RANDOM ATTENTION: RETHINKING KV CACHE EVICTION FOR EFFICIENT REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models achieve superior performance on tasks that require extended reasoning, but long chains of thought make the KV cache a severe memory bottleneck. Existing KV cache compression methods share one paradigm: score each cached token by some estimate of how much it will matter later, and keep the top-scoring ones. We show that the selection signal contributes almost nothing. *Random Attention* keeps the prompt and evicts uniformly at random within each attention head, computing no score at all; across four models and six reasoning tasks, it matches the strongest baseline in task performance while delivering 32–43% higher throughput than that method when deployed with vLLM. Controlled experiments explain this by showing that 1) the prompt is the fragile part of the cache, and most of the gap between selectors is just whether their selection signal happened to keep it; 2) the reasoning trace protects itself against eviction with redundancy at two levels, in the text (the model restates what it still needs as it works) and across attention heads (each keeps its own copy of the trace), so once the prompt is safe, a random draw retains enough copies of what the model still needs, and no score is required to pick them.

1 INTRODUCTION

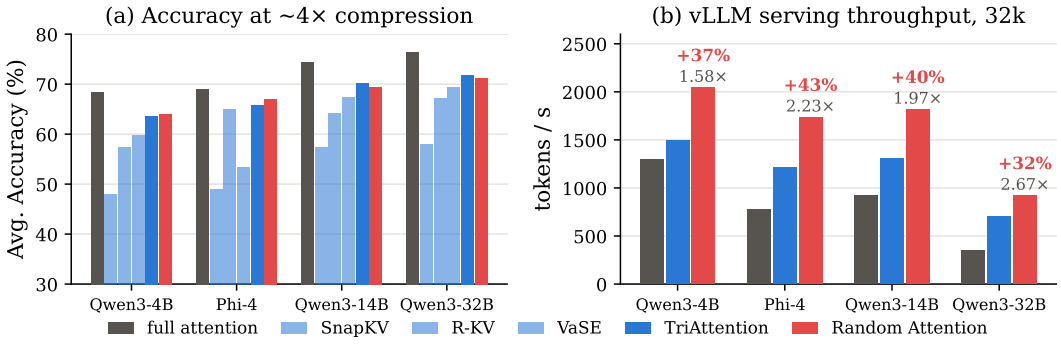


Figure 1: **(a)** Mean accuracy over the six reasoning tasks of Tables 1 and 5 at ∼4× compression: Random Attention matches the strongest baseline on every model (the small gaps to TriAttention at 14B and 32B are mostly driven by code reasoning, where long prompts consume the budget, §4.2). **(b)** vLLM serving throughput at 32k-token generations (Table 4); labels give Random Attention's multiple of full attention and its margin over TriAttention: with no scoring pass it serves 32–43% higher throughput.

Reasoning models (DeepSeek-AI, 2025; OpenAI, 2024; Abdin et al., 2025) solve hard problems by generating chains of thought that run to tens of thousands of tokens. The key-value (KV) cache grows linearly with the length of generation, creating a severe memory bottleneck. KV cache eviction methods have been developed to address this by keeping a fixed budget of KV cache entries and discarding the rest as decoding proceeds. Existing KV cache eviction methods follow the same paradigm: score each cached token by an estimate of how much it will matter later and keep the

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.

top-scoring ones. This line of work is a sequence of better scores, from accumulated attention (Zhang et al., 2023), attention from a recent window (Li et al., 2024), and attention combined with redundancy (Cai et al., 2025), to value magnitude (Chang et al., 2026) and position-dependent key statistics (Mao et al., 2026). Each new score is motivated by heuristics observed to be correlated with task accuracy, and the premise behind all of them is that the score decides accuracy under compression.

We test that premise directly and show that the selection signal contributes almost nothing. *Random Attention* keeps the prompt and evicts uniformly at random within each attention head, computing no score at all. Across four models (Qwen3-4B, 14B and 32B, and Phi-4-reasoning) and six reasoning tasks spanning math, science and code, it is comparable to the strongest baseline (Figure 1a) and even significantly ahead in 31 of the 60 baseline comparisons in the main result table. Served through vLLM integration, it delivers 32–43% more tokens per second at 32k-token generations than the strongest baseline, since it never runs a scoring pass (Figure 1b). The only cell of that table where a baseline is significantly ahead is code reasoning on the largest Qwen3 model, traced in §4.2 to long prompts that shrink the budget left for the trace. The selection signal, in other words, contributes almost nothing beyond uniform random selection.

Two controlled experiments explain why. First, **the prompt is the fragile part of the cache**. Prior evictors differ in whether the prompt survives, some pinning it by rule and others leaving it to the score, so once every method is given the same rule (keep the prompt), most of the gap between them disappears, and each method's gain is ordered by how much of the question its score has been losing (§5.1). Second, **the reasoning trace protects itself**. It is stored redundantly at two levels, in the text, because the model restates what it is still using, and across attention heads, because every head holds its own copy of every token and eviction decides per head which copies die. A planted-fact probe shows the model reading a value from whichever heads still hold it: a fact kept in one head is almost never retrieved, kept in several it almost always is, and the shape of the surviving copies does not matter (§5.2). Once the prompt is safe, a random draw keeps enough copies of what the model still needs; what a signal still buys is the rare fact stated once and never restated, which reasoning traces seldom produce (§5.3).

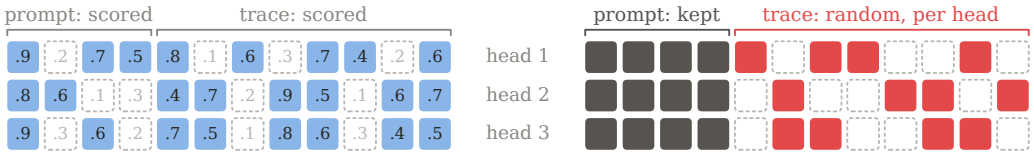
Our findings have two practical implications. First, Random Attention is a deployable method in its own right. It needs no calibration, no tuning and no scoring pass, and it is the fastest evictor we measured at equal accuracy, so it is a reasonable default for serving reasoning models under a memory budget, and the baseline that any new selection signal has to beat at matched budget and matched prompt protection. Second, the findings redirect what eviction research should optimise. The accuracy of an evictor is decided by what it protects, not by how it ranks the rest, which moves the open questions to where protection still matters: how to budget long prompts, especially in code tasks where protecting the entire prompt consumes a substantial fraction of the cache budget, and how to recover rare once-stated facts that only a content-dependent signal can preserve (§5.3).

2 PRELIMINARIES

Setting. We study KV cache *eviction during decoding*. This is the regime reasoning models create: a model answering a short (e.g. 200-token) math question may generate a very long chain of thought (e.g. more than 10,000 tokens). Eviction permanently discards key-value pairs once the cache reaches a budget, which bounds memory but risks irrecoverable loss; it is therefore distinct from sparse-attention *selection*, which attends to a subset but keeps every pair in memory and so still grows linearly with sequence length. Everything below concerns eviction.

Notation. Let t index decode steps. At step t an attention head holds N cached key-value pairs (k_i, v_i), $i = 1, \dots, N$, with $k_i, v_i \in \mathbb{R}^d$, where i is the position of the token that produced the pair; positions $1, \dots, \ell_{\mathrm{p}}$ are the prompt. The head's output for the current query q_t is $o_t = \sum_{i \le N} \alpha_i^{(t)} v_i$, with attention weights $\alpha_i^{(t)} = \mathrm{softmax}_i\big(q_t^\top k_i / \sqrt{d}\big)$. We write v_{ij} for the j-th coordinate of v_i and $\|k_i\|$ for the norm of k_i. Eviction decisions are made independently in every layer and KV head, as is standard; we drop both indices throughout.

Periodic eviction with budget K and buffer r. We adopt the decode-phase framework of Cai et al. (2025) and Chang et al. (2026). The cache keeps a persistent budget of K pairs plus a buffer



(a) Score-based eviction **(b) Random Attention (ours)**

Figure 2: Overview of the methods. **(a)** Baselines keep K positions per head by a score recomputed at every eviction; except in TriAttention, the prompt competes too. **(b)** Random Attention pins the prompt and samples the rest uniformly, independently per head. Dashed cells are evicted.

of the $r \ll K$ most recent pairs, which is never scored. Each decode step appends one pair, so the buffer fills every r steps and triggers an eviction: every candidate i in the candidate set $\mathcal{C}_t$, the cached positions outside the buffer, receives a real-valued score s_i from a policy-specific rule, which may depend on the whole cache and on the past queries, and the K highest-scoring candidates are kept,

$$\mathcal{S}_t \;=\; \text{top-}K_{\,i\in\mathcal{C}_t}\; s_i, \tag{1}$$

returning the cache to $K + r$ entries per head. Eviction is monotonic: a discarded pair is gone for good, as in a memory-bounded deployment. Eviction methods differ mostly in s.

Baselines as choices of s. We write each prior method's s in the notation above, with every score evaluated at the eviction step t. **StreamingLLM** (Xiao et al., 2024) has no score: it keeps the first few (attention-sink) positions and the recent buffer. **H2O** (Zhang et al., 2023) keeps the positions that have received the most attention since they entered the cache, $s_i = \sum_{t'=i}^{t} \alpha_i^{(t')}$. **SnapKV** (Li et al., 2024) uses only the last w queries, $s_i = \sum_{t'=t-w+1}^{t} \alpha_i^{(t')}$, max-pooled over neighbouring positions. **R-KV** (Cai et al., 2025) mixes the SnapKV score with a redundancy penalty, $s_i = \lambda\, s_i^{\text{Snap}} - (1-\lambda)\,\text{softmax}_i(\bar{c})$, where $\bar{c}_i$ is the mean cosine similarity of k_i to the other cached keys, not counting its most recent near-duplicates, so that a restated fact is not kept twice. **VaSE** (Chang et al., 2026) scores values rather than keys: it keeps the $n_{\text{large}} < K$ positions with the largest value range $\max_j v_{ij} - \min_j v_{ij}$ and fills the remaining $K - n_{\text{large}}$ slots by sampling positions with probability proportional to their SnapKV score. **TriAttention** (Mao et al., 2026) predicts the attention a key will receive from a per-head query centre calibrated offline: s_i averages $f_{k_i}(t - i + \delta)$ over future offsets $\delta \in \{1, 2, 4, \ldots, 2^{16}\}$, where f_{k_i} is a trigonometric series in the query–key distance whose coefficients depend on the centre and on k_i's pre-RoPE components, plus a norm term in $\|k_i\|$.

3 Random Attention

As illustrated in Figure 2, Random Attention is a signal-free eviction policy defined by two structural choices and nothing else. The intuition is to separate the irreplaceable input from the model-generated trace: the question is stated once and cannot be recovered if evicted, whereas the trace revisits and restates its intermediates as generation proceeds. We protect the former and use random sampling for the latter:

1. **Protect the question.** Positions $1, \ldots, \ell_{\text{p}}$, the entire prefill (the system prompt, chat template, and question), are never evicted.
2. **Scatter the rest, per head.** Every remaining cached position receives an i.i.d. uniform random score, and each KV head keeps its top-K independently. Sampling without a signal spreads the retained budget evenly over the whole trace, and differently in every head.

