times$ geometric-mean ratio over the faster competitor. Full KV is faster at $P = 4096, G = 512, B = 1$. OOM and unsupported cells are excluded from rankings.

Table 11 separates HF Backend, Online, and each implementation's Full KV control. Backend differences include runtime and software effects, and these HF Online rows do not rank PARSEE's complete online scorer. KVzip's all-kept comparison agrees on observed greedy tokens but reaches maximum KL 0.001826, above the 0.001 logit-fidelity criterion; its timing results therefore do not establish end-model logit equivalence.

F Supplementary Experiments

F.1 Component Ablations

The Qwen3-8B study uses 512 semantic coordinates and a 90% compression ratio on 202 questions from nine SCBench tasks, with 34 distinct contexts. Full KV retains all context entries. Complete

Table 9: Complete Qwen3-8B SCBench comparison at 90% compression ratio. Pre./Var./Mix./Chc./MShot/Sum./RQA denote prefix retrieval, variable tracking, mixed retrieval, choice, many-shot, summarization, and RepoQA. Avg. is the task macro; the final column is the point difference from Full KV.

Model	Method	KV	Pre.	Var.	Mix.	QA	Chc.	MShot	Sum.	RQA	Avg.	Δ
Qwen3-8B	Full KV	64.80	43.00	31.91	38.50	39.90	72.22	43.70	35.39	48.86	46.48	
	SnapKV	15.20	16.40	32.31	29.33	38.71	72.22	43.33	35.54	21.36	33.82	↓12.66
	AdaKV	47.90	21.05	27.64	32.40	31.10	66.85	38.10	30.22	35.31	36.73	↓9.75
	NestedKV	44.20	18.35	24.12	29.50	28.05	63.40	34.88	28.10	31.97	33.62	↓12.86
	ChunkKV	51.45	23.12	29.80	35.60	33.15	70.88	40.22	34.65	38.88	39.75	↓6.73
	TriAttention	0.00	0.00	30.62	31.17	30.11	**73.61**	43.33	31.45	3.86	27.13	↓19.35
	TANGRAM+Snap	0.00	0.00	23.69	27.00	34.28	**73.61**	43.70	32.55	2.27	26.35	↓20.13
	TANGRAM+KD	0.10	0.00	31.64	29.67	37.28	72.22	43.33	33.05	2.50	27.76	↓18.72
	KVzip	53.70	24.80	31.87	**37.00**	35.28	72.22	42.59	**35.72**	**42.50**	41.74	↓4.74
	FastKVzip	50.00	23.60	32.04	33.67	30.23	70.83	**45.19**	34.51	14.09	37.13	↓9.35
	PARSEE	**77.60**	**46.60**	**32.89**	27.17	**39.41**	**73.61**	42.59	35.55	30.00	**45.05**	↓1.43

Blue/orange cell tint compares compressed scores within a task. Bold/underline: best/second distinct compressed score, including ties. Δ: point decrease from Full KV.

Table 10: Backend throughput (output tokens/s). G: output length; B: batch size; KD: KeyDiff. Bold marks column maxima; – denotes unsupported configurations.

	$G = 128$				$G = 512$				$G = 2048$		
Method / B	1	4	8	16	1	4	8	16	1	4	8
Full KV (V1)	43.66	104.60	45.34	46.94	**49.63**	152.38	80.16	87.57	51.07	168.86	62.52
Full KV (eager)	18.51	47.46	OOM	OOM	21.33	62.12	OOM	OOM	22.05	61.31	OOM
Full KV (FA2)	23.56	74.68	104.23	OOM	25.84	98.32	158.20	OOM	26.94	84.02	169.52
KVzip (HF)	24.70	–	–	–	26.76	–	–	–	26.70	–	–
SnapKV (HF)	23.54	74.76	114.69	OOM	25.97	98.32	182.19	OOM	26.88	84.07	206.06
TANGRAM+KD	38.46	99.45	147.35	181.19	41.79	117.20	250.35	391.97	44.58	160.29	299.57
TriAttention	43.35	106.24	146.92	177.65	40.75	159.68	267.78	397.63	51.06	180.42	325.61
PARSEE	**44.84**	**110.92**	**152.22**	**184.73**	48.00	**163.26**	**273.38**	**406.94**	**52.30**	**183.01**	**330.03**

Higher is better within each workload column. Bold/underline: highest/second distinct throughput, including Full KV controls; tint follows that order. OOM: out of memory.

Table 11: HF throughput (output tokens/s) for fixed-selection Backend, Online, and Full KV controls. OOM marks preparation failures.

Method	G	B	Backend	Online	Full KV
KVzip	128	1	24.70	20.78	26.75
	512	1	26.76	25.46	28.96
	2048	1	26.70	26.38	29.08
SnapKV	128	1	23.54	23.43	23.56
	128	4	74.76	74.01	74.68
	128	8	114.69	111.09	104.23
	128	16	OOM	OOM	OOM
	512	1	25.97	25.89	25.84
	512	4	98.32	98.19	98.32
	512	8	182.19	179.70	158.20
	512	16	OOM	OOM	OOM
	2048	1	26.88	26.86	26.94
	2048	4	84.07	84.05	84.02
	2048	8	206.06	205.66	169.52

Backend bars share a 0–220 tokens/s scale. The three timing/control columns retain their distinct meanings.

PARSEE applies protected attention–semantic selection and physical compilation under the raw-context budget. The w/o physical projection row is its protected logical selector before page allocation. Anchor bonus replaces mandatory identifier-anchor retention with a score bonus; w/o identifier anchors removes those anchors while retaining the four sinks. Attention-only and semantic-only set $\lambda = 0$ and 1, respectively, while retaining protection; semantic-only also retains the bounded attention correction. Hard protection + L2H2B16 and Anchor bonus + L2H2B16 compile the corresponding anchor policies under page-rounded dense-capacity budgets. These are separate layout comparisons, not the complete PARSEE budget. Quality follows Appendix C.1 and is measured independently of allocator reclamation.

Table 12 reports the complete component comparisons summarized in Table 4. Scores average questions within rows, rows within tasks, and tasks equally; No MS excludes many-shot. Score differences use Full KV, whereas paired 95% intervals and Holm-adjusted tests compare with the logical reference. Intervals use 10,000 context-cluster bootstrap replicates and tests use cluster sign flips; none of the adjusted tests is significant. These tests use different contrasts from the shared-reference and position controls in Appendix F.2.

Removing anchors primarily affects short key–value retrieval. These diagnostic-subset ablations describe empirical score trends under the specified components and budgets; they do not establish that every component is statistically indispensable. The matched shared-reference and native-versus-shuffled controls provide separate evidence for semantic positional relevance; Table 3 summarizes their quality results, with full measurements and paired statistics in Table 13.

Table 12: Component-ablation scores and uncertainty. Δ is relative to Full KV; cluster confidence intervals and Holm-adjusted p compare with the logical reference.

Configuration	Score	Δ	95% cluster CI	Holm p	No many-shot
Full KV	45.79	-	-	-	-
PARSEE	**42.28**	**–3.51**	[–0.15, +8.46]	0.8784	42.41
PARSEE w/o physical projection	41.80	–3.99	—	—	**42.65**
Anchor bonus	40.97	–4.82	[–2.50, +0.00]	1.0000	41.71
Attention-only quota	39.09	–6.70	[–5.88, +0.24]	0.8742	38.98
Semantic-only quota	34.04	–11.75	[–13.09, –2.56]	0.1296	33.92
w/o identifier anchors	34.30	–11.49	[–8.89, –6.11]	0.8742	34.21
Hard protection + L2H2B16	41.47	–4.32	[–3.17, +2.30]	1.0000	42.28
Anchor bonus + L2H2B16	41.47	–4.32	[–3.17, +2.30]	1.0000	42.28

Tint follows the point estimate; bold/underline mark best/second distinct ablation values (Full KV excluded). Rankings do not imply statistical significance. CI: confidence interval.

