LAYERTRACE: LEARNING SOURCE-SPECIFIC TOKEN LATENTS FOR SPARSE ATTENTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-context decoding repeatedly reads historical keys and values, while sparse attention adds work to identify useful entries. LayerTrace moves that decision to an earlier transformer layer and learns the token representation used to make it. Each source stores a latent for every historical token, obtained from its source-block output through trainable content and positional projections. Independent destination-layer and query-head scorers use the current source output to prepare future read maps. Encoders and scorers are trained together against dense attention distributions and outputs, with a frozen backbone; destinations attend to selected original keys and values. On 503 LongBench v2 questions, Qwen3-4B at threshold 2 answers 157 correctly versus Dense's 153 while selecting 31.58% of original KV. In separate fixed-input timing at the same operating point, 32K context and batch 16, whole-decoder time falls from 62.68 to 36.34 ms per step. A 56-question DeepSeek development diagnostic scores 59.08 versus 49.40, but all-selected numerical discrepancies remain unresolved. Higher thresholds can collapse quality, and single-request execution can remain slower. These exploratory observations do not yet establish quality-preserving speedups under matched batched answer generation.

1 INTRODUCTION

Long-context decoding repeatedly reads a growing key–value cache at every attention layer. Sparse attention can reduce these reads, but the decoder must first determine which entries to use. Query-aware methods such as Quest and SparQ inspect cache information using the current layer's query (Tang et al., 2024; Ribar et al., 2024). Selection therefore adds work to that layer's critical path, before its selected attention can run. Repeating this process across layers can consume the time saved by reading fewer values. The practical objective is thus lower *whole-decoder latency*, including selection, rather than a smaller selected-cache fraction alone.

This cost motivates moving read prediction to a small set of source layers. Each source can consult its compact token history and produce distinct read plans for several later layers, passing only packed bitmaps to their attention kernels. The destinations then consume these plans without running another selection procedure. Earlier decisions are useful here because they change *where and how often selection is performed*, not because timing a prediction earlier is itself a saving. TidalDecode reduces repeated work by reusing selected positions (Chen et al., 2025), and SparDA learns next-layer lookahead (Fu et al., 2026). We study how learned source representations can support multiple destination-specific decisions while keeping their combined prediction cost below the attention time saved.

LayerTrace jointly learns a token encoder and a future-layer scorer on a frozen backbone. The encoder maps a historical token's source-block output to a compact latent; the scorer maps the current source output to a vector per destination and query head. Their dot products predict future reads. Historical latents persist across decode steps, while destination/head-specific bitmaps are recomputed for each new token. Original per-layer K/V remain resident: the learned latent is an auxiliary selection index, and actual destination queries still compute attention over selected original entries.

Figure 1 shows what each layer retains for each token and how a source prepares read maps for later layers. Across decode steps, historical latents persist while the maps change (Figure 2). A source

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

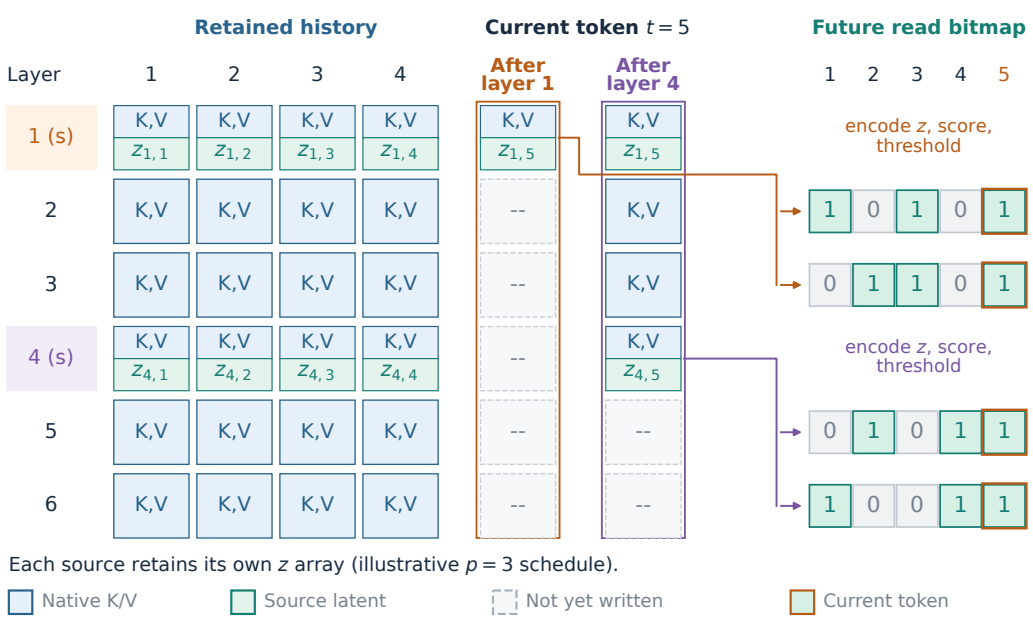

Figure 1: **Source latents predict KV reads for later layers.** Current-token columns show the state after layers 1 and 4; unchanged history is shown once. Source 1 prepares maps for layers 2–3 before their current Q/K/V exist. After layer 4, those maps have been consumed and source 4 prepares maps for layers 5–6 using its own latents. Each bit selects a token. One head and illustrative patterns are shown; experimental spacing is in Table 3.

appends the current token's latent, compares destination/head-specific score vectors with the stored latents, and thresholds the scores into read bitmaps. Destinations use their actual queries to attend to selected original K/V. Figure 3 details these operations after the notation is introduced.

Our evaluation tests whether saved attention time pays for encoding, scoring and bitmap construction. In fixed-input timing at Qwen 32K/B8, LayerTrace reduces decoder-step latency from 37.26 to 25.35ms (1.47×); all five measured baseline integrations at the reported settings remain slower than Dense. These are comparisons of the measured integrations, with their numerical limits, rather than all possible optimized implementations. Batch-one slowdowns remain visible. Low-rank ranking, grouped-layer factorization and shared routing have precedents in Loki, xKV and CLSA (Singhania et al., 2024; Chang et al., 2026; Sun et al., 2026); the systems question here is whether learned advance read maps deliver a net decoder-time benefit.

The paper presents the learned index/scorer, its storage and execution lifecycle, and measured task-quality/decoder-time tradeoffs. Separate hybrid-attention and MLA experiments test compatibility with recurrent blocks and native compressed caches: deciding *which rows to read* can complement compressing *what each row stores*.

2 Related Work

Current-query selection and cache retention. Quest estimates page importance using the current query and cached key extrema, while SparQ uses a partial-key stage to choose history to fetch (Tang et al., 2024; Ribar et al., 2024). SnapKV selects a compressed cache from prompt-attention observations, and PyramidKV varies retained capacity across layers (Li et al., 2024; Cai et al., 2025). SeerAttention-R adapts a pretrained model with a learned decoding gate (Gao et al., 2025). LayerTrace makes a destination's selection before that destination query exists and retains full original KV storage. Reduced selected volume and reduced resident-cache capacity are therefore different quantities.

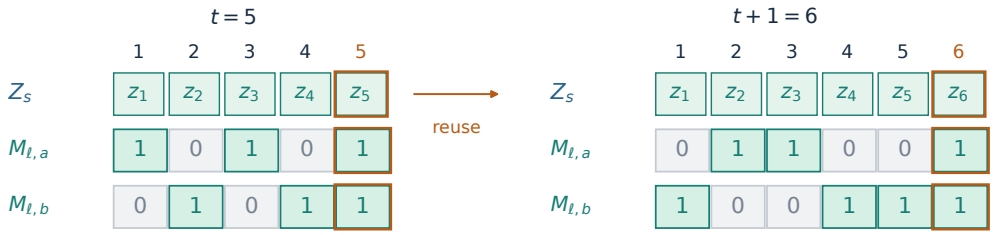

Figure 2: **Latents persist; read maps change at every decode step.** Only $z_{s,6}$ is appended, while both heads' maps are recomputed, including older-token bits. Orange outlines mark the newest latent and forced bit. Source indices on z are omitted; bit patterns are illustrative.

Cross-layer reuse and prediction. TidalDecode uses selection layers and reuses selected token positions in other layers (Chen et al., 2025). CLSA shares a routing index across layers in a KV-sharing architecture (Sun et al., 2026). LayerTrace instead produces a separate map for each destination/head from one source output. SparDA is a close lookahead comparison: its Forecast projection predicts next-layer KV blocks, with one Forecast head per GQA group (Fu et al., 2026). LayerTrace's multiple destinations, per-query-head scorers and learned source-token index define the design being studied; they do not by themselves establish a quality or speed advantage.

Low-rank representations. Loki ranks tokens in a low-dimensional key space (Singhania et al., 2024); xKV factorizes grouped-layer KV caches into a shared subspace (Chang et al., 2026). SALS combines latent-cache compression, current-query selection and selective key reconstruction (Mu et al., 2025). LayerTrace instead learns an auxiliary source-output index and predicts maps before the destination query exists; its GQA implementation retains original KV. DeepSeek-V2's MLA learns low-rank KV projections with separate rotary coordinates (DeepSeek-AI, 2024). Our MLA adaptation tests source-only advance selection over that native compressed cache; the GQA index does not itself compress resident KV.