In the notation of §2, $s_i{=}{+}\infty$ for $i \le \ell_{\text{p}}$ and $s_i \sim \text{Uniform}(0, 1)$ otherwise, drawn independently per KV head at every eviction; Eq. 1 does the rest, so an eviction costs one rand and one topk. This is the weakest selection signal we can write down: Random Attention is both a deployable method and a null hypothesis. Any signal-based selector that cannot beat it at matched budget is not extracting usable information from its signal.

4 EXPERIMENTS

4.1 SETUP

Baselines. We compare against the four eviction methods of §2, SnapKV, R-KV, VaSE and TriAttention, each run as released at the same budget (detailed configurations in Appendix F). All four are training-free and can run on any model as released, which is the comparison our question calls for. Learned eviction policies (Bui et al., 2026a;b; Dong et al., 2026), which train a scorer for each base model, are discussed in §7. **Full** attention (no eviction) is the ceiling. The diagnostics of §5.1 additionally use *recency+prompt* (keep the KV caches corresponding to the prompt, fill with the contiguous recent window, StreamingLLM-style (Xiao et al., 2024)). All methods are evaluated with FlashAttention-2 kernels (Dao, 2024), without PagedAttention (Kwon et al., 2023).

Models and tasks. We evaluate Qwen3-4B, Qwen3-14B, Qwen3-32B (Yang et al., 2025), and Phi-4-reasoning (14B) (Abdin et al., 2025) across six reasoning tasks spanning math, science, and code: MATH500 (Hendrycks et al., 2021; Lightman et al., 2024) (500 problems), GPQA-Diamond (GPQA-D) (Rein et al., 2024) (198 problems), AIME 2025 and 2026 (30 problems each, reported as one pooled AIME column) and HMMT (60 problems) via MathArena (Balunovic et al., 2026), and LiveCodeBench-v6 medium (Jain et al., 2025) (383 problems; pass@1 by real test execution). Generation uses each model's released sampling settings (temperature 0.6 for the Qwen3 models, 0.8 for Phi-4-reasoning; nucleus p=0.95 for all), and each result is repeated as independently sampled runs: 2 on MATH500, 4 on GPQA-D and LiveCodeBench, and 16 on AIME and HMMT. Every accuracy in the paper is the average over those repeated runs. The main grid fixes each task's budget at $\sim 4\times$ compression of its typical trace ($\sim 3\times$ for LiveCodeBench); the per-head budget K for each task appears in Table 1's header and is detailed in Appendix F. We set the maximum generation length to 32,768 (32k) tokens.

Metrics and statistics. The primary metric is accuracy, judged by whether the final boxed answer is correct (following Chang et al. (2026) and Gao et al. (2026)). Every claimed margin is gated by a paired, problem-clustered percentile bootstrap (95% CI) plus an exact sign test; grid cells significantly below Random Attention are grayed.

4.2 MAIN RESULTS

Table 1 presents the performance on Qwen3-4B, Phi-4-reasoning, and Qwen3-32B; Qwen3-14B replicates the pattern at an intermediate scale in Appendix A. Paired tests put Random Attention significantly ahead in 31 of the table's 60 baseline cells and significantly behind in one.

A selection signal buys nothing on math and science reasoning. On MATH500 and GPQA-D, Random Attention beats VaSE and SnapKV significantly on every model, and R-KV significantly on Qwen3-4B. No selector beats it significantly on these tasks: the leads that do appear (TriAttention by 0.3–0.6 points on two GPQA-D cells) sit inside the noise.

On competition math no selector pulls ahead. Competition math tasks including AIME and HMMT make the comparison ride on a smaller and harder sample. With 16 sampled runs per problem, SnapKV still trails Random Attention significantly on every model, as do R-KV on the Qwen3 models and VaSE on Phi-4-reasoning; no selector in Table 1 is significantly above Random Attention on either task, and the nominal leads run both ways: VaSE edges it on Qwen3-32B by 1.6 and 1.5 points, against a run-to-run standard deviation of about 4 points for both methods on 60-problem sets. These tasks separate only once the budget tightens (§4.3), and then in Random Attention's favour.

On code reasoning most signal-based selectors fall apart due to much longer prompts. LiveCodeBench is the one task with large gaps: SnapKV loses 20–35 points to Random Attention on every model, VaSE collapses on Phi-4-reasoning (0.373, 29 points behind) and is grayed on two of the three models, as is R-KV. TriAttention and Random Attention tie on Qwen3-4B and Phi-4-reasoning, while TriAttention leads on Qwen3-32B by about three points, the single significant baseline win in the main grid. The prompt is what sets code apart. LiveCodeBench prompts average 557 tokens, which is six times MATH500's under the same tokenizer, and the longest can consume

Table 1: Accuracy under KV cache eviction at each task's ∼4× compression (LiveCodeBench: ∼3×); the header gives each task's per-head KV budget K (§2). **Bold**: best eviction method per column; grayed: significantly below Random Attention (paired clustered bootstrap + sign test, 95%).

	MATH500 K=1024	GPQA-D K=2048	AIME K=4096	HMMT K=4096	LiveCodeBench K=3072
Qwen3-4B					
Full	0.939	0.562	0.642	0.462	0.807
SnapKV	0.703	0.369	0.418	0.395	0.507
R-KV	0.810	0.482	0.494	0.371	0.712
VaSE	0.809	0.461	0.596	0.421	0.700
TriAttention	0.864	**0.533**	0.592	0.437	**0.755**
Random Attention (ours)	**0.874**	0.530	**0.610**	**0.438**	0.744
Phi-4-reasoning					
Full	0.922	0.707	0.677	0.444	0.697
SnapKV	0.844	0.442	0.502	0.343	0.314
R-KV	0.909	0.636	0.643	**0.440**	0.621
VaSE	0.853	0.562	0.520	0.354	0.373
TriAttention	0.891	**0.684**	0.633	0.431	0.652
Random Attention (ours)	**0.910**	0.678	**0.662**	0.430	**0.667**
Qwen3-32B					
Full	0.950	0.703	0.715	0.559	0.886
SnapKV	0.816	0.476	0.541	0.450	0.609
R-KV	0.857	0.638	0.613	0.472	0.779
VaSE	0.868	0.597	**0.680**	**0.524**	0.797
TriAttention	0.887	**0.683**	0.677	0.508	**0.834**
Random Attention (ours)	**0.891**	**0.683**	0.664	0.509	0.806

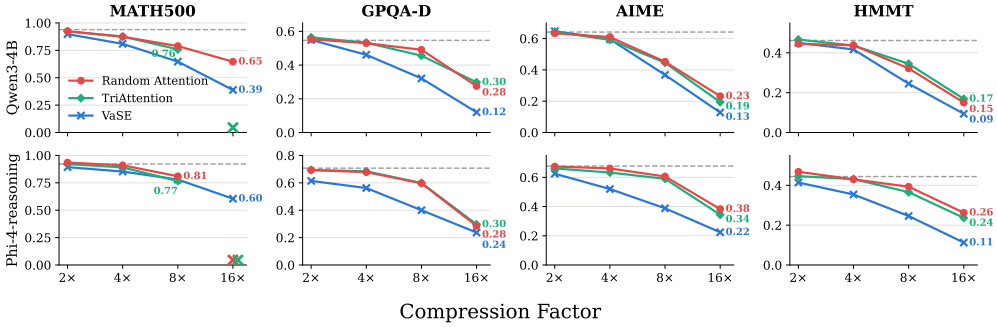


Figure 3: Accuracy from 2× to 16× compression on Qwen3-4B and Phi-4-reasoning, on the four math and science tasks; dashed lines mark full attention.

up to half of the K=3072 budget. Therefore a selector that fails to capture the prompt loses more here than anywhere else; we show in the following section that protecting it closes the SnapKV and VaSE gaps (§5.1). Random Attention itself pins every prompt token, so on code reasoning a large, variable share of its budget is spent before selection begins; much of a code prompt is scaffolding (I/O formats, harness instructions) that a smarter rule might compress rather than pin whole, which we leave to future work since Random Attention's value as a null lies in having nothing to tune.

4.3 Compression Pressure Widens the Gap, in Every Family

Figure 3 presents the performance from 2× to 16× compression on Qwen3-4B and Phi-4-reasoning, on the four math and science tasks, and both families tell the same story: at 2× every method sits near full attention; as the budget tightens, Random Attention stays tied with TriAttention, while the gap from both to VaSE opens. LiveCodeBench is left out of the sweep since its prompts alone would not fit the small budgets.

Table 2: Performance before and after protecting the prompt. R-KV never gains more than 1.9 points, SnapKV gains everywhere, VaSE gains materially only on Phi-4-reasoning.

	Qwen3-4B				Phi-4-reasoning			
	MATH500		GPQA-D		MATH500		GPQA-D	
Method	score alone	+ prompt	score alone	+ prompt	score alone	+ prompt	score alone	+ prompt
SnapKV	0.703	0.829 +12.6	0.369	0.492 +12.3	0.844	0.889 +4.5	0.442	0.667 +22.5
R-KV	0.810	0.812 +0.2	0.482	0.471 -1.1	0.909	0.902 -0.7	0.636	0.655 +1.9
VaSE	0.809	0.812 +0.3	0.461	0.470 +0.9	0.853	0.895 +4.2	0.562	0.664 +10.2
Recency window	0.246	0.843 +59.7	0.093	0.519 +42.6	0.665	0.884 +21.9	0.323	0.658 +33.5
Random Attention	0.459	**0.874** +41.5	0.231	**0.530** +29.9	0.759	**0.910** +15.1	0.434	**0.678** +24.4

5 Why the Selection Signal Buys So Little

The content of the KV cache in reasoning can be divided into two types. The prompt is stated once and never stated again. The working state (i.e., the intermediate reasoning steps a solution builds on) is written and rewritten continually as the model reasons. We show that the first is fragile, and methods differ in how they treat it; the second is redundant enough that a random draw over it keeps what the model still needs.

5.1 The Prompt Is the Fragile Part

Methods disagree about the prompt. TriAttention keeps the whole input by default, while VaSE, R-KV, and SnapKV only keep the sink tokens (Xiao et al., 2024; Han et al., 2024) and leave every slot to the score, so a comparison across papers also compares protection regimes (Chen et al., 2026). Giving every method the same rule separates the score from the protection (Table 2), and one pattern orders the outcome: *the rule pays each method according to how much of the question its score was losing*, measured for every selector by logging, round by round, how much of the prompt it keeps (Appendix D). SnapKV, whose score retains the least of the prompt, always gains, up to 22.5 points on Phi-4-reasoning GPQA-D; VaSE gains only where its retention fails, little on Qwen3-4B but $+4.2$ and $+10.2$ on Phi-4-reasoning, whose prompts are two to three times longer; R-KV, which retains the most, never gains more than 1.9 points (survival fractions in Appendix D). Once every method keeps the prompt, the three baselines land within 2.2 points of one another in every setting. On Phi-4-reasoning they also land within about two points of Random Attention, which ranks nothing; on Qwen3-4B a residual of 4–6 points below Random Attention remains for all three, and R-KV and VaSE, which already kept most of the prompt there, gain almost nothing from the rule. Code reasoning on both models, competition math on Qwen3-4B and GPQA-D at 32B (Appendix C) show the same pattern: the rule closes every gap that was large and method-specific, and what survives it is smaller and runs in Random Attention's favour. Most of the difference between the baselines was therefore the prompt. Whatever their scores add beyond it is small, and where a residual remains it is a deficit: with the prompt protected, each of the three still trails a policy that ranks nothing.