F.2 Shared-Reference and Position Controls

We compare reference sources on Qwen3-8B and Llama-3.1-8B-Instruct at $\alpha = 0.9$, $\lambda = 0.5$, and $B = K$, with fixed inputs, broadcast units, protection, layout, and generation. Attention-only sets $\lambda = 0$ and therefore also changes the quota.

Matched references and position control. The shared-attention reference reuses the native broadcast units from Equation 13:

$$H_g^{\mathrm{att}}(t) = \frac{1}{|\mathcal{B}(g)|} \sum_{h \in \mathcal{B}(g)} P_h(t). \tag{14}$$

We compare it with the post-pooling semantic base. Each base is standardized using its mean μ_F and population standard deviation s_F over free positions:

$$H_g^{\mathrm{match}}(t) = \frac{H_g(t) - \mu_F}{d_F}, \qquad d_F = \begin{cases} s_F, & s_F > 0, \\ 1, & s_F = 0. \end{cases} \tag{15}$$

The transformation applies to all context positions; temperature is one. The perturbation and raw-context softmax domain remain unchanged, and quota selection uses FP32 post-softmax scores.

Native PARSEE remains unstandardized. The shuffled control permutes native post-pooling logits only at free positions, before perturbation, using one permutation per actual broadcast unit. It reports the equal-weight mean over three shuffles, with protection, attention, quotas, and compiler unchanged.

Metrics and paired analysis. Pre-compilation counts give $D_C = 100 \sum_g D_g(S_C)/|E|$ and $D_E = 100 \sum_g D_g(E)/|E|$. Support retention is

$$R_B = \frac{|E \cap S_B|}{|E|}, \qquad R_{\text{free}} = \frac{|(E \setminus A) \cap (S_B \setminus A)|}{|E \setminus A|}. \tag{16}$$

Each row declares its own E; these metrics do not measure common true evidence. Quality intervals use 10,000 paired raw-context cluster bootstrap replicates, moving repeated-context questions together and preserving task-macro weights. The two contrasts across two models and two benchmarks form eight cluster sign-flip tests with Holm correction at familywise 0.05. The displayed 95% intervals are pointwise.

Only cost uses 4K context, one A100-80GB, and batch one. Reference time includes feature capture, representative selection, transfers, and reference construction; full scoring also includes attention acquisition and score normalization. They are measured separately using five warmups and twenty synchronized repeats per timing input. M is peak scoring workspace memory, excluding weights and KV storage. These measurements do not substitute for online serving throughput.

Results. Table 13 shows matched-semantic LongBench/SCBench gains of 1.47/1.76 points on Qwen and 1.35/1.62 on Llama, with significant Holm-adjusted contrasts. Full scoring costs 4.6 ms and 40 MiB more on each model. Shuffling reduces quality by 7.25/9.90 and 7.17/10.73 points while changing mixed synchronization by only 0.43/0.47 percentage points. The controls support useful position relevance beyond shared coordination, without requiring identical geometry or establishing cosine as uniquely effective.

F.3 Scoring Sensitivity

We vary coordinate dimensions 256/512/1024, representative counts 4/8/16, pooling radii 0/8/16/32, and radius-16 mean pooling around the native configuration. The physical budget, quota, protection, and compiler remain fixed. The dimension diagnostic fixes native representative-question indices to isolate coordinate effects. Full-D uses all residual coordinates with the same scorer and evaluation subset, rather than an importance oracle.

Quality and overlap metrics. Table 15 uses the 50-input Qwen3-8B development cohort in Appendix B. Its quality difference $\Delta Q(v) = Q(v) - Q(v_0)$ is the SCBench portion's task macro relative to native, with paired pointwise 95% intervals; geometry and overlaps use the full cohort. For Jaccard overlap $\mathcal{J}(X, Y) = |X \cap Y|/|X \cup Y|$, define

$$\begin{aligned} J_C(v) &= \mathcal{J}(S_C(v), S_C(v_0)), \\ J_E(v) &= \mathcal{J}(E(v) \setminus A, E(v_0) \setminus A), \\ J_B(v) &= \mathcal{J}(S_B(v) \setminus A, S_B(v_0) \setminus A). \end{aligned} \tag{17}$$

An empty union has overlap one. Sample-averaged J_E/J_B compare the complete free support before and after compilation. Shell follows Appendix B.2; cost uses the separate 4K protocol in Appendix F.2.

Table 14 summarizes cross-model, per-domain extrema over the scoring variants. Different extrema can come from different variants and cannot be combined into a single run. Its full-scoring time ratio is distinct from reference-branch time in Table 15.

Results. At 256 coordinates, reference time falls from 8.5 to 5.8 ms with 0.40–0.48-point subset losses. At 1024 coordinates, point gains of 0.11–0.16 have intervals containing zero and cost 13.2 ms; Full-D costs 38.5 ms for $+0.28$ $[-0.12, +0.68]$ points. Four representatives lose 0.85 points, while sixteen cost 12.8 ms for a point gain of 0.08. Removing pooling loses 1.35 points despite narrowing the P95 shell from 0.35% to 0.04%; radius 32 widens it to 1.15% without improving quality. The defaults are tested cost tradeoffs, with greater domain sensitivity on retrieval and code than summarization; intervals containing zero do not establish equivalence.

Table 13: Shared-reference and position controls. Panel (a) reports benchmark quality, pre-compilation synchronization, compiled support retention, and scoring cost. Only latency and memory use a fixed 4K context on one A100-80GB GPU, batch size one; quality and geometry retain their long-context evaluation populations. Panel (b) reports paired quality differences with 95% intervals.

(a) Configuration comparison

Configuration	LB / SC	D_C / D_E (%)	R_B / R_{free} (%)	$t_{\text{ref}} / t_{\text{score}}$ (ms)	M (MiB)
Qwen3-8B					
Attention-only quota	41.20 / 39.09	0.00 / 62.64	80.80 / 78.45	0.0 / 18.2	92
Shared attention, matched	42.35 / 43.12	0.42 / 36.85	88.68 / 87.40	4.1 / 20.3	108
Shared semantics, matched	43.82 / 44.88	0.28 / 34.62	89.42 / 88.20	8.6 / 24.9	148
Native PARSEE	**44.05 / 45.05**	0.26 / 34.47	89.54 / 88.35	8.5 / 24.8	148
Shuffled native semantics	36.80 / 35.15	0.28 / 34.90	89.20 / 87.95	8.5 / 24.8	148
Llama-3.1-8B-Instruct					
Attention-only quota	42.50 / 45.80	0.00 / 61.85	81.50 / 79.20	0.0 / 19.5	96
Shared attention, matched	43.75 / 49.10	0.48 / 37.10	89.15 / 88.02	4.3 / 21.6	112
Shared semantics, matched	45.10 / 50.72	0.32 / 34.80	90.02 / 88.90	9.1 / 26.2	152
Native PARSEE	**45.27 / 50.93**	0.31 / 34.65	90.10 / 89.01	9.0 / 26.1	152
Shuffled native semantics	38.10 / 40.20	0.33 / 35.12	89.80 / 88.65	9.0 / 26.1	152

