AGENT MEMORY THAT PAYS ITS OWN RENT

Anonymous authors
Paper under double-blind review

ABSTRACT

An LLM agent can keep each memory item as text or as its key-value (KV) cache. Text is prefilled again at every read; the cache skips that prefill but pays for storage while it is kept. Serving engines such as SGLang and Mooncake keep the cache, and agent memory systems such as MemGPT and Mem0 keep text. The cheaper choice depends on the deployment. At the same hourly price, the reuse needed for caching to pay off differs 26.3 times between two accelerators. We present METER, a controller that derives a break-even period, over which storing a cache costs as much as recomputing it, from model configuration, storage prices, and measured prefill and load times. METER admits an item once two of its reads fall within one period; each read then renews the cache for another period, after which it lapses to text. On LongMemEval and LoCoMo, across 240 combinations of reader model and memory lifetime, METER costs at most 1.51 times an offline optimum that knows all future reads. Its admission test is that optimum's own per-gap criterion, so both cache the same items by construction; the 1.51 is timing, since METER cannot see when the reads stop. Never caching reaches 1.94 and METER without admission 1.99; rules that ignore storage prices reach 160 to 3,191. Twenty-one executed runs on two accelerators stay within 1.49. On Mooncake serving traces, with over a thousand times more reuse, METER without admission costs 1.01 against 2.23 for never caching, and is provably within twice the cheapest choice on every gap between reads.

1 INTRODUCTION

An agent writes a passage into memory at one time and reads it at another. In between, the passage can be held as text and prefilled when it is read, or held as its key-value cache and loaded. The second skips the prefill and pays for bytes while the item sits idle, so the decision is a pricing question and not an eviction heuristic. Priced badly it is expensive: rules that price nothing cost up to 3,191 times what an offline optimum pays over the same reads.

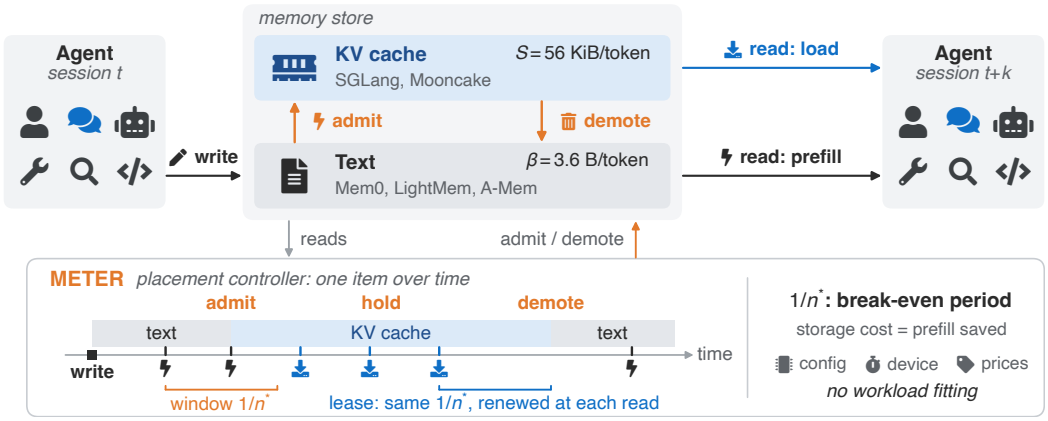


Figure 1: Overview of METER. An item is held as *text* at $\beta = 3.6$ bytes per token or as a *KV cache* at $S = 56$ KiB (Qwen2.5-7B), read by a prefill or a load. Below: one item's life, and the lease each read renews (Algorithm 1).

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.

The prices that would settle that question are not in the literature. Agent memory systems choose what to retain and retrieve (Packer et al., 2023; Fang et al., 2026); serving stores manage cache capacity and sharing (Zheng et al., 2024; Qin et al., 2025); ski-rental policies derive a cache lifetime from a holding cost and a recomputation cost handed to them (Kumar et al., 2025; Li et al., 2025). Of 23 ICLR papers whose subject is retained state or agent memory, none prices storage over the interval the state is held and none reports how often a retained object is read (Appendix D), so a published cache-versus-recompute result cannot be recomputed on different silicon, at a different storage price, or over a different residency. The quantities move: with model, workload and accelerator-hour price held fixed, the charged warm-load threshold differs by $26.3\times$ between the two devices tested here.

METER takes its prices from the device and the representation it runs on (Figure 1). From a model configuration, a price sheet, measured prefill and load times, and an item's own read timestamps it derives one length, the interval over which holding a cache costs what rebuilding it would. It spends that length twice: as the window that admits an item when two of its reads fall inside it, and as the lease that then holds the cache, renewed by every read. Where several storage tiers are available, the cheapest of them at each elapsed time extends both jobs to a multi-tier store; the hardware evaluation uses one KV tier and demonstrates no physical cascade.

Over 240 reader-by-residency cells METER caches exactly the items an offline optimum caches, at precision, recall and F_1 of 1.00, including all 127 cells where the optimum caches some items and not others. This follows from the derivation, not the data: the optimum holds a cache across a gap exactly when it is shorter than $1/n^*$, METER's own test (Section 5.2). Scoring on cumulative history instead is exact in only 47 of the 240. The agreement carries into execution: over 21 runs on two accelerators METER caches the executed reference's items every time, where caching at the write reaches 185.02 times that reference. What it still pays above the optimum lies in when it admits and how long it holds, not in which items it keeps.

We contribute three things. The first is METER itself, the rule and the calibration that supplies its length (Section 4). The second is a separation: a placement rule's choice of items and its schedule for them can be measured apart, and METER's choice is exact while what it overpays is all schedule (Section 5.2). The third is a per-gap bound: without the admission test the schedule costs at most twice the cheapest tier over any gap between reads (Theorem 1). That bound covers the schedule and not the admission test, so the rule we recommend on agent memory carries no proved guarantee, and a maximum over a finite grid is not a competitive guarantee either.

Priced this way, one controller covers both ends of a reuse axis the field treats as two problems: within 1.51 times the optimum on agent memory, and, where reuse is dense enough that the price says to lease every block, 1.01 on an hour of serving traffic against 2.23 for never caching.

2 Related Work

Agent memory systems select, summarize, page, or link information (Park et al., 2023; Packer et al., 2023; Fang et al., 2026; Xu et al., 2025; Gutiérrez et al., 2024; Xiong et al., 2026). Those operations determine what the agent remembers. METER instead takes a chosen memory item and decides where to retain its representation. Our shared-writer comparisons isolate this distinction; they do not rank complete deployed memory systems. Retained or compressed latent representations (Eyuboglu et al., 2026; Hardalov et al., 2026; Chang et al., 2025; Sun et al., 2024; Zhang et al., 2026) change footprint and may change the information available at read time. Positional transfer (Li, 2026) addresses compatibility at reuse, a condition our single-prefix execution assumes. Appendix D describes a bounded audit of 23 retrieved papers, not an exhaustive claim about the literature.

Cache compression, sharing, paging, and offloading are established serving techniques (Liu et al., 2024; Gao et al., 2024; Yu et al., 2025; Kwon et al., 2023; Zheng et al., 2024; Qin et al., 2025; Shin et al., 2026; Zhu et al., 2026). They address capacity and transfer costs that our independent-item model simplifies. METER is not a replacement for their allocation machinery: its leases provide a price-sensitive placement decision that would need integration with such machinery. The serving replay tests the cost rule, not that integration.

Break-even cache economics (Zhang, 2026), ski-rental lifetimes (Kumar et al., 2025; Li et al., 2025), file caching (Young, 2002), and expiring-purchase problems (Fleischer, 2001; Karlin et al., 1988; Zhao et al., 2024) supply the closest methodological context. We claim neither the classical factor of

two nor the principle of leasing as new. Our artifact combines device-derived prices, observed-read admission, and renewable tier placement. The lower-envelope argument specifies the no-admission variant's guarantee; the admitted rule is assessed empirically, including its failures.

3 THE PLACEMENT PROBLEM

3.1 CARRIERS, AND WHAT EACH COSTS

An agent commits content x at the write and retains an object m: either $m_{\text{txt}} = x$, the token sequence, or $m_{\text{lat}} = \Phi_\theta(x)$, the key-value cache a model with parameters θ produces from it. Three costs follow, which the literature conflates under "memory". **Context occupancy** $O(m)$ is the attention positions the carrier holds at answer time; **read time** $\tau(m)$ is the time a read spends before the carrier is usable, whether computing or loading; **storage** $S(m)$ is the bytes retained between write and read. For a prefix of N tokens under P parameters,

$$\begin{aligned} O(m_{\text{txt}}) &= O(m_{\text{lat}}) = N, \\ \tau(m_{\text{txt}}) &\approx 2PN/\Pi, \qquad \tau(m_{\text{lat}}) \approx SN/W. \end{aligned} \tag{1}$$

for a stack sustaining Π FLOPs per second and loading state at W bytes per second. Equation 1 fixes equal occupancy by construction. Read cost is the axis existing work reports, and a cached read is cheaper than a prefill but not free: restoring state still occupies the accelerator, taking 36% to 61% of the prefill it replaces on our Ascend (Table 12), so we charge it. Storage is the axis we measure. For a fixed prefix, model, positions, attention mask, and numerical computation, reusing the exact prefill state preserves the conditional distribution:

$$p_\theta(y \mid x) = p_\theta(y \mid \Phi_\theta(x)) \,. \tag{2}$$

Equation 2 is an ideal identity under those conditions. Quantization, different kernels, or an altered mask can violate its implementation assumptions. We therefore test accuracy rather than assume equality. The read budget is N slots; B below denotes prefill cost, not context capacity.

For an attention stack with L layers, h_{kv} key-value heads and head dimension d, storing both keys and values at b bytes per element costs

$$S_{\text{token}} = L \cdot h_{kv} \cdot d \cdot 2 \cdot b \tag{3}$$

per token. The conventional designs are this one equation at different h_{kv}, so the span across them is set by head count alone, a choice usually made for decode throughput rather than for memory. We verify Equation 3 against allocated caches on two independent stacks and it agrees to the byte on both. Writing β for the bytes one token occupies as text, the premium a carrier charges is $\rho = S_{\text{token}}/\beta$. Latent attention, retained residuals and a quantized cache are all priced in Appendix E. These alternatives change footprint and can change predictions. We evaluate the closed forms for 30 architectures from 135M to 120B parameters at fp16, loading no weights. The text baseline is measured rather than assumed: the reference passage is 289 bytes of UTF-8 across 80 tokens, 3.6 bytes per token.

3.2 BREAK-EVEN, AND THE LEASE PER TIER

Latent state trades storage for avoided recomputation, so the two meet at a reuse rate. Hold one token of content for T months and read it n times per month, at storage price c_s per byte-month and compute price c_c per FLOP. The text carrier stores β bytes and pays a prefill on every read; the latent carrier stores S bytes, pays that prefill once at write time, and pays a load on every read. Writing R_{txt} and R_{lat} for the dollars one read costs per token under each,

$$\begin{aligned} C_{\text{txt}} &= T\beta c_s + nT\, R_{\text{txt}}, \\ C_{\text{lat}} &= TSc_s + R_{\text{txt}} + nT\, R_{\text{lat}}. \end{aligned}$$

Equating the two and dividing by T gives the break-even reuse rate

$$n^* = \frac{(S-\beta)\, c_s}{R_{\text{txt}} - R_{\text{lat}}} + \frac{R_{\text{txt}}}{T\,(R_{\text{txt}} - R_{\text{lat}})}. \tag{4}$$

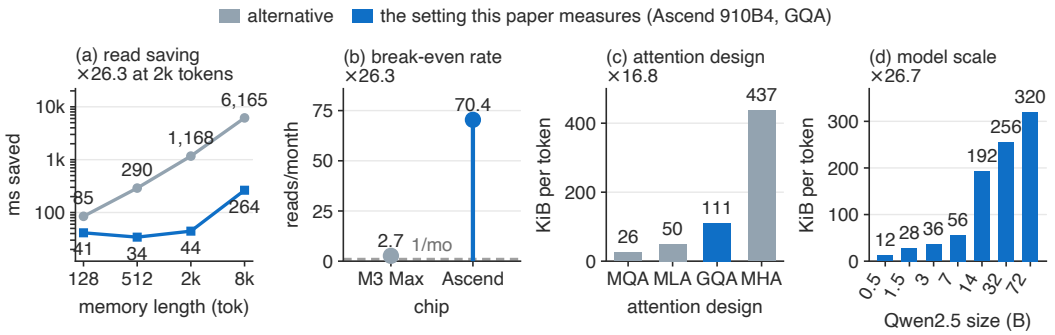


Figure 2: The threshold is a ratio: holding cost over read savings. (a) Milliseconds saved, Qwen2.5-3B, both chips. (b) The faster chip needs 26.3× more reuse. (c) Bytes per token by attention design, (d) by scale.

Because ρ runs from 3,413 to 227,556, dropping β moves n^* by at most 0.03%. The write-side prefill adds $B/(B-R)$ to the required total read count, so it must be retained in finite-lifetime write-time comparisons. Threshold tables report the $T \to \infty$ limit and keep the load: on the Ascend it moves n^* by up to 2.56× (Table 12). A machine renting at p_{acc} dollars an hour charges a read for the time it occupies, so $R_{\text{txt}} = 2Pp_{\text{acc}}/(3600\,\Pi)$ and $R_{\text{lat}} = Sp_{\text{acc}}/(3600\,W)$, giving

$$n^* = \frac{3600\,(S-\beta)\,c_s}{p_{\text{acc}}\,(2P/\Pi - S/W)}, \tag{5}$$

which is increasing in prefill throughput Π: faster hardware raises the threshold rather than lowering it. It also rises with S twice over, once in the numerator and once through the load in the denominator, so a large enough cache on a slow enough link cannot pay at any reuse rate.