The two signal-free rows make the same point from the other side. Without the rule, a recency window allocates all the budget to the recent trace and none to the prompt and scores as low as 0.09, and Random Attention, which keeps the prompt only at the uniform rate, falls to 0.23–0.76. With the rule, the same two policies lose nothing that matters: Random Attention is the best policy in every setting and a plain recency window is never more than two points below the best protected baseline. Losing the prompt is catastrophic and cutting the trace at random is not, which is what makes the prompt the fragile part of the cache. The same confound explains the large gaps others report between random retention and signal-based selection in previous works (Yuan et al., 2026; Liu et al., 2025): their random baselines perform poorly since the prompt is lost.

5.2 The Working State Protects Itself

The rest of the cache is the model's own reasoning trace (working state), which is stored redundantly at two levels. The first redundancy is in the text and is already shown by Cai et al. (2025): reasoning traces restate what they are still using, so a value that matters rarely lives at one position only. The

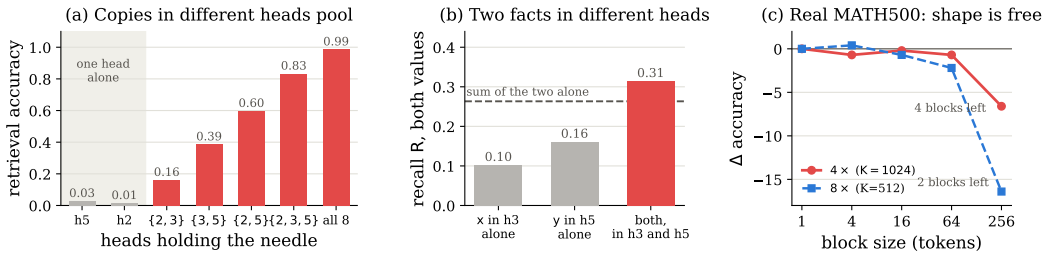


Figure 4: **(a)** A fact held in one head is almost never retrieved; held in several it is. **(b)** Two facts in *different* heads are worth more together than the sum of each alone (dashed). **(c)** Real MATH500: contiguous blocks cost nothing up to size 64; accuracy drops only once a head is left with 4 (K=1024) or 2 (K=512) blocks.

second is across heads: each of the KV heads caches its own copy of every token, and eviction decides per head which copies die. A token is only lost when all KV heads happen to drop it.

We show how the model makes use of the second, cross-head redundancy with a planted-fact probing experiment. A synthetic fact (e.g. `Let zq = 4729`; a fresh variable and value each time) is inserted into real model-generated MATH500 reasoning traces, with a question needing the value appended at the end. The fact is planted 1,536 tokens before the question, so the cache is evicted 15 times between the two (distance grid in Appendix B); the question itself is always kept. What we control is which key-value heads keep the fact: a *condition* is a chosen set of heads in which the fact's tokens are pinned, with the fact evicted from every other head and the standard per-head uniform eviction running on everything else. Two metrics measure what survives: *retrieval*, the fraction of traces whose greedy decode reproduces the value, and, where retrieval floors, a graded *recall* R, the share of the log-probability gap between deleting the fact from every cache (R=0) and keeping it in every cache (R=1) that the surviving copies recover (Appendix B). Every condition is scored on the same 250–500 planted traces.

Copies pool across heads. Attention heads specialise: only three of Qwen3-4B's eight key-value heads retain a usable trace of the fact on their own, consistent with the retrieval-head specialisation of Wu et al. (2025), and even those three are weak alone: the best single head yields the value in 3% of trials, the next in 1%. But the readout does not depend on any one head: the same two heads together yield the fact in 60% of trials, three heads in 83%, and all eight in 99% (Figure 4a). Pooling is thus strongly superadditive, a pair being worth many times the sum of its singles, and it even crosses facts: two values held in *different* heads, both needed by the answer, give R=0.31 together against 0.10 and 0.16 alone (Figure 4b). For an eviction policy the consequence is direct: a value stays usable as long as *some* heads keep a copy, which is exactly what independent per-head draws maximise. On real MATH500 traces this extra coverage is not even needed: a shared draw, the same random positions in every head, scores within 0.3 points of Random Attention at 4× and 8× (Appendix D), because the text-level redundancy already keeps a restated copy; the cross-head level carries what the text does not restate, which is the probe's regime.

The shape of the copies does not matter. We deal the fact out token by token across heads, so that no two consecutive tokens share a head and no head holds a readable span; retrieval barely moves (0.33, against 0.39 for an intact sentence at the same retained mass) and recall is essentially unchanged (R=0.75 vs. 0.76). The same insensitivity appears on real MATH500 traces at full scale: keeping the history in contiguous blocks rather than scattered tokens costs nothing as blocks grow from 1 to 64 tokens; accuracy drops only at block size 256, where the budget leaves a head just four or two blocks, and drops more with two, so what matters is blocks per head, not block length (Figure 4c). Together the two findings say the answer depends on whether some usable copy of a needed value survives *somewhere*, not on which copy, in which head, or in what shape.

5.3 What Is Left for a Selection Signal

Table 4: Serving throughput (output tok/s, and the multiple of full attention) under vLLM with PagedAttention on one H200 ($K{=}2048$, 1k-token prompts, 32k-token generations).

Method	Qwen3-4B	Phi-4-reasoning	Qwen3-14B	Qwen3-32B
Full	1296 (1.00×)	780 (1.00×)	925 (1.00×)	346 (1.00×)
TriAttention	1494 (1.15×)	1212 (1.55×)	1303 (1.41×)	700 (2.02×)
Random Attention (ours)	**2046** (1.58×)	**1737** (2.23×)	**1819** (1.97×)	**923** (2.67×)
ours over TriAttention	+37%	+43%	+40%	+32%

The one case a signal-free policy cannot cover is a fact stated once, never restated, and needed much later. We announce a passcode once, 57 compression rounds before the question, let each policy retain as it sees fit, and report two numbers (Table 3): the fraction of traces in which the model reproduces the passcode (Retr.), and the log-probability it assigns to the correct passcode at the question, averaged over traces ($\log p$). A $\log p$ of 0 means the model would produce the passcode with certainty; a number like -18 means the passcode is effectively gone from the cache. Random Attention never reproduces it. R-KV finds it 84% of the time, plausibly through its redundancy term, which favours keys unlike the rest of the cache, as a once-stated passcode is; VaSE a third of the time; SnapKV and TriAttention almost never. Wang (2026) prove that random caches must lose at pointer-chasing when nothing is redundant. Needle-finding is thus real selection skill, but it neither implies nor follows from aggregate strength: R-KV, the best needle-finder, leads only one column of Table 1, while TriAttention, the strongest baseline there, recovers almost nothing here. On real traces the case is rare, because the model keeps restating what it is still using.

Table 3: A passcode stated once, 57 compression rounds before the question.

Policy	Retr.	$\log p$
Random Attention (ours)	0.000	−18.35
VaSE	0.344	−3.88
SnapKV	0.004	−11.11
R-KV	0.836	−0.71
TriAttention	0.016	−11.11

6 Efficiency Evaluation

We measure efficiency in two settings: vLLM (Kwon et al., 2023) with PagedAttention, following TriAttention's protocol, and batched decoding in HuggingFace Transformers without paging (§4.1), at the largest batch that fits on one GPU. The first compares Random Attention with TriAttention only: among the baselines only TriAttention and R-KV have vLLM ports, and Mao et al. (2026) already show that TriAttention is more efficient than R-KV at matched budget. The second compares all methods. Details of the two protocols are given in Appendix G.

Under paged serving, Random Attention is 32–43% faster than TriAttention. Both methods run on one H200 at $K{=}2048$ with 1k-token prompts, 32k-token generations and 128 requests (at most 96 concurrent on Qwen3-32B, whose weights leave a smaller KV pool; Appendix G). At 32k tokens, full attention's cache fills the KV pool and caps how many requests run at once; compression raises that cap, so both methods beat full attention (Random Attention by 1.6–2.7× as shown in Table 4). With equal cache sizes and the same kernels, Random Attention's 32–43% margin over TriAttention comes from the scoring pass it skips (below); it also holds at capacity (+41%, +42%), at short generations and at half the load (Appendix G).

At equal memory every evictor gains, and the scoring pass decides the rest. At the largest batch each method fits on one H200 ($K{=}3072$, 32k generations; Figure 6, Appendix G), the compressed caches hold 109–200 sequences against 20–28 for full attention, the source of every evictor's 3–10× speedup. Random Attention, which computes no score, fits the largest batch and reaches 10.0× and 8.8× on Qwen3-4B and 14B (28.8× on Qwen3-4B at a smaller K of 1024). The TriAttention row there is an unfused port, so the margin we claim is Table 4's.

Why skipping the scoring pass is worth 32–43% in serving. Timed alone, the pass is cheap. One per-layer eviction costs 0.30 ms under Random Attention, which only compacts the cache, and 1.47–1.64 ms under TriAttention, which scores it first; in a single stream the extra scoring amounts to a few percent of decoding time. Serving multiplies that small cost in two ways. First, the compressions pile up: with 128 concurrent requests, each compressed every 64 of its own tokens, vLLM compresses about two requests per decoding step (61.6k compressions over the 32k run), and each compression happens at a synchronisation point between batched steps, so all 128 requests wait

while one is compressed. Second, content-dependent scoring is more expensive under paged serving than in plain batched decoding. Cache-statistic selectors need an extra pass over the paged KV state, and attention-weight selectors must recompute or explicitly expose attention statistics, because the fused kernels do not materialise them; Random Attention needs neither and only compacts. On Qwen3-14B, TriAttention's 32k run takes 910 s longer than Random Attention's, about 15 ms of whole-batch waiting per compression, against well under a millisecond for Random Attention.

7 Related Work

KV cache for long-context understanding. A major line of work on the KV cache targets long inputs and short outputs: one or many documents fill the cache at prefill and a short answer follows. Three families reduce its memory. Quantization keeps every token at lower precision (Liu et al., 2024; Hooper et al., 2024). Eviction discards tokens under a budget (Xu et al., 2025; Park et al., 2025; Xiao et al., 2025; Li et al., 2026; An et al., 2026), scored by cumulative attention (Zhang et al., 2023), its persistence across steps (Liu et al., 2023), or attention from a window of recent queries (Li et al., 2024), with budgets adapted per head or per layer (Feng et al., 2025; Cai et al., 2024), structural rules that keep the attention sinks and a recent window (Xiao et al., 2024), or a policy chosen per head (Ge et al., 2024). Query-aware selection keeps every token and attends to a subset per query (Tang et al., 2024), which saves compute but not memory. Randomness has also been studied in KV cache management. At the serving-system level, Wu et al. (2026) randomly evict unmarked leaf tokens from a prefix-sharing cache, making eviction more robust to dynamic or adversarial query arrivals. For long-context compression, Roy et al. (2026) sample for coverage of the input, and Garcia (2026) show, for long-context question answering under a global cache cap, that once the prompt-boundary tokens are guarded the choice of score is second-order and a random policy shares in the recovery.