(b) Paired contrasts

Model	Contrast	ΔLB [95% CI]	ΔSC [95% CI]
Qwen3-8B	Matched semantics − attention	+1.47 [+0.65, +2.29]*	+1.76 [+0.82, +2.70]*
	Native − shuffle mean	+7.25 [+5.80, +8.70]*	+9.90 [+8.45, +11.35]*
Llama-3.1-8B	Matched semantics − attention	+1.35 [+0.52, +2.18]*	+1.62 [+0.70, +2.54]*
	Native − shuffle mean	+7.17 [+5.62, +8.72]*	+10.73 [+9.10, +12.36]*

$\lambda = 0.5$ except attention-only (0); semantic quotas use post-softmax scores. Shuffle reports the equal-weight mean over three shuffles. D_C, D_E are normalized by each row's logical support size; both recall metrics refer to that row's declared support. M is peak scoring workspace memory. Reference and full scoring are timed separately. * denotes significance after Holm–Bonferroni correction over the eight tests in panel (b), at familywise $\alpha = 0.05$; the intervals are paired 95% intervals.

LB/SC: LongBench/SCBench. Quality bold/underline: best/second distinct point estimate within each model. Tint follows SC; recall cells are lightly shaded. Workspace bars: 0–160 MiB.

Table 14: Semantic-signal sensitivity across models and domains. Descriptive extrema over variants: columns report worst quality change, minimum support overlaps, and maximum scoring-time ratio. These extrema may come from different variants. Table 15 separately pairs Qwen3-8B development-subset quality, geometry, and reference cost by variant. QA denotes question answering.

Model	Domain	$\min \Delta Q$	$\min J_C$	$\min J_E$	$\min J_B$	$\max(t_{\text{score}}/t_0)$
Qwen3-8B	Retrieval	−4.8	0.72	0.78	0.76	1.28
	QA	−3.5	0.75	0.80	0.79	1.28
	Summary	−1.8	0.81	0.84	0.83	1.26
	Code	−4.2	0.70	0.77	0.75	1.28
Qwen3-4B	Retrieval	−5.2	0.70	0.76	0.75	1.27
	QA	−3.8	0.73	0.79	0.77	1.27
	Summary	−2.1	0.79	0.83	0.82	1.25
	Code	−4.6	0.69	0.75	0.74	1.27
Llama-3.1-8B-Instruct	Retrieval	−4.5	0.73	0.79	0.77	1.30
	QA	−3.2	0.76	0.81	0.80	1.30
	Summary	−1.9	0.80	0.85	0.83	1.28
	Code	−4.0	0.71	0.78	0.76	1.30
Qwen3.8-27B	Retrieval	−3.6	0.76	0.82	0.81	1.25
	QA	−2.8	0.78	0.84	0.83	1.25
	Summary	−1.5	0.83	0.87	0.86	1.22
	Code	−3.1	0.74	0.80	0.79	1.25

Tint shows relative magnitude within a column; blue: smaller loss, larger overlap, or lower time ratio. Extrema need not come from the same variant.

Table 15: Per-variant sensitivity on Qwen3-8B over 50 long-context development inputs. ΔQ is the paired SCBench subset macro-score difference from native, with a 95% interval; overlaps and geometry use the 50-input cohort. Full-D and its native reference use this same diagnostic subset. Latency and memory follow the separate fixed-4K protocol in Appendix F.2.

Variant	ΔQ [95% CI]	$J_C/J_E/J_B$	D_C/D_E (%)	t_{ref} (ms)	Shell P95 (%)	M (MiB)
D256/1	−0.42 [−0.85, +0.01]	0.88 / 0.93 / 0.92	0.28 / 34.60	5.8	0.36	116
D256/2	−0.48 [−0.92, −0.04]	0.87 / 0.92 / 0.91	0.29 / 34.65	5.8	0.36	116
D256/3	−0.40 [−0.83, +0.03]	0.88 / 0.93 / 0.92	0.27 / 34.58	5.8	0.35	116
D512/1 (native)	0.00 [ref.]	1.00 / 1.00 / 1.00	0.26 / 34.47	8.5	0.35	148
D512/2	−0.05 [−0.35, +0.25]	0.96 / 0.98 / 0.97	0.26 / 34.48	8.5	0.35	148
D512/3	+0.02 [−0.28, +0.32]	0.96 / 0.98 / 0.98	0.25 / 34.46	8.5	0.34	148
D1024/1	+0.14 [−0.18, +0.46]	0.91 / 0.95 / 0.94	0.24 / 34.38	13.2	0.33	210
D1024/2	+0.11 [−0.20, +0.42]	0.91 / 0.95 / 0.94	0.24 / 34.40	13.2	0.33	210
D1024/3	+0.16 [−0.15, +0.47]	0.92 / 0.95 / 0.95	0.23 / 34.36	13.2	0.32	210
Q4	−0.85 [−1.42, −0.28]	0.81 / 0.88 / 0.87	0.31 / 34.82	6.2	0.38	132
Q16	+0.08 [−0.24, +0.40]	0.94 / 0.97 / 0.96	0.24 / 34.35	12.8	0.33	176
R0	−1.35 [−2.10, −0.60]	0.74 / 0.84 / 0.83	0.21 / 34.12	8.1	0.04	148
R8	−0.32 [−0.75, +0.11]	0.89 / 0.94 / 0.93	0.24 / 34.32	8.3	0.18	148
R32	−0.52 [−1.05, +0.01]	0.83 / 0.90 / 0.89	0.38 / 35.10	8.8	1.15	148
Mean16	−0.68 [−1.22, −0.14]	0.82 / 0.89 / 0.88	0.30 / 34.75	8.5	0.42	148
Full-D (subset)	+0.28 [−0.12, +0.68]	0.86 / 0.92 / 0.91	0.22 / 34.25	38.5	0.31	420

D: coordinate dimension; numbered suffixes distinguish variants at the same dimension; Q: representative count; R: max-pooling radius; Mean16: radius-16 mean pooling. Dimension rows fix native question indices; all rows retain the native post-softmax selector. Quality uses the cohort's SCBench portion; ΔQ is not a full-benchmark score difference. Jaccard overlaps are sample averages, and J_E/J_B exclude protection. M is peak scoring workspace memory. Intervals are paired and pointwise, without a non-inferiority claim.

Quality tint: blue gain / orange loss relative to native; it does not encode significance. Workspace bars: 0–450 MiB.

F.4 Perturbation Diagnostics

The sweep compares $\eta \in \{0, 10^{-6}, 10^{-5}, 10^{-4}\}$ on Qwen3-8B and Llama-3.1-8B-Instruct with the native FP32 softmax and top-r selector. For computed scores $\widehat{C}_h^{(\eta)}(t)$ and free layer–head–position entries F, the bitwise change rate is

$$\mathrm{Chg}(\eta) = \frac{100}{|F|} \sum_{(h,t)\in F} \mathbf{1}\Big[\mathrm{bits}\Big(\widehat{C}_h^{(\eta)}(t)\Big) \neq \mathrm{bits}\Big(\widehat{C}_h^{(0)}(t)\Big)\Big]. \tag{18}$$

Reorder uses adjacent pairs in sorted baseline free scores that are tied or separated by at most one ULP, reporting changes in their ordering relation. This fixed pair set, rather than all token pairs, is its denominator. Stable index order enumerates diagnostic ties without changing selection. Overlaps use native 10^{-5} as reference; D_C/D_E are absolute synchronization percentages. Same/20 counts the modal selection-mask output over twenty executions with fixed scoring inputs and environment.