The reciprocal of n^* is a length of time: the lease. After a read, holding the cache costs incremental storage at $\sigma = (S-\beta)c_s$ per token-month and dropping it costs one net prefill $B-R$ at the next read, so holding pays exactly while the elapsed time is under

$$\ell = \frac{B-R}{\sigma} = \frac{1}{n^*}, \tag{6}$$

Each tier has its own holding rate and load cost (Table 1). The public price sheet and transfer assumptions are stated in Appendix E.4. Hosted-cache prices yield leases of about 19.6 minutes in this model, while object storage yields 4.2 days at the same reader, a ratio of 862 in price. This is a correspondence in scale, not a derivation of any provider's chosen expiration policy. A tier whose load costs at least a prefill is never admitted. Price ratios alone do not determine lease ratios because load costs also change across tiers.

Table 1: The lease METER derives per storage tier, Qwen2.5-3B at 512 and 2,048 tokens: minutes at the hosted-cache price, days on object storage. Prices are US dollars per GB-month; the accelerator columns are an Apple M3 Max and a Huawei Ascend 910B4, at a read budget in tokens.

Tier	$/GB-mo	M3 Max		Ascend	
		512	2,048	512	2,048
hosted cache	19.816	21 min	20 min	4 min	2 min
HBM	18.262	22 min	21 min	5 min	2 min
DRAM	4.821	77 min	78 min	9 min	3 min
NVMe	0.080	3.4 d	3.3 d	12.7 h	3.9 h
object store	0.023	4.6 d	4.2 d	–	–

3.3 What moves the threshold

Figure 2 separates measured read savings from model footprint. For Qwen2.5-3B at 2,048 tokens, the charged warm-load threshold differs by 26.3× between devices at a shared accelerator-hour price. Table 12 reports the timing inputs and Table 2 distinguishes analytical thresholds from stored device calibrations. The analytical sweep is not a prediction of achieved throughput for every reader. Its footprint varies by about 67× across 30 configurations; it is determined by Equation 3, not parameter count alone (Appendix A).

Table 2: Twelve readers at a 512-token budget: n^* spans 23.4 to 529.8 reads per month and 21 of 23 paired accuracy intervals include zero. Superscripts: n^* measured on the M3 Max / Ascend 910B4 (a 2.5 / 21.3; b 3.5 / 10.7; c –/ 33.2), per decimal GB-month. Bars: 95% bootstrap interval of latent minus text EM on $[-0.10, +0.10]$, zero at the tick; bold excludes zero.

				LongMemEval		LoCoMo	
Reader	Attn	KiB/tok	n^*/mo	Text EM	Latent − text	Text EM	Latent − text
Qwen2.5-7B	*GQA*	56	23.4	0.565	−.006	0.634	.005
Qwen2.5-32B	*GQA*	256	24.8	0.676	.000		*not run*
Qwen2.5-3B	*GQA*	36	37.1[a]	0.582	.018	0.662	.005
Qwen2.5-1.5B	*GQA*	28	57.7[b]	0.582	−.006	0.649	.003
Llama-3.2-1B	*GQA*	32	82.3	0.529	.000	0.554	.005
Gemma-3-1B	*MQA*	26	82.7	0.476	.006	0.528	−.005
Granite-3.1-2B	*GQA*	80	100.4	0.629	.012	0.696	−.010
Falcon3-1B	*GQA*	72	137.2	0.541	.006	0.593	.000
OLMo-2-1B	*MHA*	128	274.1	0.594	.006	0.613	.003
Phi-3-mini-3.8B	*MHA*	384	319.5	0.676	−.006	0.668	**.023**↑
Qwen3-0.6B	*GQA*	112	473.5[c]	0.435	.006	0.389	.010
SmolLM2-135M	*GQA*	22.5	529.8	0.153	−.029	0.144	**−.064**↓

4 METER

Equation 6 gives a length per tier, not a policy. METER is the rule that reads it, and the derived length does both of its jobs: the break-even period $1/n^*$ is the window that decides whether an item is worth caching, and it is the lease that decides how long the cache is then held.

Let B be the prefill cost per token and R_t the load cost from tier t, whose incremental holding rate is $\sigma_t = Sc_{s,t} - \beta c_{s,\text{text}}$ once the common text cost is subtracted. We consider tiers with $\sigma_t \geq 0$.

METER admits an item at a read when the previous read of the same item fell less than $1/n^*$ earlier. The test has no fitted constant, because the prior it needs is the one the derivation already supplies: under $\text{Gamma}(1, 1/n^*)$, a prior worth one break-even period, the posterior mean rate after a single observed gap Δ exceeds n^* exactly when $\Delta < 1/n^*$. Comparing one inter-arrival against the lease is therefore the rate test of Equation 5, not a proxy for it. Admission is free at the read that triggers it, since the prefill that read was going to pay is the build.

Both of the rule's counters live inside that window, and nothing older is consulted. This is what separates it from the obvious alternative, which scores admission on the item's whole history by comparing the prefills paid since the write against what a cache would have cost over the same elapsed age. That test has no window, so its bar collapses for a young item and grows without limit for an old one: it caches an item read once at its own write, and it cannot cache an item that lies idle and then turns hot. Section 5.2 measures both failures, and Table 3 prices them.

For a gap of length Δ after a read, define

$$f(\Delta) = \min\Big\{\min_t(\sigma_t\Delta + R_t), B\Big\}. \tag{7}$$

Once admitted, the controller holds the state attaining this lower envelope as elapsed time increases. Its slopes are non-increasing, so it moves toward cheaper storage and eventually back to text. This model assumes instantaneous, free downward moves and no shared capacity constraint. A read renews the schedule; if the previous schedule already expired, the window test is applied again to the reads that follow, which is how an item that goes quiet and later becomes busy is readmitted. Reads exactly at expiry

Algorithm 1 METER per item. Prices and timings are measured inputs.

Require: Prefill B, net holding rates σ_t, loads R_t, lease $\ell = 1/n^*$
active ← false; p ← none
for all reads at time r **do**
  Advance the previous schedule to r; expire only strictly before the read
  Serve from the current cache if active, otherwise prefill from text
  if not active and $p \neq$ none and $r - p < \ell$ **then**
    active ← true ▷ two reads inside one break-even period
  end if
  if active **then**
    Schedule the minimizer of $f(u)$ for elapsed time u after r
  end if
  $p \leftarrow r$
end for

are processed before expiration. With one tier the schedule is the lease in Equation 6.

The admission test reads off the same envelope. Admit at a read when the gap that just ended was short enough that holding across it on some tier cost less than the prefill at its end, that is, when f over that gap is not attained by text. With one tier that crossing is at $(B - R)/\sigma = 1/n^*$, so the multi-tier rule reduces to the test above and introduces no constant of its own.

Equation 7 defines the gap reference cost.

Theorem 1. *On any gap g between two reads, the cost of the cascade without admission is at most $2f(g)$, twice the cheapest tier cost for that gap, for any number of tiers with free downward moves.*

Proof sketch. The storage paid up to e is the integral of the envelope's slope, $f(e) - f(0)$, since the rule always holds the state whose slope is $f'(e)$. At the read it pays the current state's load, $R_{s(g)} \le \sigma_{s(g)} g + R_{s(g)} = f(g)$. The sum is at most $2f(g) - f(0) \le 2f(g)$. Appendix C.4 gives the full argument and the per-item corollary, $2\,\mathrm{OPT} + B$, the extra B being the lease that outlives the last read. □

The theorem bounds the schedule, which is the part of METER that runs after admission; it is proved for the variant that admits every item and is not a bound on the admitted controller. What admission contributes follows from the derivation instead: the offline optimum caches across a gap exactly when the gap is shorter than $1/n^*$, so over the 240 reader-by-residency cells of Section 5.2 METER caches the same set of items as the optimum by construction. The cost it still pays above that optimum is therefore a property of when it admits and how long it holds, not of which items it selects. Calibration must be refreshed when prices, prefill time, or load time change.

5 Experiments

5.1 Protocol

We use 170 extractive LongMemEval questions (Wu et al., 2025) and 388 single-hop LoCoMo questions (Maharana et al., 2024). Matched-budget accuracy conditions use the same reader and text prefix (Appendix B). The placement unit is an evidence memory, and a read is a question requiring that memory. LongMemEval provides question and session dates; LoCoMo provides session dates but not read dates. Its residency sweep therefore uses synthetic read timing and permuted question order, not observed deployment chronology (Appendix C.8).

Analytical replay uses twelve reader calibrations and ten residencies per corpus, from an hour to 15.7 months. Its reference storage price is $0.023 per decimal GB-month and accelerator price is $2 per hour. The benchmark grid holds one set of rounded reader thresholds fixed and reconstructs its prefill costs from those thresholds and holding rates, so every policy in a cell is priced by the same calibration. This is a cost-model comparison, not HBM priced at its physical capacity cost. The offline dynamic program knows future reads; small cases are checked by exhaustive search. The two public Mooncake traces (Qin et al., 2025) supply separate hour-long serving replays, with a 512-token prefix block as the item.

5.2 Policy comparison

Table 3 removes one part of METER at a time and prices the result over 240 reader-by-residency cells on both corpora.

The admission window selects the optimum's items exactly, and the derivation says why. The optimum takes a free transition at every read, so it settles each gap on its own, and by Equation 6 it holds a cache across a gap exactly while that gap is shorter than $1/n^*$, which is the test METER applies at the read. The two therefore select the same items by construction, at precision, recall and F_1 of 1.00 in all 240 cells and on each of the 127 open cells, where the optimum caches between 0.5% and 48% of items. We report this as a property of the rule, not as evidence for it. Selection does not settle cost, and that is where the rules separate: no other rule in the table averages above 0.5 on

Table 3: Ablations of METER and rules from prior work over 240 reader-residency cells on two corpora. Admission F_1: items a rule caches against the offline optimum's; cost: multiple of that optimum. Per-corpus columns cover the 127 *open* cells (the optimum caches some items, not all), the last column every cell. Bold is best.

	Admission	*LongMemEval*		*LoCoMo*		*all cells*
Rule	F_1	worst	mean	worst	mean	worst
METER	**1.00**	1.02	1.02	1.51	1.21	**1.51**
admission on cumulative history	0.29	1.06	1.04	1.51	1.18	**1.51**
no admission test	0.13	1.87	1.35	1.99	1.70	1.99
rate estimate instead of the window	0.42	2.16	1.38	**1.40**	**1.09**	2.16
Rules from prior work and standard practice						
rent-then-buy (ski-rental)	0.29	1.09	1.04	1.51	1.19	160
second-read retention (2Q)	0.34	1.02	1.02	44.0	7.44	745
cache at the write (always-KV)	0.13	2.55	1.46	194	29.0	3,191
never cache (always text)	0.47	**1.02**	**1.01**	1.94	1.25	1.94

selection, the next best scores 0.64 on the open cells and never caching scores 0. METER cannot see that the reads have stopped, so it holds one lease past the last read that the optimum never pays, and that is the $1.51\times$. Figure 9 separates the two ways a rule can disagree, caching what the optimum does not and missing what it keeps, across every residency. Replacing the window with a test on the item's cumulative history drops F_1 to 0.29. That rule caches 7.3% of LongMemEval items at every residency where the optimum caches none, and on a burst arriving after a long idle period it costs 47.39 times the optimum where METER costs 1.99 (Appendix C.5).

Cost separates the rules less sharply than admission does, and the reason is worth stating. Three rules share the same worst cell at 1.51: the one-hour residency on LoCoMo, where any rule admitting on a second read makes the same decision and no lease is long enough to lapse. A maximum over a grid can rank that cell rather than the method, so Table 3 reports the mean beside it, and on means no rule dominates. Never caching is cheapest on LongMemEval, at 1.01 against METER's 1.02, and estimating a rate instead of testing the window is cheapest on LoCoMo, at 1.09 against 1.21. Each is dearer on the corpus the other is cheap on, and each has a worse worst cell than METER, 1.94 and 2.16 against 1.51. The cumulative-admission ablation is closer still and matches that 1.51 exactly; what separates the two is not cost but which items they choose, at an admission F_1 of 0.29 against 1.00. Cost alone does not rank these rules, which is why the table reports admission beside it. Over the open cells, those where the optimum caches some but not all items, METER costs 1.02 and 1.21 times the optimum on the two corpora. On LoCoMo the same rule with no admission test costs 1.70, second-read retention without prices 7.44, and caching at the write 29.0.

Failing to price storage costs orders of magnitude, at whichever end of the axis the rule was not designed for. Over every cell, rent-then-buy reaches 160 times the optimum, second-read retention 745, and always-KV $3{,}191$, while never caching at all stays within 1.94. The priced rule is within 1.51 at both ends. Rent-then-buy is the classical online algorithm for rent-or-buy, and it is the closest prior rule to METER: it prices the purchase, unlike 2Q, and sees no future, unlike the offline reference. Its guarantee is a factor of two where the purchase is a one-off, and here storage accrues instead, so an item it buys is held to the end of its residency. It picks the same items as the cumulative ablation in 237 of the 240 cells and agrees on cost in only 73, which is the separation between selection and schedule that Table 3 reports in its two halves.

Two boundaries bound that claim. On LongMemEval the optimum caches nothing beyond a few days, and never caching costs 1.01 on average against METER's 1.02: there the controller earns its place by declining to cache rather than by caching well. And a maximum over a finite grid is not a competitive guarantee. Varying the derived length over a sixteenfold range moves the worst case from 1.51 to 1.95, with the derived value inside the flat minimum (Table 17), and transferring one device's calibration to another raises it from 1.51 to 3.05 over a separate 48-cell replay. Appendix C.9 prices the rate-estimator priors and lists every quantity in either rule that was chosen rather than derived, and Appendix C.1 states what each ablation isolates.