KV cache for long reasoning. Chains of thought invert the ratio: the prompt is a few hundred tokens and the cache is filled during decoding. Sparse-attention selection has been adapted to this regime (Gao et al., 2026; Yang et al., 2026), but it keeps the full cache in memory (§2); eviction is the only route that bounds it. Decode-time evictors generally estimate KV importance and use these scores to determine retention: R-KV uses a SnapKV-style score with a redundancy penalty so that restated content is not kept twice (Cai et al., 2025), VaSE scores by value magnitude and fills the remaining budget stochastically (Chang et al., 2026), following earlier score-guided randomized eviction (Chen et al., 2024), and TriAttention predicts each key's future attention from a calibrated query centre (Mao et al., 2026); LazyEviction estimates future importance from recurring attention patterns (Zhang et al., 2026). Learned evictors instead train a scorer for each base model, typically on domain-specific data: TrimKV (Bui et al., 2026b) and DBTrimKV (Bui et al., 2026a) learn per-token retention scores that decay with age, and ForesightKV (Dong et al., 2026) distills a future-attention oracle refined with GRPO. We compare only against training-free evictors and discuss how trained ones relate to our findings in Appendix H. Concurrently with this work, Prefix Sliding (Muennighoff et al., 2026) keeps only the prompt and a window of recent tokens, the recency+prompt policy of §5.1, and also explores incorporating KV cache eviction into training, as Kontonis et al. (2026) do. Evaluations of KV compression on reasoning report that eviction hurts chains of thought more than long-context tasks and that random or recency baselines fall far behind scored selection (Liu et al., 2025; Yuan et al., 2026), and Chen et al. (2026) show that protocols differing in what they protect make results across papers hard to compare. We run baselines at matched budget and show that when the prompt is kept, even evicting at random can be comparable to the strongest baseline while being much more efficient.

8 Conclusion

KV cache eviction for reasoning has been treated as a ranking problem: estimate which cached tokens will matter and keep those. We find that the ranking contributes almost nothing. A policy that keeps the prompt and evicts uniformly at random within each head matches the strongest baselines across four models and six tasks, and serves 32–43% higher throughput in vLLM deployment. We explain this by showing that 1) the prompt is the fragile part of the cache, and once every method keeps it, most of the gap between methods disappears; 2) the reasoning trace protects itself through redundancy, in the text and across heads, so once the prompt is safe a random draw keeps enough copies of what the model still needs. What remains for a selection signal is the rare fact stated once and never restated, which reasoning seldom produces.

AI USE STATEMENT

In this work, we used generative AI tools (LLM-based coding and writing assistants) to assist with drafting and editing the manuscript, implementing analysis and plotting code, and typesetting tables. We have not used generative AI tools to design the experiments, to select or register the hypotheses and thresholds, or to produce any reported measurement. All AI-assisted code and text were reviewed by the authors; reported statistics were recomputed from the underlying per-instance logs. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REFERENCES

Marah Abdin, Sahaj Agarwal, Ahmed Awadallah, Vidhisha Balachandran, Harkirat Behl, Lingjiao Chen, Gustavo de Rosa, Suriya Gunasekar, Mojan Javaheripi, Neel Joshi, Piero Kauffmann, Yash Lara, Caio César Teodoro Mendes, Arindam Mitra, Besmira Nushi, Dimitris Papailiopoulos, Olli Saarikivi, Shital Shah, Vaishnavi Shrivastava, Vibhav Vineet, Yue Wu, Safoora Yousefi, and Guoqing Zheng. Phi-4-reasoning technical report. *arXiv preprint arXiv:2504.21318*, 2025.

Yongqi An, Chang Lu, Kuan Zhu, Tao Yu, Chaoyang Zhao, Hong Wu, Ming Tang, and Jinqiao Wang. ReST-KV: Robust KV cache eviction with layer-wise output reconstruction and spatial-temporal smoothing. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=PhEHuo7oMm.

Mislav Balunovic, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. Matharena: Evaluating LLMs on uncontaminated math competitions. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2026. URL https://openreview.net/forum?id=y0zL9IZxZ7.

Ngoc Bui, Hieu Trung Nguyen, Arman Cohan, and Rex Ying. Make each token count: Towards improving long-context performance with kv cache eviction. *arXiv preprint arXiv:2605.09649*, 2026a.

Ngoc Bui, Shubham Sharma, Simran Lamba, Saumitra Mishra, and Rex Ying. Cache what lasts: Token retention for memory-bounded KV cache in LLMs. In *The Fourteenth International Conference on Learning Representations*, 2026b. URL https://openreview.net/forum?id=qCaq3jGb0S.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Baobao Chang, Junjie Hu, and Wen Xiao. PyramidKV: Dynamic KV cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*, 2024.

Zefan Cai, Wen Xiao, Hanshi Sun, Cheng Luo, Yikai Zhang, Ke Wan, Yucheng Li, Yeyang Zhou, Li-Wen Chang, Jiuxiang Gu, Zhen Dong, Anima Anandkumar, Abedelkadir Asi, and Junjie Hu. R-KV: Redundancy-aware KV cache compression for reasoning models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=2jwAjomEDB.

Ting-Yun Chang, Harvey Yiyun Fu, Deqing Fu, Chenghao Yang, Jesse Thomason, and Robin Jia. Value-aware stochastic KV cache eviction for reasoning models. In *COLM 2026 Workshop on Efficient Reasoning*, 2026. URL https://openreview.net/forum?id=HysrOiWFIA.

Alex Chen, Renato Geh, Aditya Grover, Guy Van Den Broeck, and Daniel Mingyi Israel. The pitfalls of KV cache compression. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 41530–41553, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.1926. URL https://aclanthology.org/2026.acl-long.1926/.

Yilong Chen, Guoxia Wang, Junyuan Shang, Shiyao Cui, Zhenyu Zhang, Tingwen Liu, Shuohuan Wang, Yu Sun, Dianhai Yu, and Hua Wu. NACL: A general and effective KV cache eviction framework for LLM at inference time. In Lun-Wei Ku, Andre Martins, and Vivek

Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7913–7926, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.428. URL https://aclanthology.org/2024.acl-long.428/.

Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=mZn2Xyh9Ec.

DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Zican Dong, Peiyu Liu, Junyi Li, Zhipeng Chen, Han Peng, Shuo Wang, and Xin Zhao. ForesightKV: Optimizing KV cache eviction for reasoning models by learning long-term contribution. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=znV8JHv8b8.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S Kevin Zhou. Ada-KV: Optimizing KV cache eviction by adaptive budget allocation for efficient LLM inference. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=tcisuhGsQZ.

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. Sparse attention adaptation for long reasoning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=c5BOcHM6J8.

Gabriel Garcia. Protection is (nearly) all you need: Structural protection dominates scoring in globally capped KV eviction. *arXiv preprint arXiv:2605.18053*, 2026.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. Model tells you what to discard: Adaptive KV cache compression for LLMs. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=uNrFpDPMyo.

Chi Han, Qifan Wang, Hao Peng, Wenhan Xiong, Yu Chen, Heng Ji, and Sinong Wang. LM-infinite: Zero-shot extreme length generalization for large language models. In Kevin Duh, Helena Gomez, and Steven Bethard (eds.), *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 3991–4008, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-long.222. URL https://aclanthology.org/2024.naacl-long.222/.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. URL https://openreview.net/forum?id=7Bywt2mQsCe.

Coleman Richard Charles Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Sophia Shao, Kurt Keutzer, and Amir Gholami. KVQuant: Towards 10 million context length LLM inference with KV cache quantization. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=0LXotew9Du.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=chfJJYC3iL.

Vasilis Kontonis, Yuchen Zeng, Shivam Garg, Lingjiao Chen, Hao Tang, Ziyan Wang, Ahmed Hassan Awadallah, Eric Horvitz, John Langford, and Dimitris Papailiopoulos. MEMENTO: Teaching LLMs to manage their context. In *Third Conference on Language Modeling*, 2026. URL https://openreview.net/forum?id=YaYiQDVsi0.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

Mengjie Li, Yuan Feng, Xike Xie, and William J. Song. REAL: REtrieval-reAsoning and logic-constructed attention behaviors for long-context KV cache compression. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 39035–39052, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.1811. URL https://aclanthology.org/2026.acl-long.1811/.

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. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=poE54GOq2l.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

Minghui Liu, Aadi Palnitkar, Tahseen Rabbani, Hyunwoo Jae, Kyle Rui Sang, Dixi Yao, Shayan Shabihi, Fuheng Zhao, Tian Li, Ce Zhang, Furong Huang, and Kunpeng Zhang. Hold onto that thought: Assessing KV cache compression on reasoning. *arXiv preprint arXiv:2512.12008*, 2025.

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. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=JZfg6wGi6g.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=L057s2Rq8O.

Weian Mao, Xi Lin, Wei Huang, Yuxin Xie, Tianfu Fu, Bohan Zhuang, Song Han, and Yukang Chen. Triattention: Efficient long reasoning with trigonometric KV compression. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=0tgzJK50Jz.

Niklas Muennighoff, Zhengyang Wang, Zeyi Chen, Weijia Shi, Binyuan Hui, John Yang, Dapeng Jiang, Mika Senghaas, Fares Obeid, Johannes Hagemann, Sami Jaghouar, Ludwig Schmidt, Percy Liang, Jason Wei, Andrew Y. Ng, Luke Zettlemoyer, Yejin Choi, and Mike Lewis. Prefix sliding for efficient test-time scaling. *arXiv preprint arXiv:2608.26070*, 2026.

OpenAI. Learning to reason with LLMs. https://openai.com/index/learning-to-reason-with-llms, 2024.

Junyoung Park, Dalton Jones, Matthew J Morse, Raghavv Goel, Mingu Lee, and Christopher Lott. Keydiff: Key similarity-based KV cache eviction for long-context LLM inference in resource-constrained environments. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=uBaFH7aQnC.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=Ti67584b98.

Shuvendu Roy, Mengyao Zhai, Hossein Hajimirsadeghi, and Golnoosh Samei. Coverage-driven KV cache eviction for efficient and improved inference of LLM. *arXiv preprint arXiv:2606.29563*, 2026.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. QUEST: Query-aware sparsity for efficient long-context LLM inference. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=KzACYw0MTV.

Xiao Wang. How much cache does reasoning need? depth-cache tradeoffs in KV-compressed transformers. *arXiv preprint arXiv:2604.17935*, 2026.

