DESKV: LOW-RANK ROUTING AND DEFERRED SELECTION FOR LONG-CONTEXT GENERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-context generation is limited by KV-cache storage and access. Routing-based offloading reduces GPU memory use by keeping exact K/V in host memory and retrieving only a query-dependent working set. This requires a compact router that preserves token rankings as queries change, while minimizing the routing overhead for latency. This paper presents DeSKV, a low-rank routing method with step-deferred selection. The router is fitted from query and key moments, with more weight on keys; we find that keys span more directions than queries. Step deferral selection utilizes the preceding step's token selection while preparing the next working set. On Qwen3-8B, DeSKV achieves a 48.3 average LongBench score at a 256-token prompt-KV budget versus 48.5 with full cache, and reduces decode latency by 33.6% relative to full-cache execution at a 97K-token context.

1 INTRODUCTION

Long-context generation repeatedly accesses a large key-value (KV) cache, making its storage and access a major inference bottleneck. KV-cache eviction methods (1; 2) save GPU memory by permanently discarding K/V entries, but may degrade output quality since it remove insignificant tokens that later queries may need. Routing-based offloading methods (3; 4) instead keep exact K/V in host memory and retrieve appropriate entries at each decoding step. However, its small router must accurately preserve token rankings, and the selection with K/V fetch must complete before attention.

We present DeSKV, which combines a low-rank routing sketch with step-deferred selection. The sketch is fitted by balancing query and key moments and selects tokens from compact GPU-resident state, while attention uses exact K/V fetched from host memory. In our analysis, important tokens show locality across decoding steps and positions. This observation motivates reusing the selected tokens of the preceding step while preparing the next working set concurrently to hide routing and fetching latency; *priming* and *local expansion* mitigate the resulting selection error. Experiments on retrieval and LongBench tasks show that DeSKV retains near-full-cache quality under small prompt-KV budgets while reducing long-context decoding latency.

2 BACKGROUND

We consider autoregressive decoding with KV cache of length P. For a given layer and KV head, let

$$K = [k_1, \ldots, k_P]^\top \in \mathbb{R}^{P\times d}, \qquad V = [v_1, \ldots, v_P]^\top \in \mathbb{R}^{P\times d} \tag{1}$$

denote the cached keys and values. At decode step t, the current query $q_t \in \mathbb{R}^d$ assigns each cached token the score

$$s_{t,i} = q_t^\top k_i. \tag{2}$$

Because the KV cache grows linearly with P, long-context generation is often constrained by the number of exact K/V entries that can reside in GPU memory (1; 2). Recent systems (3; 5; 6) therefore keep most or all exact K/V entries in host memory and access only a query-dependent subset during generation. Under the budget B, the score-based reference set is

$$\mathcal{S}_t = \mathrm{TopK}_i\left(s_{t,i}; B\right), \qquad |\mathcal{S}_t| = B. \tag{3}$$

Because $s_{t,i}$ depends on q_t, this set can change at every decode step.

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.

Computing $\mathcal{S}_t$ requires scoring the query against all P cached keys and reading the full key cache at every decode step. Practical systems (3; 7; 4; 8) thus keep a small *routing state* on the GPU to approximate the scores. We call the exact GPU-resident K/V entries available to attention the *working set*. A *dynamic router* uses the routing state to compute approximate scores $\widetilde{s}_{t,i}$ and selects

$$\widetilde{\mathcal{S}}_t = \mathrm{TopK}_i\left(\widetilde{s}_{t,i}; B\right). \tag{4}$$

The selected K/V entries are then fetched from host memory as working set and used for attention.

3 Related Work

KV-cache eviction limits cache size by permanently discarding unretained entries. StreamingLLM (9) keeps initial and recent tokens, while H2O (1) and Scissorhands (10) update retention using past attention during generation. In contrast, prefill-based methods select retained prompt entries once before decoding. SnapKV (2) estimates importance from an observation window at the end of prefill. PyramidKV (11) allocates larger budgets to lower layers, whereas Ada-KV (12) adapts budgets across attention heads. However, these methods cannot recover evicted entries when later queries need them.

KV-cache compression reduces the storage cost of each token entries instead of evicting them. KIVI (13) uses different quantization granularities for keys and values, while KVQuant (14) combines per-channel, non-uniform quantization with outlier handling. TurboQuant (15) applies random rotations and scalar quantization without data-dependent calibration. MiniCache (16) merges similar KV states across layers, whereas Eigen Attention (17) computes attention from low-rank key and value representations. While effective, these methods must retain sufficient K/V information for each individual token, since they need to approximate the attention output with the compressed context.

KV-cache routing selects a query-dependent working set while keeping unselected entries available to later queries. Several systems combine routing with host offloading to reduce GPU memory footprint. InfLLM (4) retrieves context blocks using representative keys and combines them with a local window. PQCache (3) uses product-quantized keys to select exact K/V entries. GPU-resident methods instead focus on reducing attention's memory bandwidth to reduce latency while keeping the full cache on the GPU. Loki (18) scores tokens using a few principal components of keys, then computes full-dimensional attention for selected entries. Quest (7) uses page-level key minima and maxima to score each page, then reads K/V only from the selected pages. SparQ (19) estimates scores by reading a few key channels corresponding to large query components. Overall, these methods share a common challenge: preserving attention-score rankings with lightweight score estimates.

4 Motivation and Routing-Subspace Design

We use low-rank projection to approximate the attention scores through query–key inner products in a shared low-dimensional space. To design the projection, we first characterize query-key geometry and select its query-key mixture and rank. Let $\Pi{=}UU^\top$ be an orthogonal rank-r projection, where $U \in \mathbb{R}^{d\times r}$ and $U^\top U{=}I$. The router approximates the score between a query q and a cached key k as

$$\widetilde{s}(q,k) = q^\top \Pi k = (U^\top q)^\top (U^\top k). \tag{5}$$

The resulting error satisfies

$$q^\top k - q^\top \Pi k = \big((I-\Pi)q\big)^\top \big((I-\Pi)k\big), \quad \Big|q^\top k - q^\top \Pi k\Big| \le \|(I-\Pi)q\|_2 \, \|(I-\Pi)k\|_2 . \tag{6}$$

With only r shared directions, the projection must capture query variation and preserve differences among cached keys. equation 6 is symmetric in the discarded query and key residuals, but queries and keys require the shared projection U to preserve different amounts of their directional variation.

4.1 Query-Key Geometry

We first compare how query and key energy (squared ℓ_2 norm) is distributed across directions. We measure effective dimensionality using the participation ratio (20). For a vector set X, let $\{\lambda_j\}$ be the eigenvalues of its uncentered second moment $X^\top X$. The measure is

$$d_{\mathrm{eff}}(X) = \frac{\left(\sum_j \lambda_j\right)^2}{\sum_j \lambda_j^2}. \tag{7}$$

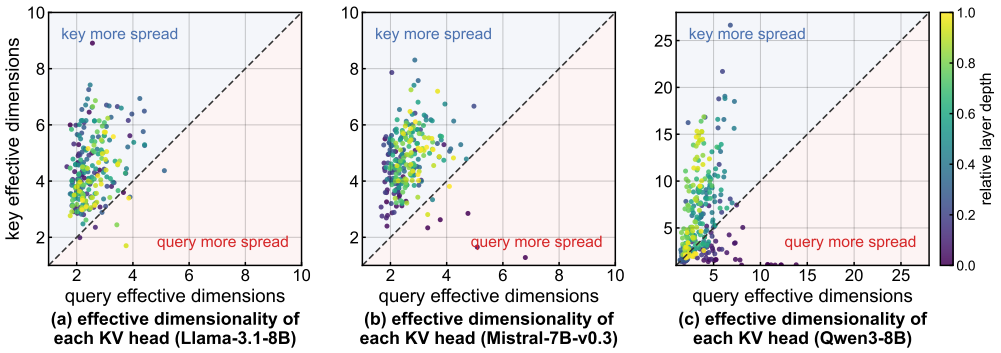

Figure 1: Effective-dimensionality measurements of prefill queries and prefix keys across layer–KV-head pairs in three models. Each point is averaged over 48 prompts; color denotes relative layer depth, and the dashed line denotes equal dimensionality.

A small value means that most of the vector energy lies in a few directions.

In the evaluated models, each KV head is shared by four query heads. For each KV head, we collect 128 queries from each of its four associated query heads, yielding 512 queries in total, and sample 512 keys from the full prefix. We compute the effective dimensionality of the two sets separately and average each layer–KV-head measurement over 48 prompts: 16 each from TriviaQA, MultiFieldQA-en, and SAMSum (21), which cover few-shot QA, long-document QA, and dialogue summarization, respectively.

Figure 1 shows the same pattern across all three models. Most layer–head pairs lie above the diagonal. The median query/key effective dimensionalities are $2.35/4.16$ for Llama-3.1-8B, $2.45/4.67$ for Mistral-7B-v0.3, and $3.27/4.65$ for Qwen3-8B. Although both effective dimensionalities are far below $d = 128$, the key effective dimensionality is 1.4–$1.9\times$ larger.

Although the score error depends on both query and key residuals, query energy is concentrated in fewer directions than key energy. Within the shared basis U, focusing on these concentrated queries may better reduce the projection error since each query is scored against all previous keys, while focusing on keys may be robust in performance as they remain fixed during decoding. We next examine how this balance affects score error on prefill queries and decode-time routing.

4.2 Selecting the Query-Key Mixture and Rank

We test whether key-weighted fitting improves decode-time routing by varying the query-key mixture and projection rank. We fit the shared basis U by truncated SVD (22): its columns are the top-r eigenvectors of the trace-normalized query-key moment:

$$C(\alpha) = \alpha \frac{Q^\top Q}{\mathrm{tr}(Q^\top Q)} + (1-\alpha)\frac{K^\top K}{\mathrm{tr}(K^\top K)}. \tag{8}$$

Here, α sets the relative query-key weight in the normalized reconstruction error minimized by U. The endpoint $\alpha = 0$ uses only key statistics, similar to Loki (18), whereas $\alpha = 1$ uses only query statistics. We first analyze how the rank and query-key weighting affect score reconstruction and routing, then evaluate how well the fitted projections preserve routing accuracy as decoding proceeds.

We test whether effective dimensionality determines the rank needed to reconstruct scores. For each Qwen3-8B KV head, we fix $\alpha = 0.2$, vary $r \in \{8, 16, 32, 64\}$, and aggregate the squared residual in equation 6 over the query set and all prefix keys. Figure 2(a) shows nearly horizontal bands, one per rank: at fixed r, score error varies little with $d_{\mathrm{eff}}(Q) + d_{\mathrm{eff}}(K)$, whereas larger r lowers each band. Effective dimensionality captures query-key asymmetry but does not determine the rank needed for accurate score reconstruction: low-energy directions can still contribute to $q^\top k$.

We next examine whether the preferred query-key mixture changes during decoding. Using the same 48-prompt set as in Section 4.1, we fit each mixture's basis during prefill and measure its overlap with exact top-128 selection over full-cache greedy trajectories of up to 100 steps.

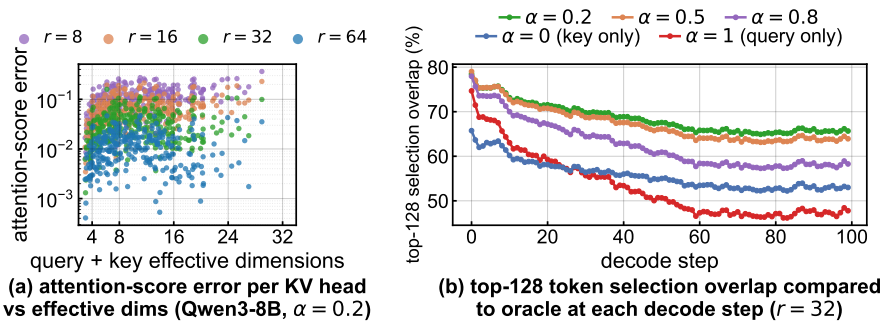

Figure 2: Rank and decode-time transfer measurements for prefill-fitted projections on Qwen3-8B. (a) Attention-score error across layer–KV-head pairs at four ranks. (b) Exact-score top-128 selection overlap over decode steps at $r = 32$.

Figure 2(b) shows that query-only fitting gives high overlap at the prefill and early decoding steps, where the fixed queries are relatively easier to reconstruct than more dispersed keys. However, as decoding proceeds, key-only fitting achieves higher overlap than query-only fitting; the prompt keys and fitted basis remain fixed during decoding, but the queries change. Therefore, later queries may be less accurately represented by the fitted basis, making query-weighted fitting less favorable.

To examine this shift more thoroughly, we evaluate each prefill-fitted basis on both its fitting queries and later decode queries. Using the same 48-prompt set and 512-query/512-key fitting procedure, we run full-cache greedy decoding for up to 100 steps and aggregate queries from the final 50 available decode steps. Figure 3(a)–(b) shows prefill (dashed) and decode (solid) results.

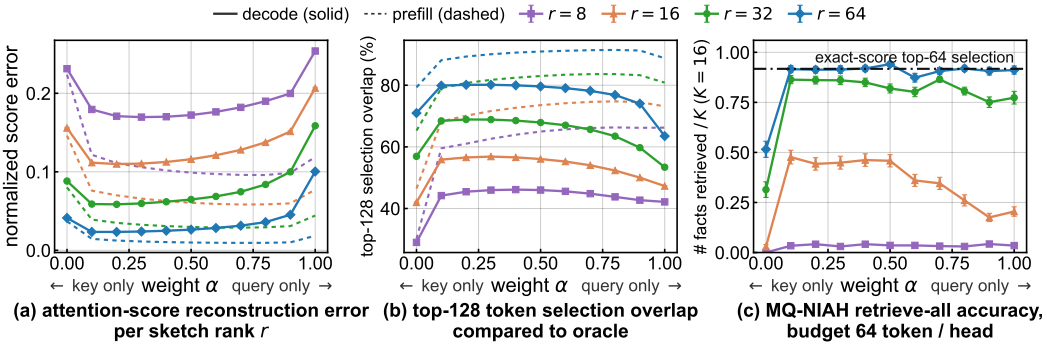

Figure 3: Query–key mixture and rank sweeps on prefill queries, accumulated decode queries, and MQ-NIAH retrieval on Qwen3-8B. (a)–(b) Attention-score error and exact-score top-128 selection overlap. (c) Retrieve-all accuracy under a 64-token-per-head budget with generation length 168.

The preferred mixture shifts with the evaluation query set. On prefill queries, both score error and top-128 selection overlap favor $\alpha \approx 0.8$; on decode queries, the preferred range shifts to $\alpha = 0.2$–0.5. Thus, prefill-only evaluation favors query-heavy fitting, whereas decode-time routing favors a larger key contribution, consistent with the key-weighted prediction in Figure 1. Increasing r improves both metrics without changing this shift.

Figure 3(c) tests whether the same mixture effect appears in model output. Each multi-query needle-in-a-haystack (MQ-NIAH) prompt contains $K = 16$ planted facts, which the model must retrieve under a 64-token-per-head KV budget with $8K + 40$ generation length. Results are averaged over 72 random seeds. The horizontal line denotes the result of selecting the same number of entries using exact $q^\top k$ scores. At $r = 16$, accuracy rises sharply when query statistics are included but then plateaus over $\alpha = 0.1$–0.5. At $r = 32$, $\alpha = 0.2$ nearly matches the exact-score reference.

