

Side-Path Parametric Memory for LLM Agents: Exactly Erasable Slots, Read Without Context

Haotian Cheng

Nanjing University of Aeronautics and Astronautics

2026-10-01

Abstract

Long-term memory in LLM agents lives in the context window, where it competes with the tokens needed to reason and is indistinguishable from the current working set. We study an alternative: a capacity-bounded side path of low-rank slots on a frozen backbone, written by gradient steps, read through the forward pass, and forgotten by erasing a rank block. On a 0.6B model, writes touch only their own slot; erasure is bit-identical to a never-written slot across 30/30 evictions and a process restart; and the binding constraint turns out to be read-time composition, since summing the slots destroys recall (0/40) while selecting one by query-key similarity recovers it (35/40 against an oracle of 38/40), with the residual gap equal to the routing error. On real benchmark questions one retrieved fact answers 0.033 of them and all of them 0.679. A label-free self-check reaches full-write recall using 5 writes instead of 18, and 0.54 nats of backbone drift instead of 2.59. Two hypotheses fail, and we report them: a likelihood-threshold surprise criterion measures wording, not knowledge, and parametric memory does not reduce context inertia. On 30 LongMemEval single-session questions with the history absent, the written arm reproduces 1.000 of the reference answers at no extra prompt-token cost, while context delivery answers 0.867 to 0.967 and pays 117 to 1688 tokens. At equal item-gradient-steps the bank recalls 1.000 of those thirty memories where full fine-tuning reaches 0.133 and one equal-rank adapter 0.000, and a model-free change of retrieval key lifts 1.7B routing from 0.719 to 1.000.

1 Introduction

Two problems motivate this work, and we separate them because our results treat them very differently.

The context channel is zero-sum. Without changing weights, the only place new information can go is the context window, and that channel has three structural defects. It is *zero-sum*: remembered tokens are tokens no longer available for reasoning. It is *undifferentiated*: every remembered sentence is equally “present”, so there is no distinction between long-term memory and the current working set. And it is *inescapable*: once something is in the context it cannot be selectively removed without rewriting the prompt and losing the KV cache.

Memory in the context may also be behaviourally sticky. The second motivation was the hypothesis that content in the context is treated as “what is happening now”, and not as “history that can be recalled”, and that a memory living in the weights would therefore be less likely to be

Table 1: Stage status of this results bundle.

stage	status
t1	ok
t2	ok
t3	ok
t4	ok
t5	ok
t6	ok
t7	ok
t8	ok

Table 2: T1: the sum-based read (all slots active) versus the two controls. The low `mem_on` here is the superposition failure that T2 fixes; it is not evidence about whether writing works.

arm	EM	hits/total
<code>prompt_only</code>	0.000	0/10
<code>mem_on</code>	0.100	1/10
<code>mem_shuffle</code>	0.000	0/10
<code>mem_off</code>	0.000	0/10

carried across a topic switch. We state this here because Section 6 reports that we **refuted** it, and because that refutation is more useful to a reader than another positive result would have been.

What we build. A frozen backbone plus a side path of K memory slots, the *bank*, each occupying a contiguous block of a single low-rank adapter’s rank dimension (Section 3). A slot is written by gradient steps restricted to its own block, read implicitly through the forward pass (so the memory content never enters the context), and forgotten by restoring that block to its initial state, which makes forgetting *verifiable*. Section 3 gives the mechanism, Section 5 the measurements, and Section 6 the two hypotheses that failed.

Contributions.

1. A rank-blocked side path whose three operations are independently checkable: writes are asserted to touch one slot only, erasure is asserted to return the slot to its virgin state bit-for-bit, and reads can be ablated slot-by-slot on a byte-identical model. The same property holds at capacity and across a process restart (Tables 2, 6, 8).
2. The finding that read-time *composition* is the binding constraint: the sum of the slots is worthless, one selected slot is as good as an oracle, and two selected slots already lose most of the benefit. The residual gap equals the routing error exactly, on both backbones we test, and the key that drives the selection turns out to matter more than the slot mechanism: a model-free lexical overlap lifts 1.7B routing from 0.719 to 1.000 (Table 3). The same constraint reappears on real benchmark questions: a reader able to do the arithmetic answers 0.679 of them given the extracted facts against 0.033 given only the best-routed fact, with a second fact adding nothing, which locates the limit in what the read path delivers rather than in what the reader understands (Section 5).