Fangzhou Wu, Sandeep Silwal, and Qiuyi Zhang. Randomization boosts KV caching, learning balances query load: A joint perspective. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=R7fv5NWfMm.

Wenhao Wu, Yizhong Wang, Guangxuan Xiao, Hao Peng, and Yao Fu. Retrieval head mechanistically explains long-context factuality. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=EytBpUGB1Z.

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

Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, junxian guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. Duoattention: Efficient long-context LLM inference with retrieval and streaming heads. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=cFu7ze7xUm.

Fangyuan Xu, Tanya Goyal, and Eunsol Choi. RefreshKV: Updating small KV cache during long-form generation. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 24878–24893, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1211. URL https://aclanthology.org/2025.acl-long.1211/.

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

Lijie Yang, Zhihao Zhang, Arti Jain, Shijie Cao, Baihong Yuan, Yiwei Chen, Zhihao Jia, and Ravi Netravali. Less is more: Fast and accurate reasoning with cross-head unified sparse attention. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=trSWJ99WzS.

Aojie Yuan, Tianqi Shen, and Dajun Zhang. Not all thoughts need HBM: Semantics-aware memory hierarchy for LLM reasoning. *arXiv preprint arXiv:2605.09490*, 2026.

Haoyue Zhang, Hualei Zhang, Xiaosong Ma, Jie Zhang, and Song Guo. LazyEviction: Lagged KV eviction with attention pattern observation for efficient long reasoning. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 36335–36352, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.1683. URL https://aclanthology.org/2026.acl-long.1683/.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Re, Clark Barrett, Zhangyang Wang, and Beidi Chen. H2o: Heavy-hitter oracle for efficient generative inference of large language models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=RkRrPp7GKO.

Table 5: Performance of Qwen3-14B with the same setting as in Table 1.

	MATH500 K=1024	GPQA-D K=2048	AIME K=4096	HMMT K=4096	LiveCodeBench K=3072
Qwen3-14B					
Full	0.951	0.645	0.706	0.557	0.856
SnapKV	0.812	0.506	0.460	0.467	0.622
R-KV	0.816	0.600	0.571	0.427	0.788
VaSE	0.852	0.543	**0.668**	0.495	0.813
TriAttention	**0.891**	0.625	0.654	0.493	**0.843**
Random Attention (ours)	0.870	**0.628**	0.642	**0.505**	0.820

A GENERALITY: ADDITIONAL MODELS

Table 5 repeats the main grid on Qwen3-14B, an intermediate scale within the headline family. The picture from Table 1 replicates: Random Attention beats VaSE and SnapKV on MATH500 and GPQA-D (both significant). Three baseline cells are significantly ahead at this scale: TriAttention on LiveCodeBench ($+2.3$ points, p=.007), matching its code win on Qwen3-32B and the prompt-length account of §4.2, TriAttention on MATH500 ($+2.1$ points, p=.02), and VaSE on AIME ($+2.6$ points, p=.007), consistent with its nominal AIME edge on Qwen3-32B. Every cell here uses the same rollout count as the corresponding cell of Table 1 (R=2 on MATH500, 4 on GPQA-D and LiveCodeBench, 16 on HMMT and AIME) for every method.

B THE PLANTED-FACT PROBE: INSTRUMENT, METRIC, AND CAVEATS

Instrument. Each instance is a short prefilled stub (chat template plus think-opener, 23 tokens) followed by a fully scripted body fed through the decode path token by token: real model-generated MATH500 reasoning as filler (screened so it never contains the planted variable or value), the fact inside a fixed 16-token box, a second fact 256 tokens later (inert unless the cell uses it), and a terminal query whose value tokens are the measurement points. Values are 4-digit numbers, distinct across instances, and exactly four tokens under Qwen3's digit tokenizer. Because every scripted token enters through the true decode path, eviction treats it exactly as generated reasoning, and the model cannot restate the fact on its own. Pinning gives the fact's span a $+\infty$ score in the designated (layer, head) sites and $-\infty$ everywhere else (evicted at first eligibility); background eviction is the standard per-head uniform draw with no protection rule anywhere, and a per-event audit log records the surviving set. Distances are eviction-event counts $E \in \{3, 15, 39, 57\}$ with per-copy survival $(1024/1088)^E$. Four checks validate the instrument: (i) token-by-token forcing matches one-shot prefill within kernel tolerance on every probe tested; (ii) the audit confirms the pinned span survives in exactly the specified sites at every eviction event; (iii) pinning is non-perturbative: keep-everywhere under eviction scores at least as high as no-eviction ($+0.22$ nats); and (iv) the endpoints are far apart (mean value logprob -0.19 nats with the fact kept everywhere vs. -13.59 deleted everywhere; exact match 0.986 vs. 0.000).

Recall metric. The graded recall of §5.2 is

$$R \;=\; \frac{\sum_i \big(\mathrm{LP}_i - \mathrm{LP}_i^{\mathrm{del}}\big)}{\sum_i \big(\mathrm{LP}_i^{\mathrm{kept}} - \mathrm{LP}_i^{\mathrm{del}}\big)}, \tag{2}$$

where LP_i is the log-probability the model assigns to trace i's correct value under the condition being tested, and $\mathrm{LP}_i^{\mathrm{kept}}$ and $\mathrm{LP}_i^{\mathrm{del}}$ are the same quantity with the fact kept in every cache and deleted from every cache, both measured per needle. R=1 means the surviving copies are worth as much as never evicting the fact, and R=0 that they are worth nothing. Retrieval leads wherever it separates conditions; R carries the comparison where retrieval floors, as it does on the two-needle probe.

Table 6: Matched protection in the settings Table 2 does not cover (LiveCodeBench: pass@1; others: accuracy; AIME pools 2025+2026). Small numbers give the gain from the rule in points, red when at least 2; bold marks the best protected method in each setting. TriAttention and Random Attention keep the prompt by construction and appear only in the protected column (values as in Table 1).

	Qwen3-4B						Phi-4-reasoning		Qwen3-32B	
	LiveCodeBench		AIME		HMMT		LiveCodeBench		GPQA-D	
Method	score alone	+ prompt	score alone	+ prompt	score alone	+ prompt	score alone	+ prompt	score alone	+ prompt
SnapKV	0.507	0.728 +22.1	0.418	0.556 +13.9	0.395	0.408 +1.3	0.314	0.666 +35.2	0.476	0.641 +16.5
R-KV	0.712	0.698 -1.4	0.494	0.499 +0.5	0.371	0.372 +0.1	0.621	0.630 +0.9	0.638	0.639 +0.1
VaSE	0.700	0.732 +3.2	0.596	0.575 -2.1	0.421	0.411 -1.0	0.373	0.646 +27.3	0.597	0.626 +2.9
TriAttention	–	**0.755**	–	0.592	–	0.437	–	0.652	–	**0.683**
Random Attention	–	0.744	–	**0.610**	–	**0.438**	–	**0.667**	–	**0.683**

Statistics. Confidence intervals are instance-clustered bootstrap percentile intervals with 4,000 replicates; every condition is scored on the same instances, so contrasts are paired. Re-running under fresh eviction randomness moves the reported values by one to two points.

Caveats. Exact retrieval has no dynamic range once a fact survives in a single head (it floors near 0.03), which is why single-copy and two-fact contrasts are reported on R. The probe runs on Qwen3-4B; which heads act as carriers is a property of the model, and no claim in the paper depends on their identity. Replicating the probe on other models is left to future work.

C Matched protection in the remaining settings

Table 2 gives every method the prompt-protection rule on Qwen3-4B and Phi-4-reasoning for MATH500 and GPQA-D. Table 6 extends the same control to the settings where Table 1 shows the largest remaining deficits: code on both models, competition math on Qwen3-4B, and GPQA-D at 32B. Two facts hold across all fifteen protected model–task settings.

The payoff follows the retention deficit. Every gain from the rule is ordered by how much of the prompt the score was losing (Appendix D). SnapKV, which retains the least, gains $+35.2$, $+22.1$, $+16.5$, $+13.9$ and $+1.3$ points; VaSE gains $+27.3$, $+3.2$ and $+2.9$ where its retention falls short, and moves by -2.1 and -1.0 on competition math, where it does not; R-KV, which already keeps most of the prompt, moves by -1.4 to $+0.9$ points, essentially zero, in all five settings. The smallest SnapKV gain is on HMMT, where its unprotected deficit is itself small at 32k-token generations, so there is little to close.

Protection closes most of the gap, not all of it. Where the deficit was prompt-driven the rule closes it completely: SnapKV and VaSE become statistical ties with Random Attention on code for both models, and SnapKV and VaSE on HMMT. Elsewhere a residual survives protection (paired tests as in §4.1): all three protected baselines remain 4–6 points below Random Attention on GPQA-D at 32B, where TriAttention ties it exactly; SnapKV, R-KV and VaSE remain 5.4, 11.1 and 3.5 points below on AIME, and R-KV 6.6 below on HMMT; and R-KV remains 4.6 and 3.7 points below on code for both models even though it never lost the prompt, a negative selection effect that mirrors TriAttention's positive one on Qwen3-32B code. Since TriAttention and Random Attention both keep the prompt by construction, every TriAttention-versus-Random Attention cell of Table 1 is already a matched-protection comparison; the confound applies only to the other baselines' deficits. The strongest reading of the 32B GPQA-D setting is the simplest: a uniform draw plus the rule beats all three protected scores there. We have not run the protected grid on Qwen3-14B.

D Keep-log measurements

The numbers in §5.1–§5.3 that describe what a policy retains come from logging every eviction round of real runs: nineteen policy–cell logs, 16 traces each, of order 10^4–10^5 rounds per log, recording for each round the keep-set of every (layer, key-value head) pair together with the age of each retained position. Three quantities are used. *Slot coverage* is the fraction of candidate positions

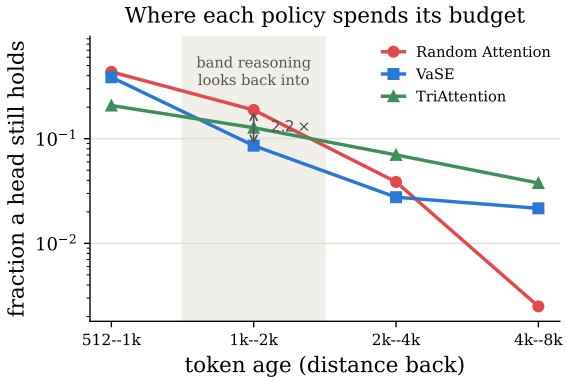


Figure 5: Fraction of positions of a given age that a head still holds (log scale; Qwen3-4B MATH500, $K{=}1024$). Random Attention decays geometrically with age; VaSE concentrates and freezes a tail of old favourites; TriAttention spends almost uniformly across ages, keeping less of the recent middle than Random Attention but several times more of the very old tail.

retained by at least one head immediately after a round; it is 0.999–1.000 for Random Attention and VaSE, 0.993 for TriAttention, and 0.938 for the shared-draw control, which by construction has no cross-head diversity. *Prompt survival* is the fraction of prompt positions still held, by any head and by a given head: 0.994–0.999 and the same per head for Random Attention, against 0.55–0.91 (union) and 0.26–0.67 (per head) for R-KV, 0.56–0.70 and 0.20–0.29 for VaSE, and 0.32–0.42 and 0.11–0.22 for SnapKV, the lowest on every one of the four model–task settings. *Survival by age* is the fraction of positions in an age band a head still holds (Figure 5); at 1–2k it is 0.188 for Random Attention, 0.161 for the shared draw, 0.145 for SnapKV, 0.105 for R-KV and 0.086 for VaSE, and the cross-head union in that band is 0.776 for Random Attention against 0.368 for VaSE and 0.161 for the shared draw, whose union cannot exceed its per-head survival. TriAttention holds 0.127 per head in that band with a cross-head union of only 0.199: it pins the prompt in full, and its heads keep nearly the same positions, the least cross-head diversity of any per-head policy measured here.