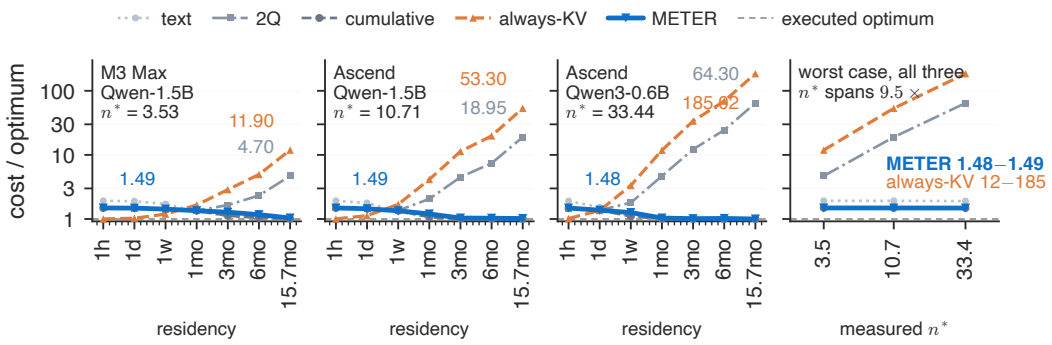


Figure 3: Single-tier execution on 80 LoCoMo items, 154 reads, and a 512-token budget, cost normalized to the offline reference executed in the same session. *cumulative* is the ablation that scores admission on the item's whole history (Table 15).

5.3 Single-tier hardware execution

Table 15 and Figure 3 report 21 executed runs: seven residencies on each of three reader-device pairs, 80 LoCoMo items and 154 reads apiece, the items sorted by key and the first 80 taken in trace order rather than sampled. Promotions execute prefills and count cache bytes, the store holds caches between reads, and every run is scored against an offline reference executed in the same session. One resident KV tier, resident hits charged no incremental compute, no tier migration and no end-to-end latency.

Selection carries over to execution as the derivation requires: in all 21 runs METER caches exactly the items the executed reference caches. What it cannot reproduce is the reference's timing. The reference builds at the read before a reuse because it knows the reuse is coming, while METER builds at the reuse, which is when the gap it prices becomes observable. Over the 21 runs that costs cache hits, 302 against 751, and it is the whole difference between them.

Price-blind rules fail here as they do in the model, and the scale follows the device. Caching at the write reaches 185.02 times the executed reference on Qwen3-0.6B at 15.7 months and second-read retention 64.30, while METER stays within 1.49 everywhere and never caching within 1.92. The rule that costs 11.90 on the M3 Max at that residency costs 53.30 on the Ascend for the same reader, whose threshold is four times higher.

That threshold is measured at the start of every run and it moves: on the M3 Max it lands between 2.13 and 3.53 reads per month over seven runs, the first timing a cold machine at 0.336 ms per prefilled token and the rest a warm one at 0.50 to 0.56, while the Ascend held at 0.11 to 0.12 ms on Qwen2.5-1.5B and 0.13 to 0.14 on Qwen3-0.6B. Cost is priced from tokens prefilled times the calibration measured up front rather than from summed wall clock, so the drift changes which threshold a run reads, not the arithmetic applied to it. On a thermally constrained device the threshold is a range, and a deployment should re-measure rather than carry one across.

5.4 Reuse and serving

Figure 4 compares corpus-derived reuse with tier-specific thresholds. The pooled memory rates are 0.146 and 0.544 reads per item-month for LongMemEval and LoCoMo, against a median 1,853 for the conversation trace's blocks over their one-hour observation window. Serving rates are computed on the blocks whose residency exceeds one median inter-arrival, 177,117 of that trace's 182,790, so the right-censored tail does not read as a low rate. The tier thresholds those rates are compared against span 6.7 reads per item-month, for object storage on the M3 Max, to 10,174 for a hosted cache on the Ascend. Four orders of magnitude of workload meet three of threshold on one axis, which is what a single derived price has to cover. These are benchmark-derived and trace-derived quantities, not estimates of all deployed agents.

Table 4 reports the replay, and it is where admission changes sides. A block is written and first read by the same request, so a test that waits for a second read cannot fire on the first one, and METER coincides exactly with second-read retention: 1.22 times the modeled optimum on the tool-agent

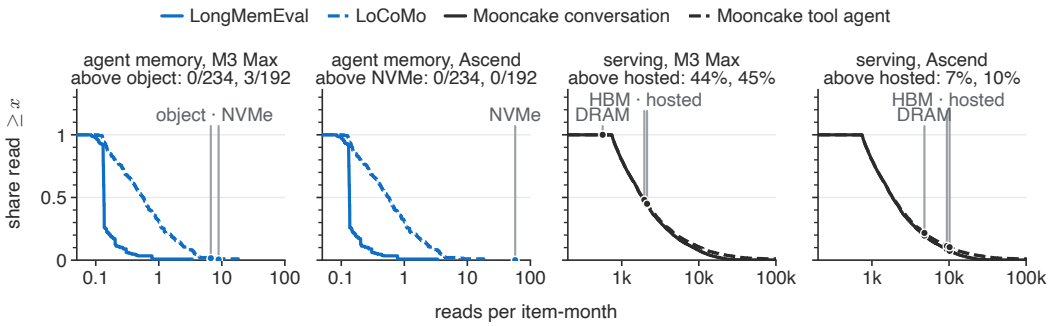


Figure 4: Reuse distributions and device-specific tier thresholds. Memory-benchmark dates define residency, while serving rates are extrapolated from one-hour traces. The tier prices and load assumptions are given in Table 1.

Table 4: Placement on one hour of Mooncake serving traffic, one 512-token prefix block per item, cost as a multiple of the offline optimum. *pays*: share of blocks the optimum caches. The two readers are the cheapest and dearest thresholds in the study.

Trace	Reader	*pays*	*text*	*always-KV*	*2Q*	METER	METER, no adm.
conversation	Qwen2.5-7B	24.2%	1.58	1.01	1.24	1.24	1.01
	SmolLM2-135M	24.2%	1.54	1.31	1.29	1.29	1.31
tool agent	Qwen2.5-7B	21.5%	2.23	1.01	1.22	1.22	1.01
	SmolLM2-135M	21.5%	2.20	1.32	1.27	1.27	1.32

trace with Qwen2.5-7B against 2.23 for never caching. Disabling the admission test, which is the configuration Theorem 1 covers, costs 1.01. Reuse this dense makes the evidence the window waits for redundant, and the prefill spent gathering it is the whole difference.

The price says where that reverses, and the reversal is in the same table. As n^* rises the advantage of caching eagerly shrinks and then inverts: on the dearest reader in the study, SmolLM2-135M, the admission test costs 1.29 against 1.31 for the same controller with admission disabled on the conversation trace, and 1.27 against 1.32 on the tool-agent trace. A deployment picks the configuration by comparing its measured reuse rate with its own n^*, the two quantities Figure 4 puts on one axis. These results do not establish throughput, latency, or physical HBM savings for a deployed serving engine.

5.5 Accuracy control

At a matched 512-slot budget, 21 of 23 paired reader-corpus intervals include zero (Table 2, Figure 7). The corresponding count is 44 of 47 over the budget sweep (Table 5) and 21 of 22 for the judged comparison; Appendix B.3 gives the cells outside this grid. These are not equivalence tests, and finite-precision storage and attention kernels can change answers. The external systems in Appendix B.1 compare write policies under a shared local writer; they are not substitutes for cache-placement baselines.

6 Conclusion

METER turns model footprint, storage prices, and measured read costs into one length, and uses it twice: to admit an item when two reads fall inside it, and to hold the cache for exactly that long afterwards. The window is the optimum's own per-gap test, so over 240 cells it selects the offline optimum's items exactly by construction, where scoring the same decision on an item's cumulative history is exact in only 47, and its schedule is 2-competitive per gap. Priced this way, one controller covers both ends of a reuse axis the field treats as separate problems: within 1.51 times the optimum on agent memory, where rules that price nothing cost up to 3,191 times, and, with the admission test disabled because reuse is dense, 1.01 on an hour of serving traffic where never caching costs 2.23. The price is also what says which of those two configurations a deployment should run. The bound covers the schedule and not the admission test, and the prices are contracts rather than measurements: a deployment should recompute the length from its own device rather than adopt ours.

USE OF LARGE LANGUAGE MODELS

We used generative AI tools to implement methods, give feedback on experimental design, interpret results, and draft and revise this manuscript. They were not used to generate synthetic data or to produce any measurement reported here. We have reviewed all AI-assisted work and take responsibility for the final content, including text, claims and artifacts.

REFERENCES

Chi-Chih Chang, Wei-Cheng Lin, Chien-Yu Lin, Chong-Yan Chen, Yu-Fang Hu, Pei-Shuo Wang, Ning-Chi Huang, Luis Ceze, Mohamed S. Abdelfattah, and Kai-Chiang Wu. Palu: KV-cache compression with low-rank projection. In *The Thirteenth International Conference on Learning Representations (ICLR 2025)*, 2025. URL https://openreview.net/forum?id=LWMS4pk2vK.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready AI agents with scalable long-term memory. In *Proceedings of the 28th European Conference on Artificial Intelligence (ECAI 2025)*, 2025. doi: 10.3233/FAIA251160. URL https://ebooks.iospress.nl/DOI/10.3233/FAIA251160.

Sabri Eyuboglu, Ryan Ehrlich, Simran Arora, Neel Guha, Dylan Zinsley, Emily Liu, Atri Rudra, James Zou, Azalia Mirhoseini, and Christopher Ré. Cartridges: Lightweight and general-purpose long context representations via self-study. In *The Fourteenth International Conference on Learning Representations (ICLR 2026)*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/4681359a7b1e94571598ad1adda35e6e-Abstract-Conference.html.

Jizhan Fang, Xinle Deng, Haoming Xu, Ziyan Jiang, Yuqi Tang, Ziwen Xu, Shumin Deng, Yunzhi Yao, Mengru Wang, Shuofei Qiao, Huajun Chen, and Ningyu Zhang. LightMem: Lightweight and efficient memory-augmented generation. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://arxiv.org/abs/2510.18866.

Rudolf Fleischer. On the Bahncard problem. *Theoretical Computer Science*, 268(1):161–174, 2001. URL https://doi.org/10.1016/S0304-3975(00)00266-8.

Bin Gao, Zhuomin He, Puru Sharma, Qingxuan Kang, Djordje Jevdjic, Junbo Deng, Xingkun Yang, Zhou Yu, and Pengfei Zuo. Cost-efficient large language model serving for multi-turn conversations with CachedAttention. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, 2024. URL https://www.usenix.org/conference/atc24/presentation/gao-bin-cost.

Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. HippoRAG: Neurobiologically inspired long-term memory for large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Momchil Hardalov, Gonzalo Iglesias, and Adrià de Gispert. Cartridges at scale: Training modular kv caches over large document collections. *arXiv preprint arXiv:2606.04557*, 2026. URL https://arxiv.org/abs/2606.04557.

Theodore Johnson and Dennis Shasha. 2Q: A low overhead high performance buffer management replacement algorithm. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB 1994)*, pp. 439–450, 1994. URL https://www.vldb.org/conf/1994/P439.PDF.

Anna R. Karlin, Mark S. Manasse, Larry Rudolph, and Daniel D. Sleator. Competitive snoopy caching. *Algorithmica*, 3(1):79–119, 1988. URL https://doi.org/10.1007/BF01762111.

Ravi Kumar, Todd Lipcon, Manish Purohit, and Tamas Sarlos. Linear elastic caching via ski rental. In *Conference on Innovative Data Systems Research (CIDR)*, 2025. URL https://vldb.org/cidrdb/papers/2025/p22-kumar.pdf.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*, 2023.

Bojie Li. Models take notes at prefill: Kv cache can be editable and composable. *arXiv preprint arXiv:2606.17107*, 2026. URL https://arxiv.org/abs/2606.17107.

Hanchen Li, Runyuan He, Qiuyang Mang, Qizheng Zhang, Huanzhi Mao, Xiaokun Chen, Hangrui Zhou, Huanchen Zhang, Alvin Cheung, Joseph E. Gonzalez, and Ion Stoica. Continuum: Efficient and robust multi-turn LLM agent scheduling with KV cache time-to-live. *arXiv preprint arXiv:2511.02230*, 2025. URL https://arxiv.org/abs/2511.02230.

Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, Michael Maire, Henry Hoffmann, Ari Holtzman, and Junchen Jiang. CacheGen: KV cache compression and streaming for fast large language model serving. In *Proceedings of the ACM SIGCOMM 2024 Conference*, 2024. URL https://doi.org/10.1145/3651890.3672274.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of LLM agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13851–13870. Association for Computational Linguistics, 2024.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023. URL https://arxiv.org/abs/2310.08560.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST)*, 2023.

Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Feng Ren, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: Trading more storage for less computation. a KVCache-centric architecture for serving LLM chatbot. In *23rd USENIX Conference on File and Storage Technologies (FAST)*, 2025.

Kun-Woo Shin, Jay H. Park, Moonwook Oh, Yohan Jo, Jaeyoung Do, and Sang-Won Lee. MatKV: Trading compute for flash storage in LLM inference. In *IEEE International Conference on Data Engineering (ICDE)*, 2026. URL https://arxiv.org/abs/2512.22195.

Yutao Sun, Li Dong, Yi Zhu, Shaohan Huang, Wenhui Wang, Shuming Ma, Quanlu Zhang, Jianyong Wang, and Furu Wei. You only cache once: Decoder-decoder architectures for language models. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024)*, 2024. doi: 10.52202/079017-0235. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/0df38cd13520747e1e64e5b123a78ef8-Abstract-Conference.html.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. Longmemeval: Benchmarking chat assistants on long-term interactive memory. In *The Thirteenth International Conference on Learning Representations (ICLR 2025)*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/d813d324dbf0598bbdc9c8e79740ed01-Abstract-Conference.html.

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 (ICLR 2024)*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/5e5fd18f863cbe6d8ae392a93fd271c9-Paper-Conference.pdf.