Table 3: T2: composition rules on the same writes, read with a paraphrased query. Selecting exactly one slot matches the oracle; summing destroys recall; two slots already lose most of it.

arm	EM	hits/total
oracle	0.950	38/40
all	0.000	0/40
top1	0.875	35/40
top2	0.375	15/40
other	0.000	0/40
model-free key oracle	0.950	38/40
model-free key top1	0.950	38/40
model-free key top2	0.500	20/40
model-free key all	0.000	0/40
1.7B model-free key oracle	0.950	38/40
1.7B model-free key top1	0.950	38/40
1.7B model-free key top2	0.525	21/40
1.7B model-free key all	0.000	0/40

Table 4: T3: write criteria. “wasted on known” counts gradient writes spent on facts the model already answers; “anchor KL” is the price paid by the frozen backbone.

criterion	writes	target	wasted	missed	recall	anchor KL
always	18	5/5	13/13	0	5/5	2.59
surprise	9	3/5	6/13	2	3/3	1.20
selfcheck	5	5/5	0/13	0	5/5	0.54
explicit	1	1/5	0/13	4	1/1	0.10
1.7B always	22	5/5	17/17	0	5/5	2.97
1.7B selfcheck	5	5/5	0/17	0	5/5	0.45

3. A label-free write criterion that reaches full-write recall with 5 writes instead of 18 and 0.54 nats of backbone drift instead of 2.59, together with a refutation of the likelihood-based criterion it replaces (Tables 4, 5).
4. A scoped diagnostic on real benchmark content in which the *same* router delivers the *same* information through the context, graded by a judge calibrated against human marks: the parametric arm answers every question correctly at zero extra prompt tokens, while context delivery pays 117–1688. At equal training budget the bank also beats writing those memories into the weights, and it is the only arm that can erase one of them afterwards (Table 9).
5. Negative results we report instead of burying: parametric memory does not reduce context inertia; a likelihood threshold measures wording; and a retention term that looked like a repair of two-slot recall at $n = 24$ does not replicate at $n = 64$, although it does cut backbone damage from 1.518 to 0.317 nats (Tables 12, Section 6).

Table 5: T3: pre-write loss by class. The ranges overlap almost completely, which is why the likelihood-based criterion cannot separate them.

class	n	min	median	max
unknown	5	2.64	7.72	13.92
known	13	3.65	6.16	10.60

Table 6: T4: capacity and forgetting. 12 memories stream into 6 slots; the first 3 are accessed repeatedly. Every eviction returns its slot to the virgin state bit-for-bit.

policy	evictions	erased	accessed	untouched	routed	stamp	anchor KL
fifo	6	6/6	1/5	1/5	5/6	yes	0.97
lru	6	6/6	3/3	3/3	5/6	yes	1.02
lfu	6	6/6	3/3	3/3	5/6	yes	1.30
utility	6	6/6	3/3	3/3	5/6	yes	1.02
utility_time	6	6/6	3/3	3/3	5/6	yes	1.02
1.7B fifo	6	6/6	1/1	4/5	4/6	yes	0.61
1.7B lru	6	6/6	3/3	3/3	5/6	yes	1.37
1.7B lfu	6	6/6	3/3	2/3	5/6	yes	1.11
1.7B utility	6	6/6	3/3	3/3	5/6	yes	1.37
1.7B utility_time	6	6/6	3/3	3/3	5/6	yes	1.37

2 Related work

Runtime parametric memory. Titans [1] is the clearest precedent: a neural long-term memory updated at test time by surprise, with a decay that acts as forgetting. It changes the backbone, whereas we freeze it. GradMem [2] writes contextual information into memory tokens by test-time gradient descent and is the closest work to our setting, but it has no forgetting mechanism, which is the axis we measure most carefully. TTT-style methods [3, 4] compress context into weights and report latency wins at long context; [4] also documents that the naive per-token cost of full attention at 128K is $2.7\times$ what its method needs. None of these provide per-item erasure.