Training and execution systems. NSA combines hierarchical sparsity with native end-to-end training and hardware-oriented execution (Yuan et al., 2025). The present LayerTrace objective instead trains selectors against frozen dense-teacher states. HiSparse manages the placement and retrieval of sparse-attention KV entries across memory levels (Xie et al., 2026); such execution concerns are complementary to deciding which positions to select. We distinguish the selection mechanism from the execution substrate and report their combined runtime rather than equating fewer selected tokens with acceleration.

3 Method

3.1 Sources, destinations and notation

Consider a decoder with L layers, residual-stream width d_{model}, H_q query heads and H_{kv} KV heads. Each native query, key and value has width d_{head}. The residual width d_{model} and head width d_{head} are different quantities; we do not assume $d_{\text{model}} = H_q d_{\text{head}}$. Layer numbers are one-based throughout the paper.

A source layer s completes its ordinary attention and MLP before planning the next destination layers, denoted $\mathcal{G}_s$. With source spacing p, sources lie at $1, 1+p, \ldots$ and $\mathcal{G}_s = \{s+1, \ldots, \min(s+p-1, L)\}$. Sources attend densely. Let $\mathcal{S}_+$ denote the sources with nonempty destination groups; an empty terminal group requires no predictor. Let i index a historical token, t the current token, a a query head, and $g(a)$ its KV head under grouped-query attention (GQA).

Figure 2 shows that latents persist while destination/head maps are regenerated.

The latent has a content width d_c and an even positional width d_r, giving total width $d_z = d_c + d_r$. These are configurable per model. The scorer's intermediate width follows the native head

width d_{head}, and its residual MLP has width d_{ff}. Table 1 defines the tensor shapes; model-specific numerical choices appear only in Table 3.

Symbol	Shape	Meaning
$x_{s,i}, \bar{x}_{s,i}$	d_{model}	Source full-block output and its parameter-free RMS normalization.
W_s^C, W_s^R	$d_c \times d_{\text{model}}$, $d_r \times d_{\text{model}}$	Trainable content and positional token encoders.
$F_{s,\ell,a}$	$\mathbb{R}^{d_{\text{model}}} \to \mathbb{R}^{d_{\text{head}}}$	Trainable current-token scorer feature for destination ℓ and head a.
$P^C_{s,\ell,a}, P^R_{s,\ell,a}$	$d_c \times d_{\text{head}}$, $d_r \times d_{\text{head}}$	Two row blocks of one trainable output projection.
R_i	$d_r \times d_r$	Positional rotation at token position i.
$z_{s,i}, u_{s,\ell,a,t}$	d_z	Stored token latent and temporary current score vector.
$q_{\ell,a,t}, k_{\ell,g,i}, v_{\ell,g,i}$	d_{head}	Native destination query, key and value.
$A_{\ell,a,t,i}, M_{\ell,a,t,i}$	scalar; bit	Predicted relevance score and read-selection decision.

Table 1: **Symbolic tensor dimensions.** Widths follow the definitions above; batch axes are omitted. Numerical hyperparameters appear in Table 3.

3.2 A SEPARATE LEARNED TOKEN INDEX FOR EACH SOURCE

For each token i, the source produces $x_{s,i}$ after its full block. Normalize this vector using the model's RMS stabilizer ϵ_n:

$$\bar{x}_{s,i} = \frac{x_{s,i}}{\sqrt{d_{\text{model}}^{-1} \sum_{j=1}^{d_{\text{model}}} x_{s,i,j}^2 + \epsilon_n}}, \qquad z_{s,i} = \begin{bmatrix} W_s^C \bar{x}_{s,i} \\ R_i W_s^R \bar{x}_{s,i} \end{bmatrix} \in \mathbb{R}^{d_z}. \tag{1}$$

The first branch encodes content; the second applies the historical token's position to separate positional coordinates. R_i uses the first $d_r/2$ frequency pairs of the model's native rotary embedding, with the paired halves required by rotate-half arithmetic. Both encoder matrices learn from the attention objective in Section 4.

The index is source-specific. In the illustrative schedule of Figure 1, sources 1 and 4 produce $z_{1,i}$ and $z_{4,i}$ from different hidden states with different encoder parameters. Each latent array is shared by the destinations and heads owned by its source, while their scoring functions are independent. The figure places layers on the vertical axis and tokens on the horizontal axis: every cell holds that layer's native K/V, and source-layer cells additionally hold a latent. A latent does not reconstruct native keys or values, and skipping a token does not delete either cache entry.

3.3 ONE CURRENT SOURCE OUTPUT, DIFFERENT FUTURE READ MAPS

At step t, the source maps its current normalized output to a feature $f_{s,\ell,a,t} = F_{s,\ell,a}(\bar{x}_{s,t}) \in \mathbb{R}^{d_{\text{head}}}$ for every destination/head. The trainable base projection is initialized from that destination's native query projection and input-normalization weights. A residual GELU MLP of width d_{ff} adapts the feature; native Q normalization is retained where present. The base and MLP are scorer parameters, separate from the frozen backbone.

The learned output projection maps this feature into content and positional score coordinates. Rotate only the latter at the current position:

$$u_{s,\ell,a,t} = \begin{bmatrix} P^C_{s,\ell,a} f_{s,\ell,a,t} \\ R_t P^R_{s,\ell,a} f_{s,\ell,a,t} \end{bmatrix}, \qquad A_{\ell,a,t,i} = \frac{u_{s,\ell,a,t}^\top z_{s,i}}{\sqrt{d_{\text{head}}}}. \tag{2}$$

The two blocks P^C, P^R together form a $d_z \times d_{\text{head}}$ matrix. The scale follows the native attention head width d_{head}, even when the latent width d_z differs. Full native-head RoPE is not applied before this projection. Content is compared without rotation; the paired positional coordinates express the current and historical positions.

For each destination/head, validation text supplies a mean $m_{\ell,a}$ and standard deviation $\sigma_{\ell,a}$ of the normalized score $\widetilde{A} = A / \max(\|u\|_2, \epsilon_s)$, where ϵ_s prevents division by zero. A threshold setting τ

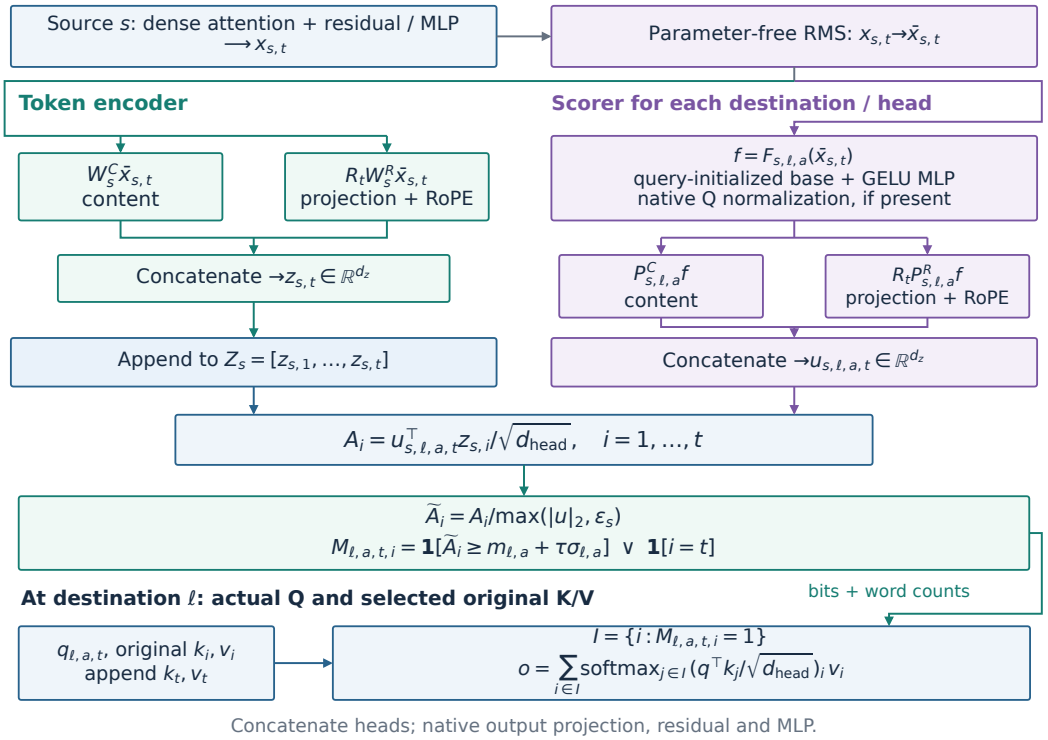

Figure 3: **From source hidden state to destination attention.** Learned content/rotary branches produce the stored latent z and destination/head score vector u. Their dot products and validation-calibrated moments m, σ yield a bitmap before the destination runs; its actual Q attends to selected original K/V. The current-token bit is forced to 1. Symbols follow Table 1.

produces

$$M_{\ell,a,t,i} = \mathbf{1}[\widetilde{A}_{\ell,a,t,i} \geq m_{\ell,a} + \tau\sigma_{\ell,a}] \vee \mathbf{1}[i = t], \qquad i \leq t. \tag{3}$$

The current token is always selected. For example, two heads at destination 2 may select different historical tokens, and destination 3 can select another set. A shared GQA fetch may read their KV union, but each head retains its own attention mask. Selected attention uses the actual destination query and original selected K/V. The predicted vector u supplies a read decision, not replacement attention weights.