Zidi Xiong, Yuping Lin, Wenya Xie, Pengfei He, Zirui Liu, Jiliang Tang, Himabindu Lakkaraju, and Zhen Xiang. How memory management impacts LLM agents: An empirical study of experience-following behavior. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2026. URL https://aclanthology.org/2026.acl-long.27/.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. In *Advances in Neural Information Processing Systems 38 (NeurIPS 2025)*, 2025. doi: 10.52202/085713-0593. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/19909c36f51abc4856b4560aff3d36d6-Abstract-Conference.html.

Neal E. Young. On-line file caching. *Algorithmica*, 33(3):371–383, 2002.

Lingfan Yu, Jinkun Lin, and Jinyang Li. Stateful large language model serving with Pensieve. In *Proceedings of the Twentieth European Conference on Computer Systems (EuroSys '25)*, 2025. URL https://doi.org/10.1145/3689031.3696086.

Luoyuan Zhang. Can I buy your KV cache? *arXiv preprint arXiv:2606.13361*, 2026. URL https://arxiv.org/abs/2606.13361.

Tao Zhang, Ziqian Zeng, Hao Peng, Huiping Zhuang, and Cen Chen. MixKVQ: Query-aware mixed-precision KV cache quantization for long-context reasoning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2026. URL https://aclanthology.org/2026.acl-long.326/.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/6ceefa7b15572587b78ecfcebb2827f8-Abstract-Conference.html.

Hailiang Zhao, Xueyan Tang, Peng Chen, and Shuiguang Deng. Learning-augmented algorithms for the Bahncard problem. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024)*, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/cd96cb9a239c37b39dbf34f3f5a4c56f-Paper-Conference.pdf.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Yongjie Zhou, Shuai Wang, Bevan Koopman, and Guido Zuccon. Reproducing LightMem: Naive RAG is just as good for memory management. *arXiv preprint arXiv:2607.29104*, 2026. URL https://arxiv.org/abs/2607.29104.

Yu Zhu, Aditya Dhakal, Yunming Xiao, Dejan Milojicic, and Gustavo Alonso. ObjectCache: Layerwise object-storage retrieval for KV cache reuse. *arXiv preprint arXiv:2605.22850*, 2026. URL https://arxiv.org/abs/2605.22850.

Appendix

CONTENTS

A Architecture sweep and break-even thresholds **14**
A.1 Attention design sets the cost . . . 14
A.2 Cost grows with scale, text does not . . . 15

B Benchmark and write-policy detail **16**
B.1 Three published memory systems do not improve on raw retrieval . . . 16
B.2 Read-cost timing . . . 16
B.3 Coverage outside the reported grid, and the judge . . . 18
B.4 Scoring, retrieval and uncertainty . . . 18
B.5 Systems we could not run, and what storage covers . . . 19
B.6 LoCoMo scoreability . . . 19
B.7 Positional transfer conditions . . . 20
B.8 Read-cost measurements . . . 20
B.9 Write policy . . . 20

C Placement policy detail **21**
C.1 What METER's design is worth . . . 22
C.2 The policies . . . 22
C.3 The offline optimum is not the obvious one . . . 23
C.4 Proof of Theorem 1 . . . 23
C.5 Fixed lifetimes, calibration, and delayed bursts . . . 23
C.6 The executed placement run, cell by cell . . . 24
C.7 Every residency, on both corpora . . . 24
C.8 Question order in the LoCoMo placement trace . . . 25
C.9 Every quantity in either rule that was chosen rather than measured . . . 26
C.10 Limitations . . . 26

D What the literature reports **26**

E Derivations and detail from Section 3 **27**
E.1 Attention designs in closed form . . . 27
E.2 Residuals are not a cheaper carrier, and what quantization actually saves . . . 27
E.3 Robustness checks from Section 3 . . . 28
E.4 The tier price sheet, and a compressed carrier on the same axis . . . 29

A ARCHITECTURE SWEEP AND BREAK-EVEN THRESHOLDS

Table 6 lists the closed-form footprint of every architecture. Figure 5 compares its relationship with parameter count and attention geometry.

A.1 ATTENTION DESIGN SETS THE COST

Grouping the sweep by attention family separates the variance cleanly. Multi-query attention averages 26.0 KiB per token, multi-head latent attention 49.5, grouped-query attention 110.6, and multi-head attention 437.3. The span from cheapest to most expensive family is $16.8\times$.

Attention design is chosen for decode throughput and serving cost. The measurement here shows it simultaneously fixes retained-state cost across more than an order of magnitude, a consequence the memory and communication literatures do not report. Multi-head latent attention is the one design that treats cache size as an explicit target, and it shows: at comparable scale, DeepSeek-V3 holds 68.6 KiB per token against 320.0 for Qwen2.5-72B, a factor of 4.7.

Table 5: Budget sweep under the retrieval-validity gate, both corpora. Oracle EM uses gold-session evidence; band rows carry each model's no-memory floor and n^*. Bars: paired 95% bootstrap intervals for latent minus text EM on $[-0.10, +0.10]$, zero at the tick. Price: latent-to-text byte ratio.

	LongMemEval EM, 170 extractive				**LoCoMo** EM, 388 single-hop				
Budget	Oracle	Text	Latent	Latent − text	Oracle	Text	Latent	Latent − text	× text
Qwen2.5-1.5B | floor .094 / .059 | n^* 57.7 reads/mo									
256	0.535	0.576	0.565	−.012	0.662	0.629	0.621	−.008	6,730
512	0.606	0.582	0.576	−.006	0.716	0.649	0.652	.003	6,383
1,024	0.612	0.588	0.588	.000	0.704	0.634	0.631	−.003	6,260
2,048	0.629	0.618	0.618	.000	0.709	0.665	0.668	.003	6,171
4,096	0.618	0.612	0.612	.000			*pool fits*		6,128
8,192	0.629	0.588	0.576	−.012			*pool fits*		6,110
Qwen2.5-3B | floor .088 / .054 | n^* 37.1 reads/mo									
256	0.582	0.618	0.612	−.006	0.716	0.637	0.642	.005	8,653
512	0.641	0.582	0.600	.018	0.781	0.662	0.668	.005	8,207
1,024	0.629	0.571	0.582	.012	0.773	0.714	0.701	−.013	8,049
2,048	0.653	0.612	0.606	−.006	0.781	0.711	0.711	.000	7,935
4,096	0.535	0.594	0.606	.012			*pool fits*		7,879
8,192	0.576	0.635	0.635	.000			*pool fits*		7,856
Qwen2.5-7B | floor .112 / .088 | n^* 23.4 reads/mo									
256	0.500	0.506	0.500	−.006	0.665	0.601	0.603	.003	13,460
512	0.612	0.565	0.559	−.006	0.729	0.634	0.639	.005	12,766
1,024	0.647	0.594	0.588	−.006	0.745	0.683	0.683	.000	12,520
2,048	0.641	0.571	0.600	**.029**↑	0.742	0.709	0.704	−.005	12,343
4,096	0.624	0.606	0.600	−.006			*pool fits*		12,257
Qwen2.5-32B | floor .106 | n^* 24.8 reads/mo									
512	0.735	0.676	0.676	.000			*not run*		58,358
2,048	0.782	0.771	0.771	.000			*not run*		56,424
Other model families 512-token budget; n^* in Table 2									
SmolLM2-135M	0.153	0.153	0.124	−.029	0.129	0.144	0.080	**−.064**↓	5,390
Qwen3-0.6B	0.494	0.435	0.441	.006	0.433	0.389	0.399	.010	25,532
Gemma-3-1B	0.547	0.476	0.482	.006	0.603	0.528	0.523	−.005	6,202
Llama-3.2-1B	0.571	0.529	0.529	.000	0.649	0.554	0.559	.005	7,271
OLMo-2-1B	0.612	0.594	0.600	.006	0.688	0.613	0.616	.003	29,012
Falcon3-1B	0.541	0.541	0.547	.006	0.660	0.593	0.593	.000	17,383
Granite-3.1-2B	0.600	0.629	0.641	.012	0.760	0.696	0.686	−.010	21,091
Phi-3-mini-3.8B	0.671	0.676	0.671	−.006	0.740	0.668	0.691	**.023**↑	103,496

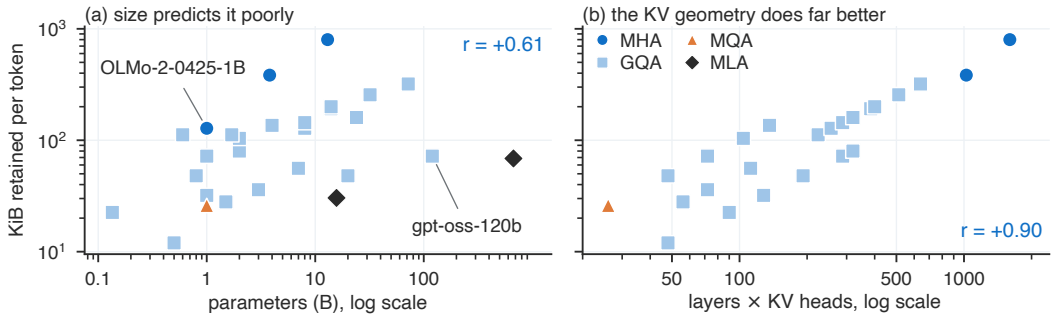


Figure 5: What sets the price of a retained cache, 30 architectures (Table 6). (a) Parameter count barely predicts it: OLMo-2-1B retains $1.8\times$ per token what gpt-oss-120b does. (b) Layers times KV heads predicts it far better; the two MLA models appear in (a) only.

Table 6: Bytes retained per token for 30 architectures, closed form from each configuration. L is layers, h_{kv} key-value heads. Ordered by cost: 12.0 KiB per token to 800.0, a span of $67\times$. No architecture comes within $3,413\times$ of text.

Model	*Attn*	L	h_{kv}	KiB/tok	$\times$ text
Qwen2.5-0.5B	*GQA/7*	24	2	12.0	3,413
SmolLM2-135M	*GQA/3*	30	3	22.5	6,400
gemma-3-1b	*MQA*	26	1	26.0	7,396
DeepSeek-R1-Distill-Qwen-1.5B	*GQA/6*	28	2	28.0	7,964
Qwen2.5-1.5B	*GQA/6*	28	2	28.0	7,964
DeepSeek-V2-Lite	*MLA*	27	512	30.4	8,640
Llama-3.2-1B	*GQA/4*	16	8	32.0	9,102
Qwen2.5-3B	*GQA/8*	36	2	36.0	10,240
Qwen3.5-0.8B-MLX	*GQA/4*	24	2	48.0	13,653
gpt-oss-20b	*GQA/8*	24	8	48.0	13,653
Qwen2.5-7B	*GQA/7*	28	4	56.0	15,929
DeepSeek-V3	*MLA*	61	512	68.6	19,520
gpt-oss-120b	*GQA/8*	36	8	72.0	20,480
Falcon3-1B	*GQA/2*	18	4	72.0	20,480
granite-3.1-2b	*GQA/4*	40	8	80.0	22,756
gemma-2-2b	*GQA/2*	26	4	104.0	29,582
Qwen3-1.7B	*GQA/2*	28	8	112.0	31,858
Qwen3-0.6B	*GQA/2*	28	8	112.0	31,858
DeepSeek-R1-Distill-Llama-8B	*GQA/4*	32	8	128.0	36,409
OLMo-2-0425-1B	*MHA*	16	16	128.0	36,409
gemma-3-4b	*GQA/2*	34	4	136.0	38,684
Qwen3-8B	*GQA/4*	36	8	144.0	40,960
Mistral-Small-24B	*GQA/4*	40	8	160.0	45,511
Qwen2.5-14B	*GQA/5*	48	8	192.0	54,613
phi-4	*GQA/4*	40	10	200.0	56,889
Qwen2.5-32B	*GQA/5*	64	8	256.0	72,818
Qwen3-32B	*GQA/8*	64	8	256.0	72,818
Qwen2.5-72B	*GQA/8*	80	8	320.0	91,022
Phi-3-mini-4k	*MHA*	32	32	384.0	109,227
OLMo-2-1124-13B	*MHA*	40	40	800.0	227,556

A.2 Cost grows with scale, text does not

Figure 2(d) holds the training recipe fixed and varies only size within the Qwen2.5 family: 12.0, 36.0, 56.0, 192.0, 256.0 and 320.0 KiB per token at 0.5B, 3B, 7B, 14B, 32B and 72B parameters. The cost rises $26.7\times$ while the same content as text stays at 3.6 bytes per token at every size.

Because the comparison is inside one family, the trend cannot be attributed to differing labs or recipes. The cost of retaining state rises with the capability of the model retaining it, and the text baseline does not move.

B BENCHMARK AND WRITE-POLICY DETAIL

Published systems. We compare against Mem0 (Chhikara et al., 2025), LightMem (Fang et al., 2026) and A-Mem (Xu et al., 2025), run from their own repositories rather than reimplemented. Each performs its own write policy and ranking over the same memory pool; we then read its top-ranked memories at the same budget, with the same reader, prompt and scorer as every other condition, so the systems supply memory rather than answers. All three run the full 170. Each assumes a hosted writer, which we replace with Qwen2.5-7B-Instruct served locally, so these results characterise them under a 7B open-weight writer rather than the model their authors used.

Uncertainty and hardware. For a question set Q we report the paired difference $\Delta = \mathrm{EM}(m_{\mathrm{lat}}) - \mathrm{EM}(m_{\mathrm{txt}})$ on the same Q, with 95% intervals from 10,000 bootstrap resamples drawn with replacement. Every accuracy number was produced on one Huawei Ascend 910B4 at bf16, and the device is not incidental: the identical script, data, budget and seed on an Apple M3 Max leaves the assembled contexts byte-identical yet moves exact match by up to 0.047, further than any latent-minus-text effect in Table 2. The paired difference moves with the device as well. These observations do not prove device-invariant accuracy equivalence. Every comparison is made within a single device and a single run.