Table 16 separates widespread score changes from limited selection changes. Removing the perturbation retains $J_C/J_E/J_B = 0.97/0.99/0.98$ on both models. Increasing it to 10^{-4} raises reorder rates to 4.25%/4.60% and P95 shell fractions to 0.82%/0.96%. Both 10^{-5} and 10^{-4} reach 20/20 in the repeatability diagnostic. These numerical observations establish neither task-quality gains nor necessity or cross-device reproducibility of the exact default.

Table 16: Perturbation diagnostics for Qwen3-8B and Llama-3.1-8B-Instruct at native precision. Chg is the bitwise change rate of FP32 post-softmax free scores versus $\eta = 0$; Reorder uses the fixed tie/near-tie pair set in Appendix F.4. Only overlaps use native 10^{-5} as reference. D_C, D_E are absolute synchronization percentages; Shell follows Appendix B.2. Same/20 counts the modal selection-mask output among 20 executions with fixed scoring inputs.

η	Chg (%)	Reorder (%)	J_C	J_E	J_B	D_C (%)	D_E (%)	Shell P95 (%)	Same/20
Qwen3-8B									
0	0.00	0.00	0.97	0.99	0.98	0.25	34.45	0.12	16 / 20
10^{-6}	41.25	0.18	0.99	1.00	0.99	0.25	34.46	0.24	19 / 20
10^{-5} (native)	84.50	0.82	1.00	1.00	1.00	0.26	34.47	0.35	20 / 20
10^{-4}	98.20	4.25	0.94	0.97	0.96	0.32	34.72	0.82	20 / 20
Llama-3.1-8B-Instruct									
0	0.00	0.00	0.97	0.99	0.98	0.30	34.60	0.14	15 / 20
10^{-6}	39.80	0.21	0.99	1.00	0.99	0.30	34.62	0.28	19 / 20
10^{-5} (native)	82.10	0.88	1.00	1.00	1.00	0.31	34.65	0.42	20 / 20
10^{-4}	97.90	4.60	0.93	0.96	0.95	0.38	34.95	0.96	20 / 20

Blue: larger overlap; orange: more changed/reordered scores. These diagnostics do not rank answer quality. Same/20 retains the modal-count denominator.

F.5 Semantic Quota

At $\alpha = 0.9$, we sweep $\lambda \in \{0, 0.25, 0.5, 0.75, 1\}$ on Qwen3-8B, Llama-3.1-8B-Instruct, and Qwen3.8-27B with identical inputs, protection, generation, and raw-context budgets within each model. Each quota defines its own compiled support. Table 17 reports benchmark macros with 95% intervals, the worst task-level difference from $\lambda = 0.5$, and recall of each quota's support.

The default has the highest benchmark point estimates on all three models, while compiled recall increases toward the semantic-only endpoint. For Qwen3-8B, recall rises from 89.54% to 99.56% between $\lambda = 0.5$ and 1, but LongBench/SCBench falls from 44.05/45.05 to 36.50/34.04. Shared coordination therefore benefits from complementary attention evidence. The benchmark intervals do not establish paired equivalence or a statistically optimal quota.

F.6 Quality and Online Serving

Table 18 combines SCBench quality with a separate Qwen3-8B serving workload across $\alpha \in \{0.8, 0.9, 0.95\}$ and $\lambda \in \{0.25, 0.5, 0.75\}$. Serving uses 4,096 prefill tokens, exactly 512 output

Table 17: Quality and compiled-support recall across semantic quotas. Benchmark macros have 95% intervals; Worst Δ is the minimum task-level difference from $\lambda = 0.5$.

λ	LongBench		SCBench		Worst Δ	Recall (%)
Qwen3-8B						
0.00	41.20	[39.1, 43.3]	39.09	[36.8, 41.4]	−6.80	80.80
0.25	<u>43.85</u>	[41.8, 45.9]	<u>44.70</u>	[42.6, 46.8]	−2.10	85.04
0.50	**44.05**	[42.1, 46.0]	**45.05**	[43.0, 47.1]	0.00	89.54
0.75	42.60	[40.5, 44.7]	42.15	[40.0, 44.3]	−3.90	94.35
1.00	36.50	[34.2, 38.8]	34.04	[31.5, 36.6]	−12.40	99.56
Llama-3.1-8B-Instruct						
0.00	42.50	[40.3, 44.7]	45.80	[43.4, 48.2]	−6.10	81.50
0.25	<u>45.10</u>	[43.0, 47.2]	<u>50.40</u>	[48.1, 52.7]	−1.80	85.80
0.50	**45.27**	[43.2, 47.3]	**50.93**	[48.6, 53.2]	0.00	90.10
0.75	43.80	[41.7, 45.9]	47.90	[45.5, 50.3]	−3.70	94.60
1.00	37.90	[35.6, 40.2]	39.50	[37.0, 42.0]	−13.20	99.60
Qwen3.8-27B						
0.00	51.10	[49.2, 53.0]	73.20	[71.1, 75.3]	−5.20	90.07
0.25	<u>53.60</u>	[51.7, 55.5]	<u>76.80</u>	[74.8, 78.8]	−1.20	92.20
0.50	**53.83**	[51.9, 55.7]	**77.26**	[75.3, 79.2]	0.00	94.56
0.75	52.90	[51.0, 54.8]	75.40	[73.4, 77.4]	−2.60	97.10
1.00	46.20	[44.1, 48.3]	66.50	[64.2, 68.8]	−11.90	99.70

Bold/underline: best/second benchmark point estimate within each model, without a significance claim. Cream marks the default quota; recall bars span 0–100% and refer to each quota's own support.

tokens with EOS stopping disabled, and concurrency one or eight. Its online clock includes prefill, scoring, selection, compilation, packing, and decoding. Quality follows the benchmark protocol; it is not measured on synthetic timing outputs.

TPS divides generated tokens by workload duration. TTFT P95 is request-level; ITL P95 pools adjacent output-token intervals. KV pool reports peak allocated generation-phase cache after compaction and reclamation, including metadata, reserves, alignment, and padding, but excluding dense prefill and scoring workspace. With fixed α, allocated capacity is unchanged across quotas.

At concurrency one, Full KV has the highest quality and throughput. At concurrency eight, $\alpha = 0.9, \lambda = 0.5$ reaches 273.4 versus 80.2 tokens/s for Full KV, with SCBench 45.1 versus 46.5 and allocated pool 1.68 versus 9.20 GiB. More aggressive compression increases throughput while reducing quality. These operating points concern the stated online workload and remain separate from the precomputed-selection backend clock.

F.7 Additional Systems Measurements

These Qwen3-8B BF16 experiments extend the native-backend workloads, measure scorer-inclusive latency in a common runtime, and compare storage representations.