Implicit age bias. Although no score is computed, the policy is not age-blind. A position that survives one eviction faces a fresh draw at the next, so its probability of still being cached after n evictions is $\left(\frac{K-\ell_{\mathrm{p}}}{K+r-\ell_{\mathrm{p}}}\right)^n$, about 0.94^n at $K{=}1024$ and $r{=}64$. Random Attention is therefore a soft recency window: recent positions are almost always present, and old ones survive as a thin tail that differs from head to head (Figure 5). §5.1 separates the two ingredients, a hard recency window with the prompt kept and the tail that the scatter adds.

Accuracy of the shared draw. The most direct task-level test of cross-head diversity is to remove it: the shared-draw control keeps the prompt and draws one random keep-set that every head uses, so its cross-head union equals its per-head survival (0.161 in the 1–2k band against 0.776 for Random Attention). On Qwen3-4B MATH500 it scores 0.871 at $K{=}1024$ and 0.788 at $K{=}512$, against 0.874 and 0.789 for Random Attention. Removing cross-head diversity therefore costs nothing on real traces at these budgets: the text-level redundancy already keeps a restated copy of what the model still needs, so the second level is not load-bearing there. It is load-bearing where the first level is absent (a fact stated once and never restated), which is the regime of the planted-fact probe (§5.2) and of the boundary in §5.3. The two redundancies are substitutes, and Random Attention preserves both.

E GENERATION LENGTHS AND RUN-TO-RUN VARIABILITY

Table 7 reports the mean number of generated tokens for every cell of Tables 1 and 5, and Table 8 the standard deviation of per-run accuracy across each cell's independently sampled runs. Two facts are worth noting. First, the budgets of §4.1 track the full-attention lengths: MATH500 traces are the shortest and competition-math traces the longest on every model, and eviction generally

Table 7: Mean generated tokens (thousands) per cell of Tables 1 and 5, measured over every run of the cell; Avg = unweighted mean over the five tasks.

	MATH500	GPQA-D	AIME	HMMT	LiveCodeBench	Avg
Qwen3-4B						
Full	5.3	8.6	17.0	18.4	11.1	12.1
SnapKV	14.2	12.0	20.1	22.7	15.8	17.0
R-KV	9.1	12.6	21.5	23.3	14.6	16.2
VaSE	8.3	11.6	18.9	21.3	14.3	14.9
TriAttention	8.0	11.3	19.4	21.2	13.6	14.7
Random Attention (ours)	7.1	9.9	18.6	20.0	14.1	13.9
Phi-4-reasoning						
Full	2.7	7.5	13.0	16.5	9.9	9.9
SnapKV	2.9	8.6	13.3	17.0	10.5	10.4
R-KV	3.1	10.7	13.9	18.1	11.3	11.4
VaSE	2.8	9.0	13.4	17.0	10.7	10.6
TriAttention	3.3	9.4	13.9	17.3	10.5	10.9
Random Attention (ours)	3.4	9.4	13.4	17.3	10.5	10.8
Qwen3-32B						
Full	4.6	7.0	14.8	17.4	10.3	10.8
SnapKV	7.5	10.1	17.2	20.4	15.0	14.0
R-KV	7.9	10.3	18.6	22.1	14.5	14.7
VaSE	7.0	9.7	16.8	20.0	15.3	13.7
TriAttention	7.2	8.3	17.0	20.0	12.8	13.0
Random Attention (ours)	6.1	8.0	16.7	19.2	18.4	13.7
Qwen3-14B						
Full	4.8	7.6	15.7	17.9	9.8	11.2
SnapKV	6.9	10.4	18.3	20.1	14.2	14.0
R-KV	8.7	11.1	19.8	22.7	13.3	15.1
VaSE	8.0	11.1	17.7	21.0	11.7	13.9
TriAttention	7.1	9.4	17.8	20.5	11.6	13.3
Random Attention (ours)	6.9	8.8	17.6	20.1	12.3	13.1

lengthens generation relative to full attention, most for the weakest selectors. Averaged over the five tasks, Random Attention is the shortest-generating evictor on Qwen3-4B and Qwen3-14B and within about 5% of the shortest on the other two models, so its accuracy parity is not bought with longer generations. Second, the variability matches the significance treatment in the main text: run-to-run standard deviation is at or under about one point on MATH500, one to three points on GPQA-D, up to two on LiveCodeBench, and two to five points on competition math, which is why every claim is decided by the paired, problem-clustered tests of §4.1 rather than by raw cell differences. MATH500 cells have R=2, so their standard deviation is a two-sample estimate.

F ENGINE DETAILS

We implement per-KV-head physical eviction on the VaSE engine: every r=64 decode steps, once the cache exceeds $K+r$, each KV head scores its candidate slots (all cached positions except the r-token recent buffer), keeps top-K, and the KV tensors are compacted with a gather. Keys are stored post-RoPE with a cumulative-length counter; keep-sets are sorted chronologically before compaction for every method; grouped-query models evict per KV head (query groups share their head's keep-set). Eviction is monotonic. Sampling and prompting follow the reference repository defaults for every method; the model decodes until end-of-sequence, up to a uniform 32k-token limit.

Baseline configurations. Every baseline runs in this engine at the same budget, trigger and buffer. VaSE uses $n_{\text{large}} = K/4$, the setting in its released run scripts (a fixed n_{large}=256 at larger budgets

Table 8: Run-to-run variability: standard deviation of per-run accuracy (points) across each cell's independent sampled runs, including LiveCodeBench, whose runs are graded individually by test execution.

	MATH500	GPQA-D	AIME	HMMT	LiveCodeBench
Qwen3-4B					
Full	0.1	1.1	4.3	4.8	0.5
SnapKV	1.3	2.9	2.9	3.5	2.0
R-KV	0.0	2.0	1.9	3.2	1.4
VaSE	0.1	1.9	2.5	3.3	1.8
TriAttention	0.2	2.5	5.2	3.9	0.8
Random Attention (ours)	0.0	1.4	3.7	2.7	1.5
Phi-4-reasoning					
Full	0.4	2.6	3.6	3.1	1.4
SnapKV	0.8	1.5	3.2	3.6	0.5
R-KV	0.1	3.3	3.6	5.5	1.3
VaSE	0.7	2.4	3.1	3.7	0.9
TriAttention	1.3	1.4	5.1	4.6	2.0
Random Attention (ours)	1.0	2.3	3.3	3.5	1.4
Qwen3-32B					
Full	0.6	1.2	3.3	4.1	0.8
SnapKV	0.6	1.8	3.2	4.0	1.7
R-KV	0.1	2.8	3.0	2.3	0.3
VaSE	0.2	1.7	3.7	4.4	0.7
TriAttention	0.7	1.2	3.7	3.9	1.0
Random Attention (ours)	0.3	1.1	3.8	4.3	1.6
Qwen3-14B					
Full	0.1	2.2	2.9	3.2	1.2
SnapKV	0.4	2.3	3.9	3.2	1.4
R-KV	0.0	1.8	3.4	3.6	0.6
VaSE	0.7	1.5	3.8	3.1	2.0
TriAttention	0.7	1.6	4.4	3.3	0.8
Random Attention (ours)	0.5	2.6	3.7	3.4	1.3

degrades it toward a recency policy). R-KV uses λ=0.5, the setting behind the R-KV rows in Chang et al. (2026), which our cells reproduce to within 2.3 points on MATH500 and 0.8 on GPQA-D; the recommended λ=0.1 scores 7–9 points lower on Qwen3-4B in both our port and a line-by-line re-implementation of the official repository. TriAttention is ported verbatim from the official implementation in its stronger per-head variant, with per-model calibration statistics that reproduce the reference ranking (mean reciprocal rank 0.99); its released harness evaluates reasoning models with the chat template disabled, which is why we do not compare against the paper's reported numbers. Our full-attention path reproduces the full-attention accuracies reported by Chang et al. (2026) on Qwen3-4B to within 0.5 points on all four shared tasks. All cells are graded at the 32k-token limit by the final boxed answer (LiveCodeBench: by test execution), and no partially completed cell enters any table.

Hardware. Accuracy generation ran on a mixed fleet of NVIDIA H200 nodes. Every efficiency measurement in §6 and Appendix G is H200-only, with a single job per GPU and nothing else on the node.

Per-task budgets. The main grid sets MATH500 K=1024, GPQA-D K=2048, AIME/HMMT K=4096, and LiveCodeBench K=3072, $\sim$3$\times$ compression of its $\sim$10k-token full-attention traces.

G EFFICIENCY: PROTOCOLS AND ADDITIONAL MEASUREMENTS

The vLLM setting (Table 4). Every run is bf16 on one H200, vLLM v0.19.0 with PagedAttention (CUDA graphs and prefix caching disabled), budget 2048 and 1k-token prompts, inside the compression integration released with TriAttention, which we run unmodified except for the selector: Random Attention is added as a scoring function inside it, so both methods share the same attention kernels, paging, scheduler and compression trigger and differ only in which positions are kept, with a request compressed every 64 generated tokens. Compression is active throughout every compressed run (55,296 eviction events at the 8k point and 61,568 at the 32k point, identical for the two methods, with no skipped round and no prefix mismatch; 59,008 at the 32k point for Phi-4-reasoning), and the same integration reproduces our accuracy at the same cadence (0.864 on Qwen3-4B MATH500 at $K{=}1024$). The compressed cells are single runs; repeating the benchmark with fresh seeds, two to three per cell in earlier rounds at these load points, moved every arm by at most 1.1% (Qwen3-14B Random Attention by 0.05%), and the full-attention cells, which the plugin never touches, are means of two runs with three repetitions agreeing within 0.4%; run-to-run variability is therefore about one percent, two orders below the margins we report. Qwen3-32B runs the same protocol; its 64 GB of weights shrink the free KV pool, which lowers the preemption-safe cap (below) and, at 32k generations, where a full-length full-attention sequence carries over 8 GB of KV, drops full-attention throughput to 346 tok/s. Phi-4-reasoning also runs the same protocol, with 31.5k-token generations, the most its 32k context allows after the 1k-token prompt. The 128-request point is close to the paged capacity plateau, the regime in which TriAttention report their headline multiple: offering 512 requests on Qwen3-4B with Random Attention at its preemption-safe ceiling of 224 resident sequences (verified, zero preemptions) raises steady-state throughput from 2046 to 2188 tok/s, +7%; the decoding step there fits 9.5 ms +0.415 ms per resident sequence, so throughput is already 85% batch-proportional at 128 and cannot rise by more than 18% at any batch. On Qwen3-14B the same probe leaves throughput within 1% of the 128-request row (1817 vs. 1819 tok/s). With both methods at the ceiling the margin holds: 2117 vs. 1501 tok/s on Qwen3-4B (+41%) and 1817 vs. 1276 on Qwen3-14B (+42%), against +37% and +40% at 128 requests. On Qwen3-32B the 128-request run already operates at its 96-sequence ceiling, so no separate probe is needed.