B.1 THREE PUBLISHED MEMORY SYSTEMS DO NOT IMPROVE ON RAW RETRIEVAL

Figure 6 sets two published systems against simply pasting the top-ranked turns. At a matched 512-slot budget neither improves on it: Mem0 trails by 0.047 $[-0.124, 0.029]$ and LightMem by 0.100 $[-0.182, -0.018]$. Both intervals run about ± 8 points and both systems assume a hosted writer we replace with a 7B open-weight one, so this is a statement about three systems under one writer at one budget and not about memory systems in general. Only LightMem's interval excludes zero, so the defensible claim is that neither improves on raw retrieval, not that either is beaten. An independent reproduction reaches the same conclusion for LightMem against naive retrieval (Zhou et al., 2026). What separates them is what their write policy left inside the budget, which exact match follows at $r = 0.86$ across the twenty cells, and each system is weakest nearest the thing its design chose not to do. Capacity is not the account, and neither is rewriting as such. Our own summariser loses 0.224, 0.135 and 0.059 at a 1.5B, 3B and 7B writer given the same budget as text, the last containing zero, so a capable writer with room to work is close to free; asked for notes at half the budget, which is what a deployed summariser banks, it loses 0.212 to 0.118 and stays significant at every scale. The published systems sit below both on that same 7B writer, so the cost is in the write policy. Reading any system's memories back as state rather than as tokens moves exact match by at most 0.012 under this protocol; on A-Mem's memories it costs about $6{,}432\times$ the bytes. This is not an equivalence test. Appendix B.9 gives the anatomy cut by cut and Table 7 every cell.

Mem0 assumes GPT-4.1-nano and LightMem GPT-4.1-mini in its own LongMemEval configuration. Writer capacity is not incidental: on a probe question a 1.5B writer caused Mem0 to store nine memories, none of them about the question, which is why we standardise on a 7B writer rather than a smaller one.

B.2 READ-COST TIMING

Both carriers end with the same cache resident on device, so the read-cost comparison is between computing it and loading it. We time each on an Apple M3 Max and a Huawei Ascend 910B4, running Qwen2.5-3B through one harness at 128, 512, 2,048 and 8,192 tokens. Every figure is the median of five repetitions, and loads are measured warm, which bounds the load path optimistically.

Tables 7a, 7b, and 7 report the external-system and summarizer comparisons at the matched read budget.

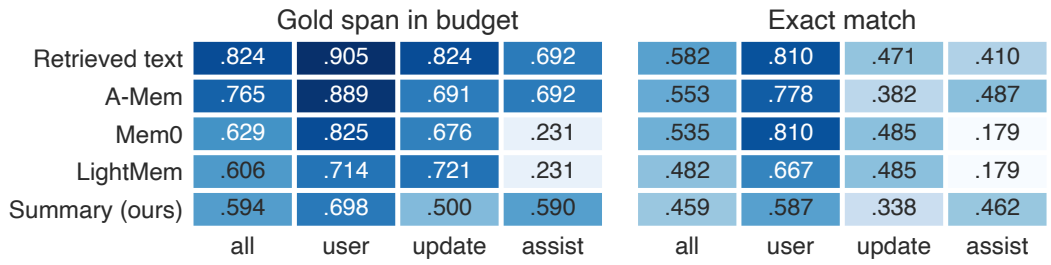


Figure 6: Write policy decides what survives, and accuracy follows. Five memory sources at a matched 512-slot budget: all 170 questions, then the user, knowledge-update and assistant cuts. Mem0 and LightMem model the user only.

Table 7: Write-policy evidence at a matched 512-slot budget on 170 LongMemEval questions. Point estimates compare retrieved memory sources; paired intervals compare summary writers under matched and half budgets.

Memory source	Span	Text	KV	KV − text
Retrieved text	**0.824**	**0.582**	0.576	−0.006
A-Mem	0.765	0.553	0.541	−0.012
Mem0	0.629	0.535	0.529	−0.006
LightMem	0.606	0.482	0.476	−0.006
Summary (ours)	0.594	0.459	0.459	0.000

(a) Who writes the memory. Every row is the same 170 questions at the same 512-slot budget. No published system reaches raw retrieval, and reading any of them back as KV state rather than text moves accuracy by at most 0.012 while costing $6,225$ to $10,613\times$ the source text bytes.

Notes	Writer	Text	Summary	Summary − text	Span	Lost
matched	1.5B	0.582	0.359	**−0.224** [−0.300, −0.147]	0.476	42%
	3B	0.582	0.447	**−0.135** [−0.212, −0.059]	0.588	29%
	7B	0.565	0.506	−0.059 [−0.124, 0.006]	0.529	36%
half	1.5B	0.582	0.371	**−0.212** [−0.294, −0.129]	0.459	44%
	3B	0.582	0.412	**−0.171** [−0.253, −0.088]	0.547	34%
	7B	0.565	0.447	**−0.118** [−0.182, −0.053]	0.500	39%

(b) The summariser, under both budget accountings. *matched* gives condition D the same read budget as text, which is this paper's invariant; *half* is the harness default, which asks for notes of budget/2 and is what a deployed summariser banks. Rewriting costs accuracy at 1.5B and 3B under both. At 7B it is significant only when the notes are allowed to be shorter.

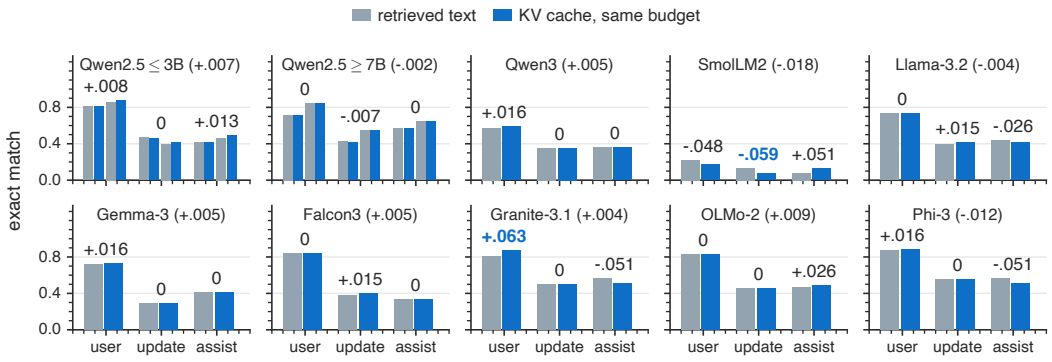


Figure 7: Paired latent-minus-text exact match at a matched 512-token budget, one panel per model family. Bars are the two carriers at the same budget and the label above each pair is their difference, bold where its 95% bootstrap interval excludes zero: two of the 36 reader-by-question-type comparisons. Intervals containing zero do not establish equivalence.

Table 8: Coverage outside the reported grid: latent minus text EM at a matched budget; CI is a 95% bootstrap interval. All 12 breadth-sweep intervals contain zero. Columns are the read budget in tokens.

	256		512		1,024		2,048	
Reader	Δ	CI	Δ	CI	Δ	CI	Δ	CI
Gemma-3-1B	−.006	[−.018, .000]	.006	[.000, .018]	.006	[.000, .018]	.012	[.000, .029]
OLMo-2-1B	.006	[.000, .018]	.006	[.000, .018]	.000	[−.018, .018]	.012	[.000, .029]
Phi-3-mini-3.8B	.006	[−.012, .029]	−.006	[−.029, .018]	−.012	[−.047, .018]	−.029	[−.071, .012]
LongMemEval computed types under the judge, 256 questions; the gate fails on both								
pooled			−.012	[−.027, .000]			−.004	[−.023, .016]

Table 9: Judge calibration on the extractive set. Agreement ranges from 0.865 to 0.890 over 5 runs; paired latent-minus-text intervals contain zero under both scorers.

Reader	agree	exact match	judge
1.5B at 512	0.865	−0.006 [−0.018, +0.000]	−0.006 [−0.035, +0.024]
32B at 2048	0.869	+0.000 [−0.018, +0.018]	−0.012 [−0.029, +0.000]
32B at 512	0.879	+0.000 [+0.000, +0.000]	+0.006 [+0.000, +0.018]
3B at 512	0.890	+0.018 [−0.006, +0.047]	−0.006 [−0.029, +0.018]
7B at 512	0.882	−0.006 [−0.029, +0.012]	−0.012 [−0.035, +0.012]

B.3 Coverage outside the reported grid, and the judge

Table 8 gives the cells Section 5.5 summarises but does not tabulate: the three breadth readers run at every budget rather than only at 512, and the computed question types at 2,048. All twelve paired intervals contain zero, the widest -0.029 $[-0.071, +0.012]$. Across the reported grid, three intervals exclude zero and point in different directions: latent leads by 0.029 on Qwen2.5-7B at 2,048 tokens and by 0.023 for Phi-3-mini on LoCoMo, and trails by 0.064 for SmolLM2-135M.

The judge is Qwen2.5-7B-Instruct served locally and decoded greedily. It is blind: it sees the question, the reference answer and the candidate, never which condition produced the candidate, so it cannot prefer a carrier. Table 9 calibrates the judge against the scorer it replaces. On the extractive set, where substring exact match is valid, the two agree on 0.865 to 0.890 of decisions over five runs, and the paired latent-minus-text difference contains zero under both. A judge that could not recover a known result would not be evidence where exact match cannot reach. Across this table and the three others that carry intervals, 54 of the 55 paired differences contain zero; the exception is SmolLM2-135M on multi-hop, the smallest reader in the study.

B.4 Scoring, retrieval and uncertainty

Retrieval is lexical on our side while the systems we compare against rank with dense embedders, so that contrast is write policy and retriever together rather than write policy alone; Appendix E shows the carrier conclusion is unchanged at 1.5B, 3B and 7B under a dense encoder, which is also the weaker retriever on this task.

We report LongMemEval's three extractive types because the other two cannot discriminate carriers at any budget we can afford: run at 512 and at 2,048 slots they clear the oracle-to-floor gate by 0.070 and 0.055 against the 0.15 it requires, since the gold span is absent from the retrievable pool for 59% of those questions and retrieval saturates at 0.410 reach by 16,384 tokens. Section 5.5 therefore extends the claim on LoCoMo's non-extractive questions, where the gate does pass. Substring exact match also charges the summariser for answers it reformatted rather than lost: of the questions text answers and the summary does not, 0%, 5% and 15% at the 1.5B, 3B and 7B writers are the gold string in another surface form, so the rewriting cost we report is an upper bound that tightens as the writer grows.

Substring exact match presumes the answer is a span of the transcript. That presumption holds for the three extractive question types and fails for multi-session and temporal-reasoning items, whose answers are computed from the dialogue rather than quoted from it, so scoring those by string match

Table 10: LoCoMo's computed question types, scored by an LLM judge because exact match cannot score them. 21 of 22 paired intervals contain zero. Same 512-slot budget, Ascend 910B4 at bf16.

	Multi-hop ($n = 275$)			**Temporal** ($n = 320$)		
Reader	Text	Δ latent	95% CI	Text	Δ latent	95% CI
Qwen2.5-7B	0.291	−.004	[−.018, .011]	0.156	.003	[−.009, .016]
Phi-3-mini-4k	0.255	.000	[−.022, .022]	0.287	−.003	[−.025, .022]
Qwen2.5-1.5B	0.251	.004	[−.011, .022]	0.144	.006	[.000, .016]
Qwen2.5-3B	0.244	−.007	[−.022, .007]	0.209	.006	[.000, .016]
granite-3.1-2b	0.244	.004	[−.018, .025]	0.247	.003	[−.022, .028]
Llama-3.2-1B	0.215	−.011	[−.025, .000]	0.175	.003	[−.006, .016]
OLMo-2-1B	0.153	.000	[−.011, .011]	0.172	−.006	[−.016, .000]
Falcon3-1B	0.149	.000	[.000, .000]	0.163	.000	[.000, .000]
Qwen3-0.6B	0.149	.000	[−.022, .022]	0.188	−.006	[−.025, .013]
gemma-3-1b	0.109	.004	[−.007, .018]	0.103	.003	[−.006, .016]
SmolLM2-135M	0.055	**−.025**	[−.047, −.007]	0.072	−.013	[−.037, .009]

would measure arithmetic rather than memory. Retrieval is lexical only, and its granularity is not incidental: the pool holds three or four sessions of roughly 13,000 tokens, so ranking whole sessions returns the entire pool and truncation then keeps a prefix instead of the evidence. Replicating a cell under three seeds returned identical numbers, because 170 is the whole subset rather than a sample drawn from it and greedy decoding makes evaluation order irrelevant. At that size the intervals are about seven exact-match points wide, and we report a smaller difference as an absence of a detectable difference rather than as a decline.

The gold string appears verbatim in the gold session for 93.7% of single-session-user questions, 91.2% of knowledge-update and 84.6% of single-session-assistant, but for only 54.1% of multi-session and 52.4% of temporal-reasoning items.

Summary notes are capped at half the read budget, and we re-run the condition with the cap lifted to the full budget to test whether the cap is what costs it. Gold-session retrieval is a reference condition rather than an upper bound: it is confined to one session at the same budget, so a retriever ranging over the whole pool can outscore it, and Table 8 reports the cells where it is.

B.5 Systems we could not run, and what storage covers

Cartridges (Eyuboglu et al., 2026) and MemGPT (Packer et al., 2023) are absent for reasons that are not incidental: the first needs a CUDA-only server we do not have, and the second's mechanism is the self-directed paging our protocol fixes, so a budget match would remove the thing being tested. Storage covers fp16 caches under Equations 3 and 8 and models no sparsity, eviction or offloading (Zhang et al., 2023; Xiao et al., 2024); serving systems that do model those (Liu et al., 2024; Gao et al., 2024) operate at a residency where our thresholds do not bind, as Appendix E.4 sets out.