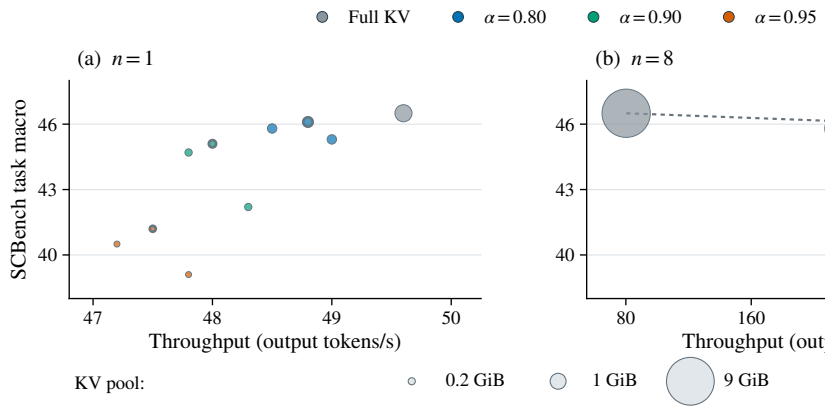
F.7.1 Backend Workload Sensitivity

The additional workloads follow Appendix C.6's Backend clock. Table 19 reports longer-prefill workloads and repeated measurements of two primary settings. Each uses three batches with $G = 512$. The repeated settings give 1.92% and 2.11% gains over the faster competitor, illustrating the scale of variation around small backend differences. They are reported separately from the primary aggregate, without a significance claim.

F.7.2 Prefill, Compression, and Generation Latency

The latency study uses two LongBench contexts near each of 4K, 8K, 16K, and 32K raw tokens, with their actual questions, and two worker runs on the two-A100 node. Each worker balances model layers across both GPUs. A batch contains four independent copies of an input with separate caches

Table 18: SCBench quality and online serving performance. Q is the task macro; TPS is output tokens/s; TTFT/ITL are P95 latencies (ms); KV pool is peak allocated cache capacity during generation (GiB).



α	λ	Q	TPS	TTFT	ITL	KV pool
Concurrency $n = 1$						
Full KV	n/a	**46.5**	**49.6**	**85**	**19.8**	1.15
0.80	0.25	45.8	48.5	110	20.2	0.35
0.80	0.50	46.1	48.8	108	20.1	0.35
0.80	0.75	45.3	49.0	107	20.0	0.35
0.90	0.25	44.7	47.8	114	20.5	0.21
0.90	0.50	45.1	48.0	112	20.4	0.21
0.90	0.75	42.2	48.3	111	20.3	0.21
0.95	0.25	40.5	47.2	118	20.8	0.14
0.95	0.50	41.2	47.5	116	20.7	0.14
0.95	0.75	39.1	47.8	115	20.6	0.14

α	λ	Q	TPS	TTFT	ITL	KV pool
Concurrency $n = 8$						
Full KV	n/a	**46.5**	80.2	380	98.5	9.20
0.80	0.25	45.8	215.4	148	36.2	2.80
0.80	0.50	46.1	228.6	142	34.1	2.80
0.80	0.75	45.3	234.0	139	33.5	2.80
0.90	0.25	44.7	258.1	135	30.2	1.68
0.90	0.50	45.1	273.4	130	28.6	1.68
0.90	0.75	42.2	280.2	128	27.9	1.68
0.95	0.25	40.5	291.5	125	26.8	1.12
0.95	0.50	41.2	304.8	121	25.5	1.12
0.95	0.75	39.1	**311.2**	**119**	**25.0**	1.12

Scatter panels use different throughput ranges. Marker area represents KV pool on a common scale; thick outlines mark $\lambda = 0.5$. Dotted segments connect the nondominated displayed Q–TPS points within $n = 8$; only Full KV is nondominated at $n = 1$. Bold/underline: best/second distinct table value within each concurrency; blue/orange: better/worse. KV bars: 0–10 GiB. Quality and timing use separate workloads.

Table 19: Additional backend workloads and repeatability measurements (output tokens/s). The upper block varies prefill length; the lower block repeats two primary configurations.

P	B	TANGRAM+ KD	TriAttention	PARSEE
Longer-prefill workloads				
8192	1	38.92	45.70	**47.83**
8192	4	123.93	130.51	**134.43**
16384	1	37.18	41.33	**41.65**
Repeated configurations				
4096	1	43.45	49.94	**50.90**
4096	4	147.05	161.16	**164.55**

Bold/underline: highest/second throughput per workload.
P: prefill tokens; B: batch size; KD: KeyDiff.

and selections. All policies compress raw context by 90%, preserving prefix, question, and generated KV; PARSEE also enforces the physical page cap.

Each input–method configuration has a 32-token warmup and three repetitions, with rotating method order. Generation produces 512 tokens, including the first token charged to question prefill. The synchronized wall clock includes context/question prefill, online scoring, selection, packing, and decode; it excludes loading, tokenization, warmup, and serialization. Dense source KV remains allocated. LongBench-input table cells average the two inputs, three repeats, and two workers. Score denotes additional post-prefill scoring: inline scoring is included in prefill and is not cost-free.

Table 20: Common-runtime prefill and compression latency (seconds).

Method	4K	8K	16K	32K
Full KV	**2.87**	**4.72**	**10.84**	**27.15**
PARSEE	6.27	5.41	13.37	28.92
KVzip	7.53	14.49	33.28	81.10
TANGRAM+KeyDiff	3.34	5.55	12.32	30.07
TANGRAM+SnapKV	3.31	5.49	12.25	30.10
TriAttention	4.08	6.89	15.25	35.78

Lower is better per context bucket. Bold/underline: lowest/second distinct latency, including Full KV; tint follows the same order.

Table 21: Batch latency decomposition for the 32K context bucket (seconds).

Method	Context	Question	Score	Select	Pack	Decode	Other	Stack
Full KV	25.933	1.214	0.000	0.000	0.000	48.644	0.002	
PARSEE	25.091	0.875	1.699	1.209	0.041	27.813	0.144	
KVzip	25.877	1.154	53.296	0.735	0.041	28.890	0.095	
TANGRAM+KeyDiff	28.013	1.201	0.000	0.822	0.037	28.888	0.080	
TANGRAM+SnapKV	28.152	1.206	0.000	0.710	0.037	28.804	0.086	
TriAttention	25.933	1.212	7.844	0.750	0.037	28.954	0.080	

Stacks use one 0–120 s scale and the column order/colors. Exact component times remain shown; inline scoring is charged to prefill.

Table 22: Complete batch latency in the common runtime (seconds).

Method	4K	8K	16K	32K
Full KV	**30.86**	**32.76**	**42.04**	75.79
PARSEE	35.06	34.23	43.24	**56.87**
KVzip	36.28	43.34	63.17	110.09
TANGRAM+KeyDiff	32.01	34.28	42.54	59.04
TANGRAM+SnapKV	31.96	34.20	42.05	58.99
TriAttention	32.81	35.67	45.53	64.81

Lower is better per context bucket; bold/underline: lowest/second distinct latency, including Full KV. Bars: 0–120 s.

At 32K, summing PARSEE's phases in Table 21 gives 56.87 seconds, versus 58.99 for TANGRAM+SnapKV and 75.79 for Full KV: reductions of 3.6% and 25.0%, respectively. PARSEE spends 1.699 seconds on scoring and 1.250 seconds on selection and packing; prefill plus compression totals 28.92 seconds. Its context prefill and decode take 25.091 and 27.813 seconds, compared with 28.152 and 28.804 for TANGRAM+SnapKV, offsetting the additional scoring work. TANGRAM+SnapKV has lower complete latency in the other three buckets. These are descriptive means under a shared runtime, not native-serving throughput.