Short generations, where compression does not pay. A second operating point offers 512 requests of 1k in, 8k out, with the compressed runs at their preemption-safe caps (224 on Qwen3-4B and 14B, 96 on Qwen3-32B) and full attention uncapped. There the arithmetic, not the cache, bounds throughput, and a compressed cache buys nothing: Random Attention serves 0.52×, 0.70× and 0.96× the full-attention throughput on Qwen3-4B, 14B and 32B, and 0.76× on Phi-4-reasoning; TriAttention 0.38×, 0.49×, 0.70× and 0.57×. The deficit shrinks with model size because per-token KV grows with the model while the weights shrink the cache pool, so the same workload becomes memory-bound as the model grows, reaching near parity on Qwen3-32B. Phi-4-reasoning, with the parameter count of Qwen3-14B but a quarter more KV per token (40 layers of 10 KV heads against 40 of 8), sits between Qwen3-14B and Qwen3-32B, as this account predicts. The margin between the two compressed methods is unchanged there: Random Attention leads TriAttention by +39%, +42% and +36% on the Qwen3 models and +35% on Phi-4-reasoning. The 64-request and single-request measurements keep the same request shape with the offered load reduced to 64 and 1. At the 64-request load Random Attention leads TriAttention by +35% on Qwen3-4B and +30% on Qwen3-14B; at a single request the two are within about one percent of each other (115.8 vs. 117.0 s on Qwen3-4B, 125.8 vs. 126.9 s on Qwen3-14B), consistent with the sub-2 ms calls of Table 9: the serving margin is the barrier-multiplied cost of reading paged cache state, not a kernel-time gap, and an integration that scored asynchronously, off the barrier, could shrink it; none is released, and Random Attention needs none. Two pitfalls in the released tooling silently make it measure the wrong thing and are worth recording: vLLM's own throughput benchmark ignores the requested output length (its default 128-token outputs never reach the compression threshold, so it times full-attention decoding), and the integration's deduplication guard can disable all later compaction after one benign under-budget round. We corrected both with bookkeeping changes that leave selection and kernel semantics untouched; every reported run is verified by its applied-event counters.

TriAttention report ∼2.5× over full attention from a capacity measurement taken at a different budget and decode length, on a different GPU; our 1.2–2.0× for their method sits below it, and the

Table 9: Cost of one per-layer eviction call (scoring + compaction), measured with CUDA events on an otherwise idle H200: $K{=}1024$, 4096 decode steps, single stream. Random Attention performs no scoring, so its round time is the compaction floor every evictor pays; the excess over it is the price of the selection signal. The ordering, and the per-call costs to within 12%, are unchanged across a 3.5× change in model size.

	Qwen3-4B		Qwen3-14B	
Method	ms / call	% of decode	ms / call	% of decode
Random Attention (ours)	0.30	0.57%	0.29	0.53%
SnapKV	0.37	0.68%	0.39	0.68%
R-KV	0.58	1.06%	0.61	1.07%
VaSE	0.74	1.35%	0.77	1.37%
TriAttention	1.47	2.49%	1.64	2.67%

comparison we draw is between the two methods we measured rather than against their published figure.

The paged table runs at $K{=}2048$ and the equal-memory table (Table 10) at 3072; the 32k point rerun at $K{=}3072$ preserves the ordering (Random Attention 2011 and 1696 tok/s against TriAttention's 1437 and 1223 on Qwen3-4B and 14B, +40% and +39%), so the budget difference between the two tables drives neither comparison.

The concurrency cap. When the KV pool is oversubscribed, vLLM V1 preempts requests silently, and the integration's compression state does not survive preemption: on resume it compacts inside block IDs the request no longer owns, and the scheduler's consistency check then kills the engine. Capacity measurements therefore need a cap that keeps the pool undersubscribed; ours is 224 requests on Qwen3-4B and 14B and 96 on Qwen3-32B at the 8k capacity point, whose 512 offered requests keep the pool under sustained admission pressure. The 32k point offers 128 requests in total and runs preemption-free at that concurrency on Qwen3-4B and 14B; on Qwen3-32B it is capped at 96. An uncapped compressed run under sustained oversubscription is not possible in this integration. Both compressed methods run under the identical cap, so the comparison between them is unaffected. Full attention tolerates preemption (vLLM recomputes the evicted request) and runs uncapped, which favours it on the two smaller models; on Qwen3-32B the 512 offered requests oversubscribe its pool, so part of its deficit there may be preemption overhead rather than capacity alone, and the comparison we rest on at that scale is the cap-matched one against TriAttention. The 32k full-attention baselines are oversubscribed in the same way on every model, so the full-attention multiples of Table 4 bundle capacity with full attention's preemption overhead; the margins over TriAttention, measured under identical load, are unaffected.

Eviction-round timing (Table 9). The full round, scoring and compaction, is timed with CUDA events on an otherwise idle node, single stream, $K{=}1024$, 4096 decode steps, 1872 (4B) and 2080 (14B) eviction calls.

The equal-memory comparison (Table 10, i.e. Figure 6 with exact throughput). Every method runs on the same engine and code path, so the comparison among them isolates the scoring pass; only the batch differs, found per method by bisecting for the largest that fits a 143 GB H200. Full attention needs its own search at each decode length (at 14B and 32k, batches of 40, 38 and 34 all run out of memory and the largest feasible is 20, at 129.2 GB peak). One caveat applies to TriAttention and to no other selector: its row runs our PyTorch port, and their paper states that no optimised kernel exists yet, so the row reflects an unfused implementation rather than the speed TriAttention can attain, which is why the method-level comparison in §6 is the vLLM one, run on the kernels they release. No throughput number in this paper comes from the batched accuracy runs, where several workers share a GPU.

Equal memory at a tighter budget. The multiple over full attention is set by the batch the cache admits, so it grows as the budget shrinks. Table 11 repeats the protocol on Qwen3-4B at $K{=}1024$, the MATH500 budget of Table 1: the compressed caches now fit 544–584 sequences against 28, and

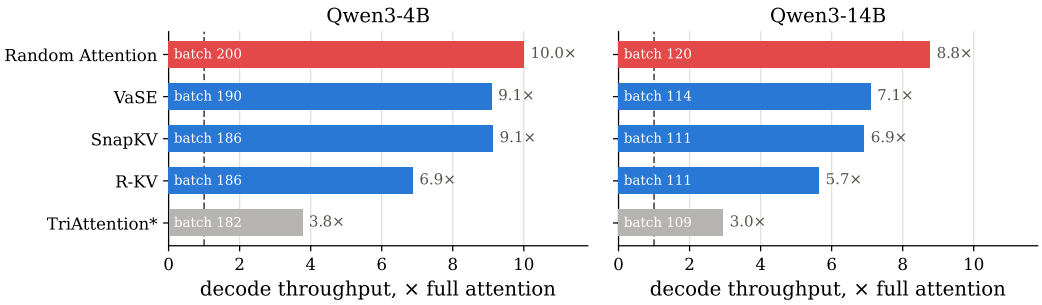


Figure 6: Equal-memory serving: decode throughput relative to full attention at each method's largest batch on one H200 (K=3072, 32k generations). *TriAttention here is an unfused re-implementation of its scorer, far slower than the vLLM version.

Table 10: Serving throughput when each method runs at the largest batch that fits one 143 GB H200, at K=3072 with 32k generations. The small cache is what buys the batch, so every evictor collects most of the win; the ordering among them follows the cost of their scoring pass. **The TriAttention row is an unfused re-implementation of its scorer, far slower than the authors' kernels, on which the gap to Random Attention on these two models is 1.4×, not 2.7–3.0× (Table 4).** Every other row runs on one shared code path.

	Qwen3-4B			Qwen3-14B		
Method	batch	tok/s	× full	batch	tok/s	× full
Full (no eviction)	28	178	1.00	20	164	1.00
SnapKV	186	1624	9.14	111	1133	6.92
R-KV	186	1223	6.88	111	925	5.65
VaSE	190	1617	9.10	114	1162	7.10
TriAttention	182	670	3.77	109	483	2.95
Random Attention (ours)	200	**1779**	**10.01**	120	**1436**	**8.78**

Random Attention serves 28.8× the full-attention throughput, 16% more than SnapKV and 20% more than VaSE at their own largest batches.

Quoting TriAttention's R-KV comparison. We cite their matched-budget row rather than their headline. Their reported throughput varies almost entirely with the budget and hardly at all with the selector (1405, 760, 564 and 414 tok/s at K=1024, 2048, 3072 and 4096), and the headline +85% over R-KV places the two methods at different budgets (1024 for TriAttention, 2048 for R-KV), so it measures the budget, not the selection signal. At equal budget their table reports 1405.2 against 1345.5, which is the +4.4% we quote and the conservative figure for our purposes. That row was measured on an A100 at 16k decode and maximum batch, neither our hardware nor our operating point, so it combines with our own measurement only as an indication, not as a controlled comparison.

What porting an evictor to a paged runtime involves. Two settings are necessary because no released runtime hosts every method. TriAttention ships a vLLM v0.19.0 plugin, which is what Table 4 uses. R-KV ships ports as well, but against a different pinned vLLM release, so the two cannot serve in one comparison without re-porting one of them; SnapKV and VaSE are released as HuggingFace-side implementations. The wiring is the larger part of the work and is independent of the score: physical eviction under paging has to rewrite each request's surviving KV into its blocks, free the tail, keep rotary positions logical while the physical slots shrink, and stay correct across preemption; R-KV's released port spans ∼849 lines of wiring across 13 upstream files and requires vLLM's V1 model runner, and the integration we serve on mishandles preemption (above). The score then decides what else is needed. A score read from the cache alone (VaSE's value range, TriAttention's calibrated key statistics) costs gathers across the block table. A score read from

Table 11: Equal-memory serving at $K{=}1024$ (Qwen3-4B, 32k generations, one 143 GB H200): the tighter budget fits a batch of 584 against 28 for full attention, and Random Attention reaches $28.8\times$ full-attention throughput. R-KV and TriAttention were not measured at this budget.

	Qwen3-4B		
Method	batch	tok/s	× full
Full (no eviction)	28	178	1.00
SnapKV	544	4400	24.79
VaSE	552	4273	24.07
Random Attention (ours)	584	**5110**	**28.79**

attention weights (SnapKV, R-KV's importance term, VaSE's attention-proportional fill) cannot be read at all from a fused paged kernel, which never materialises the weights, so it must either be recomputed as an explicit window-query product against every candidate key (chunked, since the transient is heads × window × cache per request) or be extracted by modifying the attention kernel itself. Random Attention needs neither: its keep-set is a random permutation of slot indices, so it reads nothing and the runtime's existing compaction path is the whole integration, which is why adding it to TriAttention's plugin took a single function.