B.6 LoCoMo scoreability

LoCoMo is multimodal, and 1,226 of its 5,882 turns carry an image caption that the text-only protocol restores; restoring it lifts single-hop verbatim-in-gold from 83.2% to 85.6%. At a matched budget the same allowance buys roughly fourteen LoCoMo turns against four LongMemEval ones. The two corpora are not on a common scale, because a LoCoMo turn has a median of 136 characters against 449, so we report them separately.

Verbatim-in-gold rates across all four LoCoMo categories, with no answer-length cap, are 57.9% for single-hop, 20.3% multi-hop, 9.0% open-domain and 8.8% temporal.

Multi-hop and temporal answers are computed rather than quoted, so substring matching cannot score them and we rerun both under an LLM judge. Table 10 gives every reader separately: 21 of 22 paired intervals contain zero.

Table 11: Descriptive positional check on the four readers beyond the Qwen2.5-3B probe of Appendix B.7. Each cell is a splice's mean gold log-probability minus its reader's oracle; the largest observed deviation is 0.079 nats. Splices use offset 64, except Gemma-3-1B at offset 0.

	oracle-to-floor	*splice minus its own oracle, nats*		
Reader	*gap, nats*	*inject at 0*	*offset, no repair*	*offset, RoPE repaired*
Llama-3.2-1B	7.49	-0.019	+0.018	-0.019
Qwen3-0.6B	7.54	+0.022	+0.079	+0.022
SmolLM2-135M	8.31	+0.000	+0.021	-0.001
gemma-3-1b	10.66	+0.000	*n/a*	*n/a*

B.7 Positional transfer conditions

To test whether latent state survives being read at a position other than the one it was written at, we compare an oracle, a no-passage floor, and three re-injection conditions on a passage of six invented facts the model cannot guess. Condition A places passage and question in one context, giving the oracle; condition B presents the question alone, giving the floor; conditions C, D and E capture the KV state of the passage in isolation, discard the passage, and re-inject the state at offset 0, at offset k without correction, and at offset k with a rotary re-rotation of the cached keys by k. The probe is informative only if the facts are unrecoverable without the passage, so we require the oracle-to-floor gap to exceed 1.5 nats and abort otherwise; on Qwen2.5-3B it is 7.44 nats, oracle -0.004 against floor -7.441 in mean log-probability of the gold continuation, and each condition c is reported as the fraction of that gap it recovers,

$$r(c) = \frac{\ell_c - \ell_\emptyset}{\ell^* - \ell_\emptyset},$$

where ℓ is the mean log-probability the reader assigns to the answer tokens, ℓ^* with the passage in context and $\ell_\emptyset$ without it.

The probe runs on Qwen2.5-3B, Qwen3-0.6B, SmolLM2-135M, Llama-3.2-1B and Gemma-3-1B, covering grouped-query attention at three group sizes and multi-query attention, so the result is not read off one training recipe. On Gemma-3-1B, whose sliding-window and full-attention layers carry separate rotary bases, re-injection at offset 0 is exact but a shifted splice does not run without rebuilding the hybrid mask, so we make no claim about interleaved sliding-window models beyond offset 0.

Table 11 repeats the probe across families. Every model under global attention recovers the full oracle from an uncorrected splice at offset 64, on gaps from 7.49 to 8.31 nats, so the validity guard clears by five times its threshold in every case. Gemma-3-1B clears the largest gap in the set, 10.66 nats, but only at offset 0.

B.8 Read-cost measurements

Each timing brackets one operation with an explicit device synchronisation and frees its tensors between repetitions. Prefill is a single forward pass with `use_cache=True`; the load path deserialises per-layer key and value tensors from disk and copies them to the device. The cache occupies 4.7, 18.9, 75.5 and 302.0 MB at the four lengths, which is Qwen2.5-3B's 36.0 KiB per token (Table 6) at each. Per length, restoring the state beats re-encoding by $4.3\times$, $12.0\times$, $34.0\times$ and $91.4\times$ on the M3 Max at 128, 512, 2,048 and 8,192 tokens, against $2.8\times$, $2.1\times$, $1.6\times$ and $2.3\times$ on the Ascend, the disagreement between devices that Section 3 reports as $26.3\times$ at the charged threshold. The load enters n^* through R_{lat}; Table 12 reports the threshold with it charged and with it free. The M3 Max runs fp16 and the Ascend bf16, since fp16 overflows on this Ascend family; both are two bytes, so the byte counts and n^* are unaffected.

B.9 Write policy

Our own summariser, written to the same interface as the published systems, is the weakest of the four at 0.459 and preserves 59.4% of spans. We report it as a lower bound on what rewriting costs

Table 12: Prefill and state-load milliseconds for Qwen2.5-3B on both chips. The break-even rate charges the load at the prefill rate; the load-free column is retained as a diagnostic. At 2,048 tokens, charging load changes the cross-machine threshold ratio from 10.6 to 26.3×.

	M3 Max				**Ascend 910B4**			
			n^* (reads/mo)				n^* (reads/mo)	
Tokens	prefill	load	*load free*	*charged*	prefill	load	*load free*	*charged*
128	110.7	26.1	1.8	2.3	64.4	23.1	3.0	4.7
512	316.2	26.4	2.5	2.7	66.2	32.0	11.8	22.9
2,048	1,203.8	35.4	**2.6**	**2.7**	113.5	69.1	**27.6**	**70.4**
8,192	6,233.7	68.2	2.0	2.0	473.4	209.5	26.4	47.4

Table 13: Cost as a multiple of the offline optimum over twelve readers and ten residencies. Every cell is the mean over the twelve readers; the last row is the worst single open cell, which is a different population and so need not appear above it. *pays*: percentage of items the optimum caches; *cum.*: admission scored on the item's whole history. Read the permuted LoCoMo block (Appendix C.8).

	LoCoMo order permuted						**LongMemEval** real dates				**LoCoMo** file order			
Residency	*pays*	*text*	*KV*	*2Q*	*cum.*	METER	*pays*	*KV*	*2Q*	METER	*pays*	*KV*	*2Q*	METER
1 h	48.1	1.86	1.16	1.49	1.49	1.49	1.7	1.10	1.02	1.02	48.1	1.23	1.57	1.57
6 h	44.2	1.59	1.76	1.45	1.40	1.42	1.5	1.59	1.02	1.01	48.1	2.35	1.85	1.73
1 d	33.6	1.30	3.81	1.74	1.26	1.31	0.9	3.35	1.02	1.01	48.1	6.22	2.83	1.87
3 d	20.7	1.11	9.27	2.91	1.14	1.20	0.4	8.07	1.04	1.00	48.1	15.8	5.25	1.90
1 wk	13.0	1.06	20.4	5.43	1.06	1.13	0.0	17.5	1.09	1.00	48.1	32.6	9.49	1.86
2 wk	6.4	1.02	39.7	10.0	1.03	1.06	0.0	33.9	1.18	1.00	48.0	56.7	15.6	1.77
1 mo	3.5	1.01	84.6	19.1	1.01	1.03	0.0	72.2	1.38	1.00	40.0	101	26.8	1.59
3 mo	1.3	1.00	247	55.0	1.00	1.01	0.0	210	2.12	1.00	31.9	261	67.8	1.36
6 mo	0.4	1.00	477	113	1.00	1.00	0.0	409	3.18	1.00	15.9	486	125	1.17
15.7 mo	0.2	1.00	1,152	269	1.00	1.00	0.0	994	6.31	1.00	7.9	1,153	297	1.05
worst		**1.94**	**194**	**44.0**	**1.51**	**1.51**		**2.55**	**1.02**	**1.02**		**196**	**50.8**	**1.94**

rather than as a stand-in for either published system, which it is not: both real systems do better than it does.

The per-scale intervals for the summariser are $[0.129, 0.294]$ at 1.5B, $[0.088, 0.253]$ at 3B and $[0.053, 0.182]$ at 7B. Of the 82.4% of questions whose gold span retrieval places inside the budget, the span survives the rewrite for 55.7%, 66.4% and 60.7% at the three scales, so between 34% and 44% of the located evidence is destroyed by the act of summarising it. The 3B preserves the most and the 1.5B the least, with the 7B between them.

Budget is the obvious objection: the notes occupy 267 tokens against 537 for text. We re-ran all three writers with the cap lifted. The loss becomes 0.224 with interval $[0.147, 0.300]$ at 1.5B, 0.135 with $[0.059, 0.212]$ at 3B and 0.059 with $[-0.006, 0.124]$ at 7B. More room did not help the smallest writer, which locates its loss in the rewriting rather than in the space, and the interval at 7B contains zero: with a capable writer and a full budget we cannot resolve a loss at this sample size. That cell separates two explanations the aggregate confounds. Were rewriting harmful only because small models write poor notes, a 7B writer would repair the deployed systems too. It does not.

Asked the user's previous occupation (`lme_5d3d2817`, gold "Marketing specialist at a small startup"), the reader answers correctly from retrieved text and "managing a team of interns at a startup" from the 7B summary of the same turns.

C Placement policy detail

Table 13 prices every rule over twelve readers and ten residencies, and Figure 8 gives the boundary reader by reader: the share of items whose total reads clear the write-time rule, walking left as n^* rises, which is the hardware result of Section 3.3 on a second axis.

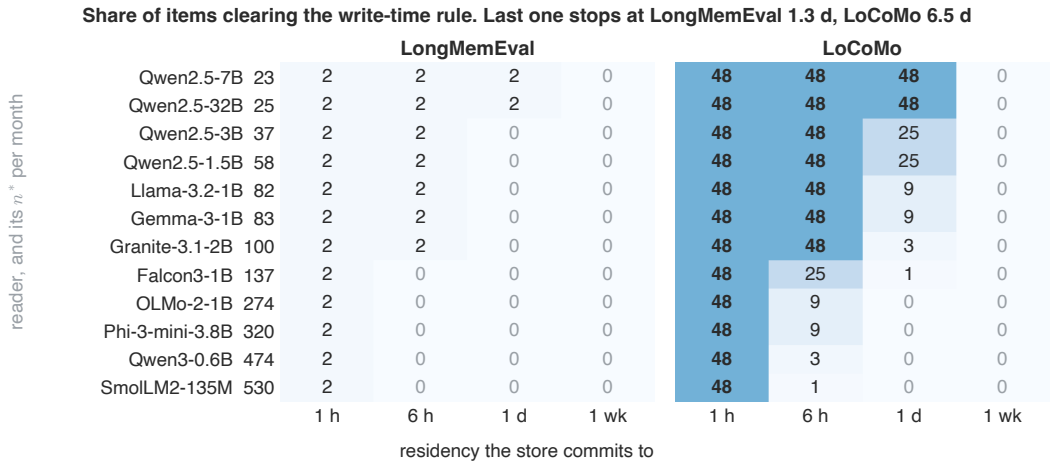


Figure 8: Where the boundary falls, reader by reader: share of items whose reads clear the write-time rule $n^*T + 1$, by committed residency, twelve readers ordered by n^*. A zero is a measurement; residencies past one week are all zero and omitted.

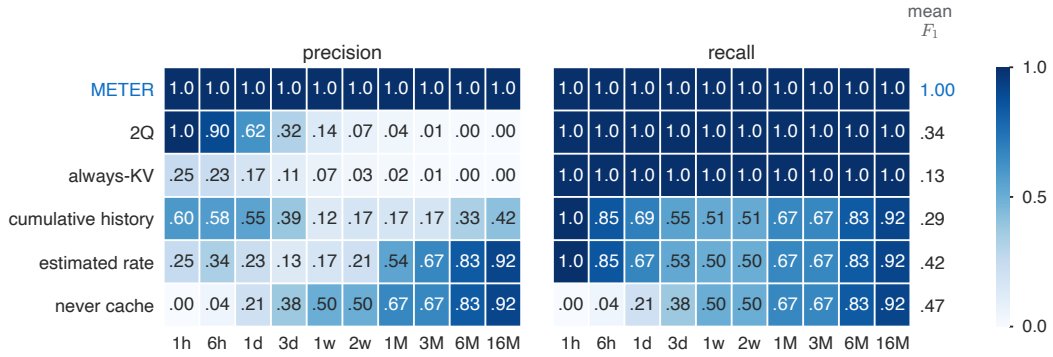


Figure 9: METER caches exactly the offline optimum's items in every cell; no other rule does. Precision and recall of each rule's cached set against the optimum's, averaged over 240 reader-by-residency cells; columns are committed residency, the right margin is mean F_1.

C.1 What METER's design is worth

Table 3 has two halves under a shared cost model. The ablations remove one component at a time, isolating one cost each: holding after every read, admitting without the window, and scoring admission on cumulative history rather than inside it. The rules below them are not ablations: rent-then-buy is the classical rent-or-buy algorithm, 2Q a replacement heuristic, and always-KV and never-cache the two standing assumptions of serving stores and agent memory systems. Appendix C.5 adds fixed lifetimes, threshold perturbations, and late-burst cases.

C.2 The policies

Always-text prefills on every read. Always-KV builds at write time and retains the cache. Second-read retention builds on the second actual read and never expires; it is inspired by multi-hit admission, not an implementation of full 2Q replacement (Johnson & Shasha, 1994). METER applies Algorithm 1: two actual reads within $1/n^*$ admit the item, and the lease of that length renews at every read. A write timestamp is not a read, which matters because an implementation that treats write-to-first-read time as an inter-read gap admits items that were read once. A comparator that does so is labelled legacy lease wherever it appears.

The cumulative-admission ablation scores the same decision over the item's whole life, admitting at the j-th read e months after the write when $jB \geq B + \min_t(\sigma_t e + jR_t)$. No-admission schedules a lease after every read. Admission-only retains admitted items indefinitely. The rate ablations estimate

reuse under their recorded priors, with details in Table 17. All simulated policies integrate holding cost over time and charge each text-served read. Cache creation at an admitting read reuses that read's prefill rather than charging it twice. The single-tier replay charges resident hits zero; loaded tiers have explicit R_t. The online rules use only observed reads. Rate-prior ablations additionally assume known item residency; the write-time oracle and dynamic program use future reads.