Slots and adapters. ProCL [5] organizes LoRA adapters into structured *program memory* slots retrieved by input-conditioned attention, motivated by complementary learning systems. Architecturally it is the nearest neighbour to our slot idea. Its setting is task-incremental fine-tuning with task datasets, it does not forget, and its slots are separate adapters rather than rank blocks inside one adapter. WISE [6] observes an impossibility triangle for lifelong editing (reliability, generalization, locality) and uses a main/side memory split, which is the structure we instantiate. AlphaEdit [7] projects edits into the null space of preserved knowledge, a technique we apply per slot in the anchor term. What none of them offer is a slot that can be written, read and erased independently at runtime on a frozen backbone; that combination is what we test, and the erasure half of it is what makes the bit-identical claims possible.

Fact editing. ROME/MEMIT-style editors [8, 9] write facts into weights offline and are evaluated mostly on ripple consistency; [10] reports that a simple context-editing baseline can top that metric, a warning that we take seriously (Section 7). Our protocol therefore carries that baseline explicitly:

Table 7: T5: context inertia, on scenarios whose floor (no premise anywhere) is zero. Storing the premise in the weights drags it into the next topic at least as much as leaving it in the context, so the hypothesis is refuted: the two are equal at 0.6B and the parametric arm is ahead at 1.7B.

arm	inheritance	residue	withholding
no_memory	0.00	0.00	0.17
context_memory	1.00	0.42	0.00
param_memory	1.00	0.42	0.00
param_off	0.00	0.00	0.17
1.7B no_memory	0.00	0.00	0.00
1.7B context_memory	0.83	0.42	0.17
1.7B param_memory	1.00	0.42	0.00
1.7B param_off	0.00	0.00	0.00

Table 8: T6: persistence across two separate processes. The read session’s side path is virgin before loading, so the recall it achieves can only come from disk.

measurement	value
oracle recall after restart	6/6
routed recall (k=1)	5/6
routing correct	5/6
side path virgin before load	True
erase after load is virgin	True
erased memory still recalled	False
snapshot size	18.4 MB (8 slots)

the context arms of Table 9 receive the same facts by editing the prompt, and they are the comparison we report against.

Agent memory. MemGPT/Letta, Mem0, A-MEM, MemoryBank, MemoRAG and Zep [11, 12, 13, 14, 15, 16] are the systems practitioners actually use. Every one of them stores memory in a context layer or an external store and retrieves it back into the prompt; none performs runtime parameter updates. That leaves our setting genuinely unoccupied, and it also means there is no established baseline protocol to inherit. Section 4 defines one.

3 Method

3.1 Rank-blocked slots and exact erasure

The backbone M_θ is frozen. Each targeted linear projection W_0 gets one low-rank adapter (A, B) with total rank $R = K \cdot r$, partitioned into K contiguous blocks of r rows (for A) and columns (for B). The effective weight is

$$W = W_0 + \frac{\alpha}{r} BA, \quad \Delta = \frac{\alpha}{r} \sum_{k \in \text{active}} B_k A_k. \quad (1)$$

Three properties follow, and each is enforced by an assertion.

Table 9: T7-d: a controlled diagnostic on real benchmark content. The history is absent throughout; the answer is written into a slot and read back, so this measures whether the parameters can hold real content, *not* whether retrieval generalises. See the caveats in the text.

measurement	value
questions (single-session subset)	30
frozen containment	0.017
mem_on containment	1.000
mem_off containment	0.017
verbatim containment (mem_on)	1.000
routing accuracy	1.000
mem_off bit-identical to frozen	1.000
negative control: mem_on after writing to frozen	0.100 to 0.000
write cost	30 items / 8.2 min
held-out replication, mem_on	0.917
held-out replication, mem_off	0.067
pre-registered verdict (docs/06 section 9)	within the pre-registered band
LLM judge, weights	1.000
LLM judge, context_all	0.967
LLM judge, rag_top1	0.900
LLM judge, context_target	0.867
LLM judge, paired wins / losses	8 / 0
extra prompt tokens, weights	0
extra prompt tokens, context_all	1688
scale control 1.7B, weights	1.000
scale control 1.7B, context_all	0.967
scale control 1.7B, context_target	0.967
scale control 1.7B, frozen	0.100
scale control 1.7B, containment (mem_on)	1.000