3.4 Prefill, decoding and storage

During dense prefill, each source encodes and stores every prompt-token output. During decoding, it appends the new token's latent and generates packed head-specific maps with per-word counts. Destinations consume these maps; the next source repeats with its own index. Historical latents persist, but maps are regenerated each step (Figure 2).

Historical source hidden states are training records; deployment retains their latents and the full native K/V cache. Packed maps add one bit per token and query head, plus count metadata. Latents, predictor weights and workspace are additional costs, so fewer selected reads do not imply smaller resident KV storage. Appendix A gives the storage formulas.

4 Learning the token index and the scorers

Dense frozen-model execution supplies the source hidden state for every historical token and the native destination Q/K/V. The encoders must see all historical token states, even when only a subset of query positions is sampled for a training update. Teacher states and native model parameters remain fixed; token latents are recomputed with the current encoder weights on every update.

For a sampled query (ℓ, a, t), let p_i be the native dense attention distribution over historical positions $i < t$, and let $o = \sum_{i<t} p_i v_{\ell,g(a),i}$ be its output. The predicted scores give $\hat{p}_i = \mathrm{softmax}_{i<t}(A_i)$ and $\hat{o} = \sum_{i<t} \hat{p}_i v_{\ell,g(a),i}$. For a source's parameter set $\theta_s = \{W_s^C, W_s^R, F_{s,\ell,a}, P_{s,\ell,a}^C, P_{s,\ell,a}^R\}$, optimize

$$\mathcal{L}_s(\theta_s) = \mathbb{E}_{(\ell,a,t)}\left[\mathrm{KL}(p\|\hat{p}) + \lambda_o \frac{\|\hat{o} - o\|_2^2}{\max(\|o\|_2^2, \epsilon_o)}\right], \qquad \mathcal{L} = \sum_{s:\mathcal{G}_s \neq \varnothing} \mathcal{L}_s. \tag{4}$$

Here λ_o weights the output term and ϵ_o stabilizes its denominator. Their numerical values are experimental settings in Table 9. The loss teaches attention distributions and outputs; it is not regression to the native query vector. Threshold calibration follows checkpoint selection and is not part of the gradient objective.

Encoders and destination/head scorers receive gradients jointly within each source group. Fixed dense-teacher states make groups independent, although they can share teacher passes and training data. This is not end-to-end training through the sparse decoder: generation combines all groups, and earlier sparse decisions can change later source states.

Quantity	Training	Deployment
Native model and Q-norm buffers	Frozen teacher	Fixed original model.
Token encoders and scorers	Joint updates within each source group	Fixed learned checkpoint.
Historical source hidden	Reused dense-teacher records	Not retained after encoding.
Token latent $z_{s,i}$	Recomputed with current weights	Retained; append only new token.
Threshold moments	Validation calibration	Fixed per destination/head.
Native K/V	Teacher attention/output operands	Full cache; selected originals read.

Table 2: **What learns and what persists.** Source-group objectives follow Eq. 4.

The teacher excludes the current token, which runtime always includes. This distinction and the shift from dense-teacher to sparse-generation states separate validation loss from generated-answer quality.

5 Experimental setup

Models and training. We freeze Qwen3-4B and DeepSeek-R1-Distill-Llama-8B and train their source encoders and scorers on 64 general-text books, with three separate validation books. Each source receives 24,576 updates over three epochs using AdamW, learning rate 3×10^{-5} and output-loss weight $\lambda_o = 0.1$. Validation selects an epoch checkpoint; exported BF16 weights are then calibrated on validation text. No benchmark answers enter training. Table 3 gives model-specific choices; Appendix A records the remaining configuration.

Model	L	d_{model}	H_q/H_{kv}	d_{head}	d_c/d_r	d_{ff}
Qwen3-4B	36	2560	32/8	128	192/64	256
DeepSeek-R1-Distill-Llama-8B	32	4096	32/8	128	192/64	256

Table 3: **Model geometry and selector hyperparameters.** Both use source spacing $p = 5$ and seven nonempty groups; dense-source/destination counts are 8/28 for Qwen and 7/25 for DeepSeek. The latent width is $d_z = d_c + d_r$.

Quality and operating points. Qwen uses all 503 LongBench v2 questions with the official no-CoT prompt, a 32K pre-chat truncation cap and native non-thinking chat: 128 new tokens, temperature 0.1, top-k 20, top-p 0.95 and seed 42 (Bai et al., 2024). Appendix A details input construction. DeepSeek uses a 64K RULER-derived development diagnostic: eight questions in each of seven task families, greedy decoding, 256 new tokens and EOS 128001 (Hsieh et al., 2024). Half of its 56 questions were previously inspected; it is not the full 13-task RULER benchmark. Baselines match

IDs, input-token hashes and generation protocols. Each threshold gets an independent generation; twelve thresholds range from -0.5 to 6.

Our target is decoder-time reduction at maintained Dense quality, allowing different read budgets across methods (Tang et al., 2024; Ribar et al., 2024; Li et al., 2024; Cai et al., 2025; Gao et al., 2025). Here we report **observed task scores and separate fixed-input timings**. Score at least Dense is an exploratory screen, not a noninferiority test; selected points still need disjoint quality confirmation and batch-specific implementation checks. For times T_m, T_D, reduction is $100(1 - T_m/T_D)$ and speedup is T_D/T_m.

Timing and quality reuse. Synchronized eager timing covers the whole decoder on fixed inputs, excluding prefill; it is not answer-generation throughput. Quality reuse requires matching model, checkpoint, threshold, input/context protocol and numerical implementation; changed batch kernels or padding require a consistency check. Qwen uses YaRN factor 4 at 64K/128K and 8 at 256K (original context 32768); DeepSeek has no override, with 256K beyond its native 128K. These context extensions lack quality validation.

6 RESULTS

Observed task quality and fixed-input decoder timing. At $\tau = 2$, Qwen answers 157/503 correctly versus Dense's 153/503 using 31.58% of original KV; DeepSeek scores 59.08 versus 49.40 (Table 4). Both pass the observed-quality screen, which does not establish statistical equivalence. Appendix B gives paired uncertainty, all thresholds and the separate 6500-question baseline-only suite.

Qwen quality/timing share checkpoints, calibration and core kernels. Its quality runs pass 24/24 fast/reference and 4/4 native checks; separate full-model KL checks at B8/B16 are unavailable. DeepSeek fails 2/4 native and 7/24 fast/reference checks. Scores use actual fast-path generation; these observations do not confirm quality-preserving batched answer-generation speedups.

	Qwen3-4B			**DeepSeek-R1-Distill-Llama-8B**		
Method	Setting	Score	KV (%)	Setting	Score	KV (%)
Dense	n/a	30.42	100.00	n/a	49.40	100.00
Quest	$0.2N$	29.62	31.20	$0.2N$	53.93	32.08
SparQ	$N/8$	30.42	24.05	$N/8$	54.14	13.52
SnapKV	4K	30.22	27.68	16K	56.34	39.00
PyramidKV	4K	30.62	26.99	16K	50.71	37.34
SeerAttention-R	4K	30.62	14.34	NI	NI	NI
LayerTrace	$\tau = 1$	31.01	56.47	$\tau = 1$	63.30	61.95
LayerTrace	$\tau = 1.5$	31.61	41.87	$\tau = 1.5$	56.88	48.51
LayerTrace	$\tau = 2$	31.21	31.58	$\tau = 2$	59.08	37.17
LayerTrace	$\tau = 2.5$	31.21	25.99	$\tau = 2.5$	47.11	29.51

Table 4: **Task quality and selected original KV.** Qwen: LongBench v2, 503 questions; DeepSeek: 56-question development diagnostic. Scores: 0–100; KV includes dense sources. Seer quality uses reference attention. n/a: no setting; NI: gate/adapter not integrated (Appendix C.1).

Whole-decoder time. Tables 5–6 show scaling and absolute times at $\tau = 2$. DeepSeek 64K gains rise from 1.03× at B1 to 1.21× at B4 and 1.34× at B8. Longer contexts need not help: from 128K to 256K at fixed B2, Qwen improves from 1.23× to 1.51×, but DeepSeek falls from 1.10× to 0.97× (58.85 to 60.75ms). Extended-context quality remains unvalidated.

Measured baseline integrations. At Qwen 32K/B8, the five baseline integrations take 47.95–235.05ms against Dense's 37.26ms; LayerTrace takes 25.35ms, about 32% less time (Table 6). This compares measured integrations, not a quality-matched ranking of the methods. Qwen B8 checks fail 2/2 for compiled SparQ and 1/2 each for SnapKV/PyramidKV; Seer fails 4/8 across configurations. Reference quality does not certify these speed paths. Appendix C gives implementation details, checks and per-method controls.