Why the two settings report such different multiples. On Qwen3-14B at 32k generations, Random Attention runs at 1436 tok/s here and 1819 in vLLM, while full attention runs at 164 here and 925 there: the compressed method gains $\sim 1.3\times$ from the better engine and the full-attention baseline gains $\sim 5.6\times$. Paging is the reason for the asymmetry: reserving a full cache per sequence is exactly the constraint compression relieves, and a paged allocator relieves it too, admitting requests as memory frees rather than capping the batch at 20. The compressed methods have little left to gain, since their caches are small under either allocator. The two settings also differ in budget (3072 here, 2048 there) and in request count, so this attribution is indicative rather than a controlled decomposition; what it explains is why an unpaged capacity multiple of $8.8\times$ and a paged one of $1.97\times$ are consistent measurements of the same effect.

H Discussion and limitations

What the evidence covers. Our experiments study decode-phase eviction for long chain-of-thought reasoning: prompts of a few hundred tokens, traces of several thousand to 32k tokens, budgets from $2\times$ to $16\times$ compression, four reasoning models, six tasks in math, science and code, and training-free evictors as baselines. The claim that the selection signal contributes almost nothing is a claim about this regime, and "strongest baseline" means the strongest of the training-free methods we evaluate. It is also a claim about the aggregate rather than every cell: four of the 80 baseline comparisons in Tables 1 and 5 favour a baseline significantly (TriAttention on LiveCodeBench at 14B and 32B and on MATH500 at 14B, and VaSE on AIME at 14B). Long-input workloads, where the prompt itself fills the cache, fall outside this regime and have their own evidence (Garcia, 2026).

What the claim rests on. The 31 significant wins in the main grid are against the baselines as released, and §5.1 shows that most of them come from prompts those baselines lose; they establish that Random Attention is competitive, not that scores carry no information. The evidence about the score itself comes from the comparisons in which protection is matched. Against TriAttention, which keeps the prompt as Random Attention does, the 20 cell differences in Tables 1 and 5 range from -2.8 to $+2.9$ points and average 0.0. With the prompt given to every other baseline, none of the 27 protected baseline cells of Tables 2 and 6 scores above Random Attention. We state this parity descriptively rather than through a formal equivalence test, and a non-significant cell is not by itself evidence of equality; the claim rests on the pattern across these matched cells, not on any one of them.

Signal-free, not structure-free. Random Attention computes no score, but it is not free of structure. It pins the prompt, and because every surviving position faces a fresh draw at each eviction, the survival of a trace position decays geometrically with its age, about 0.94^n after n evictions at

$K{=}1024$ (Appendix D); the policy behaves as a soft recency window. This age profile is not a design choice: any memoryless draw produces it, and at each eviction every trace position is treated alike. By "selection signal" we mean what the baselines add on top of such structure, a score computed from the cache (attention weights, values or keys). Our claim is that this score adds little once the prompt is kept, not that structure is unimportant; protection is the largest effect we measure.

Protected recency. The nearest signal-free alternative is therefore a hard recency window with the prompt kept. In the four matched settings of Table 2 it is never more than two points below the best protected baseline, and Random Attention is above it in all four, by 1.1–3.1 points. The two spend the same budget differently: the window keeps every recent position and nothing older than the budget, while Random Attention keeps a thinning sample of both and still holds 19% of the 1–2k band in each head and 78% across heads at $K{=}1024$ (Appendix D). The tail can help only when the model needs something older than the budget that it has not restated within it, and it cannot help once too few copies survive, as in the passcode test of §5.3. We ran the protected window only in these diagnostic settings, not across the full grid or the compression sweep; the two policies cost the same, and where we ran both, Random Attention scored higher. We also centre the paper on Random Attention because a null for a selection signal should not assume which trace positions matter, and a window assumes that only recent ones do. Were a protected window to match Random Attention everywhere, our main conclusion would stand, since neither computes a score.

Relation to other protection results. This conclusion agrees with Garcia (2026), who finds that protection dominates scoring for long-context question answering under a global cache cap, and with the concurrent Prefix Sliding (Muennighoff et al., 2026), which uses protected recency during reasoning; what we add is matched evidence in decode-phase reasoning against four scored evictors, an account of why the trace tolerates signal-free eviction, and the serving cost of scoring.

Matched protection across the grid. The main grid runs every baseline as released, because that is what a practitioner would deploy, so each cell mixes the effect of the score with whatever protection the method applies by default. We separate the two where the grid showed large, method-specific gaps: the four settings of Table 2 and the code, competition-math and 32B GPQA-D settings of Appendix C; every TriAttention-versus-Random Attention cell is already matched, since both keep the prompt. We have not run the protected grid on Qwen3-14B, but the missing cells would not explain away the three baseline wins there: two are TriAttention's, already matched, and VaSE's AIME win is on competition math, where the rule did not help VaSE on Qwen3-4B (-2.1 and -1.0 points on AIME and HMMT, Table 6). What the missing cells would change is the size of Random Attention's margins over SnapKV, R-KV and VaSE, which the rule narrows where a baseline had been losing the prompt.

Which redundancy does the work. The two levels of redundancy have different evidential status. That copies of a fact pool across heads is shown directly, but only in the planted-fact probe, where the text is not redundant by construction. On real MATH500 traces a shared draw, which keeps the same positions in every head, scores within 0.3 points of Random Attention at $4\times$ and $8\times$ compression (Appendix D). On the traces we tested, then, the text carries the load, and cross-head redundancy is a second line of defence for what the text does not restate; we present the probe as evidence of that mechanism, not as the cause of the benchmark results. Text-level redundancy is itself inferred, not measured: it is documented for reasoning traces by Cai et al. (2025), and it is what reconciles our two observations that random eviction preserves benchmark accuracy while a once-stated passcode is lost. Independent per-head draws cost nothing over a shared draw, so we keep them, but we have run the shared draw only on MATH500 and the probe only on Qwen3-4B.

Where a selection signal still pays. The clearest case is a fact stated once, never restated, and needed much later: Random Attention never recovers a passcode announced 57 compression rounds before the question, while R-KV recovers it 84% of the time (Table 3). Workloads built around such facts, for example state that an agent reads once and consults much later, are where a content-dependent signal earns its cost, and Random Attention is the wrong default there. We have not measured how often real reasoning traces contain such facts, that is, how often a value the model uses was last stated more than a budget earlier; the aggregate results suggest they are rare on our benchmarks. A natural hybrid keeps the random draw for most of the budget and reserves a few

slots per head for a cheap content signal, much as VaSE reserves slots for large values; we have not evaluated one.

The code gap. Code is the only task on which a content signal wins at more than one scale: TriAttention leads on LiveCodeBench at 14B and 32B, by 2.3 and 2.8 points. Both methods pin the prompt, so the prompt does not produce this gap by itself; what it does is shrink the budget left for the trace, to as little as half of K on the longest code prompts. In the compression sweep on Qwen3-4B and Phi-4-reasoning, far smaller trace budgets on the math and science tasks leave Random Attention within a few points of TriAttention or ahead of it (Figure 3), so a small trace budget alone does not open a gap there. Code traces that define names and interfaces once and use them much later would sit closer to the once-stated regime above, which would explain the gap; we have not run the prompt-length-stratified analysis, or the sweep at the larger models, that would test this.

Budgets and long prompts. Random Attention spends part of its budget on the prompt before any selection happens, and this is its main cost. On LiveCodeBench, the longest prompts take up to half of the $K{=}3072$ budget, and at $16\times$ compression on MATH500 ($K{=}256$) the protected prompt no longer fits for both prompt-keeping methods on Phi-4-reasoning and for TriAttention on Qwen3-4B (the $\times$ markers in Figure 3); VaSE, which scores the prompt, degrades more gracefully there. The method is therefore defined for budgets that hold the prompt. Long-context and multi-turn settings, where the prompt or conversation history can exceed any practical budget, need a rule for which parts to protect, and compacting the scaffolding of long code prompts is a smaller version of the same question; we leave both to future work.

Architectures. All four models use grouped-query attention with eight or ten key-value heads per layer, and three of them are Qwen3 sizes; Phi-4-reasoning differs in family, tokenizer and training data. We have not tested architectures where per-head independence is unavailable or different in kind: multi-head latent attention, whose heads share one latent cache per layer, multi-query attention, or hybrids of full and sliding-window layers. The shared-draw result suggests that text-level redundancy alone carries real traces at our budgets, which is the case that matters when per-head draws are unavailable, but this is an inference, not a measurement.

Efficiency. The paged-serving comparison includes only TriAttention, the one baseline whose released integration runs on the vLLM version we use (Appendix G), and its 32–43% margin is a property of that integration as much as of the algorithm. The score's own cost is small: about two percent of single-stream decoding time (Table 9), and about one percent end to end at a single request. The integration multiplies it by compressing at a synchronisation point between batched steps, so the margin grows with load, from 30–35% at 64 concurrent requests to 41–42% at capacity; a fused or asynchronous scorer could remove much of this amplification, though not the score's own cost, which Random Attention does not pay. Compression itself pays only when the cache limits concurrency: at 8k-token generations every compressed method serves less than full attention (Random Attention 0.52–$0.96\times$, TriAttention 0.38–$0.70\times$), while Random Attention's lead over TriAttention holds. The throughput runs use fixed output lengths, whereas under natural stopping the methods generate different amounts: averaged over the five task columns, Random Attention generates fewer tokens than TriAttention on three of the four models and 5% more on Qwen3-32B (Appendix E), well inside its throughput margin. The exception is LiveCodeBench on Qwen3-32B, where Random Attention's traces are 44% longer and its end-to-end advantage would not hold.

Statistics and randomness. Every margin we call significant passes both a paired, problem-clustered bootstrap and an exact sign test at the 5% level, each comparison on its own, without a correction for multiple comparisons. Over 80 comparisons a few cells in either direction may be significant by chance, including the weakest of the baseline wins ($p{=}.02$, Appendix A), which is why no conclusion rests on a single cell. Eviction randomness adds to the randomness of sampled decoding; every accuracy averages 2 to 16 independently sampled runs, whose spread Appendix E reports, and a per-request seed makes Random Attention reproducible when needed.

Learned evictors. Learned evictors such as TrimKV (Bui et al., 2026b), DBTrimKV (Bui et al., 2026a) and ForesightKV (Dong et al., 2026) require training a scorer for each base model on domain-

specific data, so we compare only against training-free evictors, which run on any model as released. They relate to our findings in two ways: TrimKV and DBTrimKV learn how fast each token's score decays with age, a learned, per-token version of the uniform rate at which a position's survival decays under Random Attention (Appendix D), while ForesightKV learns a content score from a future-attention oracle, the kind of content-dependent signal that recovered the once-stated passcode in §5.3.