C.3 The offline optimum is not the obvious one

The bound is the cheapest placement available when every read time is known in advance, and it is not "cache the items that clear Equation 4 at the write". Promoting at a read costs no prefill that read was not already paying, so an oracle restricted to promoting at the write is beaten by one allowed to wait, by up to 24% on LoCoMo. We compute it per item by dynamic programming over the read times, with two states per interval and a free transition at every read. The exhaustive check uses items with at most eight reads from the original two corpus orders, three horizons (one hour, one month, and the full span), and every fifth reader in the configuration list. The dynamic program and enumerated placement costs agree on that tested subset. This is not exhaustive validation of every shuffled cell. Separate regression tests check reads at expiry.

C.4 Proof of Theorem 1

Fix one item and one gap of length g between consecutive reads. Index the states by t, with $t = 0$ the resident tier and the last state text, each with storage rate $\sigma_t \geq 0$ per token-month and a cost $R_t \geq 0$ paid at the next read (the load, or for text the prefill B). Holding state t across the whole gap costs $\sigma_t g + R_t$, and allowed downward moves have nonincreasing holding rates. Holding the eventual destination throughout the gap therefore incurs no greater storage cost and the same terminal load, so it dominates such a sequence when transitions are free. The offline optimum on the gap is therefore the lower envelope $f(g) = \min_t(\sigma_t g + R_t)$, a concave piecewise-linear function with $f(0) = \min_t R_t \geq 0$ and slopes that are nonincreasing in e.

The cascade holds, at elapsed time e, the state $s(e)$ that attains $f(e)$, so its instantaneous storage rate is $f'(e)$ wherever f is differentiable and the storage it pays over the gap is $\int_0^g f'(e)\,de = f(g) - f(0)$. At the read it pays $R_{s(g)}$, and $R_{s(g)} \leq \sigma_{s(g)} g + R_{s(g)} = f(g)$. Its total on the gap is at most $f(g) - f(0) + f(g) \leq 2f(g)$, which is the claim. The bound is tight for two states: with a resident tier and text, the cascade drops at $e = \ell = (B - R_0)/\sigma_0$, and a read arriving just after pays $\sigma_0 \ell + B = 2B - R_0$ against an optimum of B.

For the per-item corollary, both policies start from text and charge the first required prefill. Building at write time cannot avoid that build cost and adds holding cost. After the last read, the no-admission cascade pays at most B before it expires. Summing the gap bounds therefore gives $\text{cost} \leq 2\,\text{OPT} + B$ under these initial-state and free-movement assumptions. This corollary does not extend to admission: a declined short gap costs B even when the no-admission cascade costs much less. The delayed-burst diagnostic below gives a concrete counterexample.

C.5 Fixed lifetimes, calibration, and delayed bursts

Table 14 summarizes matched CPU comparisons on the same 240 benchmark cells, 48 cross-device cells, and 27 constructed bursts. Every policy in a cell sees the same true prices, reads, horizon, and exact offline optimum; a controller's estimate can change what it admits and how long it holds, never what it is charged.

Fixed renewable lifetimes of five minutes, one hour, and one day reach maximum ratios 1.86, 1.67, and 16.66. METER reaches 1.51 at a mean of 1.08, and the cumulative-admission variant the same maximum at a mean of 1.10, cheaper than METER in 73 cells, tied in 47, and dearer in 120. The shared maximum therefore does not mean shared behaviour.

Multiplying the derived threshold by 0.25, 0.5, 1, 2, and 4 gives maxima 1.95, 1.51, 1.51, 1.51, and 1.55. The derived value sits inside that flat region rather than at a tuned optimum, and doubling it improves the mean slightly, so the derivation is an interpretable input and not the empirically best setting for every workload. Calibration does have to be local: transferring the lowest-threshold

Table 14: Counterfactual validation on analytic cells, v2 economics and exact offline DP. On the worst delayed-burst cell (50 reads after 100 break-even periods) cumulative admission costs $47.39\times$ the optimum, METER $1.99\times$, no admission $1.05\times$. The last three columns count cells cheaper than, tied with, or dearer than METER.

Policy or calibration	max cost/OPT	mean cost/OPT	cheaper	tied	dearer
Matched benchmark grid, 240 cells					
METER	1.51	1.08	0	240	0
cumulative admission	1.51	1.10	73	47	120
METER, no admission	1.99	1.78	26	0	214
TTL 5 min	1.86	1.11	55	0	185
TTL 1 h	1.67	1.24	37	0	203
TTL 1 d	16.66	4.66	26	0	214
Calibration perturbations, 240 cells					
$n^* \times 0.25$	1.95	1.18	3	131	106
$n^* \times 0.5$	1.51	1.11	3	152	85
$n^* \times 1$	1.51	1.08	0	240	0
$n^* \times 2$	1.51	1.07	81	150	9
$n^* \times 4$	1.55	1.07	86	142	12
METER, transferred n^*	5.78	1.32	3	131	106
cross-device transfer (48 cells)	3.05	1.26	9	26	13
Synthetic delayed bursts, 27 cells					
METER	2.10	2.06	0	27	0
cumulative admission	47.39	9.87	6	9	12
METER, no admission	1.10	1.08	27	0	0
TTL 5 min	47.67	13.55	15	0	12
TTL 1 h	47.67	7.28	21	0	6
TTL 1 d	1.10	1.04	27	0	0

reader's estimate across the grid raises the maximum to 5.78, and across 48 cross-device cells transfer reaches 3.05 against 1.51 for local calibration, while still being cheaper in 9 of them.

The 27 constructed bursts isolate what admission depends on. On the cell where cumulative admission is worst, 50 reads arriving 100 break-even periods after the write, that rule costs 47.39 times the optimum, METER costs 1.99, and no admission at all costs 1.05. Over all 27, METER's maximum is 2.10 against the cumulative rule's 47.39: reading the window rather than the whole history is what lets an item that has gone quiet be readmitted when it becomes busy. These are synthetic stress cells, not hardware observations.

C.6 The executed placement run, cell by cell

Table 15 is Figure 3 as numbers, including the offline optimum in dollars and the device-measured n^* that each block's rule read. On those 80 items the analytic pricer and the raw-event repricing agree within 1% for METER (worst 1.50 against 1.49 over the residency grid) and exactly for never caching (1.92); always-KV and 2Q cannot be compared this way, because the analytic grid prices 208 items and the executed run 80.

C.7 Every residency, on both corpora

Table 13 gives every residency on both corpora and both LoCoMo question orders. On LongMemEval the offline optimum caches nothing beyond a few days at any of the twelve readers, so the policies converge and that block carries no information about them; that is why Section 5.2 runs on LoCoMo. Its third block reports LoCoMo in the benchmark's own question order, which Appendix C.8 shows is a burst rather than a property of agent memory, and which the first block therefore permutes.

Table 16 gives the separate LongMemEval hardware replay.

Table 15: Executed placement on 80 LoCoMo items (154 reads, 512 tokens) on two accelerators, cost as a multiple of an executed offline reference. n^* is measured on each device at the start of every run; EM is identical across placements; *cumulative* is the ablation of Table 3.

residency	n^* (reads/mo)	optimum (USD)	text	always-KV	2Q	METER	cumulative
M3 Max, Qwen2.5-1.5B							
1 h	3.53	7.57e-03	1.92	1.00	1.49	1.49	1.49
1 d	2.33	1.16e-02	1.89	1.03	1.48	1.48	1.48
1 wk	2.38	1.26e-02	1.71	1.19	1.43	1.43	1.43
1 mo	2.29	1.70e-02	1.31	1.65	1.35	1.35	1.36
3 mo	2.13	2.16e-02	1.11	2.87	1.65	1.28	1.12
6 mo	2.18	2.26e-02	1.04	4.94	2.32	1.17	1.04
15.7 mo	2.22	2.29e-02	1.01	11.90	4.70	1.04	1.01
Ascend 910B4, Qwen2.5-1.5B							
1 h	10.71	2.50e-03	1.92	1.01	1.49	1.49	1.49
1 d	10.77	2.68e-03	1.78	1.13	1.45	1.45	1.45
1 wk	10.54	3.75e-03	1.30	1.69	1.35	1.34	1.36
1 mo	10.49	4.61e-03	1.06	4.14	2.06	1.20	1.06
3 mo	11.06	4.60e-03	1.01	11.32	4.50	1.04	1.01
6 mo	9.92	5.15e-03	1.01	19.79	7.41	1.03	1.01
15.7 mo	10.40	4.93e-03	1.00	53.30	18.95	1.02	1.00
Ascend 910B4, Qwen3-0.6B							
1 h	33.44	3.22e-03	1.91	1.02	1.48	1.48	1.48
1 d	35.22	3.83e-03	1.52	1.36	1.37	1.37	1.37
1 wk	34.29	5.49e-03	1.09	3.34	1.80	1.26	1.10
1 mo	34.80	5.85e-03	1.01	11.84	4.68	1.04	1.01
3 mo	34.29	5.98e-03	1.00	33.75	12.22	1.01	1.00
6 mo	35.56	5.78e-03	1.00	69.19	24.42	1.02	1.00
15.7 mo	36.40	5.65e-03	1.00	185.02	64.30	1.00	1.00

Table 16: Executed placement on 234 LongMemEval sessions (238 reads), Qwen2.5-1.5B on Ascend 910B4, cost as a multiple of the executed offline optimum. The optimum caches almost nothing on this corpus, so never caching stays within 2% of it at every residency while caching at the write reaches 4.88× at one month.

residency	optimum (USD)	*text*	*always-KV*	*2Q*
1 h	7.25e-03	1.02	1.01	1.02
1 d	6.68e-03	1.02	1.13	1.02
1 wk	7.28e-03	1.01	1.85	1.01
1 mo	7.04e-03	1.00	4.88	1.02
worst		**1.02**	**4.88**	**1.02**

C.8 Question order in the LoCoMo placement trace

LoCoMo dates its sessions but not its questions, so its writes are placeable in time (Appendix E) and its reads are not. For the counterfactual sweep, all items are written at time zero and reads are spread over the residency being swept. Recovered session write dates are used only in the separate corpus reuse analysis. This reset matters because admission depends on age since write. The unpermuted question order is not neutral: the benchmark groups every question about one memory into consecutive indices, so the reads arrive as a burst that a cache can be leased across even at a long residency. At the 15.7-month residency the offline optimum caches 47.6% of items under that order and 1.0% with the order permuted on Qwen2.5-7B, and 7.9% against 0.2% averaged over the twelve readers, so the burst is the benchmark's file layout rather than a property of agent memory. Table 13 reports the permuted order throughout, taking the permutation whose optimum cost is the median of thirty. LongMemEval needs none of this, since it dates its questions as well as its sessions.

Table 17: Sensitivity to the two quantities the rules do not derive, as cost over the offline optimum across all residencies and readers. Top: the lease multiplied by c, with $c = 1$ derived. Bottom: the rate ablation's prior α, where larger admits sooner and 1.00 is its default.

cost / optimum at lease c/n^*, $c =$	0.25	0.5	0.75	1	1.5	2	4
LongMemEval, open cells	1.02	1.02	1.02	1.02	1.02	1.02	1.02
LoCoMo, open cells	1.55	1.51	1.51	1.51	1.51	1.51	1.95
worst, open cells	1.55	1.51	1.51	1.51	1.51	1.51	1.95
worst, all cells	1.55	1.51	1.51	1.51	1.51	1.51	1.95
the rate ablation's Gamma prior, swept; the admitted rule has none							
α, pseudo-reads per residency	0.05	0.10	0.25	0.50	1.00	2.00	
LongMemEval, worst open cell	1.59	1.63	1.73	1.85	2.16	2.27	
LoCoMo, worst open cell	1.41	1.40	1.39	1.39	1.40	1.52	

C.9 Every quantity in either rule that was chosen rather than measured

Table 17 sweeps both: the renewable lease length, and the rate estimator's prior. These isolate different choices. The price-derived lifetime is not fitted to a benchmark, but a sweep can still improve some empirical costs. Appendix C.5 reports this directly rather than claiming that calibration is universally optimal.

C.10 Limitations

Hardware execution covers single-tier placement on two devices, not a physical multi-tier serving engine. The analytical storage price is a reference accounting choice, not a measurement of actual HBM expenditure. Load timing is warm unless stated otherwise. LoCoMo read chronology is synthetic; serving traces cover one hour. Accuracy intervals do not establish equivalence. New GPU execution, cold-transfer measurements, capacity contention, and end-to-end latency evaluation remain needed for broader deployment claims. Four more matter to anyone reproducing the numbers. Carrier accounting charges the selected representation. A separately retained fallback text copy adds its byte cost during cached intervals; metadata and retrieval index costs are not included. We charge the text carrier for its own bytes alone: an fp32 retrieval index adds roughly 14 bytes per token, dividing our ratios by five. Run rather than priced, METER is two readers at one budget on two accelerators, over seven residencies each. The store caches a whole memory item and serves it as a prefix, so a read drawing on two cached items cannot concatenate their caches without the position repair of Table 11, which we measure separately and do not exercise inside the policy. Both prices are contracts rather than measurements, and n^* moves with their ratio, which is the reason to measure a threshold on the deployment rather than quote ours.

D What the literature reports

We audited the reporting of cost inputs in a query-selected corpus. We took the ICLR 2024 and 2025 papers indexed under the KV cache, long context, agent memory and context compression queries, kept the 23 whose subject is retained state or agent memory, and scored each for the four quantities Equation 4 requires: the accelerator, its prefill throughput Π, the storage price c_s, and the reuse count n.

Table 18 gives the result. Fourteen of 23 name an accelerator and 11 report a throughput figure, so roughly half the corpus states enough about its hardware for a reader to locate the measurement. Neither of the two quantities that decide whether retention pays appears anywhere: no paper prices storage over the interval the state is held, and no paper reports how many times the retained object is read. The absence is not a matter of degree. It is 0 of 23 on both.