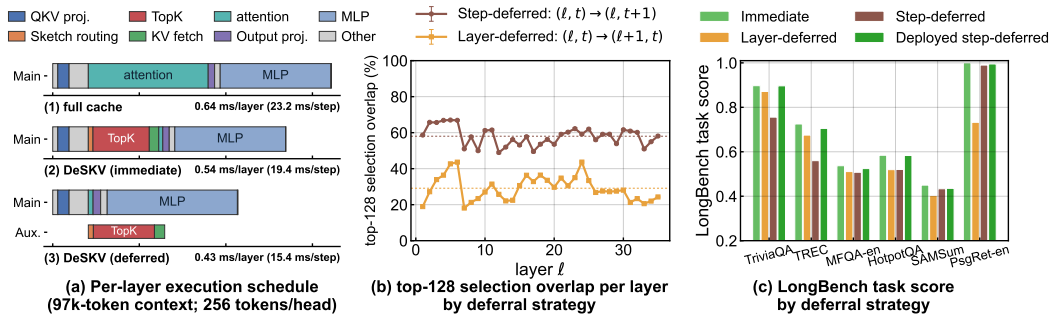

Figure 4: Execution-schedule and selection-transfer experiments on Qwen3-8B. (a) Per-layer schedules at an approximately 97K-token context; labels report measured decode latency. (b) Overlap with the top-128 selection when reusing the selection from the previous layer or previous decode step. (c) LongBench task scores under immediate, layer-deferred, and step-deferred schedules.

5 DeSKV: Routing and Execution

DeSKV keeps two representations of the KV cache. Host memory stores all exact keys and values, while GPU memory stores a compact routing sketch for every token and the exact K/V entries in the current working set. The sketch selects the working set, while attention uses its exact K/V entries.

During routing, *immediate selection* uses the current query to select and fetch the working set before attention. For efficiency, DeSKV instead uses *step-deferred selection*: after initializing the first working set, attention uses the previous step's same-layer selection based on locally expanded scores.

5.1 Routing Sketch and Immediate Selection

For each layer ℓ and KV head h, let $K_{\ell,h} \in \mathbb{R}^{P\times d}$ denote the keys produced during prefill. As in equation 8, we take $U_{\ell,h} \in \mathbb{R}^{d\times r}$ as the top-r eigenvectors of the normalized query–key moment. Because routing scans all P projected keys at each layer and decode step, we quantize the low-dimensional sketch with channel-wise granularity and store it in INT4 to reduce both the routing-sketch size and the memory traffic of the scan:

$$Z_{\ell,h} = K_{\ell,h}U_{\ell,h} \in \mathbb{R}^{P\times r}, \quad \left(Z^{\mathrm{int4}}_{\ell,h}, \theta_{\ell,h}\right) = \mathrm{Quantize}_{\mathrm{INT4}}(Z_{\ell,h}), \tag{9}$$

where $Z^{\mathrm{int4}}_{\ell,h}$ contains the packed codes and $\theta_{\ell,h}$ contains the quantization metadata. The packed sketch requires $r/2$ bytes per cached token and KV head. At $d = 128$ and $r = 32$, this is 16 bytes, or 3.1% of the corresponding BF16 K/V state. We overlap GPU prefill with K/V offloading to host memory, CPU-side basis fitting, and sketch construction, details are expained in Appendix E.1.

Under grouped-query attention (23), one KV head serves multiple query heads. Let $\mathcal{G}(h)$ denote the query heads associated with KV head h. At decode step t, DeSKV projects each live query into the same basis and computes

$$\widehat{z}_{\ell,h,i} = \mathrm{DeQuant}_{\mathrm{INT4}}\left(z^{\mathrm{int4}}_{\ell,h,i}; \theta_{\ell,h}\right), \quad \widetilde{s}^{(\ell,h)}_{t,i} = \max_{g\in\mathcal{G}(h)} \left(U^{\top}_{\ell,h} q_{\ell,g,t}\right)^{\top} \widehat{z}_{\ell,h,i}. \tag{10}$$

The maximum operation favors tokens assigned a high score by any query head sharing the KV head. We implement equation 10 with a Triton kernel (24) to compute the scores directly from the packed INT4 sketch. The sketch is used only for selection; attention always uses exact keys and values.

Following prior KV-cache methods (2; 11; 12), we keep a fixed window of the last few prompt tokens without selection. Let $\mathcal{R}$ denote this window, $\mathcal{G}_t$ the generated tokens until step t, and B the per-head KV budget including $\mathcal{R}$. Immediate selection retrieves $B-|\mathcal{R}|$ entries using the sketch, yielding the immediate selection $\mathcal{I}^{(\ell,h)}_t$ and attention working set $\mathcal{E}^{\mathrm{imm},(\ell,h)}_t$:

$$\mathcal{I}^{(\ell,h)}_t = \mathrm{TopK}_{i\notin\mathcal{R}}\left(\widetilde{s}^{(\ell,h)}_{t,i}; B - |\mathcal{R}|\right), \qquad \mathcal{E}^{\mathrm{imm},(\ell,h)}_t = \mathcal{I}^{(\ell,h)}_t \cup \mathcal{R} \cup \mathcal{G}_t. \tag{11}$$

The $\mathcal{R} \cup \mathcal{G}_t$ remain GPU-resident, with generated tokens appended each step. The $\mathcal{I}_t$ is recomputed from the current query q_t and fetches its exact K/V from host memory before attention can begin.

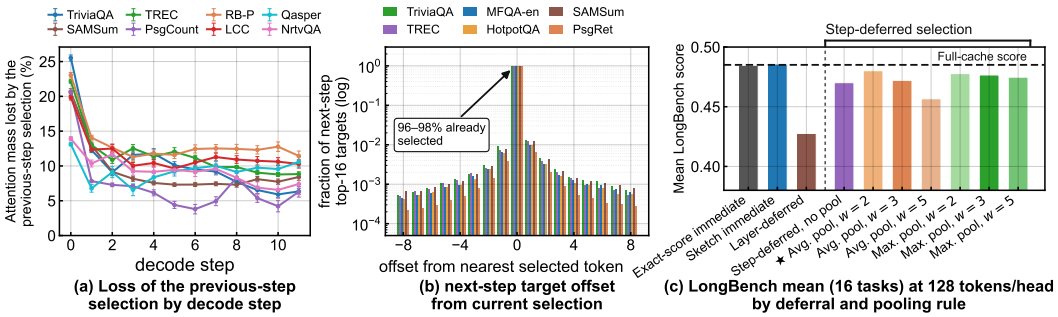

Figure 5: Selection-reuse and local-pooling experiments on Qwen3-8B. (a) Attention-mass loss from reusing the preceding selection during the first few decode steps. (b) Signed offsets of next-step top-16 attention targets from the current top-128 selection. (c) Mean LongBench score over 16 tasks at 128 tokens per KV head across deferral and pooling variants.

5.2 Step-Deferred Selection

Immediate selection must score the sketch, select tokens, and fetch their exact K/V entries before attention. To hide the latency of routing and K/V fetch, deferred selection prepares a selection one step ahead and uses it in the next attention operation. Figure 4(a) compares full-cache, immediate, and step-deferred execution at a 97K-token context. Their measured latencies are 23.2, 19.4, and 15.4 ms per step, respectively. Most of the additional cost comes from token selection and the associated index processing. The deferred schedule moves this preparation to an auxiliary stream while the main stream continues the current forward pass.

We consider two ways to defer selection. *Layer-deferred selection* reuses at layer ℓ the selection computed at layer $\ell - 1$ in the same decode step, whereas *step-deferred selection* reuses the selection computed in the same layer at the preceding decode step. The former transfers token indices between layer representations and provides one layer of lead time. The latter stays within the same layer and provides one decode step to route and fetch the next working set.

Figure 4(b) measures the overlap between each deferred selection and its destination's top-128 set. Step-deferred selection overlaps with average 58% of this set, compared with 29% for layer-deferred selection. This gap persists across model depth, and thus DeSKV uses step-deferred selection.

Figure 5(a) measures the attention mass lost when the preceding selection is reused for the current query. Across eight tasks, the loss is highest at the start of decoding and drops sharply in the next few steps. Thus, we initialize step deferral with a three-step *priming phase* of immediate selection. The first two steps use immediate routing scores. The third applies *local score expansion* in Section 5.3 and prepares a selection consumed both immediately and at step 4, when step deferral begins.

Let $\mathcal{D}_t$ denote the locality-aware selection from equation 14 computed at step t. The working set is

$$\mathcal{E}_t^{\mathrm{def},(\ell,h)} = \begin{cases} \mathcal{I}_t^{(\ell,h)} \cup \mathcal{R} \cup \mathcal{G}_t, & t = 1, 2, \\ \mathcal{D}_3^{(\ell,h)} \cup \mathcal{R} \cup \mathcal{G}_3, & t = 3, \\ \mathcal{D}_{t-1}^{(\ell,h)} \cup \mathcal{R} \cup \mathcal{G}_t, & t \geq 4. \end{cases} \tag{12}$$

For $t \geq 4$, attention uses the prepared selection $\mathcal{D}_{t-1}$. Once the current query is available, an auxiliary stream computes $\mathcal{D}_t$. After the current layer's attention finishes reading the working-set buffer from step $t-1$, the selected exact K/V overwrite its routed portion in place while the main stream continues the forward pass. A completion event orders the next step's attention after this fetch.

Figure 4(c) shows the task-level effect of this schedule. Step-deferred selection alone loses accuracy because it uses the preceding query without accounting for changes between consecutive steps. The deployed schedule combines step-deferred selection with priming and the local selection rule introduced next, recovering much of the gap to immediate selection across the evaluated tasks.

5.3 Local Expansion for Deferred Selection

For deferred selection to work, the selection must cover the tokens needed at the next step. Figure 5(b) measures, for each token in the exact top-16 selection at step $t+1$, its signed positional offset from the nearest token in the top-128 selection at step t. Across six tasks, 96–98% of these targets are already selected, and the remaining targets are near selected positions. This locality motivates local score expansion, which extends routing scores to nearby tokens.

We compare average and max pooling of routing scores with widths $w \in \{2, 3, 5\}$. Figure 5(c) shows the mean score over 16 LongBench tasks at 128 tokens per KV head. Width-two right-neighbor average pooling achieves the highest mean among step-deferred variants, narrowing the gap to immediate selection. We thus apply this local score expansion before selecting the next working set:

$$\overline{s}_{t,i}^{(\ell,h)} = \left(\widetilde{s}_{t,i}^{(\ell,h)} + \widetilde{s}_{t,i+1}^{(\ell,h)}\right)/2, \qquad 1 \le i < P. \tag{13}$$

with the neighborhood clipped at sequence boundaries. The selected indices are

$$\mathcal{D}_t^{(\ell,h)} = \mathrm{TopK}_{i \notin \mathcal{R}} \left(\overline{s}_{t,i}^{(\ell,h)}; B - |\mathcal{R}|\right). \tag{14}$$

Average pooling shares routing evidence between adjacent positions, allowing them to compete for the same working-set slots. Detailed results are reported in Appendix D.7.

6 Evaluation

6.1 Experimental Setup

We implement DeSKV using Python 3.12.3, PyTorch 2.9.0, CUDA 13.0, Transformers 5.12.1, and FlashAttention 2.7.4 (25). Experiments run with NVIDIA RTX PRO 6000 GPU and AMD EPYC 9755 CPU. The GPU is connected through PCIe 5.0 $\times 16$, and the offloaded exact KV cache is stored in pinned host memory. We evaluate Llama-3.1-8B-Instruct (26), Mistral-7B-v0.3 (27; 28), and Qwen3-8B (29). Main-text results use Qwen3-8B, with the other models reported in Appendix D.1. DeSKV uses $\alpha = 0.2$, sketch rank $r = 32$, an INT4 routing sketch, a fixed 32-token window, and width-two right-neighbor average pooling.

We implement all baselines based on their official repositories. All selective-attention layers use a prompt-KV budget B per KV head, including sink, recent, and local tokens; for methods with head- (12) or layer-wise allocations (11), B denotes the average KV budget. All methods retain K/V representations of generated tokens separately; these do not count toward B. StreamingLLM (9) keeps 4 sink tokens and $B - 4$ tail tokens; H2O splits B equally between heavy-hitter and tail tokens; SnapKV (2) scores from 32-token prompt-tail window; and Ada-KV (12) and PyramidKV (11) use their native head-wise and layer-wise allocations, respectively.

For routing methods, we target a 16-byte-per-token scoring payload, matching DeSKV, with metadata accounted for separately. Quest (7) uses 32-token pages without a tail window and full attention in the first two layers, following its official implementation. InfLLM (4) uses 16-token blocks with one representative key. PQCache (3) uses $m = 16$ and $b = 8$, yielding $mb/8 = 16$ bytes codes and a 64 KB codebook per layer and KV head fitted by per-prompt k-means. SparQ (19) reads $r = 8$ query-selected key channels while retaining the full BF16 keys on the GPU. Loki (18) uses $r = 8$ per-head PCA fitted offline on WikiText-2 keys after rotary position encoding, while retaining the full BF16 KV cache on the GPU.

6.2 Evaluation on Multi-Query Retrieval

We construct a controlled multi-query needle-in-a-haystack (MQ-NIAH) experiment based on RULER (30) to examine how retrieval changes with the number of requested facts under fixed prompt-KV budgets. Each prompt places K distinct city–code facts at evenly spaced depths in an approximately 8.9K-token WikiText-2 (31) context and asks the model to return all codes. We report results for up to 16 facts at KV budgets $B \in \{128, 192, 256\}$ tokens per KV head. Greedy decoding uses an output limit of 80 tokens for $K \le 8$ and $8K + 40$ otherwise. Accuracy is the fraction of target codes recovered by exact substring matching, averaged over 24 prompts per setting. Exact-score performs immediate selection using exact $q^\top k$ scores under the same budget.

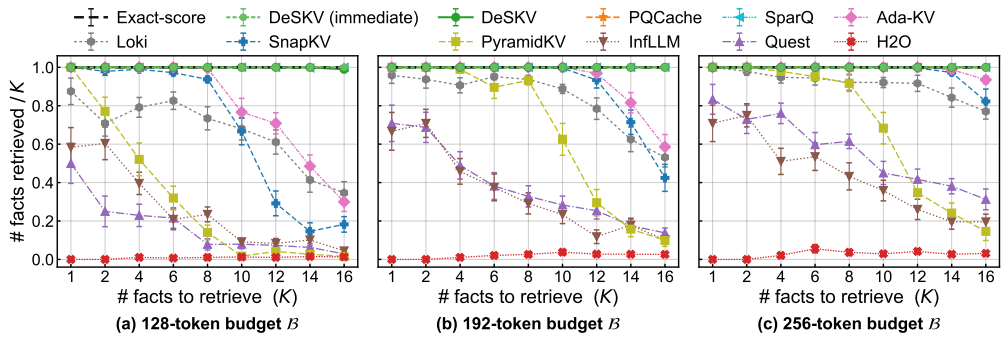

Figure 6: Multi-query needle-in-a-haystack retrieval on Qwen3-8B across fact counts and KV budgets. Accuracy is the fraction of target codes recovered, and error bars show standard error over 24 random seeds. Exact-score uses immediate selection with exact attention scores.

Figure 6 shows that DeSKV (immediate) matches Exact-score throughout the plotted range. The deployed DeSKV maintains 99–100% accuracy across these settings along with PQCache and SparQ. In contrast, page- and block-level routers are limited by their coarse selection granularity, while prompt-time selection degrades as more facts must be retained. At $B = 128, K = 16$, SnapKV and Ada-KV recover only 18.2% and 29.9% of the facts, respectively; larger budgets improve their coverage but do not close the gap. Thus, the compact sketch preserves exact-score retrieval quality in immediate execution, and the deployed step-deferred schedule retains nearly all of this accuracy.