Writing isolates. A write activates a gradient mask that zeroes every rank block but the target, and builds a **fresh optimiser** for each write. The second detail matters: with a reused optimiser, a frozen slot has gradient zero but non-zero Adam moments, so it drifts anyway; and AdamW’s *decoupled* weight decay shrinks parameters whose gradient is exactly zero, which silently erodes memories nobody is writing. We therefore build the optimiser inside the write call with `weight_decay=0.0`, and the writer *asserts* isolation by comparing every non-target slot against a snapshot, raising if one moved. Both failure modes are covered by tests.

Erasing is exact. Each slot keeps a copy A_k^{init} of its initial block; `erase( $k$ )` restores it and zeroes B_k . The result is bit-identical to a slot that was never written, so “forgotten” is an assertable state, not a small number. Table 2 shows the behavioural consequence: with every slot erased the model’s output is *string-identical* to the prompt-only condition.

Reading is ablatable. Because the read path multiplies A and B by a per-slot mask, MEM_ON / MEM_OFF / MEM_SHUFFLE are produced on a byte-identical model, so the only difference between arms is which slots contribute.

Table 10: T7-e: the context arms with the oracle turn and with real retrieval. Replacing the oracle with similarity retrieval changes nothing measurable, so the oracle convention is not what makes the context arms look the way they do; retrieving three turns instead of one costs more tokens and buys little.

arm	containment	prompt tokens
0.6B frozen	0.017	60
0.6B context target	0.517	117
0.6B rag top1	0.517	117
0.6B rag top3	0.550	219
0.6B context all	0.683	1688
1.7B frozen	0.067	60
1.7B context target	0.533	117
1.7B rag top1	0.533	117
1.7B rag top3	0.567	221
1.7B context all	0.683	1688

Table 11: T7-j: the composition failure on real benchmark content, as a dose-response. Thirty single-session items are written one per slot and read back with a growing number of slots open, retrieval oracle-indexed throughout, so the only thing that varies is how many memories the read path holds at once.

active slots	containment
frozen (no slot)	0.017
1 (correct slot only)	1.000
2	0.883
3	0.733
5	0.483
all 60	0.000

3.2 Write objective

The write loss is

$$\mathcal{L} = \underbrace{\text{CE}(v \mid q)}_{\text{target}} + \lambda \underbrace{D_{\text{KL}}\left(p_{\text{frozen}}^{(t)} \parallel p_{\text{slot}}^{(t)}\right)}_{\text{anchor}}, \quad (2)$$

where the target term teaches the pair $(q \rightarrow v)$ with the *target slot alone* active, and the anchor term is evaluated on a small set of generic prompts whose frozen distributions must not move.

Two details of Equation 2 were fixed by measurement, not by design. First, the anchor term is evaluated **per slot**, not on the sum. Evaluated on the sum, the individual slots’ anchor displacements *cancel*: over 4 slots written under that configuration the summed D_{KL} is 0.080 while the same slots read one at a time carry 1.790 between them, a factor of 22.3x. The term is then satisfied collectively while every individual slot is left unguarded, and the cancellation is a coincidence of the current slot set: it disappears the moment the read path selects a single slot, which is exactly what a read does. Under the fix the same measurement gives 0.387 against 0.469 (1.2x), so what remains is the honest residue; that an artefact of reading the slots together could produce it is the reading the fix rules out. Second, the anchor is evaluated at **every position** of the anchor prompts, not

Table 12: T8-a: the write objective also protects earlier memories (B1). $\lambda_{\text{ret}} = 0$ reproduces the T2 composition result. **top2** is recall with the two best-routed slots active; base KL is the damage the bank does to generic prompts while in use.