Model	Context	$B = 1$	2	4	8	16
Qwen	32K	0.84	0.99	1.25	1.47	1.72
Qwen	64K	1.01	1.25	1.49	1.73	CAP
Qwen	128K	1.27	1.23	1.61	CAP	CAP
Qwen	256K	1.25	1.51	CAP	CAP	CAP
DeepSeek	32K	0.75	0.89	1.08	1.26	1.49
DeepSeek	64K	1.03	1.10	1.21	1.34	CAP
DeepSeek	128K	1.07	1.10	1.15	CAP	CAP
DeepSeek	256K	0.98	0.97	CAP	CAP	CAP

Table 5: **Context/batch speedup at $\tau = 2$.** Matched Dense/LT time; below 1: slower. NM: unmeasured; CAP: native KV alone exceeds VRAM (not an observed OOM). Appendix C gives full times and capacity bounds. Extended-context quality is unvalidated.

	Qwen 32K			**DeepSeek 64K**			
Method	Setting	$B = 1$	8	Setting	$B = 1$	4	8
Dense	n/a	14.16	37.26	n/a	19.07	36.09	58.94
Quest	$0.2N$	37.97	235.05	$0.2N$	60.44	211.67	OOM
SparQ	$N/8$	24.59	52.96†	$N/8$	32.31	49.47	OOM
SnapKV	4K	21.01	51.99†	16K	30.98	85.28	OOM
PyramidKV	4K	14.65	51.96†	16K	30.45	84.60	OOM
SeerAttention-R	4K	26.28†	47.95†	NI	NI	NI	NI
LayerTrace	$\tau = 2$	16.42	25.35	$\tau = 2$	18.37	29.78	43.84

Table 6: **Measured decoder-step latency (ms).** Fixed-input eager execution. †: reported timing checks fail. NM: unmeasured; NI: not integrated; OOM: observed in this integration. Reasons and separate method controls: Appendix C. Dense B1: baseline controls; B4/8: paired LayerTrace runs.

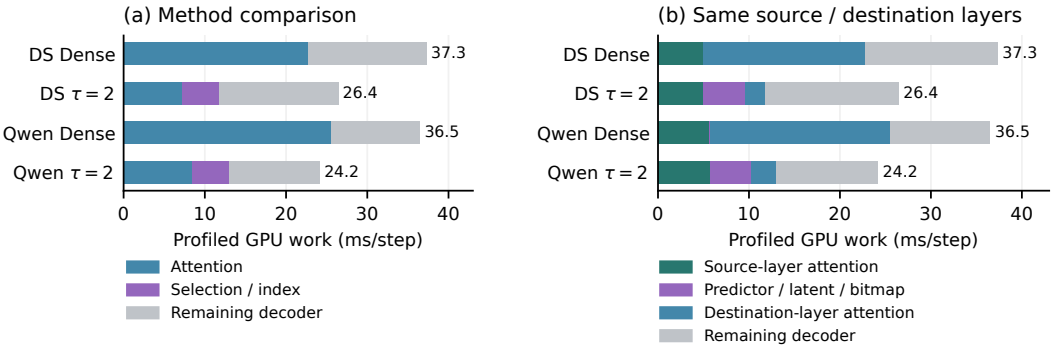

Figure 4: **Exclusive GPU work: DeepSeek 16K/B16; Qwen 32K/B8.** In (b), identical destination-layer groups fall from 17.80 to 2.20ms / 19.88 to 2.80ms (DeepSeek/Qwen); source attention stays near 4.98/5.69ms, while selection adds 4.59/4.57ms. Gray includes QKV/output. Events are counted once; these separate profiled sums are not elapsed latency.

Applicability to hybrid attention and MLA. Table 7 tests both architectures on 128 fixed LongBench v2 questions per model. Hybrid two-source selection reduces native row reads by 61.89% at 33 correct answers versus Dense's 29. The MLA shared-map adaptation reduces reads by 31.58% while matching Dense's 26 correct answers. **Read selection can coexist with native KV compression**: MLA keeps its compressed cache and selects its rows directly. This adaptation trains a scorer over frozen native source-cache rows, separately from the learned-z encoder. Architecture-specific latency is not measured.

Configuration	Index width	τ	Correct/128	Parsed/128	Rows read (%)
Hybrid: Qwen3.5-4B					
Dense	n/a	n/a	29	126	100.00
One source	1024	0.5	34	116	74.90
One source	1024	1	25	84	53.25
One source	256	0.5	19	114	74.23
One source	256	1	19	100	52.40
Two sources	256	0.5	35	126	73.29
Two sources	256	1.5	33	125	38.11
MLA: DeepSeek-V2-Lite-Chat					
Dense	n/a	n/a	26	109	100.00
Learned source z	256	1	16	102	88.75
Learned source z	256	1.5	31	113	74.55
Native shared map	512+64	0	26	95	68.42
Native shared map	512+64	0.5	22	98	58.58

Table 7: **Compatibility with hybrid attention and compressed MLA caches.** All fresh fits and their development-selected thresholds. Hybrid sources are attention ordinal 1, or 1 and 5; recurrent blocks are unchanged. MLA retains native compressed KV: the shared map uses native content+positional coordinates, whereas learned-z adds an index. Rows include dense sources but exclude selector traffic. Appendix D gives protocol, paired statistics and small-data controls.

7 Discussion and conclusion

Larger batches allow decoder weight loads to serve multiple sequences, reducing weight-transfer cost per sequence and potentially making sequence-specific KV reads a larger share of latency. Predictor weights can likewise be reused across a batch. These effects offer a mechanism for larger gains when attention reads dominate; the measured batch trend is consistent with it, but our timings do not isolate these causes. *Prediction work is not shared across sequences*: each sequence has its own latents, score computation and bitmaps, which are regenerated at each decode step. Small batches can still lose because encoding, scoring and map construction outweigh the saved attention time.

Aggressive thresholds collapse quality while reads approach the dense-layer floor (8/36 for Qwen; 7/32 for DeepSeek), plus current-token reads. DeepSeek falls below Dense at $\tau = 2.5$; Qwen reaches zero at 5. The floor reflects the schedule, not guaranteed quality-preserving sparsity.

Above-Dense scores can reflect sampling, changed trajectories or development selection; quality preservation needs independent confirmation and numerical agreement.

The hybrid results motivate studying source placement alongside latent capacity. MLA demonstrates complementary selection over an already compressed cache, rather than replacing its compression. Its evidence is weaker: Dense scores only 20.31%, and the shared-map setting parses 95/128 answers versus Dense's 109/128 despite equal correct counts. These exploratory results do not establish statistical equivalence or architecture-specific acceleration. Logical row reductions alone measure neither physical traffic nor resident-memory savings.

AI USE STATEMENT

Generative AI tools assisted research ideation, literature discovery, implementation, experiment orchestration, analysis, mathematical exposition, drafting and figure preparation. Human authors retain responsibility for the final manuscript and reported claims. The author disclosure and submission-form statement will be confirmed before submission.

REPRODUCIBILITY STATEMENT

The manuscript specifies dimensions, training, data flow and evaluation protocols. Anonymous source code, configurations, training/inference utilities and result tables are available at `https://anonymous.4open.science/r/layertrace-B1E4/README.md`. The release covers the two GQA models, excluding the architecture adaptations, model weights, learned checkpoint binaries and benchmark inputs. Its portable examples are not an exact replay of the paper's evaluation. Retained evidence includes question-level outputs, input/checkpoint hashes, timing repetitions and numerical-control reports.

REFERENCES

Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. Longbench v2: Towards deeper understanding and reasoning on realistic long-context multitasks, 2024. URL https://arxiv.org/abs/2412.15204.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and Wen Xiao. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling, 2025. URL https://arxiv.org/abs/2406.02069.

Chi-Chih Chang, Wei-Cheng Lin, Chien-Yu Lin, Hung-Yueh Chiang, Yash Akhauri, Xilai Dai, Huiqiang Jiang, Yucheng Li, Luis Ceze, Kai-Chiang Wu, and Mohamed S. Abdelfattah. xkv: Cross-layer kv-cache compression via aligned singular vector extraction, 2026. URL https://arxiv.org/abs/2503.18893.

Zhuofu Chen, Zhihao Zhang, Lijie Yang, Zikun Li, and Zhihao Jia. Tidaldecode: Fast and accurate llm decoding with position persistent sparse attention. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/pdf?id=EkfLaCJ7bk.

DeepSeek-AI. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model, 2024. URL https://arxiv.org/abs/2405.04434.

Yaosheng Fu, Guangxuan Xiao, Xin Dong, Song Han, and Oreste Villa. Sparda: Sparse decoupled attention for efficient long-context llm inference, 2026. URL https://arxiv.org/abs/2606.04511.

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

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?, 2024. URL https://arxiv.org/abs/2404.06654.

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, 2024. URL https://arxiv.org/abs/2404.14469.

Junlin Mu, Hantao Huang, Jihang Zhang, Minghui Yu, Tao Wang, and Yidong Li. SALS: Sparse attention in latent space for KV cache compression, 2025. URL https://arxiv.org/abs/2510.24273.

Luka Ribar, Ivan Chelombiev, Luke Hudlass-Galley, Charlie Blake, Carlo Luschi, and Douglas Orr. Sparq attention: Bandwidth-efficient llm inference, 2024. URL https://arxiv.org/abs/2312.04985.

Prajwal Singhania, Siddharth Singh, Shwai He, Soheil Feizi, and Abhinav Bhatele. Loki: Low-rank keys for efficient sparse attention, 2024. URL https://arxiv.org/abs/2406.02542.