Table 1: Latency and KV memory costs on Qwen3-8B with a fixed 256 token budget per KV head. TTFT estimates combine full-cache prefill with the additional router preparation time.

Method	KV Cache Source†	TTFT (s)			TPOT (ms/step)			KV memory cost at 97K⋆	
		8K	32K	97K	8K	32K	97K	GPU Footprint	Read Volume
Full cache	GPU	0.50	2.63	13.28	14.24	16.53	23.17	100%	100%
Routing-free	–	0.50	2.63	13.28	13.48	13.46	13.45	0.3%	0.3%
PQCache	Host	+0.28‡	+0.30‡	+39.2‡	20.89	21.59	26.47	0.3%(+3.3%)	0.3%(+3.3%)
SparQ	GPU	+0.08	+0.34	+1.18	17.15	17.86	19.91	100%(+0.0%)	0.3%(+3.1%)
Loki	GPU	+0.08	+0.35	+1.20	16.44	17.07	18.59	100%(+0.0%)	0.3%(+3.1%)
InfLLM	Host	+0.36	+1.59	+4.35	15.97	16.24	16.82	0.3%(+3.1%)	0.3%(+3.1%)
Quest (our impl.)	GPU	+0.13	+0.81	+2.18	15.87	15.91	16.43	100%(+3.1%)	0.3%(+3.1%)
DeSKV (imm.)	Host	**+0.00**‡	**+0.00**‡	**+0.07**‡	17.17	17.79	19.40	0.3%(+3.1%)	0.3%(+3.1%)
DeSKV	Host	**+0.00**‡	**+0.00**‡	**+0.07**‡	**14.59**	**14.69**	**15.39**	0.3%(+3.1%)	0.3%(+3.1%)

†Full KV cache location. ‡CPU preparation overlaps GPU prefill; only the remaining delay is added.
⋆KV storage/reads with additional routing costs in parentheses, normalized to full BF16 KV cache at 97K.

6.3 Evaluation on Router Overhead

We evaluate routing costs in a fixed-shape benchmark on one RTX PRO 6000 GPU at batch size one, capturing the entire decode step in a CUDA Graph. Table 1 reports time to first token (TTFT), decoding latency per output token (TPOT), and KV memory costs. All methods share full-attention prefill; TTFT estimates add the preparation time that remains after any overlap. CPU-side construction leaves little additional delay for DeSKV. PQCache also overlaps construction with prefill, but its clustering exceeds the prefill latency at 97K even with their recommended minimum iteration under the $m = 16, b = 8$ configuration. For Quest, SparQ, Loki, and InfLLM, preparation is sequentially executed after the prefill operation.

Equal scoring payloads do not imply equal routing latency. PQCache performs codebook lookups, which are more costly than our sketch scan in these measurements. SparQ gathers a small number of query-selected channels, whereas Loki scans fixed PCA components for token selection. Quest and InfLLM rank pages or blocks rather than individual tokens, reducing selection cost at the coarser granularity evaluated in the retrieval experiments. Immediate execution still places scoring, selection, and KV materialization before attention. Step deferral overlaps this preparation with the current forward pass, giving DeSKV the lowest TPOT among the evaluated routers. Its remaining overhead comes primarily from contention for GPU compute and memory bandwidth.

Table 2: LongBench score of eviction and routing methods on Qwen3-8B under fixed KV budgets.

			Single-Doc. QA			Multi-Doc. QA			Summarization			Few-shot			Synthetic		Code		
Budget	Type	Method	*NrtvQA*	*Qasper*	*MFQA-en*	*HotpotQA*	*2WikiMQA*	*MuSiQue*	*GovReport*	*QMSum*	*MultiNews*	*TREC*	*TriviaQA*	*SAMSum*	*PCount*	*PRe*	*LCC*	*RB-P*	*Avg.*
Full	–	Full cache	26.0	48.1	53.9	59.2	43.0	35.5	33.5	24.1	24.7	71.5	90.7	44.2	7.2	100.0	57.7	56.7	48.51
128	Eviction	StreamingLLM	14.4	23.5	24.1	33.9	29.9	13.3	15.7	19.1	15.3	34.0	59.5	37.1	3.5	68.5	52.4	48.7	30.80
		H2O	17.4	32.7	33.0	49.4	33.4	21.6	26.2	20.7	21.3	52.0	56.7	37.0	4.5	61.0	52.5	49.2	35.54
		PyramidKV	16.8	27.0	38.1	44.0	31.8	14.8	16.2	20.2	14.9	37.5	67.2	35.5	1.9	92.5	50.2	45.3	34.62
		SnapKV	17.9	29.3	44.7	51.9	35.3	20.7	17.9	21.1	16.2	42.0	85.4	38.4	2.1	98.5	52.0	50.4	38.99
		Ada-KV	18.8	30.5	46.9	53.9	37.2	22.1	18.1	20.9	16.4	47.0	83.7	37.9	2.3	99.5	53.5	51.8	40.03
	Routing	Quest	4.7	25.7	31.6	21.3	22.4	7.0	25.4	18.2	22.3	15.0	24.4	14.3	2.2	53.1	34.2	25.5	21.71
		Loki	16.1	31.4	42.1	43.7	31.7	19.3	32.0	24.0	24.9	57.5	49.6	37.1	3.5	52.0	35.2	28.8	33.06
		InfLLM	**27.1**	35.6	45.0	52.9	34.2	25.3	27.8	22.6	22.6	46.0	78.0	39.1	2.7	67.0	53.0	51.8	39.43
		SparQ	23.2	44.4	**54.0**	58.0	**44.2**	32.6	32.2	24.1	24.4	69.5	89.1	43.2	4.6	94.5	57.1	55.0	46.87
		PQCache	25.7	45.7	53.7	**59.9**	43.7	35.2	33.5	23.9	24.8	**74.0**	89.0	44.5	4.2	**100.0**	**58.8**	56.2	48.29
		DeSKV (imm.)	26.0	**47.8**	53.7	58.4	43.0	**35.8**	**33.6**	**24.4**	24.8	72.5	**89.7**	**45.0**	5.2	**100.0**	58.6	**58.8**	**48.58**
		DeSKV	26.2	46.7	52.4	58.3	43.0	35.2	32.8	23.6	**25.5**	70.5	89.6	43.5	**7.9**	99.5	56.6	56.9	48.01
256	Eviction	StreamingLLM	15.2	24.8	23.6	34.1	29.1	13.6	18.4	19.4	17.9	40.0	67.2	39.3	3.7	68.0	55.1	51.6	32.57
		H2O	19.6	32.6	35.7	50.7	33.3	26.4	27.6	21.4	22.2	58.0	67.5	40.0	3.7	63.5	55.7	51.7	38.11
		PyramidKV	17.7	31.5	44.2	52.0	32.7	19.5	20.2	21.3	17.6	47.5	82.8	38.7	**7.1**	99.5	53.9	49.6	39.74
		SnapKV	21.5	36.5	49.3	58.2	40.1	29.8	21.7	22.3	19.4	53.5	**90.4**	40.6	5.4	**100.0**	56.2	54.5	43.71
		Ada-KV	21.2	38.2	50.6	59.2	40.6	28.5	21.9	22.6	19.5	60.0	87.6	40.4	5.3	**100.0**	56.8	55.5	44.25
	Routing	Quest	9.8	36.8	43.0	39.0	31.5	16.9	29.0	20.3	23.8	35.0	51.2	23.5	4.0	84.2	42.9	35.2	32.88
		Loki	19.3	37.0	46.1	44.0	32.6	21.5	**33.9**	24.1	**26.5**	65.0	70.7	39.6	3.3	84.5	43.4	36.9	39.29
		InfLLM	**26.9**	40.6	49.3	57.4	40.2	28.4	30.4	23.4	23.5	63.5	83.9	42.6	3.8	80.0	56.9	55.8	44.16
		SparQ	23.1	46.1	**54.5**	58.4	**43.5**	31.6	33.4	24.1	24.6	71.0	88.8	43.9	5.7	98.5	57.5	56.5	47.56
		PQCache	25.6	**47.1**	54.1	**60.9**	43.3	34.5	33.6	24.1	24.9	**72.5**	90.0	**45.2**	4.3	**100.0**	**58.5**	56.8	**48.47**
		DeSKV (imm.)	26.1	46.6	53.7	60.1	41.5	**35.4**	33.5	**24.6**	24.8	72.0	89.9	45.1	5.1	**100.0**	58.0	**58.6**	48.43
		DeSKV	26.1	45.6	54.1	59.8	41.4	**35.4**	33.2	24.0	25.5	72.0	89.8	44.1	7.0	**100.0**	58.0	57.6	48.35

The memory columns distinguish GPU storage from per-step reads. SparQ and Loki retain the full KV cache, while Quest additionally stores page statistics; reading less data therefore does not yield comparable storage savings. InfLLM, PQCache, and DeSKV instead keep the full exact KV in host memory and only a working set and routing state on the GPU. The eviction row provides a fixed-working-set reference without routing or fetching. DeSKV remains slightly slower than full-cache attention at 8K, where routing contention outweighs the attention savings, but its TPOT grows more slowly with context length and falls below full cache at longer contexts.

6.4 Evaluation on LongBench test

We evaluate Qwen3-8B on 16 LongBench tasks spanning single- and multi-document QA, summarization, few-shot learning, synthetic tasks, and code completion. Table 2 reports the task scores and their average at prompt-KV budgets $B \in \{128, 256\}$. DeSKV (imm.) achieves mean scores of 48.58 and 48.43 across these budgets, compared with 48.51 for full cache, indicating that the compact routing sketch preserves overall task performance even at the smallest budget. The deployed step-deferred schedule reaches 48.01 and 48.35, matching the mean score of immediate selection at $B = 256$. It scores above all evaluated eviction methods and all routing baselines except PQCache, trailing the latter by at most 0.3 points. These results show that step-deferred selection retains near-full-cache task quality even under small prompt-KV budgets. At $B = 128$, TREC and LCC show the largest decreases relative to immediate selection by 2.0 points. Increasing the budget closes the gaps.

7 Conclusion

DeSKV addresses two challenges in KV-cache offloading: preserving query-dependent token rankings with a compact representation and hiding the latency of preparing selected exact K/V. It fits an INT4 routing sketch from a key-weighted mixture of query and key moments, reflecting the broader directional spread of keys. To hide routing and K/V-fetch latency, step deferral reuses the preceding step's same-layer selection and prepares the next working set alongside the current forward pass; priming and local expansion limit the accuracy loss from the reuse. On Qwen3-8B, DeSKV retains near-full-cache quality on retrieval and LongBench under small prompt-KV budgets, with a 33.6% reduction in decode latency relative to full-cache execution at a 97K-token context.

REFERENCES

[1] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in neural information processing systems*, 36:34661–34710, 2023.

[2] Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. Snapkv: Llm knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37:22947–22970, 2024.

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

[4] Chaojun Xiao, Pengle Zhang, Xu Han, Guangxuan Xiao, Yankai Lin, Zhengyan Zhang, Zhiyuan Liu, and Maosong Sun. Infllm: Training-free long-context extrapolation for llms with an efficient context memory. *Advances in neural information processing systems*, 37:119638–119661, 2024.

[5] Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. {InfiniGen}: Efficient generative inference of large language models with dynamic {KV} cache management. In *18th USENIX symposium on operating systems design and implementation (OSDI 24)*, pages 155–172, 2024.

[6] Di Liu, Meng Chen, Baotong Lu, Huiqiang Jiang, Zhenhua Han, Qianxi Zhang, Qi Chen, Chengruidong Zhang, Bailu Ding, Kai Zhang, et al. Retrievalattention: Accelerating long-context llm inference via vector retrieval. *Advances in Neural Information Processing Systems*, 38:54358–54385, 2026.

[7] Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context llm inference. *arXiv preprint arXiv:2406.10774*, 2024.

[8] Huanxuan Liao, Yixing Xu, Shizhu He, Guanchen Li, Xuanwu Yin, Dong Li, Emad Barsoum, Jun Zhao, and Kang Liu. Spark: Query-aware unstructured sparsity with recoverable kv cache channel pruning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 31961–31969, 2026.

[9] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, volume 2024, pages 21875–21895, 2024.

[10] Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *Advances in Neural Information Processing Systems*, 36:52342–52364, 2023.

[11] Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, et al. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*, 2024.

[12] 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. *Advances in Neural Information Processing Systems*, 38:113152–113188, 2026.

[13] Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. Kivi: A tuning-free asymmetric 2bit quantization for kv cache. *arXiv preprint arXiv:2402.02750*, 2024.

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

[15] Amir Zandieh, Majid Daliri, Majid Hadian, and Vahab Mirrokni. Turboquant: Online vector quantization with near-optimal distortion rate. In *International Conference on Learning Representations*, volume 2026, pages 56418–56439, 2026.

[16] Akide Liu, Jing Liu, Zizheng Pan, Yefei He, Gholamreza Haffari, and Bohan Zhuang. Minicache: Kv cache compression in depth dimension for large language models. *Advances in Neural Information Processing Systems*, 37:139997–140031, 2024.

[17] Utkarsh Saxena, Gobinda Saha, Sakshi Choudhary, and Kaushik Roy. Eigen attention: Attention in low-rank space for kv cache compression. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 15332–15344, 2024.

[18] Prajwal Singhania, Siddharth Singh, Shwai He, Soheil Feizi, and Abhinav Bhatele. Loki: Low-rank keys for efficient sparse attention. *Advances in Neural Information Processing Systems*, 37:16692–16723, 2024.

[19] Luka Ribar, Ivan Chelombiev, Luke Hudlass-Galley, Charlie Blake, Carlo Luschi, and Douglas Orr. Sparq attention: Bandwidth-efficient llm inference. *arXiv preprint arXiv:2312.04985*, 2023.

[20] Ashok Litwin-Kumar, Kameron Decker Harris, Richard Axel, Haim Sompolinsky, and LF Abbott. Optimal degrees of synaptic connectivity. *Neuron*, 93(5):1153–1164, 2017.

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

[22] Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.

[23] Joshua Ainslie, James Lee-Thorp, Michiel De Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pages 4895–4901, 2023.

[24] Philippe Tillet, Hsiang-Tsung Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pages 10–19, 2019.

[25] Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations*, volume 2024, pages 35549–35562, 2024.

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

[27] Albert Q Jiang, A Sablayrolles, A Mensch, C Bamford, D Singh Chaplot, Ddl Casas, F Bressand, G Lengyel, G Lample, L Saulnier, et al. Mistral 7b. arxiv. *arXiv preprint arXiv:2310.06825*, 10(3), 2023.

[28] Mistral AI. Mistral-7B-v0.3. Hugging Face model card. Accessed September 10, 2026.

[29] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

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

[31] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

A Disclosure of AI Assistance

We used generative AI tools solely for language polishing and figure design, with assistance limited to improving the clarity and readability of the manuscript and the visual presentation of figures. Generative AI tools were not used for research ideation, method development, theoretical analysis, implementation of the proposed methods, experimental design or execution, data generation, or analysis and interpretation of results. The authors reviewed and verified all AI-assisted text and figures and take full responsibility for the final content of this paper.

B Algorithmic Summary

Algorithm 1 summarizes immediate and step-deferred selection. Let $\mathcal{R}$ contain the final W prompt indices and $\mathcal{G}_t$ the generated-token indices available at step t. The prompt budget B includes $\mathcal{R}$ but excludes generated history. For each layer and KV head, $\mathcal{A}_t$ contains the $B - W$ non-tail prompt indices selected from the current query, and $\mathcal{E}_t$ is the complete index set used by attention.