The fixed-length sweep uses total prefill $P \in \{4096, 8192, 16384, 32768\}$, question length $Q \in \{32, 128\}$, output length $G \in \{512, 2048\}$, and batch four; raw context has $P - Q$ tokens. Table 23

reports three repetitions in each worker run per configuration. PARSEE is 0.06–2.06% slower than the faster TANGRAM mean in this sweep. Its stronger benchmark quality (Tables 8 and 9) and these latency measurements describe a quality–cost tradeoff across the stated workloads.

Table 23: Complete batch latency on fixed-length inputs (seconds). P, Q, and G denote total prefill, question, and generated lengths; T-Key/T-Snap denote TANGRAM+KeyDiff/SnapKV.

P	Q	G	PARSEE	T-Key	T-Snap
4096	32	512	31.836	31.675	**31.674**
4096	32	2048	119.254	118.618	**118.594**
4096	128	512	32.211	31.621	**31.561**
4096	128	2048	119.901	118.660	**118.594**
8192	32	512	34.516	34.522	**34.448**
8192	32	2048	121.555	121.609	**121.432**
8192	128	512	34.825	34.535	**34.391**
8192	128	2048	122.153	**121.601**	121.648
16384	32	512	41.059	41.159	**41.032**
16384	32	2048	128.881	128.806	**128.468**
16384	128	512	41.293	41.122	**41.019**
16384	128	2048	128.926	128.992	**128.758**
32768	32	512	58.771	57.856	**57.779**
32768	32	2048	145.109	**144.096**	144.583
32768	128	512	58.561	58.071	**57.805**
32768	128	2048	144.285	143.905	**143.482**

Bold/underline: lowest/second latency within each row; tint follows this order. All three decimal places are retained.

F.7.3 Storage Geometry and Attention Replay

For each policy's selections on the eight inputs, we compare compact per-head banks, per-head pages, uniform-capacity head pages, and policy-group pages. Uniform capacity reserves the largest head page count for every head. Groups use paired heads for PARSEE, calibrated groups for TANGRAM, individual heads for KVzip, and all heads for TriAttention. Within a policy, all representations retain identical token identities.

The study samples layers 0, 17, and 35 at batch sizes 1, 4, 8, and 16, with BF16 Q/K/V, head dimension 128, and 16-token pages. Table 24 aggregates payload over inputs and sampled layers at batch one; it excludes model weights, metadata, and allocator reserves. PARSEE saves 13.08% relative to its uniform-capacity representation. Selected payloads differ across policies, so cross-row memory ratios do not measure equal-quality efficiency.

Table 24: KV representation overhead for fixed selections within each policy. Saving is relative to the uniform-capacity representation.

Policy	Selected (MiB)	Group/selected	Uniform/selected	Saving
PARSEE	181.77	1.0000	1.1505	13.08%
KVzip	114.79	1.0064	2.2840	55.94%
TANGRAM+KeyDiff	143.06	1.0000	1.7899	44.13%
TANGRAM+SnapKV	143.06	1.0000	1.3517	26.02%
TriAttention	143.34	1.0059	1.0059	0.00%

Saving bars use 0–100% relative to the uniform-capacity representation within each policy. Selected payloads differ across policies.

Kernel timing uses 20 warmups and seven repetitions of 100 calls, measured by CUDA events. Table 25 reports graph replay with the dense source released and medians over input–layer pairs. Compact attention is faster than grouped pages in the reported cells; single-layer replay does not determine full-model serving throughput. Outputs are compared with FP32 QK–softmax–V on the

selected indices at absolute and relative tolerance 0.02. These checks and active-allocation release are distinct from scheduler reclamation.

Table 25: Attention-kernel replay time (μs). Each cell reports compact / policy-group-paged attention.

Policy	$B = 1$	$B = 4$	$B = 8$	$B = 16$
PARSEE	**14.09** / 19.85	**27.38** / 33.30	**41.84** / 49.25	**131.28** / 137.60
KVzip	**13.31** / 13.84	**18.01** / 18.64	**23.21** / 24.13	**37.24** / 39.02
TANGRAM+KeyDiff	**14.64** / 20.14	**26.08** / 31.32	**40.25** / 46.26	**94.76** / 96.80
TANGRAM+SnapKV	**14.19** / 19.38	**25.01** / 30.28	**37.03** / 43.34	**88.55** / 94.81
TriAttention	**13.66** / 14.47	**23.42** / 28.71	**34.51** / 41.05	**83.81** / 90.72

Bold/underline: lower/higher time within each compact/paged pair. Orange grows with the paged/compact ratio; policy rows use different selections.

G Additional Related Work and Implementation Discussion

Compaction and heterogeneous allocation. PagedAttention maps logical request blocks to reusable physical blocks, while FlashAttention reduces attention memory traffic through tiling and fused computation (Kwon et al., 2023; Dao et al., 2022; Dao, 2023). TriAttention estimates importance from pre-RoPE query/key geometry and consolidates survivors in its vLLM integration, including order-preserving repacking and hole filling (Mao et al., 2026b;a). Its reported uniform head budget permits different token identities at a shared length. TANGRAM groups heads using calibrated budgets and ragged page tables (Kim et al., 2026a). PARSEE instead coordinates input-dependent member counts during selection and analyzes their effect on physical support completion.

DiffKV combines mixed precision, token-importance tiers, and per-head sparsity with parallel compaction (Zhang et al., 2025). CHESS aggregates keys across heads and layers and selects grids, chunks, and pages (Fei et al., 2026). GraceKV uses prototype trees and dependent Add/Split operations to allocate coverage and resolution (Tian et al., 2026). PARSEE's analysis concerns original-vector storage in fixed page groups; its rank, synchronization, rounding, and competition terms describe that representation.

Restoration beyond selection. RestoreKV supplements retained entries with a compact restore cache from a context-conditioned, LoRA-adapted pass, charging those states to the total budget (Baek et al., 2026). PARSEE retains original KV vectors and coordinates their allocation using the available question. Reconstruction of discarded content and selection of original entries address different representation choices and may be complementary.

G.1 Selection, Compact Storage, and Reclamation

Logical exclusion, compact attention, and reusable allocator capacity are distinct capabilities. A mask can exclude entries without changing cache allocation. Gathering survivors into variable-length head banks reduces payload and can process heads in one fused attention call, but multi-request serving also needs request-specific lengths, cache growth, and ownership management.

PARSEE's grouped pages provide independent request–head-group lengths and retain independent token identities within each shared page shape. The commit protocol in Appendix C.5 makes released dense blocks available to the scheduler. These capabilities distinguish physical deployment from mask-based quality evaluation; they do not imply that compact head banks are inherently serial or unable to support batching.

Question-attention acquisition remains additional work before compression. Accordingly, the native Backend clock, scorer-inclusive common-runtime latency, and task-quality protocols are reported separately in Appendices C.6, F.7, and C.1.

H Discussion, Limitations, and Future Work

H.1 Discussion: Successful Retrieval Cases

The KV and prefix–suffix retrieval gains in Table 1 concern two requirements: selecting the correct record and reproducing its complete contents. Table 26 illustrates both with four Qwen3-8B examples at 90% compression. Each is an observed PARSEE success and KVzip failure from the existing evaluation, using the same context, question, generation limit, and first output token. The comparison follows Appendix C.1 and reuses the historical KVzip reference. Compression is conditioned on each question using the retained dense context; these examples do not evaluate a single compressed cache reused for unseen questions.