Yutao Sun, Yanqi Zhang, Li Dong, Jianyong Wang, and Furu Wei. You only index once: Cross-layer sparse attention with shared routing, 2026. URL https://arxiv.org/abs/2606.06467.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context llm inference, 2024. URL https://arxiv.org/abs/2406.10774.

Zhiqiang Xie, Zhangheng Huang, Tingwei Huang, Ziyi Xu, Ruiyang Ma, and Christos Kozyrakis. Hisparse: Scaling sparse-attention decoding with hierarchical kv cache management, 2026. URL https://arxiv.org/abs/2608.07009.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Y. X. Wei, Lean Wang, Zhiping Xiao, Yuqing Wang, Chong Ruan, Ming Zhang, Wenfeng Liang, and Wangding Zeng. Native sparse attention: Hardware-aligned and natively trainable sparse attention, 2025. URL https://arxiv.org/abs/2502.11089.

A ADDITIONAL CONFIGURATION DETAILS

Quantity	Symbol	Qwen3-4B	DeepSeek-R1-Distill-Llama-8B
Backbone geometry			
Number of layers	L	36	32
Residual hidden width	d_{model}	2560	4096
Native head width	d_{head}	128	128
Query / KV heads	H_q/H_{kv}	32 / 8	32 / 8
Selector choices in the present configuration			
Content latent width	d_c	192	192
Positional latent width	d_r	64	64
Total latent width	$d_z = d_c + d_r$	256	256
Scorer MLP width	d_{ff}	256	256
Source spacing	p	5	5
Nonempty source groups	$\lvert\mathcal{S}_+\rvert$	7	7
Dense sources / destinations	counts	8 / 28	7 / 25
Dense-layer read floor	fraction of layers	22.222%	21.875%
Latent / native KV precision	b_z, b_{KV}	BF16; 2 bytes each	BF16; 2 bytes each

Table 8: **Model-specific dimensions and configurable hyperparameters.** Qwen's final layer is a dense source with no destinations.

Each model uses 64 distinct general-text books and three disjoint validation books. Source hidden states are collected for the full causal history. The sampled query positions determine which losses are evaluated, not which historical tokens are available to the encoder. No benchmark answers enter selector training. Table 9 records the common optimization choices. Validation selects the best checkpoint among initialization (epoch 0) and the trained epoch checkpoints. The selected checkpoint is exported to BF16 before validation text supplies score moments for each destination/head.

Setting	Symbol, where used	Reported configuration
Training / validation documents	—	64 / 3 per model
Training prefixes (8K/16K/32K/64K)	—	Qwen: 16/16/32/0 books; DeepSeek: 16/16/16/16
Validation prefixes	—	One book each at 8K, 16K and 32K
Supervised positions / document	—	512 consecutive teacher-forced positions after the prefix
Training initialization seed	—	3407; document shuffle seed $3407 + e$ for epoch index $e = 0, 1, 2$
Epochs; query batches / document	—	3; 128
Queries / batch; updates / source	—	4; 24,576
Optimizer; learning rate	—	AdamW; 3×10^{-5}
Weight decay; gradient norm clip	—	0.01; 1.0
Output-loss weight	λ_o	0.1
Output / score stabilizer	ϵ_o, ϵ_s	10^{-5}; 10^{-12}
RMS stabilizer	ϵ_n	Native model setting
Evaluated threshold sweep	τ	$\{-0.5, 0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 5, 6\}$
Training arithmetic	—	FP32 master weights; BF16 autocast
Checkpoint export	—	BF16 encoder and scorer weights

Table 9: Optimization and calibration settings shared by the two-model configuration.

Training windows are centered in each tokenized book: for book length M and prefix length N, the offset is $\lfloor (M - N - 513)/2 \rfloor$. Each update samples four of the 512 supervised positions uniformly with replacement; all preceding tokens remain available for causal supervision. Document identifiers, URLs, text/token hashes, offsets, splits and selected checkpoint fingerprints are retained in the experiment manifests. They are not supplied by the current illustrative source release.

Qwen evaluation input construction. We render the official LongBench v2 no-CoT prompt, tokenize it and, if it exceeds 32,768 tokens, keep its first and last 16,384 tokens. The retained tokens are decoded with special tokens skipped, then wrapped in the native Qwen chat template with a generation prompt and thinking disabled. This final retokenization and chat envelope can exceed the pre-template cap: the 503 input prefixes range from 10,154 to 32,781 tokens. Each prefix plus the 128-token generation allowance fits the configured 40,960-token limit. Dense and sparse methods reuse the same serialized input tokens; original benchmark length categories refer to text before this truncation.

Storage accounting. Historical source hidden states are retained as training records, but deployment keeps the latents instead of an additional hidden-state history. For batch size B, token capacity N, S_{alloc} allocated source arrays and b_z bytes per latent scalar, their storage is $b_z S_{\text{alloc}} B N d_z$. Original K/V storage remains $2 b_{\text{KV}} B L N H_{\text{kv}} d_{\text{head}}$, where b_{KV} is bytes per native cache scalar. Packed maps add roughly $B H_q N/8$ bytes per destination, plus metadata. We include allocated empty source slots when reporting storage. Weights, temporary scores and workspaces are additional costs.

For D destination layers and S nonempty sources, the encoder/scorer parameter count is

$$D H_q (d_{\text{head}} d_{\text{model}} + 2 d_{\text{head}} d_{\text{ff}} + d_{\text{head}} d_z) + S d_{\text{model}} d_z.$$

The present Qwen and DeepSeek selectors contain 386,269,184 and 505,413,632 trainable parameters, respectively. Their BF16 parameter payloads are 0.72 and 0.94 GiB, excluding fixed normalization buffers, latents, maps and workspace. These are calculated weight sizes, not measured peak memory or memory savings.

B Complete quality measurements

Table 10 reports every tested threshold, including the settings where quality collapses. Selected KV is averaged per question over original-cache unions, with dense sources included; it excludes latents and bitmap traffic. The full 56-question DeepSeek Dense control combines 54 recovered questions with two additional questions whose IDs are disjoint and whose input-token hashes and decoding settings match the sparse runs. Qwen uses exactly the same 503 IDs for every method. Neither comparison mixes reference-only quality with a different fast implementation.

τ	Qwen correct/503	Qwen score	KV (%)	DeepSeek score	KV (%)
Dense	153	30.42	100.00	49.40	100.00
-0.5	155	30.82	93.86	55.89	92.28
0	154	30.62	85.55	51.07	85.28
0.5	156	31.01	72.40	58.24	74.99
1	156	31.01	56.47	63.30	61.95
1.5	159	31.61	41.87	56.88	48.51
2	157	31.21	31.58	59.08	37.17
2.5	157	31.21	25.99	47.11	29.51
3	140	27.83	23.61	38.87	25.36
3.5	125	24.85	22.75	17.68	23.45
4	102	20.28	22.43	6.49	22.73
5	0	0.00	22.25	1.19	22.35
6	0	0.00	22.23	0.00	22.29

Table 10: Complete learned-z threshold sweeps. Qwen reports correct answers and percentage; DeepSeek uses each task's official score, averaged over the balanced 56-question diagnostic.

Paired uncertainty. We resample question-wise method-minus-Dense differences within each task family, preserving each family's sample size (10,000 draws; seed 260926). Table 11 gives percentile intervals in score points. These are descriptive, without multiplicity correction across thresholds. A positive interval in the development-exposed DeepSeek diagnostic does not turn a selected threshold into a held-out result; an interval containing zero is not evidence of equivalence. Tables 12 and 13 expose task and length heterogeneity.

Model	τ	Difference (points)	95% paired interval
Qwen	1	0.60	[-0.20, 1.59]
Qwen	2	0.80	[-0.40, 1.99]
Qwen	2.5	0.80	[-0.60, 2.19]
DeepSeek	1	13.90	[4.70, 22.62]
DeepSeek	2	9.67	[2.38, 16.52]
DeepSeek	2.5	-2.29	[-11.16, 6.37]

Table 11: Paired differences from native Dense and descriptive 95% intervals.

Model	Task family	n	Dense	$\tau = 1$	$\tau = 2$
Qwen	Code Repository Understanding	50	32.00	32.00	34.00
Qwen	Long In-context Learning	81	32.10	33.33	32.10
Qwen	Long Structured Data Understanding	33	24.24	27.27	24.24
Qwen	Long-dialogue History Understanding	39	33.33	33.33	33.33
Qwen	Multi-Document QA	125	28.00	28.00	28.00
Qwen	Single-Document QA	175	31.43	32.00	33.14
DeepSeek	cwe	8	0.00	2.50	6.25
DeepSeek	fwe	8	20.83	37.50	41.67
DeepSeek	niah_multikey_1	8	87.50	87.50	87.50
DeepSeek	niah_multivalue	8	62.50	78.12	78.12
DeepSeek	niah_single_1	8	100.00	100.00	100.00
DeepSeek	qa_2	8	37.50	62.50	25.00
DeepSeek	vt	8	37.50	75.00	75.00

Table 12: Task-family breakdowns. DeepSeek task names follow RULER.

Qwen length stratum	n	Dense	$\tau = 1$	$\tau = 2$
long	108	31.48	33.33	34.26
medium	215	23.72	24.19	24.65
short	180	37.78	37.78	37.22

Table 13: Qwen scores by the original LongBench v2 length labels, before model-specific input truncation.