ROUTE scores the prompt sketch using the live query group. Raw mode returns the immediate selection $\mathcal{I}_t$; local mode applies right-neighbor average pooling and returns $\mathcal{D}_t$. Immediate selection fetches $\mathcal{A}_t$ before attention. Deferred execution begins with two raw immediate steps and one pooled immediate step, retaining the third selection as the working set for step four.

From step four, attention consumes $\mathcal{E}_t = \mathcal{A}_{t-1} \cup \mathcal{R} \cup \mathcal{G}_t$ while the auxiliary stream computes $\mathcal{A}_t$. Routing can overlap the current attention, but fetching waits for that layer's attention to finish reading the working-set buffer. The selected exact K/V then overwrite its routed portion in place, leaving the prompt tail and generated history unchanged. This fetch can overlap the remaining forward-pass work. Before the next step's attention reads the buffer, the main stream waits for ready_{t+1}, which marks fetch completion. No second exact-KV buffer is required. Bases, sketches, and host stores are implicit in the helper procedures; Appendix E.1 describes their construction.

C Experimental Settings

C.1 Models and Benchmarks

Table 3 lists the model configurations. All three models use four query heads per KV head. Experiments use BF16 weights and exact K/V, greedy decoding, and batch size one. The platform has a 96 GB RTX PRO 6000 GPU and an AMD EPYC 9755 CPU. Latency measurements use one GPU connected through PCIe 5.0 $\times$16, with offloaded exact K/V stored in pinned host memory.

Table 3: Model configurations used in the experiments. Configured positions refer to the checkpoint settings before any experiment-specific extension.

Model	Layers	Q heads	KV heads	Head dim.	Configured positions
Qwen3-8B	36	32	8	128	40,960
Llama-3.1-8B-Instruct	32	32	8	128	131,072
Mistral-7B-Instruct-v0.3	32	32	8	128	32,768

The main-text and cross-model LongBench comparisons use all 3,750 examples across the 16 tasks in Table 4, following the benchmark's task templates and scoring functions. Single- and multi-document QA, document summarization, and PassageRetrieval-en use the model's chat template. Few-shot, code, and PassageCount retain their answer-continuation format. Qwen3 runs with thinking disabled, and generation stops at EOS or the task-specific output limit.

Qwen3-8B and Llama-3.1-8B use a 130,000-token prompt cap, exceeding every evaluated prompt, so no examples are truncated. Mistral-7B uses middle truncation at 31,499 tokens, affecting 153 examples (4.1%), including 115 from NarrativeQA. All methods and the full-cache reference use the same input-processing policy within each model. LongBench applies no additional positional scaling. On Qwen3-8B, 41 prompts (1.1%; 38 NarrativeQA and three GovReport) exceed 40,960 configured

Algorithm 1 Immediate and step-deferred DeSKV decoding

Require: Prefill queries and keys; host KV stores; rank r; mixture α; prompt length P; budget B; tail size W; execution track τ

$\mathcal{R} \leftarrow \{P - W + 1, \ldots, P\}$, $\mathcal{G}_0 \leftarrow \emptyset$
for each layer ℓ and KV head h **do**
  Fit $U_{\ell,h}$ from the normalized moment in equation 8
  $(Z^{\text{int4}}_{\ell,h}, \theta_{\ell,h}) \leftarrow \text{Quantize}_{\text{INT4}}(K_{\ell,h}U_{\ell,h})$
  Retain exact prompt-tail K/V on the GPU
for $t = 1, 2, \ldots$ **do**
  $\mathcal{G}_t \leftarrow \mathcal{G}_{t-1} \cup \{P + t\}$
  for each layer ℓ **do**
    Compute all live queries $\{q_{\ell,g,t}\}_g$ and new K/V
    Append new exact K/V to the host stores and retain them on the GPU
    for each KV head h **do**
      $q \leftarrow q_{\ell,\mathcal{G}(h),t}$
      if $\tau =$ IMMEDIATE or $t < 3$ **then**
        $\mathcal{A}^{(\ell,h)}_t \leftarrow \text{Route}(q, \text{RAW})$
        Fetch exact K/V for $\mathcal{A}^{(\ell,h)}_t$ synchronously
        $\mathcal{E}^{(\ell,h)}_t \leftarrow \mathcal{A}^{(\ell,h)}_t \cup \mathcal{R} \cup \mathcal{G}_t$
      else if $t = 3$ **then**
        $\mathcal{A}^{(\ell,h)}_3 \leftarrow \text{Route}(q, \text{LOCAL})$
        Fetch exact K/V for $\mathcal{A}^{(\ell,h)}_3$ synchronously
        $\mathcal{E}^{(\ell,h)}_3 \leftarrow \mathcal{A}^{(\ell,h)}_3 \cup \mathcal{R} \cup \mathcal{G}_3$
        $\mathsf{ready}^{(\ell,h)}_4 \leftarrow \text{Done}$
      else
        $\text{Wait}(\mathsf{ready}^{(\ell,h)}_t)$ on the main stream
        $\mathcal{E}^{(\ell,h)}_t \leftarrow \mathcal{A}^{(\ell,h)}_{t-1} \cup \mathcal{R} \cup \mathcal{G}_t$
    Enqueue this layer's attention on the main stream using exact K/V indexed by $\{\mathcal{E}^{(\ell,h)}_t\}_h$
    if $\tau =$ DEFERRED and $t \geq 4$ **then**
      Record $e^{\text{attn}}_{\ell,t}$ after attention on the main stream
      for each KV head h **do**
        Launch $\text{PrepareNext}(q_{\ell,\mathcal{G}(h),t}, e^{\text{attn}}_{\ell,t})$ on the auxiliary stream ▷ $\mathcal{A}^{(\ell,h)}_t$ and $\mathsf{ready}^{(\ell,h)}_{t+1}$
function ROUTE(q, m)
  Compute prompt scores $\widetilde{s}_{1:P}$ using equation 10
  $s \leftarrow \text{AvgPool}_2(\widetilde{s})$ if $m =$ LOCAL; otherwise $s \leftarrow \widetilde{s}$
  return $\text{TopK}_{1 \leq i \leq P,\ i \notin \mathcal{R}}(s_i; B - W)$
function PREPARENEXT(q, e_{attn})
  $\mathcal{A} \leftarrow \text{Route}(q, \text{LOCAL})$
  Wait for e_{attn} on the auxiliary stream
  ready $\leftarrow \text{AsyncFetchExact}(\mathcal{A})$
  return $(\mathcal{A}, \text{ready})$

positions and use unscaled RoPE extrapolation. YaRN is used separately in depth–length runs above 32K and latency runs above 40,960 positions; the latter measure timing only.

For examples with multiple references, we take the highest reference score. TREC, TriviaQA, and SAMSum use the first output line. Task scores are multiplied by 100, and the LongBench average weights all 16 tasks equally before display rounding. Protected prompt entries count toward B; generated K/V are retained separately by every method.

MQ-NIAH inserts city–code facts at evenly spaced depths in WikiText-2 filler and asks for every planted code. The main experiment uses approximately 8.9K-token prompts and measures the fraction of codes recovered by exact substring matching. The output limit is 80 tokens for $K \leq 8$ and $8K + 40$ otherwise. Each setting is averaged over 24 prompts.

C.2 Calibration and Baseline Configurations

DeSKV fits one basis per prompt, layer, and KV head from post-RoPE activations. LongBench calibration uses the final 64 prompt positions from the four associated query heads, yielding 256

Table 4: LongBench datasets, metrics, and maximum generated-token counts.

Task	Examples	Metric	Output limit
NarrativeQA	200	QA F1	128
Qasper	200	QA F1	128
MultiFieldQA-en	150	QA F1	64
HotpotQA	200	QA F1	32
2WikiMQA	200	QA F1	32
MuSiQue	200	QA F1	32
GovReport	200	ROUGE-L	512
QMSum	200	ROUGE-L	512
MultiNews	200	ROUGE-L	512
TREC	200	Classification	64
TriviaQA	200	QA F1	32
SAMSum	200	ROUGE-L	128
PassageCount	200	Count score	32
PassageRetrieval-en	200	Retrieval score	32
LCC	500	Edit similarity	64
RepoBench-P	500	Edit similarity	64

query samples per KV head. For NIAH, the sampling window equals the tokenized question length and is also taken from the prompt tail. We sample the same number of keys uniformly from the prompt and fit the trace-normalized moment with $\alpha = 0.2$ and $r = 32$. This calibration window is separate from the 32 prompt-tail tokens retained for attention.

Projected keys are quantized independently by feature channel into INT4, with each channel's parameters shared across prompt positions within a layer and KV head. Immediate selection uses raw sketch scores. The deployed track uses two raw immediate steps and one pooled immediate step, followed by step-deferred selection with width-two right-neighbor average pooling. Scores are maximized over the query heads associated with each KV head.

Table 5 summarizes the baseline scoring rules and prompt-budget allocations. For head- or layer-adaptive methods, B denotes the average prompt budget per KV head. The 16-byte-per-token scoring payload refers to index codes or selected key components; metadata and full-cache residency are accounted for separately.

Table 5: Baseline configurations for the reported prompt-KV comparisons. Protected prompt entries count toward B. Page and block methods retain whole units, while adaptive allocations use B as an average budget.

Method	Scoring or allocation	Prompt-budget allocation
StreamingLLM	Fixed sink-and-tail rule	4 sinks; $B - 4$ tail tokens
H2O	Accumulated prefill attention	$B/2$ heavy hitters; $B/2$ tail tokens
SnapKV	Observation window 32; average pooling, width 5	32 prompt-token tail
PyramidKV	SnapKV scores; layer allocation, $\beta = 20$	32 prompt-token tail per layer
Ada-KV	Observation window 32; head allocation, floor 0.2; max pooling, width 7	32 prompt-token tail
Quest	32-token pages; first two layers use full attention	No forced tail; at least 3 pages
InfLLM	16-token blocks; one representative key	1 sink block; 2 local blocks
PQCache	$m = 16$ subvectors; 8-bit codes	2 sinks; 32 prompt-token tail
SparQ	8 query-selected key channels	32 prompt-token tail
Loki	8 components of an offline key-PCA basis	32 prompt-token tail

In the evaluated eviction paths, the retained prompt subset is selected once at the end of prefill and remains fixed during generation. Newly generated K/V are appended and always attended; no further eviction is performed. StreamingLLM retains four sink tokens and the $B - 4$ most recent prompt tokens. H2O accumulates each prompt key's causal attention mass over all prefill queries and retains $B/2$ heavy hitters plus $B/2$ recent prompt tokens. SnapKV scores keys from the last 32 prompt queries, applies width-five average pooling, and retains the top $B - 32$ keys outside that window. PyramidKV uses the same observation window and pooling kernel with its layer-wise allocation ($\beta = 20$). These three attention-based scorers sum scores across the four query heads sharing a KV head. Ada-KV instead averages the associated query heads over a 32-query observation window, applies width-seven max pooling, and redistributes the layer's prompt budget across KV heads with a minimum allocation of $0.2B$ per head. All retained prompt sets are represented by token-level masks.

Dynamic routers produce one selection per KV head. PQCache sums the associated query-head scores; the other evaluated routers take their maximum. Quest selects $\max(3, \lfloor B/32 \rfloor)$ whole 32-token pages per KV head, giving four pages at $B = 128$ and eight at $B = 256$. Its first two layers retain full-prompt attention. InfLLM selects $\lfloor B/16 \rfloor$ whole 16-token blocks, including one sink block and two local blocks. This leaves five routed blocks at $B = 128$ and thirteen at $B = 256$. Each block is represented by the key receiving the highest attention mass from that block's own queries. The evaluated attention paths retain the original RoPE positions without position remapping.

PQCache fits a separate codebook for each prompt, layer, and KV head using all prompt keys of that head. Each of its $m = 16$ subvectors has 256 centroids and an 8-bit code. Quality runs use batched k-means with up to 64 iterations and early stopping at tolerance 10^{-4}, followed by asymmetric-distance scoring summed over the query group. Two sinks and 32 recent prompt tokens are included within B. Construction-time measurements instead use PQCache's own system and its latency-budgeted iteration rule, with a minimum of three iterations. In the reported MQ-NIAH check, three iterations and fitting to convergence give the same retrieval quality.

Loki fits its basis offline from 16 WikiText-2 sequences of 4,096 tokens using post-RoPE keys. Its rotated keys and unprojected values remain on the GPU. SparQ also retains the full K/V cache and is evaluated without mean-value correction. In contrast, InfLLM, PQCache, and DeSKV keep full exact K/V in host memory. Quest's latency measurements use batched PyTorch evaluation of the page min/max bound, with `einsum` over the query group followed by page top-k selection. The released CUDA kernels could not be built on the `sm_120`/CUDA 13 platform. The page statistics and selection rule are retained; reported latency refers to this custom implementation.

D Additional Evaluation

D.1 LongBench across Models and Budgets

Table 6 and 7 extend the LongBench comparison to Llama-3.1-8B-Instruct and Mistral-7B-Instruct-v0.3 at prompt-KV budgets $B \in \{128, 256\}$. Each table reports all 16 task scores and their equal-weight average. As in the main text, retained prompt entries count toward B and generated history is retained separately.

On Llama, the deployed method achieves averages of 48.59 and 48.70, compared with 48.56 for full cache. Immediate selection reaches 49.18 and 48.99, so the additional decrease from deferral is smaller at the larger budget. Overall quality remains close to full cache, with gains on 2WikiMQA and RepoBench-P alongside lower GovReport scores.

Mistral shows a larger gap at both budgets: deployed averages are 43.64 and 43.61 against 46.58 for full cache. SparQ achieves the highest average among the evaluated routers. Immediate sketch selection also trails full cache, with deferral lowering its average by a further 0.53 and 0.46 points. The budget response is task-dependent: increasing B raises deployed 2WikiMQA from 30.19 to 34.84 but lowers PassageRetrieval-en from 91.25 to 82.67. The larger gap and mixed budget response therefore also appear outside deferred execution.

We examine this model dependence with a separate replay on eight prompts each from HotpotQA, 2WikiMQA, and TREC. Greedy decoding runs to EOS or 32 steps, averaging about 14 steps per prompt. For each KV head and step, we first compute $s_j = \max_{g \in \mathcal{G}(h)} q_g^\top k_j$ across its four query heads, then define the diagnostic distribution $a_j = \mathrm{softmax}_j(s/\sqrt{d})$ over prompt tokens only.

Table 6: LongBench task scores on Llama-3.1-8B-Instruct. DeSKV denotes the deployed schedule. Avg. gives equal weight to the 16 tasks; task scores use one decimal and averages two. Bold marks the highest budgeted score per column, including ties at the displayed precision.