Table 26: Selected SCBench retrieval successes for Qwen3-8B at 90% compression. Full questions retain their original wording and punctuation; line breaks are reformatted. Enclosing output quotation marks are omitted. The reference and PARSEE answer coincide in every case, while KVzip receives zero task score. Row/question indices are zero-based within the short task variants. Anchor counts refer to PARSEE's identifier protection.

A. KV: key → value	**(row 3, question 4; 144 anchor tokens)**
Question	`Key: "f1123b6a-f860-4b45-9aab-956f535836ed"` The value associated with the specified key is:
Reference / PARSEE	`927d0b70-5935-46d5-aacc-cb3b51bd5680`
KVzip	`9acd5f7f-0ee6-457d-92fa-309e1b86108a`
Observed error	Returns the value of a different key in the dictionary.
B. KV: value → key	**(row 1, question 7; 144 anchor tokens)**
Question	`Value: "b0825e78-9c71-49fe-a098-943ca5717468` The key associated with the specified value is:
Reference / PARSEE	`8ab31076-37cb-4425-bbec-84bed2df3e47`
KVzip	`01d11578-ce8f-40aa-bb47-3178f9f0e177`
Observed error	Returns a key associated with a different value.
C. Prefix–suffix: incomplete copy	**(row 4, question 3; no anchors)**

Question	`Given prefix = "Owufn"`
	`suffix = "yOdXc"`
	What is the word in the dictionary that has both the prefix and the suffix?

Reference / PARSEE	`OwufnXb^vro-uVnzHOLNyOdXc`
KVzip	`OwufnXc`
Observed error	Omits the interior and fails to preserve the required suffix.
D. Prefix–suffix: interior corruption	**(row 9, question 1; no anchors)**

Question	`Given prefix = "i^SEg"`
	`suffix = "AtuQZ"`
	What is the word in the dictionary that has both the prefix and the suffix?

Reference / PARSEE	`i^SEglXRxPJyLcDaFHqhAtuQZ`
KVzip	`i^SEglXRxPJyLcDaFHq^Zk^AtuQZ`
Observed error	Matches both boundaries but changes the interior.

KV retrieval: preserving the requested association. Case A asks for the value of a key; case B reverses the lookup. KVzip returns a well-formed identifier in each case, but it belongs to another record in the corresponding dictionary. PARSEE returns the correct partner in both directions. This distinction matters because the identifiers carry little natural-language meaning: reproducing a plausible identifier cannot recover the requested association. PARSEE matches qualifying identifiers from the question to the input context and protects a bounded neighborhood, using the policy in Appendix C.3. Both examples activate 144 identifier-anchor positions, all retained after physical compilation and charged to the common budget. Protecting the neighborhood can keep the queried identifier together with its associated field. This explanation is consistent with the short-KV degradation upon removing anchors in Appendix F.1; the examples alone do not isolate the anchor contribution. Anchor construction uses only the question and context, without reference answers.

Prefix–suffix retrieval: preserving the complete string. Case C exposes incomplete copying: KVzip emits `OwufnXc`, losing the intervening content and the required suffix. In case D, it preserves both boundaries but replaces an interior `h` with `^Zk^`; the resulting word does not occur in the dictionary. PARSEE copies the full target in both cases. Neither case activates identifier anchors, so their success cannot be attributed to that protection rule. Question attention can locate matching boundaries, while the shared residual reference and radius-16 local maximum pooling can raise the priority of neighboring content needed to copy the whole word. Here, residual similarity is a contextual matching signal, not evidence that random strings have linguistic semantics. Pooling expands relevance scores without guaranteeing contiguous retention.

Why this combination can help retrieval. KVzip ranks entries by their role in reconstructing the context, supporting query-agnostic reuse (Kim et al., 2025). PARSEE instead allocates its attention and shared-reference quotas to the current question, with explicit identifier protection when applicable. At a small retained budget, this can favor the particular association or complete string required by an exact retrieval query. Query conditioning alone is not a sufficient explanation for the differences from all baselines: SnapKV also observes prompt queries. The relevant design combines question-specific evidence, local relevance expansion, and bounded protection, then compiles the selected support within the physical budget. These cases illustrate the failure modes this design can address; they do not identify which entries caused KVzip's errors or establish an independent accuracy gain from the page layout. Aggregate comparisons remain those in Table 1, and component evidence is reported separately in Appendix F.1.

H.2 Discussion: Failure Cases

Strong identifier retrieval does not imply equally strong evidence composition, numerical comparison, or source-code reproduction. Tables 27 and 28 present three Qwen3-8B failures at 90% compression, drawn from LongBench multi-hop QA, SCBench mixed retrieval, and SCBench RepoQA. In each case, PARSEE receives zero native task score, while the historical Full KV and KVzip references receive one. Contexts, questions, first output tokens, and generation limits agree across the three methods for these examples. All PARSEE runs finish within the physical budget and below their output limits. These are selected counterexamples, not estimates of failure frequency.

Table 27: Selected multi-hop QA and mixed-retrieval failures of Qwen3-8B at 90% compression. Questions and answers are shown in full, with display whitespace normalized. The reference, Full KV, and KVzip answers coincide; PARSEE is incorrect in both cases. Row/question indices are zero-based.

E. LongBench 2WikiMultihopQA (row 146, question 0)	
Question	Question: Where was the director of film 00 Schneider – Jagd Auf Nihil Baxter born?
Reference	Mülheim an der Ruhr
Full KV / KVzip	Mülheim an der Ruhr
PARSEE	Münster, West Germany (now part of Germany)
Observed error	Gives a different birthplace despite the two supporting passages.
F. SCBench mixed retrieval (row 50, question 5)	
Question	What is the largest number of the list? You should answer with only one number, no other words. The largest number of the list is:
Reference	`119`
Full KV / KVzip	`119`
PARSEE	`118`
Observed error	Returns a list member that is smaller than the maximum.

Multi-hop QA: relevance must preserve an evidence chain. Case E requires identifying the film's director and then finding his birthplace. The input's Passage 9 names Helge Schneider as the director, and Passage 2 gives his birthplace as Mülheim an der Ruhr. PARSEE instead answers Münster, a location absent from the supplied context. Thus, the error is substantive rather than a formatting mismatch. The two supporting passages establish the required chain, but the saved answer

alone does not show which link became ineffective after compression. A reference based on the original question can prioritize directly matching content without guaranteeing sufficient support for every intermediate entity. Shared relevance and local pooling therefore provide no general guarantee for composing evidence across passages.

Mixed retrieval: relevance is not numerical ordering. Case F asks for the maximum of a 15,004-element list. PARSEE returns 118, although the correct maximum is 119 and appears repeatedly in the input. Both Full KV and KVzip return 119. Unlike an identifier query, the question does not contain the desired value; the model must compare the list's contents. Question attention and residual similarity rank relevance to that request, but neither explicitly enforces numerical order or complete coverage of the comparison. This case illustrates a gap between preserving relevant numerical context and preserving the evidence needed for an exact global operation. It does not establish that every occurrence of 119 was evicted.

Table 28: G. SCBench RepoQA failure (row 50, question 4), Qwen3-8B at 90% compression. The full question is reproduced with Markdown formatting rendered. Code rows show noncontiguous signature and guard excerpts; omitted comments and body lines are not compared here. Full KV and KVzip receive native task score one, while PARSEE receives zero.