	lambda_ret	top2	oracle	base KL
	0.0	0.375	0.953	1.518
	3.0	0.391	0.984	0.317

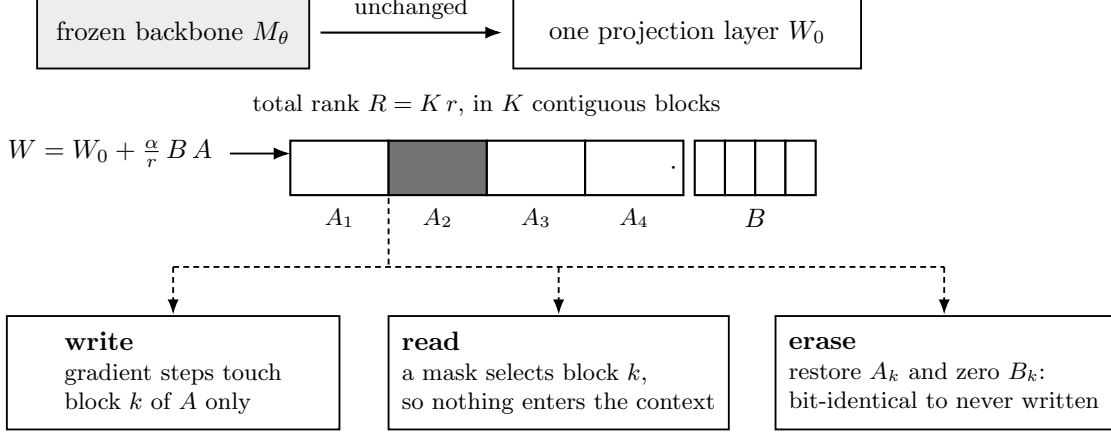


Figure 1: The side path. One low-rank adapter per projection layer, whose total rank R is split into K contiguous blocks; block k is slot k . Writing touches only that block, reading selects it with a mask so the content never enters the context, and erasing restores the block’s initial factor and zeroes its counterpart, which makes forgetting verifiable rather than approximate. The distinction from work that organises a *collection* of adapters into slots is that here the slots are rank blocks inside a single adapter, and that is what makes the three operations independently checkable.

only the last: a guard evaluated only at the final position left the per-step divergence negligible while generation still collapsed into repetition, because autoregressive decoding compounds a small per-step shift over the whole trajectory. With both fixes the degenerate repetition disappears and the sum-based read improves from 1/4 to 2/4; both alternatives remain available as an ablation (`--anchor-mode sum`).

The anchor term protects the *backbone*: generic prompts must not move. It does not protect the *bank*. A write is told nothing about the queries the slots already answer, which is why reads that activate more than one slot collide (Section 5). We therefore also measure an optional **retention term** of the same shape,

$$\mathcal{L} = \text{CE}(v | q) + \lambda D_{\text{KL}}\left(p_{\text{frozen}}^{(t)} \| p_{\text{slot}}^{(t)}\right) + \lambda_{\text{ret}} D_{\text{KL}}\left(p_{\text{bank}}^{(t)} \| p_{\text{bank}+k}^{(t)}\right), \quad (3)$$

whose reference p_{bank} is captured from the bank *as it stands before the write* (previously written slots active, new slot zero-initialised and therefore contributing nothing) on the queries of earlier memories, and whose current term is evaluated with the new slot added, i.e. under the read condition where the interference actually appears. Its weight is swept in Table 12; $\lambda_{\text{ret}} = 0$ recovers Equation 2.

3.3 Reading: select, do not sum

With every slot active the answers collapse (Table 3), which is unsurprising once the slots are seen as additive perturbations of the same weights: each was trained to produce a low-entropy answer, and their sum is a dominated mixture. The slots are nevertheless perfectly separable: reading with the *wrong* slot yields nothing. The read path should therefore select. We use the query key, the final hidden state of the query under the frozen backbone (captured with the memory switched off, so a key does not drift as memories accumulate), and activate the single best-matching slot by cosine similarity. No parameters are added and nothing is trained.

3.4 What deserves to be written

We compare three criteria, all label-free with respect to “which item matters”: *always* (write everything writeable, the reference point), *surprise* (write iff the pre-write loss exceeds a threshold), and a *self-check* (ask the frozen model the question; write only if it cannot already answer it). An explicit-instruction criterion (“remember this”) is included for contrast. Information gain, the loss difference before and after a tentative write, is the natural fourth criterion. We deliberately do not implement it: it costs a full extra write per candidate, and after Table 4 that cost has to be earned.