	Single-Doc. QA			Multi-Doc. QA			Summarization			Few-shot			Synthetic		Code		
Method	*NrtvQA*	*Qasper*	*MFQA-en*	*HotpotQA*	*2WikiMQA*	*MuSiQue*	*GovReport*	*QMSum*	*MultiNews*	*TREC*	*TriviaQA*	*SAMSum*	*PCount*	*PRe*	*LCC*	*RB-P*	Avg.
Full cache	29.3	44.9	56.1	57.6	49.0	33.2	34.4	25.5	26.9	72.5	91.7	43.9	6.0	100.0	54.3	51.8	48.56
Prompt-KV budget $B = 128$ per KV head																	
Eviction																	
StreamingLLM	20.9	21.6	31.1	44.1	40.4	21.6	18.0	21.4	18.1	26.0	56.7	37.9	5.8	99.5	51.9	48.5	35.21
H2O	26.0	28.6	35.8	48.3	44.3	26.8	24.7	22.9	23.5	42.0	77.4	37.4	6.4	98.5	49.9	47.6	40.01
PyramidKV	26.2	26.3	50.4	54.5	47.6	26.2	18.5	22.9	19.3	31.0	87.2	39.5	4.9	98.5	49.3	48.7	40.68
SnapKV	25.8	23.8	48.5	56.4	47.4	28.7	18.5	22.5	19.1	31.0	85.6	39.0	4.1	98.5	49.9	50.3	40.57
Ada-KV	25.1	25.0	49.9	55.4	48.3	27.5	19.4	23.0	19.8	32.5	88.1	39.7	4.5	99.5	52.1	53.1	41.43
Routing																	
Quest	17.4	25.3	38.2	38.6	28.9	20.4	26.6	20.7	25.2	20.5	59.4	19.3	6.8	85.6	40.8	41.3	32.18
Loki	9.2	24.8	29.7	29.0	24.1	10.2	13.5	22.2	21.7	33.5	47.5	36.0	2.8	74.0	31.9	31.0	27.56
InfLLM	29.7	38.4	51.5	54.6	44.6	31.4	27.6	23.8	24.2	36.5	85.3	41.6	2.9	94.5	52.5	52.3	43.22
SparQ	**31.1**	42.3	56.6	54.9	50.4	33.1	32.3	**25.0**	26.2	57.5	91.8	42.6	7.0	99.5	**56.1**	53.7	47.50
PQCache	29.2	44.4	56.9	56.0	50.6	32.7	**34.3**	24.9	**27.0**	69.5	91.0	44.0	4.1	**100.0**	55.8	55.0	48.46
DeSKV (imm.)	29.5	**44.8**	**57.0**	**58.3**	52.6	**34.8**	33.7	24.6	26.6	**70.0**	**92.6**	**44.4**	**7.1**	99.5	56.0	**55.4**	**49.18**
DeSKV	29.6	44.7	56.1	57.5	**52.9**	34.4	30.7	24.9	25.7	**70.0**	92.0	43.1	6.7	**100.0**	54.9	54.2	48.59
Prompt-KV budget $B = 256$ per KV head																	
Eviction																	
StreamingLLM	21.5	23.7	32.9	42.9	40.9	21.0	20.6	21.1	20.2	35.5	64.7	40.0	6.6	98.5	54.0	50.0	37.12
H2O	26.0	31.2	42.1	52.4	47.4	29.2	25.6	22.5	24.1	48.0	82.1	40.9	3.3	97.5	52.1	51.0	42.21
PyramidKV	28.1	31.9	53.9	55.1	48.6	28.8	21.5	23.7	21.5	42.5	89.2	41.3	3.8	99.5	51.4	50.8	43.22
SnapKV	27.6	32.1	52.8	56.0	49.9	29.5	21.6	23.6	21.6	39.5	91.4	41.1	4.6	99.0	53.3	53.0	43.54
Ada-KV	26.5	34.0	54.4	56.8	49.7	29.0	22.2	23.7	21.6	45.0	91.4	41.3	7.0	99.8	53.8	54.6	44.42
Routing																	
Quest	22.3	34.9	45.4	47.0	39.5	25.7	30.6	22.2	26.3	36.0	75.2	25.2	6.8	96.2	48.1	49.3	39.41
Loki	15.1	32.0	40.5	44.4	32.2	19.3	20.4	23.8	26.1	48.0	71.2	40.6	4.2	92.5	41.5	34.5	36.65
InfLLM	29.4	42.8	55.6	57.3	48.0	31.8	29.8	24.5	25.1	52.5	90.1	42.8	3.3	94.5	54.9	54.5	46.06
SparQ	29.4	43.9	**57.8**	56.2	51.3	33.9	33.8	25.0	26.7	63.0	91.6	44.0	6.2	**100.0**	55.6	55.1	48.34
PQCache	**30.6**	44.7	56.4	56.6	**52.2**	32.3	**34.8**	25.4	26.7	**71.0**	**92.4**	44.1	5.1	**100.0**	**55.8**	55.1	48.95
DeSKV (imm.)	29.1	**45.0**	56.2	**57.7**	50.8	**34.1**	34.3	**25.5**	**26.8**	69.0	92.1	**44.4**	**8.4**	99.5	**55.8**	**55.3**	**48.99**
DeSKV	29.3	44.0	56.7	57.5	51.5	33.6	32.5	25.1	26.5	69.0	92.2	43.9	7.5	**100.0**	55.3	54.6	48.70

Table 8 reports overlap with the exact top-128 set and the mass $\sum_{j \in \mathcal{S}} a_j$ captured by each 128-token selection. Exact, sketch, and SparQ sets use their raw top-128 scores without forcing the last prompt tail. Generated history is excluded, and results are averaged over layers, KV heads, decode steps, and prompts. This probe isolates the routing representation from the deferred schedule.

Exact top-128 selection captures 79.98% of this diagnostic mass on Mistral, compared with 84.32% on Llama and 86.39% on Qwen3. INT4 sketch overlap is similar on Mistral and Llama (61.72% and 61.25%), and sketch mass coverage is about three percentage points below exact selection in all three models. On these probe prompts, the same selection size therefore covers less mass on Mistral without a corresponding loss in exact-set recovery. This suggests greater sensitivity to the working-set constraint even when the sketch approximates the exact ranking comparably.

D.2 Needle Retrieval across Context Lengths and Depths

We evaluate single-needle retrieval across context lengths and insertion depths on Qwen3-8B, complementing the fact-count sweep in MQ-NIAH. Budgeted methods use the $B = 128$ prompt-KV setting per KV head, with method-specific configurations described in Appendix C.2; full cache provides an unbudgeted reference. For DeSKV, the budget comprises 96 selected prompt tokens and a fixed 32 prompt-token tail. We evaluate both immediate selection and the deployed three-step priming and Avg2 configuration.

Each prompt contains a single Zurich–access-code statement inserted into concatenated WikiText-2 text, followed by a question requesting the code. We sweep ten target prompt lengths, $L \in$

Table 7: LongBench task scores on Mistral-7B-Instruct-v0.3. DeSKV denotes the deployed schedule. Avg. gives equal weight to the 16 tasks; task scores use one decimal and averages two. Bold marks the highest budgeted score per column, including ties at the displayed precision.

	Single-Doc. QA			Multi-Doc. QA			Summarization			Few-shot			Synthetic		Code		
Method	*NrtvQA*	*Qasper*	*MFQA-en*	*HotpotQA*	*2WikiMQA*	*MuSiQue*	*GovReport*	*QMSum*	*MultiNews*	*TREC*	*TriviaQA*	*SAMSum*	*PCount*	*PRe*	*LCC*	*RB-P*	Avg.
Full cache	26.8	37.9	50.2	51.1	38.9	28.2	34.0	25.7	26.6	76.0	88.6	47.2	6.0	98.0	53.4	56.7	46.58
Prompt-KV budget $B = 128$ per KV head																	
Eviction																	
StreamingLLM	17.4	18.5	28.6	36.6	28.1	17.2	17.9	19.6	16.9	26.0	79.5	40.5	4.5	51.0	47.2	49.1	31.16
H2O	20.6	25.6	33.4	28.8	28.5	11.8	25.3	21.7	23.5	50.0	84.3	39.9	1.5	66.8	47.7	46.5	34.74
PyramidKV	21.5	22.6	42.5	43.7	32.1	18.9	21.1	21.9	19.9	23.0	88.4	43.0	4.5	**93.0**	47.9	49.3	37.09
SnapKV	20.3	23.0	43.2	41.7	31.8	19.6	21.0	22.1	19.3	27.5	87.2	42.7	4.0	91.0	47.8	49.3	36.97
Ada-KV	21.0	24.6	44.7	44.9	32.4	21.0	21.5	21.8	20.0	31.0	87.9	43.0	**6.5**	90.0	49.3	51.5	38.19
Routing																	
Quest	10.9	19.8	37.3	33.8	22.6	14.2	26.5	21.0	25.3	29.5	74.6	24.6	6.0	71.5	42.5	41.9	31.37
Loki	15.9	21.0	36.7	29.3	21.1	12.6	27.7	22.7	**27.2**	55.0	83.5	39.7	2.7	54.5	45.5	45.7	33.79
InfLLM	24.2	34.0	46.4	**49.0**	34.9	23.6	27.4	23.0	24.1	41.5	85.5	44.4	4.5	74.5	49.2	51.1	39.83
SparQ	**27.7**	36.0	**50.3**	45.0	**38.2**	24.4	32.7	24.4	26.9	70.0	**89.4**	44.9	5.5	89.0	52.8	54.8	**44.49**
PQCache	25.8	**36.7**	49.4	38.9	28.4	18.7	32.4	23.6	26.7	**74.0**	88.9	46.8	4.5	91.9	53.3	55.7	43.49
DeSKV (imm.)	24.8	33.6	48.2	41.7	31.1	**26.5**	**33.9**	24.6	26.7	73.0	89.1	**47.0**	4.0	90.0	**54.4**	**58.1**	44.17
DeSKV	25.9	32.3	47.7	42.3	30.2	25.3	29.7	**24.8**	26.0	73.0	**89.4**	46.2	4.0	91.3	53.3	57.0	43.64
Prompt-KV budget $B = 256$ per KV head																	
Eviction																	
StreamingLLM	18.4	19.9	26.4	38.4	29.0	17.6	19.7	19.9	19.5	36.5	81.1	43.3	3.5	50.5	50.5	51.0	32.83
H2O	21.9	30.0	37.9	36.4	31.5	14.6	26.2	21.6	24.3	52.5	85.6	43.3	2.5	71.0	50.3	49.7	37.45
PyramidKV	23.6	27.4	47.2	45.9	34.8	22.7	22.8	22.5	21.5	35.5	**89.3**	44.9	4.0	95.0	50.3	52.9	40.01
SnapKV	24.1	28.0	46.2	45.4	33.6	22.6	23.2	22.6	22.3	43.0	89.1	43.9	4.5	**97.5**	50.6	54.1	40.65
Ada-KV	24.3	29.9	48.5	47.2	34.0	25.9	23.0	22.7	22.3	52.5	88.1	43.3	4.0	94.0	52.0	54.1	41.61
Routing																	
Quest	12.7	27.9	41.9	41.8	29.7	21.3	29.8	21.5	25.9	48.5	84.4	30.5	**6.5**	88.0	47.0	49.0	37.89
Loki	19.1	28.1	43.5	31.3	24.3	13.9	30.6	24.0	**27.6**	66.0	87.3	42.3	3.6	69.5	48.6	48.4	38.00
InfLLM	26.3	36.4	50.2	**48.3**	37.1	25.0	29.5	24.0	25.7	58.5	87.4	43.7	6.0	69.5	52.4	54.9	42.18
SparQ	**28.5**	**37.6**	**51.6**	45.4	**38.4**	**27.1**	33.1	**25.2**	26.6	70.0	87.8	45.9	4.0	91.2	54.4	54.9	**45.10**
PQCache	26.7	35.7	50.1	41.6	32.3	19.5	33.2	23.7	26.7	**74.5**	89.2	46.5	5.0	93.0	**55.0**	57.4	44.38
DeSKV (imm.)	23.5	36.7	49.9	42.4	35.7	21.4	**33.8**	24.9	26.7	73.0	**89.3**	**47.4**	5.0	82.7	54.4	**58.4**	44.07
DeSKV	22.4	35.6	50.5	42.6	34.8	21.9	31.5	24.8	25.9	73.0	89.1	46.7	5.0	82.7	53.9	57.4	43.61

Table 8: Exact-set overlap and diagnostic mass coverage across models. The probe uses eight prompts each from HotpotQA, 2WikiMQA, and TREC, with up to 32 greedy decode steps. Overlap is agreement with exact top-128 selection. Mass sums the prompt-token softmax probabilities obtained *after* GQA max reduction over the selected positions. Both are percentages. Selections exclude generated history and do not force the prompt tail; sketch rows are evaluated without step deferral.

	Qwen3-8B		Llama-3.1-8B		Mistral-7B	
Selection	Overlap	Mass	Overlap	Mass	Overlap	Mass
Exact top-128	100.00	86.39	100.00	84.32	100.00	79.98
Sketch ($r = 32$, FP16)	73.74	84.44	72.68	82.61	73.05	78.40
Sketch ($r = 32$, INT4)	66.60	83.41	61.25	81.20	61.72	76.96
SparQ (8 channels)	42.54	74.23	40.82	72.80	38.71	68.73

$\{4, 8, 12, 16, 24, 32, 48, 64, 96, 128\}$K, and eleven insertion depths from 0% to 100% in 10% increments. Length counts the complete input, including the haystack, needle, and question; depth specifies the insertion position as a fraction of the haystack's characters. Each cell contains eight prompts with different haystack cuts and four-letter access codes, shared across all methods. This gives 110 cells and 880 evaluations per method. Greedy decoding stops at EOS or 80 generated tokens, and an output is correct when it contains the target code as a case-sensitive exact substring.

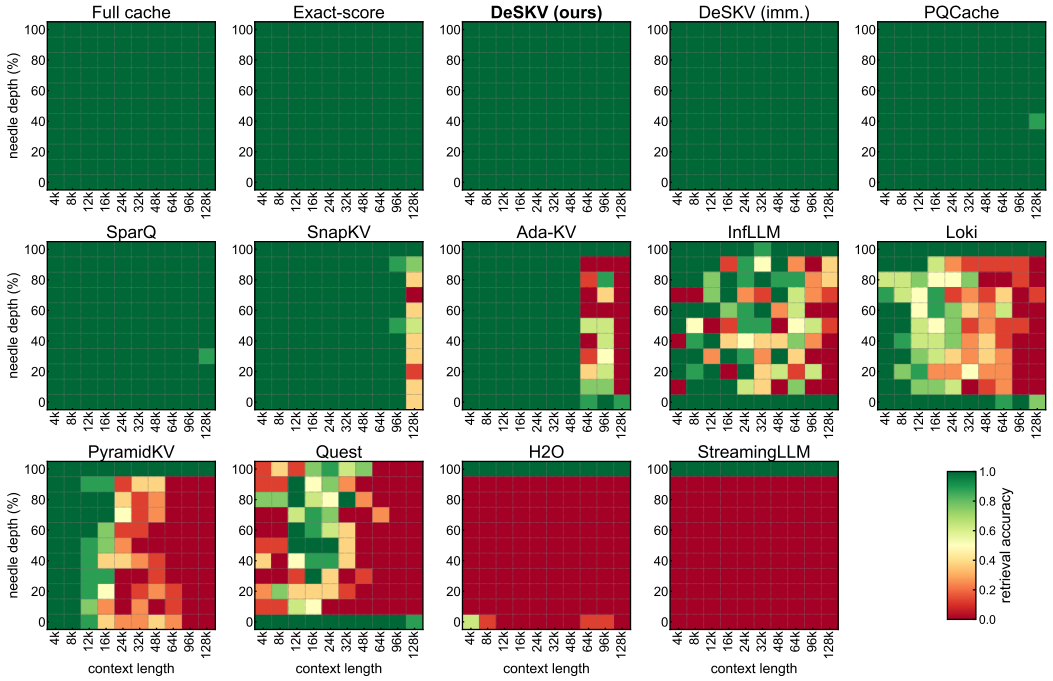

Figure 7: Single-needle retrieval across context lengths and insertion depths on Qwen3-8B at the $B = 128$ prompt-KV setting. Length denotes the target total input length, and depth is the insertion position within the haystack. DeSKV denotes the deployed schedule; Exact-score uses immediate selection from exact attention scores. Full cache is unbudgeted.

Above 32K, all methods and references use YaRN with factors 2 at 48K and 64K, 3 at 96K, and 4 at 128K.