Baseline settings. Quest uses 16-token pages, keeps its first two layers dense and selects a fraction of pages for each query head. Qwen's SparQ policy uses $r = d_{\text{head}}/4$ query coordinates and $k = N/8$ selected entries; DeepSeek settings below report the registered compression ratio. SnapKV and PyramidKV use the pinned published head-selection rules, observation window 32, average-pooling width 5 and PyramidKV parameter $\beta = 20$. Their reference integrations retain a backing KV cache and construct head-specific selected caches. SeerAttention-R quality uses the published pretrained gate with masked reference attention; its separately timed official Triton kernel is tested for output agreement in Appendix C. These quality integrations alone do not establish optimized cache capacity or kernel performance.

Model	Baseline/setting	Score	Selected KV (%)
Qwen	Quest	29.62	31.20
Qwen	SparQ	30.42	24.05
Qwen	SnapKV	30.22	27.68
Qwen	PyramidKV	30.62	26.99
Qwen	SeerAttention-R	30.62	14.34
DeepSeek	PyramidKV $c = 16384$	50.71	37.34
DeepSeek	PyramidKV $c = 32768$	51.73	66.40
DeepSeek	PyramidKV $c = 4096$	42.95	12.11
DeepSeek	PyramidKV $c = 8192$	44.85	21.34
DeepSeek	Quest $f = 0.1$	54.05	18.01
DeepSeek	Quest $f = 0.2$	53.93	32.08
DeepSeek	Quest $f = 0.3$	61.99	44.39
DeepSeek	Quest $f = 0.4$	63.57	55.44
DeepSeek	SnapKV $c = 16384$	56.34	39.00
DeepSeek	SnapKV $c = 32768$	56.10	66.41
DeepSeek	SnapKV $c = 4096$	39.17	12.74
DeepSeek	SnapKV $c = 8192$	50.06	22.26
DeepSeek	SparQ ratio 16	52.65	7.21
DeepSeek	SparQ ratio 2	55.03	40.54
DeepSeek	SparQ ratio 4	58.66	23.88
DeepSeek	SparQ ratio 8	54.14	13.52

Table 14: All available baseline settings on Qwen503 and DeepSeek56. Qwen uses Quest fraction 0.2, SparQ's policy above, and 4096-token SnapKV/PyramidKV/SeerAttention-R budgets.

B.1 Full RULER baseline coverage

This separate suite contains 13 tasks with 500 questions each per method. Inputs are identical across its five methods within every task. It uses the pinned official RULER prompts, task-specific generation caps and scoring. Quest uses fraction 0.2, SparQ the registered ratio 4, and SnapKV/PyramidKV capacity 8192. Table 15 provides the completed baseline scores; LayerTrace has not been evaluated on this full suite. Its 56-question diagnostic above is a different population and must not be compared to these aggregate scores.

Task (500 each)	Dense	Quest	SparQ	SnapKV	PyramidKV
cwe	1.00	3.16	0.92	2.96	2.12
fwe	45.73	42.07	47.00	44.07	38.20
niah_multikey_1	87.00	88.60	87.80	87.20	89.80
niah_multikey_2	85.40	87.60	90.20	18.60	10.20
niah_multikey_3	8.80	13.60	14.80	1.20	1.80
niah_multiquery	83.05	75.75	74.15	82.60	82.05
niah_multivalue	70.55	67.05	71.35	39.45	39.10
niah_single_1	100.00	100.00	100.00	100.00	100.00
niah_single_2	94.00	92.00	94.20	92.60	91.80
niah_single_3	90.80	84.60	92.20	19.20	18.60
qa_1	39.20	40.20	38.80	41.20	37.40
qa_2	36.40	34.40	36.00	38.20	35.60
vt	0.00	0.04	0.48	0.92	1.12
Mean	57.07	56.08	57.53	43.71	42.14

Table 15: DeepSeek full 13-task RULER baseline suite: 6500 questions per method, with the unweighted task mean in the last row. No full-suite LayerTrace score is claimed.

B.2 Numerical controls

Qwen's quality runs pass 24/24 fast/reference checks, including 4/4 at $\tau = 2$. Quality and repeated timing specifications agree on all 21 trained-artifact path/hash pairs, seven source-group mappings and 14 shared source-file hashes, including the core selector and sparse-attention kernels. The timing

run's seven encoder checks pass, but its B8 rows contain no separate whole-model fast/reference KL check. Encoder checks do not substitute for that test. These records support implementation identity without establishing full-trajectory or statistical quality equivalence. All-selected checks test whether changing the attention implementation, without selection, matches native logits at KL $\leq 10^{-3}$. Initial Qwen checks pass 4/4 and DeepSeek checks pass 2/4; the largest reported initial DeepSeek failure is KL 0.001789. The DeepSeek fast/reference checks fail 7/24; those generated fast-path task scores are reported directly. Its initial quality sweep uses one H100 and three RTX GPUs, so the later RTX diagnostic does not establish identical numerical behavior across the earlier inputs/hardware. These tests answer an implementation-equivalence question separately from the observed task-score comparison. A subsequent DeepSeek diagnostic uses identical inputs for native, all-selected reference and all-selected fast execution. At 4K, both alternatives pass (KL 9.57×10^{-7} and 4.41×10^{-6}). At 64K, reference KL is 0.001544 (fail) and fast KL is 0.000944 (pass); top-1 logits agree in these four cases. The follow-up does not establish a universal pass or identify one cause of the earlier discrepancies. The observed higher task scores therefore remain reported as such, without attributing their gains solely to the selector.

C TIMING, GPU-WORK ATTRIBUTION AND CAPACITY

Timings use an RTX PRO 6000 Blackwell Server Edition with 96GB VRAM and synchronized eager whole-decoder execution. The initial batch sweeps use eight warmup and 24 measured steps per repetition, repeated three times. The independent Qwen 32K repeat retains the original ordered sweep (batches 1/4/8; Dense then $\tau = 0, \ldots, 6$) with five repetitions. Tables report the median of repetition medians. Four fixed-corpus streams repeat into independently allocated cache rows. Prefill and answer sampling are outside these decoder-step numbers. Baseline single-batch runs use their recorded two-repetition controls; larger-batch and official Seer runs use three repetitions. Compilation and first-step startup are excluded from steady-state time.

C.1 CONTEXT AND BATCH COVERAGE

Missing values use explicit codes throughout the measurement tables. **NM** means no completed measurement; the remaining unmeasured baseline settings are Qwen 32K/B8 SnapKV and PyramidKV at the 16K budget. Table 21 includes the four LayerTrace B2 configurations at 32K/64K and all completed DeepSeek 64K/B4/B8 baseline outcomes, with each run's own Dense control. **CAP** marks an unattempted native-KV capacity bound, calculated below. **OOM** instead records an actual out-of-memory outcome for DeepSeek 64K/B8 Quest, compiled SparQ, SnapKV and PyramidKV. These outcomes concern the measured integrations: Quest materializes FP32 gathered keys, SparQ retains a dual key layout, and the SnapKV/PyramidKV adapters retain full backing KV plus query-head-expanded caches. They do not establish an inherent method limitation or the capacity of native optimized implementations. **NI** marks DeepSeek SeerAttention-R, for which no validated model-specific gate/adapter was integrated in this study. **NC** means no numerical-check result in the timing record, not a passed check; **n/a** means no selector setting. Measured numerical failures retain their times and check counts.

Tables 16 and 17 report complete integer-threshold sweeps. Qwen 32K batches 1/4/8 use the independent repeat with matching input streams and learned checkpoints; other cells retain their original run and Dense control. At batch 8, Dense's five repetition medians span 37.2517–37.2768ms and LayerTrace $\tau = 2$ spans 25.3437–25.3582ms, with aggregate 37.2591/25.3496ms. The initial run's 37.2544/25.3508ms agrees at this operating point. The range describes those repetitions, not all possible deployments or historical executions.

Tables 18 and 19 retain other context/batch cells, each with its own Dense control. Qwen 64K/128K uses YaRN factor 4 with original context 32768; 256K uses factor 8. DeepSeek has no override and 256K exceeds its native 128K. These are execution measurements, not expanded-context quality evidence. In particular, DeepSeek 256K/B2 at $\tau = 2$ is slower than Dense (60.75 vs 58.85ms), while Qwen 16K/B1 is also slower (28.68 vs 21.16ms). The table retains these losses.

Context	B	Dense	$\tau = 0$	1	2	3	4	5	6
16K	1	21.16	28.67	28.69	28.68	28.65	28.60	28.66	28.63
16K	4	21.78	29.73	29.79	29.74	29.79	29.76	29.81	29.81
16K	8	27.54	31.32	29.92	29.96	29.95	30.00	29.98	29.96
16K	16	40.44	42.21	33.21	33.20	33.28	33.23	33.02	32.96
16K	32	64.53	64.90	46.33	39.41	39.45	39.46	39.39	39.46
32K	1	13.81	16.96	16.50	16.42	16.42	16.44	16.43	16.39
32K	4	24.00	26.44	22.08	19.27	19.23	19.22	19.70	19.24
32K	8	37.26	39.84	31.24	25.35	23.72	23.48	23.47	23.45
32K	16	62.68	65.36	48.23	36.34	33.29	32.84	32.82	32.80
64K	8	62.36	63.91	46.89	36.02	33.00	32.52	32.48	32.47
128K	2	39.52	44.29	36.63	32.09	32.10	32.19	32.24	32.23
256K	2	61.77	70.30	55.13	41.03	34.91	33.51	33.34	33.32

Table 16: Qwen integer-threshold sweeps, whole-decoder ms/step. The 32K batches 1/4/8 use five repeats; remaining cells use three.