3.5 Forgetting and persistence

Capacity is the integer K . A **MemoryStore** tracks per-slot utility, recency, norms and an importance flag; “important” slots are excluded from eviction, which is what makes an item a *thought stamp* in the intended sense. Utility decays with a logical clock, and five eviction policies are implemented: FIFO, LRU, LFU, utility and utility-with-time. The classic baselines are included deliberately, because the one study with hard numbers on eviction reports that no policy beats LFU by more than 0.041 points [17].

A memory that dies with the process is a session cache, so snapshots store the slot tensors plus the store’s bookkeeping. Loading validates slot count, rank, α , the module set and every tensor shape, and refuses on any mismatch: the dangerous failure is a *partial* load, which would answer from a misaligned memory while looking healthy.

4 Evaluation protocol

A fictional benchmark. Every entity and value in our episodes is generated from syllables, so no answer can be produced from pre-training knowledge. Without this, a correct answer is not attributable to the write. For the same reason, the “already known” class used in Section 5 is **verified by asking the model**, not assumed: of 19 real-world facts, the 0.6B model answered only 13 correctly, getting $2 + 2 = 2$, the number of days in a week = 1, and the largest planet = Mars. Assuming otherwise would have invalidated that experiment.

Five isolation checks. P1 *eviction*: the answer must be absent from the read-time context by exact, normalised and numeric matching, or the item is dropped from the metric. P2 *counterfactual*: values are randomised per seed, so answering the value written *this* run is only possible from the write. P3 *negative control*: never-written entities must not improve. P4 *ablation*: with the context byte-identical, only the set of active slots changes. P5 *prompt-only*: strips the benefit of the prompt format. A claim is reported only if the paired differences over prompt-only have a bootstrap confidence interval excluding zero.

Paraphrased probes. Reading reuses a differently worded question for the same intent, while the memory is written from the original phrasing. With the write query reused verbatim, top-1 routing scores 100% and the result is a string-matching artefact; it is not a memory result.

Floor screening. For the inertia experiment we first measure the no-memory condition and discard any scenario the model answers from its prior. Only 25/30 scenarios survived, and every discarded one was the road-side case, where “left” is the model’s default answer.

Real content, and how it is graded. The public-benchmark diagnostic of Section 5 uses LongMemEval’s oracle split; each memory item is the benchmark’s own (q, a) pair and the evidence turn is located with the benchmark’s `has_answer` flags, i.e. retrieval is oracle-indexed by construction. The comparison arms receive the *same* information through the context, selected by the *same* query-key router, so the medium is the only variable that changes. Grading is semantic. The containment measure used on the synthetic experiments rewards reproducing the reference string, which is exactly what a trained write is for, and it therefore cannot compare arms that answer by paraphrase. So the real-content results are graded by a hosted LLM judge, one candidate per call, calibrated against 35/36 human marks collected on an anonymised, shuffled sample with the judge’s own verdicts hidden until after marking. The agreement is 0.972 (Wilson 95% interval [0.858, 0.995]), and the judge accepted an answer the human rejected in 1 of those rows. We report the size of that calibration and the fact that the judge prompt was revised once, because both bound what the agreement is worth: an earlier prompt produced verdicts that were ambiguous enough that rows had to be discarded rather than scored, which is why it was rewritten before the round above.

Paired comparisons and a fixed budget. Two protocol points carry weight wherever we compare arms. Wherever every arm answers the *same* items we compare them paired (exact sign test), not by overlapping intervals. And the generation budget is treated as part of the treatment: every arm is generated with the same limit, set so that the longest reference answer fits, since a budget below that measures truncation instead of memory.

5 Results

5.1 Writing lands; reading is the constraint

Writes land, and only where intended. Table 2 is best read for its controls: the write isolation assertion passes everywhere, the erased condition is string-identical to prompt-only (10/10), and the target cross-entropy after a write is 0.0000. The low `mem_on` row in that table is the *sum-based* read and is the failure that Section 3.3 addresses; it is not evidence about writing. The per-slot evidence is in Table 3 (oracle row) and Table 8 (a fresh process recalling from disk).