Figure 7 shows that both immediate and deployed DeSKV recover all 880 target codes, matching Full cache and Exact-score in every cell. PQCache and SparQ also achieve nearly complete retrieval, each recovering 879 of 880 codes. Their single misses occur at 128K, at depths 40% and 30%, respectively. Thus, the compact sketch and deployed schedule preserve full-cache single-fact retrieval across the tested length–depth grid.

Other methods exhibit stronger length or position dependence. SnapKV recovers 828 of 880 codes, with most failures at 128K, whereas Ada-KV recovers 713 and begins to lose coverage at 64K. InfLLM, Loki, PyramidKV, and Quest show broader regions of reduced accuracy. H2O and StreamingLLM succeed predominantly near the prompt end. At 100% depth, the needle immediately precedes the final question and partly overlaps the protected prompt tail, making this endpoint accessible to tail retention. In contrast, DeSKV succeeds at interior positions as well as at this endpoint, maintaining retrieval coverage as the prompt grows under the fixed budget.

D.3 Projection Rank and Quantization Precision

Figure 8 compares ranks $r \in \{8, 16, 24, 32, 40, 48, 56, 64\}$ and FP16, INT8, INT4, and INT2 sketches on Qwen3-8B, Llama-3.1-8B, and Mistral-7B. Each model uses 16 prompts each from TriviaQA, MultiFieldQA-en, and SAMSum, totaling 48 prompts. With $\alpha = 0.2$, each KV-head basis is fitted from the last 128 prompt positions of its four query heads (512 query rows) and 512 prompt keys sampled uniformly at random. Evaluation uses the final 50 of 100 decode steps.

Let Q stack these decode queries across the associated query heads, K contain all prompt keys, and $\widehat{Z}$ denote the projected keys after the stated quantization and dequantization. We compare $S = QK^\top$ with $\widetilde{S} = (QU)\widehat{Z}^\top$ in FP32 and report normalized score error $\|S - \widetilde{S}\|_F^2 / \|S\|_F^2$. For an unquantized

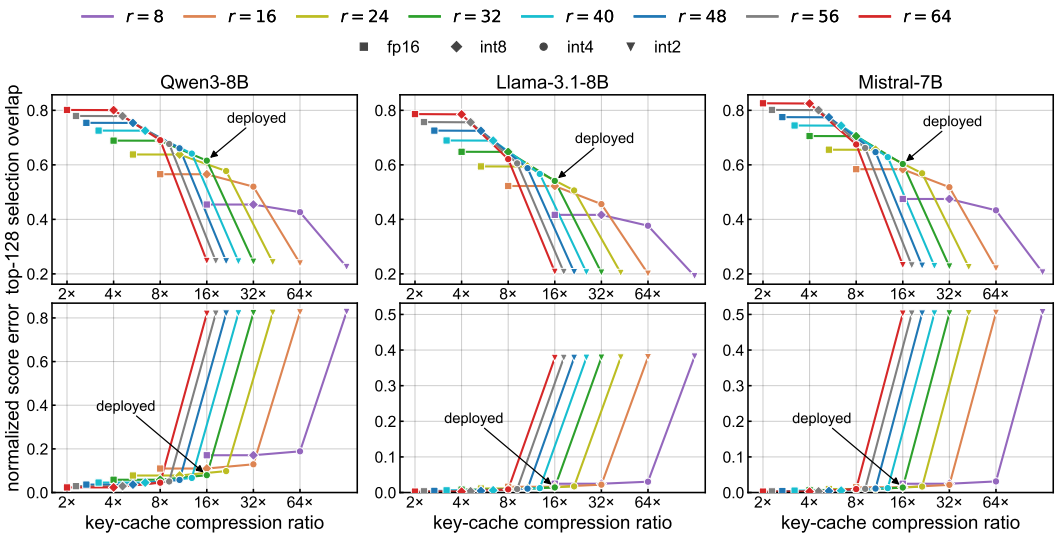

Figure 8: Projection rank and quantization precision across three models. Top: top-128 selection overlap. Bottom: normalized score error. Colors indicate rank and markers indicate precision. Compression ratios compare 16-bit keys with sketch codes, excluding metadata; arrows mark the deployed rank-32 INT4 configuration.

projection, the score residual reduces to $Q(I - UU^\top)K^\top$. Top-128 overlap is computed per step and averaged over the same 50 steps. A rank-r, b-bit sketch uses $rb/8$ code bytes per token, giving $16d/(rb)$ compression relative to a 16-bit key, excluding the basis and quantization metadata.

At fixed rank, FP16 and INT8 give similar metrics. INT4 lowers overlap more noticeably, while INT2 sharply reduces overlap and increases score error. Increasing rank improves overlap at FP16, INT8, and INT4, but recovers little at INT2: more projection directions do not offset this precision loss.

At the same $16\times$ code compression, rank-32 INT4 gives higher overlap than rank-8 FP16, rank-16 INT8, and rank-64 INT2 in all three models. These configurations allocate the same code budget to different numbers of directions and bits per direction. Their comparison supports rank-32 INT4 as a balance between projection capacity and precision.

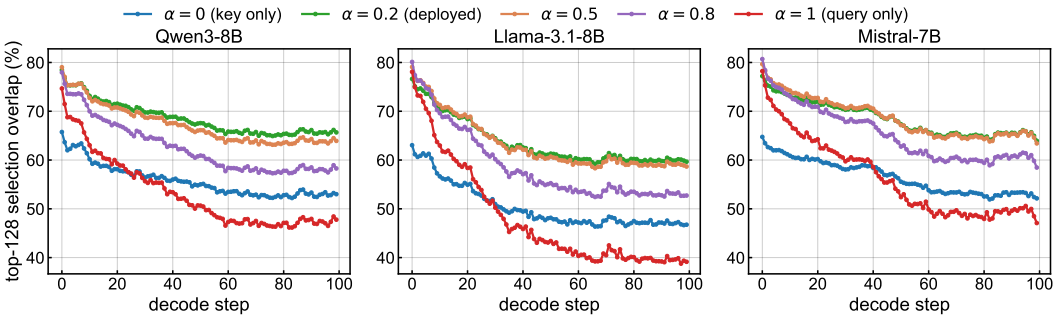

Figure 9: Decode-time routing overlap across three models. Each panel reports overlap with exact-score top-128 selection over the first 100 decode steps for five query–key mixture weights. The deployed mixture uses $\alpha = 0.2$.

D.4 Sensitivity to the Query–Key Mixture

D.4.1 Prefill-to-Decode Transfer across Models

We vary the query–key mixture across the three model families, using the same 48 prompts per model and 512-query/512-key calibration procedure as Appendix D.3. Figure 9 traces exact-score top-128 overlap over 100 decode steps for five mixture weights.

Query-only fitting starts with high overlap but declines more strongly during decoding. Joint mixtures with $\alpha = 0.2$ or 0.5 retain higher late-step overlap than either single-source fit in all three models.

Figure 10 compares the calibration queries with queries from the final 50 of 100 decode steps. We sweep α from 0 to 1 in increments of 0.1 at $r \in \{8, 16, 32, 64\}$, computing top-128 overlap per step before averaging. Higher rank reduces score error and raises overlap. At $r = 32$, decode overlap favors intermediate mixtures, with a broader preferred range on Mistral, whereas prefill overlap generally favors greater query weight. The mixture preferred on fitting queries thus differs from that preferred on later queries.

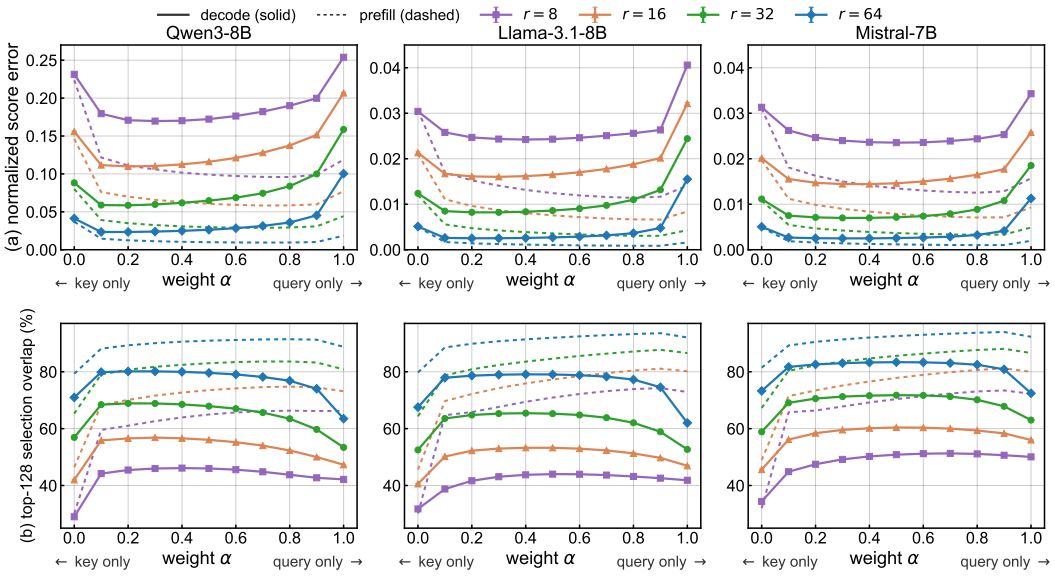

Figure 10: Query–key mixture and rank across three models. Top: normalized score error. Bottom: exact-score top-128 selection overlap. Solid and dashed curves report decode and prefill queries, respectively; colors indicate projection rank.

D.4.2 Routing over Longer Generation

We extend the replay to 512 greedy decode steps on Qwen3-8B using 16 prompts each from GovReport and QMSum. Each basis is fitted once during prefill and evaluated with immediate selection without pooling. Figure 11(a) traces the generation trajectory; panel (b) averages overlap over steps 462–511 at ranks 16, 32, and 64.

After the initial decline, the deployed mixture stabilizes near 67% overlap. Query-only fitting falls to around 40% before a modest late recovery, while key-only fitting remains flatter but below the joint mixture. The preferred late-window mixture remains around $\alpha = 0.1$–0.3 across the three ranks, extending the benefit of joint fitting beyond the first 100 steps on these summarization prompts.

D.4.3 Downstream Sensitivity

Table 9 evaluates immediate sketch selection on Qwen3-8B across all 16 LongBench tasks. We use the first 64 examples per task in dataset-file order and vary $\alpha \in \{0, 0.1, 0.2, 0.3, 0.5, 0.8, 1\}$ at $B \in \{128, 256\}$. Exact-score immediate selection and full cache are evaluated on the same subsets.

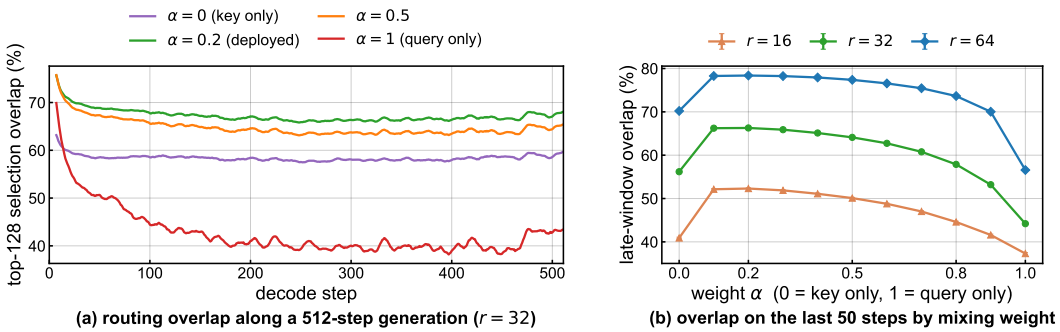

Figure 11: Routing quality over 512 decode steps on Qwen3-8B. Results use 16 prompts each from GovReport and QMSum. (a) Exact-score top-128 selection overlap at $r = 32$. (b) Mean overlap over steps 462–511 versus the query weight at $r \in \{16, 32, 64\}$. Selection is evaluated immediately without pooling.

Key-weighted joint mixtures give the highest subset means at both budgets: 47.69 with $\alpha = 0.2$ at $B = 128$ and 48.14 with $\alpha = 0.3$ at $B = 256$. The default $\alpha = 0.2$ improves over key-only fitting by approximately 3.0 and 1.5 points, respectively, and remains close to the best mean at $B = 256$ with a score of 48.01.

The preferred weight nevertheless varies across tasks. At $B = 128$, moving from key-only fitting to $\alpha = 0.2$ raises MuSiQue from 23.15 to 36.02 and RepoBench-P from 40.20 to 51.39. MultiFieldQA-en and 2WikiMQA instead achieve their highest scores with query-only fitting at this budget, while MultiNews favors key-only fitting at both budgets. The subset averages thus favor a key-weighted joint mixture, while individual tasks benefit from different query–key balances.

D.5 Contributions of Priming and Local Expansion

Table 10 compares one and three priming steps with no pooling, centered width-three max pooling, or right-neighbor width-two average pooling. We evaluate Qwen3-8B on the first 64 examples of each of six LongBench tasks in dataset-file order, using identical subsets for all variants. The prompt cap is 130,000 tokens. Here, T_p counts initial immediate-selection steps, and Mean6 gives equal weight to the six task scores. Full cache, exact-score immediate selection, and raw sketch immediate selection provide references.

The benefit of pooling depends on the priming length. Without pooling, extending priming from one to three steps raises Mean6 by 0.67 and 0.95 points at $B = 128$ and 256, respectively, largely through MultiFieldQA-en. Additional priming alone is not uniformly beneficial across tasks. At $B = 128$, Avg2 slightly lowers the one-prime mean from 59.76 to 59.64, but raises the three-prime mean from 60.43 to 61.73. At $B = 256$, it improves the three-prime mean from 61.89 to 63.02. Local pooling is therefore more effective after the longer priming sequence in these comparisons.

With three priming steps, Avg2 and Max3 yield similar means at $B = 128$ (61.73 versus 61.69), while Avg2 gives a higher mean at $B = 256$ (63.02 versus 62.30). Their task-level effects differ: Avg2 scores higher on GovReport at both budgets, whereas Max3 scores higher on MultiFieldQA-en and RepoBench-P at $B = 128$. The deployed combination reduces the loss from deferred execution, leaving gaps of 0.85 and 0.44 points to immediate sketch selection at the two budgets.

Table 11 and 12 compare outputs under exact-score immediate selection, raw sketch immediate selection, one-step priming without pooling, and the deployed three-step Avg2 schedule. The examples show complete execution choices; the matched comparisons in Table 10 separate the individual components.

Table 11 shows details recovered by the deployed schedule: “other toxic products” in MultiFieldQA-en and “when he gets home” in SAMSum are absent from the one-prime outputs. In RepoBench-P, deployed identifies the reference sound object at $B = 128$ but omits division by `100f`; at $B = 256$,

Table 9: Downstream sensitivity to the query–key mixture on Qwen3-8B. Results use the **first 64 examples per task**; the same examples for every column. All mixture settings and references use the same subsets. Immediate sketch selection is evaluated at prompt-KV budgets $B \in \{128, 256\}$ per KV head. The mean gives equal weight to the 16 subset-level task scores. Here, $\alpha = 0$ and 1 use key-only and query-only fitting, respectively; $\star$ marks the default $\alpha = 0.2$.