This bounds what can be inferred from the corpus. A cache-versus-recompute result is a reading taken at one point on the surface Equation 5 describes, and n^* is linear in c_s, but nonlinear in Π when loading is charged. A reader who wants to know whether a published conclusion survives on different silicon, at a different storage price, or over a different residency cannot recompute it, because the inputs are not in the paper. We report our own three numbers in Section 3.2 for this reason.

Table 18: Bounded audit of 23 ICLR 2024–25 papers on KV cache and agent memory. The columns record the four quantities Equation 4 needs. In this queried corpus, neither retained-state storage price nor reuse rate is reported.

Paper	Yr	Acc.	Thr.	Stor.	Reuse
CCM	2024	•	•	–	–
StreamingLLM	2024	•	–	–	–
ICAE	2024	•	–	–	–
FastGen	2024	•	•	–	–
RECOMP	2024	–	–	–	–
Retrieval vs. LC	2024	–	–	–	–
Dyn-VQA	2025	•	–	–	–
CAKE	2025	•	•	–	–
DuoAttention	2025	•	–	–	–
FlexPrefill	2025	•	•	–	–
EM-LLM	2025	–	–	–	–
Activation Beacon	2025	–	•	–	–
LongMemEval	2025	–	–	–	–
MELODI	2025	–	–	–	–
HeadKV	2025	–	–	–	–
Palu	2025	•	•	–	–
RazorAttention	2025	•	•	–	–
SCBench	2025	•	•	–	–
SeCom	2025	–	–	–	–
SqueezeAttention	2025	•	•	–	–
Cascading KV	2025	•	•	–	–
VL-Cache	2025	•	•	–	–
Attention sink	2025	–	–	–	–
Reported by		**14/23**	**11/23**	**0/23**	**0/23**

The corpus also puts the compression literature's savings on a common scale with the architectural spread of Appendix A. The reductions these papers claim run from $3.3\times$ for RazorAttention, through $4\times$ to $8\times$ for the context compression methods, to $10\times$ for VL-Cache and $11.4\times$ for Palu. Choosing a different attention family moves the same cost by $16.8\times$ and the full sweep by $67\times$, neither of which requires a method. Most of the corpus saves less than the base model choice already decides.

E Derivations and detail from Section 3

E.1 Attention designs in closed form

The three conventional designs are this one equation at different h_{kv}: multi-head attention sets $h_{kv} = h$, grouped-query attention $h_{kv} = h/q$ for group size q, and multi-query attention $h_{kv} = 1$. Cost is therefore linear in h_{kv} at fixed L, d and b, and

$$S_{\text{token}}^{\text{MHA}} / S_{\text{token}}^{\text{MQA}} = h,$$

which is the span quoted in Section 3.1. Multi-head latent attention caches a compressed joint representation instead, giving

$$S_{\text{token}}^{\text{MLA}} = L \cdot (r_{kv} + d_{\text{rope}}) \cdot b \tag{8}$$

where r_{kv} is the latent rank and d_{rope} the dimension carrying rotary position. Both follow from configuration alone, so the quantity is exact.

E.2 Residuals are not a cheaper carrier, and what quantization actually saves

Keys and values are deterministic projections of the residual stream, so a system could retain residuals and regenerate the cache. That is not a cheaper carrier. Only the input-side residual reconstructs the full cache, since a layer-k checkpoint cannot recover the keys and values of layers 1 to k that attention still reads, and that checkpoint is the token embedding, which we verify is bit-identical to the embedding lookup. It costs $320\times$ to $4{,}551\times$ text for the same information and the same forward

Table 19: Rounding the retained cache moves its price, not its accuracy: three readers at a 512-slot budget, Δ cache minus text with a bootstrap interval. Four bits is 3.9× off fp16 and still 1,621× text on Qwen2.5-1.5B.

Retained as	× text	EM	Δ	95% CI
Qwen2.5-1.5B, text 0.582				
fp16 cache	6,383	0.576	−.006	[−.018, .000]
int8 cache	3,216	0.576	−.006	[−.018, .000]
int4 cache	1,621	0.588	.006	[−.029, .047]
Qwen2.5-3B, text 0.582				
fp16 cache	8,207	0.600	.018	[−.006, .047]
int8 cache	4,135	0.594	.012	[−.012, .035]
int4 cache	2,084	0.588	.006	[−.035, .047]
Qwen2.5-7B, text 0.565				
fp16 cache	12,766	0.559	−.006	[−.029, .012]
int8 cache	6,433	0.576	.012	[.000, .029]
int4 cache	3,241	0.547	−.018	[−.059, .024]

pass. Equation 8 and quantization are the real intermediate points. Quantizing the retained cache to b_q bytes per element with a per-channel scale and zero point, each b_s bytes, over a block of K tokens costs

$$S^q_{\text{token}} = 2L\, h_{kv}\, d \left(b_q + \frac{2b_s}{K} \right), \tag{9}$$

so the saving over fp16 is $b/(b_q + 2b_s/K)$ and not b/b_q. At $b = b_s = 2$, $b_q = 1$ and $K = 512$ that is $1.98\times$, which is what we measure.

E.3 Robustness checks from Section 3

Quantizing the retained cache. Quantizing the retained cache does not close this. Retaining Qwen2.5-1.5B's cache at int8 and int4 as in Equation 9, then reading it back at the same 512-token budget, gives exact match 0.576 and 0.588 against 0.582 for text and 0.576 for the fp16 cache, both paired intervals containing zero. What moves is the price, and by a factor of four rather than a change of regime: the premium falls from $6{,}383\times$ text at fp16 to $1{,}621\times$ at int4 (Table 19). The perturbation is real: int8 changes the decoded continuation on 26% of questions and int4 on 78%, despite the small changes in aggregate exact match. Nonsignificant score differences do not imply identical predictions.

Retriever, recipe and scale. Neither the retriever nor the recipe carries the result. Swapping BM25 for a dense encoder (all-MiniLM-L6-v2, everything else held) leaves the difference containing zero at all three scales: -0.012, -0.006 and $+0.006$ against -0.006, $+0.018$ and -0.006 under BM25. Dense reaches the gold span on only 76.5% of questions against BM25's 82.4%, which bounds every retrieval-based condition. Eight of the twelve readers widen the recipe rather than the budget: seven labs, all three attention designs, a factor of four in capability from SmolLM2-135M at 0.153 to Phi-3-mini at 0.676. The no-memory floor runs from 0.000 on SmolLM2-135M to 0.112 on Qwen2.5-7B, so the memory does the work rather than the model's priors. Doubling the retained bytes exactly, Qwen2.5-1.5B to 7B, changes accuracy by -0.018 $[-0.094, 0.065]$.

Positional transfer. Position is not the obstacle either. Re-injected state recovers the entire 7.44-nat oracle-to-floor gap at every offset tested, 0 through 24,576 tokens, and rotary re-rotation makes no measurable difference. Four further families return 99.7 to 101.0% of their own oracle on gaps of 7.49 to 10.66 nats. The one boundary case is architectural, not positional: Gemma-3-1B's hybrid attention needs its mask rebuilt for a shifted splice (Appendix B.7). The position probes test numerical recovery, not the wall-clock cost of arbitrary cache composition. Their tested offsets do not establish that migration or repair is free in a deployed multi-tier store (Li, 2026).

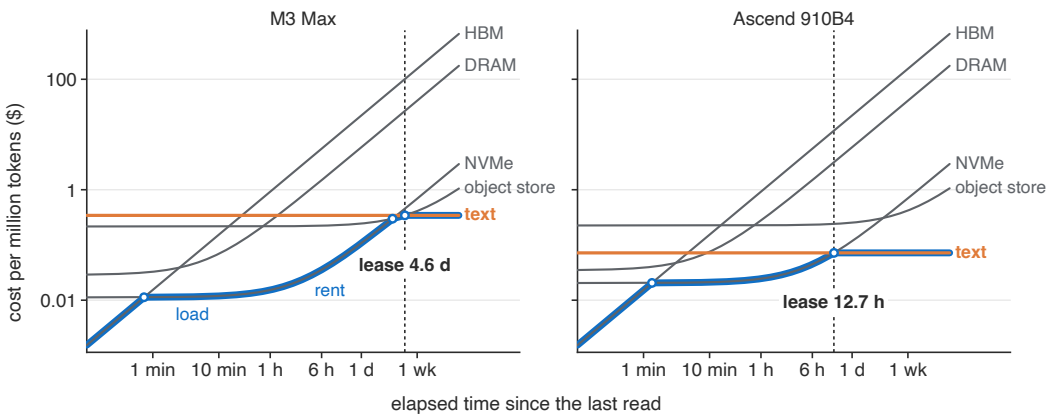


Figure 10: The lease is an intersection, not a knob. Each gray line is one tier's cost to hold a token for the elapsed time and serve one read, against text paying a prefill (orange); METER follows their lower envelope. The drop line is the longest lease in that machine's column of Table 1.

Reuse measured on both corpora's own dates. The objection Section 5.4 raises, that neither corpus is a deployment trace, is answered hardest on LongMemEval. That benchmark carries real dates, a question date on every question and a write date on every session, spanning 15.7 months, so residency and reuse are measurable in the same units rather than compared across them. Priced on those dates, an offline optimum that knows every future read, and may hold a cache for exactly the interval it wants rather than for the item's whole life, caches nothing at any of the twelve readers. Four of 234 items are read twice and none more often; those two reads are 128 to 160 days apart, against a break-even period of 1.3 days at the cheapest reader and 1.4 hours at the dearest. LoCoMo is datable too: all 288 of its sessions carry a stamp, dropped from the text because a stamp costs about a fifth of the read budget. Recovered for the accounting, an item's write is its latest evidence session and its residency the remainder of that conversation, a median of 2.9 months, giving 0.544 reads per item-month pooled over item-months, 43× under the cheapest threshold and 973× under the dearest. Sixteen of 208 items are written within one median inter-session gap of the transcript ending and are right censored, their residency being too short to carry a rate; left in, nine would clear the threshold on the strength of the one-day floor rather than of reuse. Censored out, the busiest item reaches 18.3. Against the thresholds the chips measure rather than the closed form, 7 of the 192 uncensored items pay under the write-time rule at the M3 Max's 2.50 and 19 exceed the rate; at the Ascend's 10.71 one does; at 21.3 and above none.

E.4 The tier price sheet, and a compressed carrier on the same axis

Table 1 prices five tiers from public list prices read on 2026-09-02, per GB-month: a rented H100-class accelerator's memory at $2 per hour over 80 GB ($18.26), on-demand memory-optimised instance capacity at $0.0066 per GB-hour ($4.82), EBS gp3 ($0.08), S3 Standard ($0.023), and a hosted API's context-cache storage charge of $1.00 per million tokens per hour, which for Qwen2.5-3B's 36 KiB per token is $19.82 per GB-month. HBM is resident and loads for free; DRAM is the measured warm load; NVMe is the cold model of Appendix B; object storage adds a fetch at 100 MB/s, an assumption we sweep: the tier is worth reading on the Ascend only above about 2.5 GB/s, because below that the load exceeds the 113 ms prefill it replaces. At fixed net read savings, the lease is inversely proportional to the incremental holding price. Figure 10 draws these costs against elapsed time, where a lease is a crossing and the cascade is a lower envelope. DRAM reaches that envelope on neither machine; on the Ascend neither does object storage.

A compressed cache is the same carrier at fewer bytes, and the axis prices it too. Cartridges (Eyuboglu et al., 2026) reports 38.6× less memory than the context it replaces. At that ratio Qwen2.5-7B's threshold falls from 23.4 to 0.61 reads per month if the write is free, just above LoCoMo's measured 0.544. The write is not free: a cartridge is trained by self-study, and charging that training at even ten prefills' worth of compute amortised over a year raises the threshold to 1.4, and at a thousand prefills over three months to 334. Compression moves an item along the axis; it does not remove the axis, and what it costs to write is the term of Equation 4 that the $T \to \infty$ limit discards.

Sweeping the two prices. Equation 5 makes the threshold responsive to storage and compute prices. The historical sweep at \$0.001 per GB-month admits 27 of 192 uncensored LoCoMo items under its write-time criterion; at \$0.0005 it admits 52. Thus cheap archival storage can change which items pay. These counts use the sweep's own calibration and are not a substitute for a threshold measured on the deployment.

What a system should do about the threshold. The threshold needs no experiment. Bytes retained follow in closed form from the configuration, so a system should compute n^* from its own config and its deployment's prefill throughput, and recompute it whenever either changes. What the closed form cannot supply is the rate to compare it against, which has to be estimated as reads arrive; Section 4 specifies that test. Both literatures should report bytes retained, expected reads and prefill throughput, so a reader can place a result on the axis.

Serving and memory workloads can be compared with the same cost model, but not with the same assumed chronology. Equation 4 includes the initial build; its asymptotic threshold omits that term. A fixed-lifetime write-time cache requires $n^*T + B/(B - R)$ reads, not merely n^*T. A cache created at a required read has a different incremental decision because that prefill has already been paid. Table 4 uses the latter convention. Neither convention supports a universal recommendation to retain text or cache every item.

Where each published system fails, and why. The deficits follow each design. Mem0 and LightMem both model the user, LightMem through the setting `messages_use: "user_only"`, and both collapse on assistant-answered questions, 0.179 against 0.410 for retrieved text $[-0.410, -0.077]$, span survival falling from 0.692 to 0.231: the evidence sat in turns their write policy never read. A-Mem inverts this. Storing every speaker, it is the only published system to lead text there, by 0.077 $[-0.051, +0.205]$, and falls back on knowledge-update instead, because its write path adds and links memories but never retracts a superseded one. LightMem's released configuration declares a `kv_cache` flag that no code path reads,[1] which measures how far the retained-state option has been taken here.

[1] A full-text search for `kv_cache` across `src/lightmem/` in the released repository `zjunlp/LightMem` returns three occurrences, all inside `configs/base.py`, and no code path reads either field. Read 2026-08-22.