Context	B	Dense	$\tau = 0$	1	2	3	4	5	6
16K	1	14.64	17.22	16.72	16.46	16.41	16.40	16.40	16.40
16K	4	19.14	21.79	19.92	18.65	18.35	18.33	18.35	18.37
16K	8	25.34	27.45	23.46	21.10	20.44	20.39	20.39	20.39
16K	16	37.79	39.93	32.01	27.39	26.16	26.03	26.03	26.03
16K	32	60.68	62.92	46.87	37.52	35.26	35.01	35.01	35.00
32K	1	16.68	22.69	20.94	22.23	23.07	20.99	23.29	23.75
32K	4	25.30	30.10	26.65	23.53	24.10	23.17	23.07	22.98
32K	8	37.00	41.48	34.65	29.30	25.86	25.52	25.48	25.87
32K	16	60.60	66.82	52.99	40.55	35.99	35.16	35.15	35.13
64K	1	18.96	21.96	20.22	18.37	17.75	17.67	17.66	17.63
64K	4	36.09	41.72	36.20	29.78	25.89	24.51	24.30	24.28
64K	8	58.94	67.72	56.72	43.84	36.04	33.49	33.07	33.04
128K	2	36.04	42.79	38.52	32.78	28.38	26.07	25.10	24.60
256K	2	58.85	73.96	68.63	60.75	53.71	48.29	43.99	39.97

Table 17: DeepSeek integer-threshold sweeps, whole-decoder ms/step.

Context	B	Sweep	Dense	a	b	c	d	e	f
64K	1	I	16.96	20.40	19.50	18.35	17.31	16.71	16.71
64K	1	E	17.04	16.64	16.62	16.60	16.58	16.56	16.54
64K	4	I	36.53	41.72	38.46	34.23	30.00	26.58	24.47
64K	4	E	36.62	23.45	22.92	22.72	22.67	22.64	22.66
128K	1	I	23.62	27.41	26.15	24.29	22.08	19.98	18.60
128K	1	E	23.72	18.73	18.71	18.40	18.45	18.34	18.35
128K	4	I	61.63	71.82	66.79	59.56	51.54	43.88	38.17
128K	4	E	61.73	34.88	33.08	32.31	31.92	31.82	31.85
256K	1	I	36.84	44.97	42.75	39.25	35.26	31.18	29.49
256K	1	E	36.34	24.33	23.05	22.59	22.90	22.64	23.44

Table 18: Other Qwen context/batch cells, ms/step. Sweep I columns a–f denote $\tau = -0.5, 0, 0.5, 1, 1.5, 2$; sweep E denotes $2.5, 3, 3.5, 4, 5, 6$.

Context	B	Sweep	Dense	a	b	c	d	e	f
128K	1	I	24.97	29.25	28.69	27.71	26.34	24.76	23.29
128K	1	E	24.91	22.03	21.16	20.48	20.10	19.61	19.34
128K	4	I	58.45	71.39	69.51	66.24	61.43	56.00	50.61
128K	4	E	58.47	45.88	41.84	38.73	36.69	34.45	33.49
256K	1	I	36.06	44.65	43.96	42.77	41.08	38.98	36.87
256K	1	E	36.04	34.88	33.26	31.89	30.63	28.45	26.51

Table 19: Other DeepSeek context/batch cells, ms/step; sweep definitions as in Table 18.

Capacity bounds. Resident native KV requires $2BLTH_{\mathrm{kv}}d_{\mathrm{head}}b_{\mathrm{KV}}$ bytes for batch B and context T. DeepSeek 16K/B64,32K/B32,64K/B16,128K/B8 and 256K/B4 each require 128GiB; the same Qwen cells require 144GiB. All exceed GPU VRAM before weights, source latents, bitmaps and workspace. Lower logical reads do not remove resident KV or imply proportional memory-capacity savings.

C.2 Measured baseline integrations

Table 20 consolidates Qwen 32K measurements. Each method/batch column is an independently measured integration; NM cells lack a completed run. The Dense B1 value here is the native-baseline run, while Table 5 uses the Dense control from the same repeated LayerTrace job. We do not substitute one control for another to calculate paired speedups.

Quest uses the pinned page-selection policy with reference gather/attention rather than its official optimized kernel. SnapKV/PyramidKV use published selection policies and FlashAttention, with a full backing cache and per-token compressed-cache append in the adapter. Compiled SparQ uses official arithmetic and a dual key layout: at Qwen 32K/B8, ratio 4 fails 1/2 sampled output checks and ratio 8 fails 2/2; at B1 the corresponding failures are 1/2 and 0/2. Thus reference-path benchmark quality cannot automatically certify these compiled configurations. These measurements do not assume equally optimized integrations.

Measured integration	Setting	$B = 1$ ms	$B = 8$ ms
Dense	0	14.16	37.26
Quest (gather reference)	0.2	37.97	235.05
Quest (gather reference)	0.4	58.34	403.36
SparQ (compiled)	8	24.59	52.96†
SparQ (compiled)	4	20.40†	85.84†
SnapKV (FlashAttention)	4096	21.01	51.99†
SnapKV (FlashAttention)	16384	30.88	NM
PyramidKV (FlashAttention)	4096	14.65	51.96†
PyramidKV (FlashAttention)	16384	29.89	NM
SeerAttention-R (official kernel)	4096	26.28†	47.95†
SeerAttention-R (official kernel)	16384	30.25	58.41†
LayerTrace	$\tau = 2$	16.42	25.35

Table 20: **Qwen 32K measured whole-decoder latency.** Settings denote Quest fractions, SparQ registered ratios, cache budgets or LayerTrace threshold. NM: not measured; †: failed reported timing check. Quality-path and speed-path agreement is qualified in the text.

Additional batch measurements and controls. Table 21 retains each added configuration's same-run Dense control, including controls recovered from OOM runs. Timings use three fresh-cache repetitions and unchanged selection policies. The four LayerTrace B2 runs report no whole-model numerical-check result (NC). DeepSeek 64K/B4 compiled SparQ, SnapKV and PyramidKV each pass both sampled logits checks; Qwen 32K/B8 SnapKV and PyramidKV each fail one of two (maximum KL 0.001872/0.001949 versus 10^{-3}). Their measured latency is not certified by reference-path task quality. OOM is a capacity outcome, not an infinite latency or a numerical-check failure.

Model	Context/B	Integration	Setting	Dense ms	Method ms	Checks passed
DeepSeek	32K/2	LayerTrace	$\tau = 2$	19.58	22.09	NC
DeepSeek	64K/2	LayerTrace	$\tau = 2$	24.89	22.57	NC
Qwen	32K/2	LayerTrace	$\tau = 2$	17.56	17.78	NC
Qwen	64K/2	LayerTrace	$\tau = 2$	24.05	19.25	NC
DeepSeek	64K/4	PyramidKV	16384	36.27	84.60	2/2
DeepSeek	64K/8	PyramidKV	16384	59.02	OOM	NC
DeepSeek	64K/4	Quest	0.2	36.80	211.67	NC
DeepSeek	64K/8	Quest	0.2	59.76	OOM	NC
DeepSeek	64K/4	SnapKV	16384	36.80	85.28	2/2
DeepSeek	64K/8	SnapKV	16384	60.10	OOM	NC
DeepSeek	64K/4	Compiled SparQ	8	36.17	49.47	2/2
DeepSeek	64K/8	Compiled SparQ	8	59.03	OOM	NC
Qwen	32K/8	PyramidKV	4096	37.28	51.96†	1/2
Qwen	32K/8	SnapKV	4096	37.24	51.99†	1/2

Table 21: **Added batch configurations and their own Dense controls.** OOM: observed integration capacity failure; NC: no numerical check supplied. Controls are retained per execution and are not substituted into another run's speedup.

SeerAttention-R kernel integration. The pinned official Triton sparse-decode kernel is connected to the Hugging Face model and cache through our adapter. At budget 4096 it takes 26.28ms at B1 and 47.95ms at B8. Table 22 reports all four configurations and their repeated timing ranges. Four of eight same-policy, same-pre-step-cache checks against masked reference attention exceed KL 10^{-3}. This is not itself an observed benchmark-score drop, but it leaves quality agreement unverified; the reference quality in Table 4 must not be combined with these times to claim a confirmed quality-matched comparison.

B	Budget	ms/step	Repetition-median range	Failed/checks
1	4096	26.28	26.22–26.35	1/2
1	16384	30.25	30.18–30.66	0/2
8	4096	47.95	46.90–48.48	2/2
8	16384	58.41	58.02–61.82	1/2

Table 22: Official SeerAttention-R kernel with our HF/cache adapter, Qwen 32K. Ranges are three repetition medians; failures retain the prespecified KL threshold.