	DeSKV immediate: query weight α							References	
Task	0	0.1	$0.2^\star$	0.3	0.5	0.8	1	Exact score	Full cache
Prompt-KV budget $B = 128$ per KV head									
NarrativeQA	21.82	27.79	27.52	27.27	**30.29**	27.75	28.35	28.33	25.20
Qasper	45.91	47.02	**48.04**	45.75	47.76	47.96	46.58	47.53	45.81
MultiFieldQA-en	56.56	58.69	56.82	58.38	57.76	58.39	**59.26**	58.77	59.18
HotpotQA	43.52	**54.72**	51.15	51.60	51.59	51.16	53.33	54.11	53.46
2WikiMQA	36.07	37.72	36.86	37.35	37.00	37.51	**39.60**	36.47	41.06
MuSiQue	23.15	31.03	**36.02**	35.33	31.12	32.95	32.10	33.32	34.04
GovReport	34.24	33.94	**34.33**	34.32	34.07	34.22	32.04	33.68	33.90
QMSum	23.69	23.71	24.32	23.51	24.35	24.20	**24.53**	23.90	24.08
MultiNews	**24.91**	24.11	23.92	23.75	23.98	23.99	23.80	23.99	24.08
TREC	68.75	**70.31**	**70.31**	68.75	68.75	67.19	**70.31**	70.31	68.75
TriviaQA	88.07	**89.17**	88.65	88.65	88.65	88.65	88.65	89.17	90.73
SAMSum	42.16	**44.97**	44.30	44.16	44.36	44.55	44.76	45.52	43.14
PassageCount	6.12	6.60	**7.41**	4.45	5.45	5.13	5.80	5.69	6.45
PassageRetrieval-en	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	100.00	100.00
LCC	59.98	61.50	62.05	61.41	**62.17**	60.81	61.02	60.94	62.28
RepoBench-P	40.20	51.25	**51.39**	51.09	50.23	50.45	51.11	52.19	49.73
Mean (16 tasks)	44.70	47.66	**47.69**	47.24	47.35	47.18	47.58	47.75	47.62
Prompt-KV budget $B = 256$ per KV head									
NarrativeQA	25.25	26.85	27.86	26.40	**28.52**	27.82	27.56	26.56	25.20
Qasper	44.91	45.79	45.51	46.90	46.06	**46.95**	46.09	46.08	45.81
MultiFieldQA-en	55.63	59.29	59.50	**59.91**	58.73	58.92	58.59	60.19	59.18
HotpotQA	48.34	52.38	**53.94**	52.38	51.16	**53.94**	51.34	53.94	53.46
2WikiMQA	35.67	38.91	38.39	39.95	**40.48**	38.45	38.91	38.43	41.06
MuSiQue	26.19	33.52	34.19	36.41	**36.80**	35.93	33.59	34.77	34.04
GovReport	**35.14**	34.65	34.56	33.97	34.69	33.85	33.59	33.61	33.90
QMSum	23.58	24.24	24.65	24.12	24.60	24.25	**24.77**	24.30	24.08
MultiNews	**24.52**	24.40	24.31	24.29	24.30	24.04	23.80	24.11	24.08
TREC	68.75	70.31	**71.88**	70.31	68.75	68.75	68.75	68.75	68.75
TriviaQA	**91.51**	89.17	89.17	89.17	89.17	89.17	88.65	89.17	90.73
SAMSum	45.04	44.68	44.74	44.82	**45.05**	44.42	44.04	44.88	43.14
PassageCount	6.43	**7.65**	4.17	6.05	6.86	2.96	5.30	5.20	6.45
PassageRetrieval-en	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	100.00	100.00
LCC	**63.83**	61.61	62.44	62.55	61.66	61.95	62.69	62.12	62.28
RepoBench-P	49.80	52.12	52.78	**53.03**	52.97	51.31	52.09	50.69	49.73
Mean (16 tasks)	46.54	47.85	48.01	**48.14**	48.11	47.67	47.48	47.67	47.62

it also preserves that scaling. The budget change thus affects a specific argument in the generated continuation, not only its aggregate score.

Table 12 show the contrasting outcomes. In RepoBench-P, deployed generates `edit`, while exact-score and immediate selection generate the reference API, `add_reaction`, with a different emoji constant. In TriviaQA, all four modes produce the same incorrect answer. The final SAMSum example receives a higher benchmark score despite naming the wrong driver. These cases distinguish recovered details from remaining continuation errors and from differences between benchmark scores and semantic agreement with the reference.

D.6 Sensitivity to Priming Length

Figure 12 varies $T_p \in \{1, \ldots, 8\}$ at $B \in \{128, 256\}$ on Qwen3-8B. We use all 200 TriviaQA, 150 MultiFieldQA-en, and 200 SAMSum examples, together with the first 200 of 500 RepoBench-P examples in dataset-file order. No prompts are truncated under the 130,000-token cap.

Table 10: Contributions of priming and local expansion. Scores on a six-task LongBench subset (**first 64 examples per task**; the same examples for every row); Mean6 gives its equal-weight average. T_p is the number of initial immediate-selection steps. Max3 uses centered width-three max pooling; Avg2 averages each score with its immediate right neighbor.

Selection	T_p	Pool	TriviaQA	MFQA-en	SAMSum	PsgRet-en	RB-P	GovReport	Mean6
Full cache			90.73	59.18	43.14	100.00	49.73	33.90	62.78
Prompt-KV budget $B = 128$ per KV head									
Exact-score (immediate)			89.17	58.77	45.52	100.00	52.19	33.68	63.22
Sketch (immediate, raw)			88.65	56.82	44.30	100.00	51.39	34.33	62.58
Deferred	1	None	88.96	50.67	41.82	**100.00**	47.34	29.79	59.76
Deferred	3	None	88.85	55.85	41.38	**100.00**	46.88	29.64	60.43
Deferred	1	Max3	**89.17**	55.92	41.80	98.44	49.38	30.04	60.79
Deferred	3	Max3	88.85	**59.15**	41.86	**100.00**	**50.11**	30.19	61.69
Deferred	1	Avg2	85.26	53.82	40.95	98.44	46.16	33.23	59.64
Deployed	3	Avg2	88.52	57.34	**42.16**	**100.00**	48.98	**33.37**	**61.73**
Prompt-KV budget $B = 256$ per KV head									
Exact-score (immediate)			89.17	60.19	44.88	100.00	50.69	33.61	63.09
Sketch (immediate, raw)			89.17	59.50	44.74	100.00	52.78	34.56	63.46
Deferred	1	None	88.96	53.48	42.86	**100.00**	48.23	32.09	60.94
Deferred	3	None	88.65	58.57	42.57	**100.00**	49.66	31.91	61.89
Deferred	1	Max3	88.85	55.77	43.37	**100.00**	49.72	32.36	61.68
Deferred	3	Max3	**89.17**	60.34	42.29	**100.00**	49.94	32.08	62.30
Deferred	1	Avg2	88.33	53.79	43.23	**100.00**	**50.67**	33.88	61.65
Deployed	3	Avg2	88.85	**61.27**	**44.50**	**100.00**	49.19	**34.28**	**63.02**

The pooling rule is fixed to width-two right-neighbor averaging. Steps $1, \ldots, T_p - 1$ use raw immediate selection, and step T_p uses pooled immediate selection to initialize the working set for deferred execution. For $T_p = 1$, the single priming step is therefore pooled. From step $T_p + 1$, the same pooling rule is used with the deferred schedule. The immediate reference uses raw same-step selection throughout.

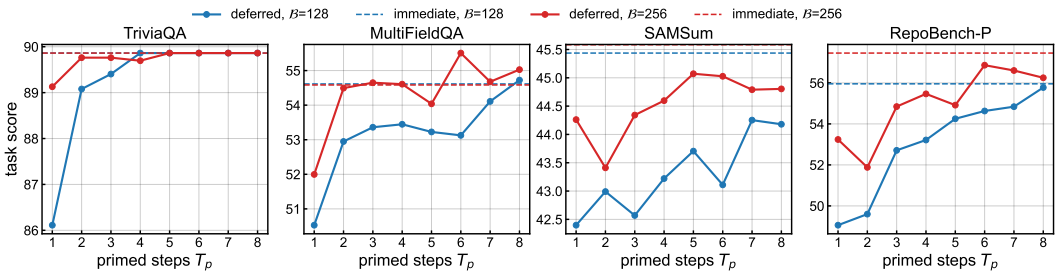

Figure 12: Sensitivity to priming length on Qwen3-8B. The first $T_p - 1$ steps use raw immediate selection and step T_p uses right-neighbor Avg2 before switching to deferred execution. Solid curves report task scores; dashed lines show raw immediate selection at the same prompt-KV budget. Each panel uses its own score range.

TriviaQA plateaus after four to five priming steps. MultiFieldQA-en approaches its immediate reference within two to three steps at $B = 256$, but fluctuates at longer lengths. At $B = 128$, it approaches the reference at $T_p = 8$ after a dip at intermediate lengths. SAMSum and RepoBench-P benefit from longer priming overall, but the gains are not monotonic: at $B = 256$, their highest plotted scores occur at $T_p = 5$ and 6, respectively, followed by small declines. SAMSum remains below immediate selection throughout the tested range. RepoBench-P nearly reaches its $B = 128$ reference at $T_p = 8$, but retains a gap at $B = 256$.

Table 11: Recovered details under the deployed schedule. The shaded strip gives the reference; right-aligned numbers are the original per-example scores (0–1). Code cells show the first statement; scores are taken from the original run and are not recomputed from the excerpts.

(a) MultiFieldQA-en

Ex. 19 $B = 256$ 3,869 prompt tokens

Reference	**Other toxic products.**	
Exact / imm.	Answer: **other toxic products**	0.86
1 prime, no pool	Answer: **saline solution**	0.00
Deployed	**other toxic products**	1.00

(b) SAMSum

Ex. 15 $B = 128$ 9,810 prompt tokens

Reference	Jim will check out Max's latest music project **when he gets home**.	
Exact-score	Max is not a famous music producer, but he shared his latest project with Jim, who will listen to it **later**.	0.12
Immediate	Max is a music producer who is not famous. He shared his latest project with Jim, who will listen to it **later** and give his opinion.	0.17
1 prime, no pool	Max shared his latest music project with Jim, who promises to listen to it and give feedback **later**.	0.21
Deployed	Max is not a famous music producer. He shares his latest project with Jim. Jim will listen to it **when he gets home** and tells Max what he thinks later.	0.46

(c) RepoBench-P

Ex. 28 $B = 128, 256$ 11,452 prompt tokens

Reference

```
Assets.perfectTapSound.play(GlobalConfiguration.feedbackVolume / 100f);
```

$B = 128$

Exact / imm. 0.92

```
Assets.perfectTapSound.play(
 GlobalConfiguration.feedbackVolume
 / 100f);
```

1 prime, no pool 0.38

```
Assets.playSound("tap_perfect");
```

Deployed 0.87

```
Assets.perfectTapSound.play(
 GlobalConfiguration.feedbackVolume);
```

$B = 256$

Exact / imm. 0.91

```
Assets.perfectTapSound.play(
 GlobalConfiguration.feedbackVolume
 / 200f);
```

1 prime, no pool 0.31

```
Assets.playSound(
Assets.PERFECT_SOUND);
```

Deployed 0.92

```
Assets.perfectTapSound.play(
 GlobalConfiguration.feedbackVolume
 / 100f);
```

Priming therefore presents a task- and budget-dependent quality–execution trade-off. Longer priming can recover additional quality, but the gains may saturate or reverse, while each additional priming step postpones the transition to deferred execution.

D.7 Local Pooling Rules

Table 13 compares local pooling on Qwen3-8B at $B \in \{128, 256\}$, using all 3,750 examples across the 16 LongBench tasks without truncation. Deferred variants share the joint sketch and three-step priming schedule. Width-two rules combine each score with its right neighbor; widths three and five use centered neighborhoods. The comparison varies the pooling operator and neighborhood together.

Avg2 gives the highest deferred average at both budgets, 48.02 and 48.35, improving over no pooling by 1.00 and 0.84 points. Its gap to immediate selection narrows from 0.56 to 0.08 points as B increases. Task-level preferences differ: at $B = 128$, widening the average from two to five tokens raises MultiNews from 25.5 to 27.3 but lowers RepoBench-P from 56.9 to 44.3. Max2 is stronger than Avg2 on LCC, whereas Avg2 is stronger on GovReport and MultiNews. Pooling changes the ranking without enlarging the exact-KV budget; the narrow average gives the best overall balance among the evaluated rules.

Table 12: **Remaining differences and score-sensitive outputs.** Formatting and selection modes follow Table 11. Panel (d) compares generated API calls; (e) shows the same incorrect answer in all modes; (f) contrasts the reported scores with the named driver. Code cells show the first statement only.

(d) RepoBench-P
Ex. 20 $B = 256$ 6,538 prompt tokens

Reference	`await wild_report.add_reaction(MyEmojis.DESPAWNED)`	
Exact / imm.	`await wild_report.add_reaction(MyEmojis.ON_MY_WAY)`	0.78
1 prime, no pool	`await wild_report.add_reaction(self.HERE_EMOJI)`	0.69
Deployed	`await wild_report.edit(content="", embed=wild.expire_embed())`	0.44

(e) TriviaQA
Ex. 18 $B = 128, 256$ 17,165 prompt tokens

Reference	Court (volleyball); Volley ball	
All four modes	Badminton	0.00

(f) SAMSum
Ex. 1 $B = 256$ 2,601 prompt tokens

Reference	Mico and Jeff will go to the village party. **Jeff will drive.**	
Exact / imm.	Mico wants to go to the village party and invites Jeff. Jeff is not excited but agrees to go. Lia is too tired and doesn't want to go. Jim jokes about the village boys.	0.47
1 prime, no pool	Mico and Jeff are going to the village party. **Mico will drive.** They are not excited about it, but are going anyway.	0.64
Deployed	Mico wants to go to the village party to meet hot boys. Jeff and Mico will go. **Mico will drive.**	0.75

E Efficiency Analysis

We analyze routing-state construction, decode-time preparation, and cache-related storage. Unless stated otherwise, measurements use Qwen3-8B on one RTX PRO 6000 GPU at batch size one.

E.1 Routing-State Construction and Prefill Overlap

Construction proceeds layer by layer during GPU prefill. Each layer offloads exact K/V to preallocated pinned host memory and transfers the query samples used for fitting. CPU workers form the normalized query–key moment and compute its leading eigenspace U. Once the basis and host keys are ready, the CPU computes KU, quantizes the projected entries, and packs them into INT4. The sketch, basis, and quantization parameters are uploaded to the GPU while exact K/V remain on the host. Projection and packing use 16 CPU threads; separate transfer streams overlap offloading and uploading with GPU prefill.

Tables 14 report the construction timings and the interval between successive layers' fitting-sample arrivals. K/V offload is the CUDA-event duration of two device-to-host copies, without host synchronization. Fitting-sample readiness is measured from the GPU Q/K/V event until both sample matrices are usable on the CPU. It includes the preceding K/V copies, question-span query transfer, CPU gathering of 76 key rows per KV head. Basis fitting and projection/packing use CPU wall-clock timings; routing-state upload uses an isolated CUDA-event copy measurement.

Basis fitting varies little with context length (1.9–3.3 ms per layer), while projection, quantization, and packing dominate at longer contexts. At 97K, these CPU operations take 113.4 ms per layer, and the reported construction intervals sum to 131.3 ms against a 361.5 ms GPU inter-layer interval. The corresponding comparisons are 3.9 versus 5.7 ms at 4K and 164.4 versus 571.2 ms at 128K. The layer-wise pipeline overlaps CPU construction and data transfers with GPU prefill, largely hiding their latency from adding a separate construction pass afterward. The measured complete-prefill latency remains nearly unchanged.

Table 13: LongBench scores for pooling rules in Qwen3-8B. Avg. is the 16-task mean; task scores use one decimal and means two. Bold indicates the best displayed deferred score, including ties. † is the deployed rule.