G. Exact function reproduction: changed signature and guard

Question. Function Description:
1. **Purpose**: The function is designed to remove all child elements from each selected element, while leaving text and comment nodes intact.
2. **Input**: The function takes no parameters and operates on the current selection of elements.
3. **Output**: It returns the instance itself, allowing for method chaining.
4. **Procedure**: The function iterates over each selected element. For each element, it checks if it has children. If it does, it sets the `next`, `prev`, and `parent` properties of each child to `null`, effectively detaching them. Finally, it clears the `children` array of the element.

Based on the function description and code context, please retrieve and repeat the exact described function from the code context in a code block wrapped by ```:

Reference excerpts	`export function empty<T extends AnyNode>`
	`(this: Cheerio<T>): Cheerio<T> {`
	`if (!hasChildren(el)) return;`

Full KV / KVzip excerpts	`export function empty<T extends AnyNode>`
	`(this: Cheerio<T>): Cheerio<T> {`
	`if (!hasChildren(el)) return;`

PARSEE excerpts	`export function empty(this: Cheerio): Cheerio {` `if (!isTag(el)) return;`

RepoQA: locating a function is insufficient for exact copying. In case G, PARSEE reproduces the correct function name and much of its loop structure, yet replaces `hasChildren` with `isTag` and drops the generic type parameter from the signature. The displayed Full KV and KVzip excerpts preserve the reference signature and guard. This is a source-fidelity failure despite substantial structural agreement. The description makes the function's purpose salient, but the task requires the actual implementation rather than a plausible reconstruction. The example does not determine whether the changed lines were absent from the retained cache or whether generation failed to use retained evidence.

Implications for the method. These failures identify demands that a relevance proxy does not explicitly certify: dependency coverage, exact comparison, and implementation fidelity. None of the three inputs activates identifier anchors. Their valid physical allocations also show why capacity feasibility and the semantic count bound must remain separate from task correctness. The cases motivate evaluating dependency-aware support expansion and fidelity-oriented protection under the same budget, rather than assuming that stronger retrieval averages establish reliable reasoning or copying on every input.

H.3 Discussion: Scope and Design Choices

Query-conditioned compression and cache reuse. PARSEE uses the available question to construct both its shared semantic reference and head-specific attention evidence. This design targets settings in which the active query is known before context compression, rather than requiring a single compressed support to remain optimal for arbitrary future queries. However, this does not preclude PARSEE from participating in a reuse-oriented serving system. A reusable query-agnostic cache can remain as the persistent substrate, while PARSEE performs query-conditioned selection and physical compilation for the active request or query-specific working set. Such a two-level design would separate long-lived cache reuse from short-lived query adaptation. Our current evaluation does not study the amortization, storage policy, or update strategy of such a reused-cache system, and therefore does not establish its end-to-end benefit.

Page layout and grouping granularity. The member-demand disparity identified by PARSEE is not specific to the L2H2B16 layout: the representation analysis is formulated for a general group size and page length. The concrete amount of synchronization and page-rounding overhead, however, depends on the chosen grouping and page granularity. Smaller groups can reduce demand coupling but increase metadata and scheduling granularity, whereas larger groups may improve execution regularity at the cost of stronger synchronization. Our implementation uses L2H2B16 as an executable operating point rather than claiming it to be universally optimal. Accordingly, the main contribution is the coupling between selection and a given physical representation, rather than a particular page shape.

Identifier protection. Identifier anchors address a narrow but important failure mode in exact retrieval, where semantically weak strings such as keys, UUIDs, or code identifiers may nevertheless be essential to the answer. They are a bounded protection mechanism outside the free-entry semantic and attention quotas, rather than a general relevance estimator. Their benefit is therefore expected to be task dependent: they can preserve exact associations when recognizable identifiers occur, but provide little direct help when the task instead requires multi-hop composition, numerical aggregation, or source-code fidelity. The successful retrieval cases without activated anchors further indicate that PARSEE's behavior is not reducible to this protection rule alone.

Hybrid architectures. For hybrid models, PARSEE compresses the KV states of full-attention layers while retaining recurrent state separately. The same page-aware selection principle therefore applies to the compressible attention component, but the resulting end-to-end memory reduction depends on how much of the model state is represented by that component. Consequently, compression ratios and memory savings should not be interpreted identically between fully attention-based and hybrid architectures.

H.4 Limitations

Structural guarantees and relevance. The semantic count bound follows from shared logits and bounded member perturbations under the stated cutoff, tie, and complete-tile assumptions. It characterizes a source of representation overhead but neither identifies true token importance nor guarantees that the retained support is sufficient for downstream reasoning. Mixed support can still contain attention-driven disparity, while page rounding and allocation competition remain separate physical costs. Likewise, the reference controls support the usefulness of residual semantics in the tested settings without establishing residual cosine as a uniquely effective scorer.

Evaluation scope. Models, tokenizers, and attention architectures differ, so cross-model results do not isolate scale effects. Parameter diagnostics on a development subset identify practical operating points rather than universal optima, and macro averages can conceal domain-specific losses. Backend throughput uses precomputed selections, whereas the online and common-runtime measurements additionally include scoring and selection under their stated timing boundaries. Capacity and latency gains depend on context length, generation length, concurrency, and allocator behavior; the measured TP1 eager backend therefore does not establish the same performance under every serving configuration.

Transient memory during compression. PARSEE reduces generation-phase KV capacity after sparse pages are committed and the corresponding dense blocks are reclaimed. Compression itself, however, requires destination sidecar pages to be reserved before the dense source can be released. Dense prefill KV and sparse destinations therefore coexist transiently, together with scoring workspace and metadata. As a result, the reduction in steady-state KV capacity should not be interpreted as an equivalent reduction in peak prefill or compression-time memory.

Repeated compression overhead. Applying compression repeatedly during long generation can incur additional scoring, gather, synchronization, page-table update, and ownership-transition costs. The current decode-only adaptation demonstrates that the selection framework can be reapplied to generated KV, but we do not jointly optimize when compression should be triggered or how frequently repacking should occur. For workloads with short intervals between compression events, these migration costs may offset part of the memory and decoding benefit.

H.5 Future Work

Adaptive physical layouts. PARSEE currently optimizes selection for a fixed executable grouping. A natural extension is to jointly adapt group layouts and support selection, explicitly accounting for the metadata, migration, and repacking costs induced by regrouping. Such a formulation could trade reduced member-demand disparity against the systems cost of changing the physical organization itself.

Broader model and serving coverage. The evaluated models cover several scales and attention or expert architectures but remain concentrated in the Qwen and Llama families. Broader evaluation across additional model families, accelerator generations, tensor- and pipeline-parallel deployments, prefix-sharing workloads, preemption policies, and heterogeneous request mixtures would clarify how broadly the observed representation and serving tradeoffs transfer.

Adaptive compression scheduling. The current system uses predefined compression points and budgets. Future systems could trigger compression according to cache pressure, decode length, request concurrency, or predicted remaining generation, and jointly choose the compression ratio, semantic quota, and recompression interval. Such adaptive scheduling could avoid paying selection and migration costs when the expected downstream benefit is small.

Composition with complementary compression mechanisms. PARSEE reduces physical cost by coordinating which original KV entries are retained and how they are represented in pages. This dimension is complementary to methods that reduce the bytes per retained entry through quantization or recover discarded information through reconstruction or restoration. Jointly optimizing selection, page allocation, precision, and restoration under a common physical-memory budget may provide a broader quality–capacity tradeoff than any one mechanism alone.