Model	Integration	Setting	ms/step ($B = 1$)	Checks passed
DeepSeek	Native Dense	0	19.07	NC
DeepSeek	PyramidKV	4096	16.74	NC
DeepSeek	PyramidKV	16384	30.45	NC
DeepSeek	Quest	0.2	60.44	NC
DeepSeek	Quest	0.4	94.93	NC
DeepSeek	SnapKV	4096	20.47	NC
DeepSeek	SnapKV	16384	30.98	NC
DeepSeek	Compiled SparQ	4	32.34	2/2
DeepSeek	Compiled SparQ	8	32.31	2/2

Table 23: DeepSeek 64K/B1 baseline integration times. Settings and output-check conventions follow the Qwen comparison. NC means the timing record supplies no check, not a pass.

Model	Context	B	Method/setting	ms/step	Checks passed
DeepSeek	16K	16	Quest 0.2	212.56	NC
DeepSeek	16K	16	Quest 0.4	360.38	NC
DeepSeek	16K	16	Compiled SparQ 4	82.08	1/2
DeepSeek	16K	16	Compiled SparQ 8	52.33	0/2
DeepSeek	16K	16	Dense	38.41	NC
DeepSeek	16K	16	LayerTrace $\tau = 1$	32.77	NC
DeepSeek	16K	16	LayerTrace $\tau = 2$	28.11	NC

Table 24: DeepSeek 16K/B16 follow-up integrations, with separate Dense/LayerTrace controls. NC: no check recorded. No matched 16K task-quality sweep is available.

C.3 Exclusive component accounting

Figure 4 uses separate four-step traces at Qwen 32K/B8 and DeepSeek 16K/B16. Each physical CUDA kernel/copy event inside the decoder is assigned once to its innermost applicable operation; no event is unassigned or outside the model. The Dense bars use the same layer indices as LayerTrace's source/destination groups. Prediction includes the scorer, latent encoding/append, thresholding, bitmap packing and metadata. QKV/output projections, positional/cache operations, FFN, normalization/residual work and the language-model head belong to the remaining decoder category.

GPU event-duration sums measure GPU work, not elapsed time. Concurrency, host launch gaps and profiling overhead preclude additive wall-time attribution. In particular, these Qwen traces and the independent repeated latency run are separate executions; their totals are not fitted to each other. All components remain unscaled. Baseline selection and attention are combined where the trace does not separate them, rather than inventing a split. Only the corrected exclusive attribution of the eight baseline traces is used; nested parent/child durations are never stacked together.

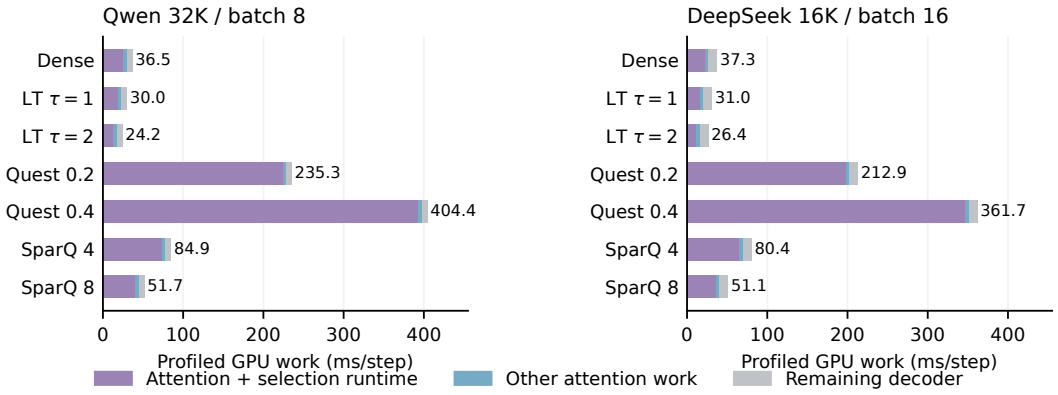

Figure 5: **Exclusive GPU work for the measured integrations.** Purple combines attention kernels with selection/index work. Blue separates projections and other attention-block work from the remaining decoder in gray; these blue/gray components are combined in Figure 4. Quest uses reference gather/attention and SparQ uses compiled arithmetic with the numerical limits above. These independent trace sums are not elapsed latency or a ranking of equally optimized kernels.

D APPLICABILITY TO HYBRID ATTENTION AND SHARED MLA CACHES

This experiment evaluates Qwen3.5-4B and DeepSeek-V2-Lite-Chat on 128 fixed LongBench v2 questions per model. Questions are stratified by domain and length and fixed before outcomes; every configuration within a model uses identical IDs. These questions exclude the eight used in an initial pilot. Evaluation uses the official two-stage protocol (1024 CoT tokens, then 128 final-answer tokens), temperature 0.1, seed 42, the official parser and 30K input truncation. It is separate from the Qwen3-4B 503-question and DeepSeek-R1-Distill-Llama-8B 56-question comparisons.

Configurations and training. Hybrid one-source configurations use the first full-attention layer to predict layers 2–8. The two-source configuration uses attention ordinal 1 for 2–4 and ordinal 5 for 6–8; recurrent blocks remain unchanged. Historical-token encoders and per-destination/head scorers are trained jointly within each source group on frozen Dense trajectories. Widths are given in Table 25. MLA uses source spacing five across its 27 layers. Its learned-z control follows per-head selection with a newly learned historical-token encoder. The native shared-map adaptation instead uses each source's pretrained compressed content row and positional component (widths in Table 26), and trains a destination-specific vector from the current source output hidden state. Their inner product gives one score per historical token, shared across heads. The map selects native destination cache rows; the pretrained MLA compression remains frozen. Actual destination queries enter only the subsequent attention, never the selector. Neither variant uses fallback.

Five fresh configurations train on 32 general-text documents with two separate development documents for three epochs, 6144 updates per source. Checkpoints minimize development distillation/reconstruction loss. Two thresholds per fit are selected from $\{0, 0.5, 1, 1.5, 2, 2.5, 3\}$ to approach 40% and 60% whole-attention row reads on development text, without benchmark answers. Generated-sequence read rates need not match these targets. Each model also retains a fixed small-data learned-z control (16 training documents, two epochs; one hybrid source or MLA spacing five). It is a separate control, not an exact optimizer continuation or an isolated training-step comparison.

Complete quality and read measurements. Tables 25 and 26 retain all 14 conditions, including those below Dense. Correct and parsed counts have denominator 128. W/L counts paired wins/losses against the model's Dense control; p is the exact two-sided McNemar value, unadjusted for multiple configurations. No selected setting has a subsequent confirmatory holdout. Reads count logical native-cache rows, combining CoT and final-answer stages with generated-token weighting within each question, then averaging across questions. Dense source attention is included; recurrent computation and latent/scorer/bitmap traffic are excluded. These percentages are not physical bytes, bandwidth, latency or resident-memory savings.

Configuration	Width	τ	Correct	Parsed	Reads (%)	W/L	p
Dense	n/a	n/a	29	126	100.00	0/0	1.000
One source	1024	0.5	34	116	74.90	22/17	0.522
One source	1024	1	25	84	53.25	16/20	0.618
One source	256	0.5	19	114	74.23	12/22	0.121
One source	256	1	19	100	52.40	13/23	0.132
Two sources	256	0.5	35	126	73.29	21/15	0.405
Two sources	256	1.5	33	125	38.11	24/20	0.652
Fixed small-data control	256	1	24	86	55.08	14/19	0.487

Table 25: **Qwen3.5-4B hybrid attention: all configurations.** Width is the learned historical-token latent dimension. Dense: 29/128 correct (22.66%).

Configuration	Width	τ	Correct	Parsed	Reads (%)	W/L	p
Dense	n/a	n/a	26	109	100.00	0/0	1.000
Learned source z	256	1	16	102	88.75	7/17	0.064
Learned source z	256	1.5	31	113	74.55	18/13	0.473
Native shared map	512+64	0	26	95	68.42	12/12	1.000
Native shared map	512+64	0.5	22	98	58.58	10/14	0.541
Fixed small-data control	256	1	16	83	94.43	9/19	0.087

Table 26: **DeepSeek-V2-Lite-Chat MLA: all configurations.** Width is learned-z dimension or native content+positional width for the shared-map adaptation. Dense: 26/128 correct (20.31%).

Interpretation and controls. At similar observed hybrid row reads, one source with width 1024 scores 34/128 at 74.90%, versus 19/128 at 74.23% for width 256; two width-256 sources score 35/128 at 73.29%. At the stricter two-source threshold, 33/128 correct at 38.11% reads and 125 parsed answers support feasibility with substantial logical row reduction. Single fits cannot establish a universal source schedule or a causal capacity effect.

The MLA shared-map setting at $\tau = 0$ matches Dense's aggregate correct count with 31.58% fewer row reads, but parsed answers fall from 109 to 95. Its low Dense accuracy limits a strong quality-preservation conclusion. The learned-z control scores 31/128 at $\tau = 1.5$ and 74.55% reads, but only 16/128 at $\tau = 1$ despite 88.75% reads; the cause of this nonmonotonic result is unestablished. Native/all-selected and map-contract gates pass in all 144 evaluation jobs at short contexts 512/4096. These are not whole-30K numerical-equivalence tests. Reference implementations establish quality/read behavior; no architecture-specific latency or optimized-kernel equivalence was measured.