	References			Step-deferred selection						
Task	Full cache	Exact-score (imm.)	Sketch (imm.)	No pooling	Avg.† $w=2$	Avg. $w=3$	Avg. $w=5$	Max. $w=2$	Max. $w=3$	Max. $w=5$
Prompt-KV budget $B = 128$ per KV head										
NarrativeQA	26.0	26.4	26.0	25.6	26.2	24.7	24.6	26.4	26.8	**26.9**
Qasper	48.2	47.4	47.8	43.5	**46.7**	44.8	44.5	45.9	46.1	45.7
MultiFieldQA-en	54.0	54.2	53.7	51.2	52.4	51.9	50.8	52.8	**53.9**	53.0
HotpotQA	59.1	61.2	58.4	**58.9**	58.3	58.3	58.7	58.8	58.8	**58.9**
2WikiMQA	43.0	42.1	43.0	43.0	43.0	**43.4**	42.4	43.2	42.9	43.2
MuSiQue	35.6	33.4	35.7	33.6	**35.2**	34.3	34.0	34.3	34.6	34.9
GovReport	33.5	33.3	33.6	29.9	32.8	33.2	**33.4**	30.8	30.4	29.7
QMSum	24.1	24.1	24.4	23.4	23.7	23.9	23.2	23.8	23.3	**24.0**
MultiNews	24.7	24.6	24.8	23.6	25.5	25.9	**27.3**	23.5	23.9	23.5
TREC	71.5	71.5	72.5	70.0	**70.5**	**70.5**	**70.5**	**70.5**	**70.5**	**70.5**
TriviaQA	90.7	89.6	89.7	**89.8**	89.7	89.2	84.1	89.6	89.4	89.7
SAMSum	44.2	45.2	45.0	42.2	**43.5**	41.4	37.9	43.0	42.4	42.9
PassageCount	7.2	5.5	5.2	6.8	7.9	6.9	**8.3**	7.0	6.0	5.5
PassageRetrieval-en	100.0	100.0	100.0	**100.0**	99.5	98.0	95.5	99.0	99.0	97.0
LCC	57.7	58.1	58.6	55.8	56.6	56.0	51.2	**58.4**	57.8	57.6
RepoBench-P	56.7	59.1	58.7	54.8	56.9	53.0	44.3	**57.4**	56.9	56.7
Avg.	48.51	48.47	48.58	47.01	**48.02**	47.21	45.67	47.78	47.66	47.48
Prompt-KV budget $B = 256$ per KV head										
NarrativeQA	26.0	25.5	26.1	26.2	26.1	25.8	24.9	26.4	**26.5**	**26.5**
Qasper	48.2	47.3	46.6	43.9	45.6	45.0	45.6	45.8	**46.4**	46.1
MultiFieldQA-en	54.0	54.3	53.7	52.9	**54.1**	53.8	53.5	53.5	53.4	53.6
HotpotQA	59.1	60.9	60.0	60.3	59.8	60.3	60.1	59.9	60.1	**60.6**
2WikiMQA	43.0	41.3	41.5	41.5	41.4	**42.2**	41.4	41.1	41.3	41.2
MuSiQue	35.6	35.3	35.4	32.9	**35.3**	34.7	33.9	34.3	34.8	34.7
GovReport	33.5	33.4	33.5	31.2	33.2	**34.0**	33.9	31.7	31.7	31.5
QMSum	24.1	24.1	24.6	23.4	24.0	**24.2**	23.8	23.6	23.7	23.4
MultiNews	24.7	24.8	24.8	23.9	25.5	25.8	**27.3**	24.3	24.4	24.2
TREC	71.5	73.0	72.0	**72.0**	**72.0**	**72.0**	**72.0**	**72.0**	**72.0**	**72.0**
TriviaQA	90.7	90.0	89.9	**90.0**	89.8	89.4	87.0	89.9	89.6	89.7
SAMSum	44.2	45.2	45.1	43.1	44.1	**44.7**	41.1	43.8	43.4	43.5
PassageCount	7.2	6.3	5.0	6.6	6.9	5.6	6.8	**7.5**	6.6	7.0
PassageRetrieval-en	100.0	100.0	100.0	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
LCC	57.7	58.2	58.0	57.4	58.0	57.6	56.5	58.0	**58.3**	58.2
RepoBench-P	56.7	59.0	58.6	54.9	**57.6**	56.5	52.6	**57.6**	**57.6**	57.4
Avg.	48.51	48.66	48.43	47.50	**48.35**	48.22	47.51	48.09	48.12	48.10

E.2 Decode Latency and Scaling

We capture the complete decode step, including routing, in a CUDA Graph. At $B = 256$, DeSKV retrieves 224 prompt tokens per KV head alongside its fixed 32 prompt-token tail. The routing-free reference uses the same prompt working set without scoring, selection, or fetching. Runs above 40,960 positions use YaRN and measure latency only.

Each latency measurement decodes one sequence from a fixed prompt of 7,988–127,751 tokens. After prefill, index construction, and offload, 16 warm-up steps include CUDA-graph capture and the timing harness's single same-step initialization pass. We then time five repetitions of 128 decode steps and report the median repetition time divided by 128. Generated history grows by one token per step, ranging from 16 to 144 tokens beyond the prompt working set during timing, at its original positions. Prefill, construction, offload, warm-up, and priming are excluded from the timed window. Table 15 compares the resulting per-token latencies.

Full-cache attention reads increasingly more K/V as the prompt grows, whereas selective attention uses a budgeted prompt working set. Routing still covers the growing prompt, but deferral overlaps

Table 14: Construction stages and their overlap with prefill on Qwen3-8B. All times are in milliseconds. Per-layer timings are averaged over layers, and the stage total sums rows 1–5. CPU fitting uses the offloaded samples; projection and packing follow once the basis and host keys are ready, followed by routing-state upload. These operations overlap with GPU prefill. The inter-layer sample interval measures the time between successive layers' fitting samples becoming available.

Operation	Hardware	4K	8K	16K	32K	64K	97K	128K
Per-layer construction (ms; mean over layers)								
1. Offload K/V	GPU→CPU	0.3	0.6	1.2	2.3	4.6	7.0	9.3
2. Fitting-sample readiness	GPU→CPU	0.7	0.9	1.5	2.7	5.0	7.5	9.7
3. Fit U	CPU	1.9	2.3	2.6	2.8	3.0	3.2	3.3
4. Project KU/quantize/pack	CPU	1.0	1.7	12.7	26.5	73.6	113.4	141.8
5. Upload routing state	CPU→GPU	0.03	0.04	0.05	0.09	0.16	0.23	0.31
Sum of stages 1–5	–	3.9	5.5	18.1	34.4	86.4	131.3	164.4
Latency of a layer	GPU	5.7	13.2	28.9	70.4	188.1	361.5	571.2
Complete prefill (ms)								
Prefill alone	GPU	220.7	501.5	1,094.1	2,639.8	6,952.1	13,304.0	20,883.1
Prefill + construction	GPU + CPU	218.3	501.9	1,093.3	2,640.5	6,974.9	13,354.2	20,987.9

Table 15: Decode latency across context lengths. Times are milliseconds per output token at $B = 256$ on Qwen3-8B. Routing-free uses the same fixed-size attention working set without scoring, selection, or fetching. Bold marks the lowest latency among the evaluated routing methods.

Context	8K	16K	32K	64K	97K	128K
Full cache	14.24	15.03	16.53	19.63	23.17	26.14
Routing-free	13.48	13.46	13.46	13.42	13.45	13.40
PQCache	20.89	21.66	21.59	23.62	26.47	28.64
SparQ	17.15	17.93	17.86	18.66	19.91	20.90
Loki	16.44	17.21	17.07	17.59	18.59	19.31
InfLLM	15.97	15.91	16.24	16.51	16.82	17.14
Quest (our impl.)	15.87	15.93	15.91	16.10	16.43	16.76
DeSKV (immediate)	17.17	18.06	17.79	18.39	19.40	20.17
DeSKV (deferred)	**14.59**	**14.43**	**14.69**	**14.88**	**15.39**	**15.90**

scoring, selection, and K/V preparation with the current forward pass. From 8K to 128K, deferred latency grows from 14.59 to 15.90 ms, compared with 14.24 to 26.14 ms for full cache. At 8K, attention savings do not offset the remaining preparation overhead; at longer contexts in this grid, avoiding full-cache access is beneficial.

Deferral does not remove the resource requirements of preparation. Its excess over routing-free grows from 1.11 to 2.50 ms, consistent with auxiliary-stream work contending for GPU compute and memory bandwidth. At 97K, deferral nevertheless reduces latency by 20.7% relative to immediate execution and 33.6% relative to full cache. The scheduling benefit is a smaller preparation cost exposed on the attention dependency path.

Table 16 isolates score computation and selection by disabling fetching and subtracting the routing-free reference. The retrieved-token budget bounds selected K/V volume, while the set of routing candidates grows with context length.

Routing costs reflect both selection granularity and memory access. Quest and InfLLM rank page or block summaries, whereas Loki and DeSKV score individual tokens from low-dimensional representations. SparQ gathers query-selected channels, and PQCache accumulates codebook lookups. These gather-based access patterns help explain their higher costs in the evaluated implementations. At 97K, DeSKV's 5.16 ms route term is close to Loki's 5.12 ms and below SparQ's 6.44 ms and PQCache's 12.37 ms. Step deferral overlaps the resulting score-and-select work with model execution.

Table 17 reports the smaller and larger budgets; $B = 256$ appears in Table 15. Increasing B leaves the number of scored prompt positions unchanged but requires selecting and preparing more K/V. At 97K, moving from 128 to 512 tokens adds 1.63 ms to immediate execution and 1.13 ms to deferred

Table 16: Routing overhead excluding K/V fetching. Times are milliseconds per decode step at $B = 256$ on Qwen3-8B. Each entry is the step latency with fetching disabled minus the routing-free reference, including score computation and selection.

Context	8K	16K	32K	64K	97K	128K
InfLLM	1.78	1.77	2.15	2.47	2.72	3.11
Quest (our impl.)	2.31	2.39	2.38	2.69	2.98	3.36
Loki	2.86	3.67	3.60	4.20	5.12	5.91
SparQ	3.67	4.38	4.40	5.28	6.44	7.50
PQCache	6.60	7.42	7.50	9.57	12.37	14.61
DeSKV (imm.)	3.08	3.57	3.70	4.36	5.16	5.98

execution. Routing-free latency changes little over this range, indicating that preparation accounts for most of the observed budget sensitivity.

Table 17: Working-set budget scaling on Qwen3-8B. Times are milliseconds per output token. All budgets include 32 fixed prompt-tail tokens.

Context	8K	32K	64K	97K	128K
Prompt-KV budget $B = 128$ per KV head					
Routing-free	13.45	13.44	13.45	13.38	13.39
Immediate	16.38	17.05	17.69	18.62	19.26
Deferred	14.34	14.32	14.69	15.06	15.54
Prompt-KV budget $B = 256$ per KV head					
Routing-free	13.48	13.46	13.42	13.45	13.40
Immediate	17.17	17.79	18.39	19.40	20.17
Deferred	14.35	14.54	14.95	15.36	15.86
Prompt-KV budget $B = 512$ per KV head					
Routing-free	13.51	13.49	13.41	13.43	13.42
Immediate	18.03	18.59	19.24	20.25	21.02
Deferred	14.84	15.43	15.84	16.19	16.68

Table 18 extends the comparison to Llama-3.1-8B-Instruct and Mistral-7B-Instruct-v0.3. Immediate and deferred execution share prefill and routing-state construction, yielding the same reported TTFT estimates. These estimates remain close to the routing-free reference across all three models, consistent with construction being largely overlapped with prefill. During decoding, routing-free TPOT remains nearly constant as context grows, whereas immediate execution becomes more expensive as routing covers a longer prompt. Deferral overlaps working-set preparation with the current forward pass, reducing TPOT by 3.25 ms on Llama and 3.41 ms on Mistral at 97K. The scheduling benefit thus extends across models without a corresponding increase in initialization latency.

Table 18: Execution-schedule comparison across models at $B = 256$. TTFT is the full-cache prefill plus the routing-state construction that is not hidden under it; TPOT in milliseconds per output token.

		TTFT (s)				TPOT (ms/step)			
Model	Schedule	32K	64K	97K	128K	32K	64K	97K	128K
Qwen3-8B	Routing-free	2.64	6.95	13.29	20.88	13.46	13.42	13.45	13.40
	Immediate	2.64	6.97	13.35	21.01	17.79	18.39	19.40	20.17
	Deferred	2.64	6.97	13.35	21.01	14.69	14.88	15.39	15.90
Llama-3.1-8B	Routing-free	2.36	6.25	11.81	18.66	12.35	12.29	12.30	12.30
	Immediate	2.37	6.27	11.88	18.77	16.17	16.62	17.39	18.13
	Deferred	2.37	6.27	11.88	18.77	13.49	13.55	14.14	14.64
Mistral-7B	Routing-free	2.31	6.12	11.65	18.44	11.75	11.76	11.75	11.76
	Immediate	2.31	6.13	11.73	18.54	15.62	16.19	17.01	17.71
	Deferred	2.31	6.13	11.73	18.54	12.94	13.01	13.60	14.18

Table 19: Accounted GPU cache and routing-state storage of DeSKV. Values are in decimal MB and include one $B = 256$ exact working set. Model weights and workspaces are outside this total. Components and totals are rounded independently.

Context	INT4 codes	Bases + scales	Exact working set	Total
8K	36.8	2.38	37.7	**76.9**
32K	147.0	2.38	37.7	**187.2**
64K	294.3	2.38	37.7	**334.4**
97K	446.3	2.38	37.7	**486.5**
128K	588.7	2.38	37.7	**628.8**

E.3 GPU Memory Footprint and Routing Reads

Table 19 breaks down the accounted prompt-cache and routing-state storage at $B = 256$. INT4 codes grow with prompt length, while the exact prompt working set occupies 37.7 MB. Feature-channel quantization shares parameters across prompt positions, so metadata storage remains independent of prompt length. At 97K, the listed components total 486.5 MB, including 446.3 MB of sketch codes. This accounting includes one prompt working set rather than peak process memory.

Read volume and storage measure different costs. Table 20 compares representations at 97K. DeSKV, Quest, SparQ, and Loki each read approximately 0.446 GB of routing payload per step, but the latter three implementations retain the 14.28 GB full K/V cache on the GPU. SparQ and Loki route from their resident keys rather than a separate compact index. DeSKV, InfLLM, and PQCache instead keep exact K/V on the host and only routing state and a working set on the GPU.

Table 20: Routing state, full-cache residency, and logical reads at 97K. Routing state excludes the exact working-set buffer. For SparQ and Loki, the routing state is contained within their resident key cache, not an additional allocation. All rows move approximately 33 MB of selected K/V per step: a host-to-GPU transfer for offloaded caches, or a GPU-local gather for resident caches. Read volumes account for the routing payload, not temporary score buffers or repeated accesses.

Method	Full K/V source	Routing state (GB)	Full GPU K/V (GB)	Index read (GB/step)	K/V movement
DeSKV	Host	0.449	0.00	0.446	Host to GPU
InfLLM	Host	0.446	0.00	0.446	Host to GPU
PQCache	Host	0.465	0.00	0.465	Host to GPU
Quest (our impl.)	GPU	0.446	14.28	0.446	GPU gather
SparQ	GPU	7.141	14.28	0.446	GPU gather
Loki	GPU	7.142	14.28	0.446	GPU gather

At fixed budget, selected K/V movement remains approximately 33 MB per step from 8K to 128K, while DeSKV's routing payload grows from 0.037 to 0.589 GB. Offloading reduces GPU capacity requirements and bounds selected-K/V traffic, but the sketch scan still scales with prompt length. The reported reads describe logical payloads, not measured physical memory transactions.