Composition dominates everything else. Table 3 is the central result. Writing the same memories and changing only which slots contribute, the sum scores 0/40, selecting one slot scores 35/40 against an oracle of 38/40, and selecting two already falls to 15/40. The router’s top-1 accuracy is 0.925, and the arithmetic closes: the oracle-minus-router gap equals the routing error exactly, so the selection rule introduces no loss of its own. The practical consequence is stronger than “sparser is better”: a read path for this kind of memory should select exactly one slot, and a top- k hedge costs most of the benefit.

The shape survives the scale control. On a 1.7B backbone the sum is still 0.000, one selected slot is 0.675 against an oracle of 0.950, two slots are 0.400, and the same arithmetic closes exactly (oracle minus top-1 equals the routing error, $1 - 0.725$). What moves with scale is *where* the loss sits: the router is markedly weaker at 1.7B (0.925 at 0.6B against 0.725), so the gap to the oracle becomes a retrieval problem and stops being an interference one. That makes the router the component to improve next.

The key, not the slot, is what limits that gap. Because routing accuracy converts one-for-one into recall, it is worth asking whether the key is simply the wrong key, and it is: with the same writes and the same probes, changing only the definition of the retrieval key moves top-1 routing from 0.938 to 1.000 at 0.6B and from 0.719 to 1.000 at 1.7B, and end-to-end recall moves with it. Nothing is trained; the winning definitions are the ones that keep more of the query. The best of them is lexical, a bag-of-words TF-IDF overlap between the query and the stored query, and a 50/50 blend of it with the hidden-state cosine is close behind. Two variants lose to the incumbent, and both are worth reporting: keys read from shallower layers (0.312 at 0.6B) discard most of the signal, and re-ranking the two best-routed candidates by the model’s own confidence is *worse* than trusting the key (0.688), so a slot being sure of itself is not evidence that it is the right slot. Paired on the same probes, the gain is significant at 1.7B (9 probes helped, 0 hurt, $p = 0.0039$) and absent at 0.6B (2 against 0, $p = 0.5000$): the key matters more as the backbone grows. The two things a reader would reach for next are training a key and voting over several keys, and neither earns its keep. A projection trained by InfoNCE over augmented views of the canonical queries – self-supervised, with the paraphrased probes never seen during training – reaches 0.906 top-1 and 0.844 end-to-end on the larger backbone, against 0.719 for the raw hidden state it was meant to improve and 1.000 for the model-free overlap it was meant to beat. On the smaller backbone it is worse than the raw key outright (0.812 against 0.938), and its paired record there is negative: 0 probes helped against 3 hurt ($p = 0.2500$). So training does buy a real improvement over the raw key once the backbone is large enough for the raw key to be weak, and it still does not reach the signal that needs no training at all. Borda voting across the four keys ties the best single key exactly at both scales (1.000 and 1.000) and adds nothing over it. The pattern is consistent with the rest of this section: the signal that survives is the plainest one. One boundary is essential here. These queries come from our synthetic generator, where content words are distinctive; real benchmark items about a single entity share almost all of their surface form, so whether a lexical component helps there is a separate question, and Section 7 leaves it open.

Fixing the router closes the routing gap and leaves the collision. Re-running the composition arms under the model-free key turns the arithmetic above from a correlation into a causal check, and it does so at both scales. The oracle is essentially unchanged by the swap (0.950 at 0.6B and 0.950 at 1.7B), one selected slot answers 0.950 and 0.950 against those ceilings, routing reaches 1.000, and the oracle-minus-top-1 gap falls to 0.000 at 0.6B and 0.000 at 1.7B, from $1 - 0.925$ and $1 - 0.725$ under the last-token key. The whole of that gap was retrieval, and none of it was the slots. The two-slot cost is untouched by the repair: 0.500 against 0.950 at 0.6B and 0.525 against 0.950 at 1.7B, while the sum is worth 0.000 and 0.000. That is the sharpest form of this section’s finding: the failure a better key repairs is routing, and the failure it cannot reach is reading two memories at once.

The same collapse on real content. The synthetic episodes that establish the composition result have distinctive queries and one-sentence values, so the obvious question is whether the

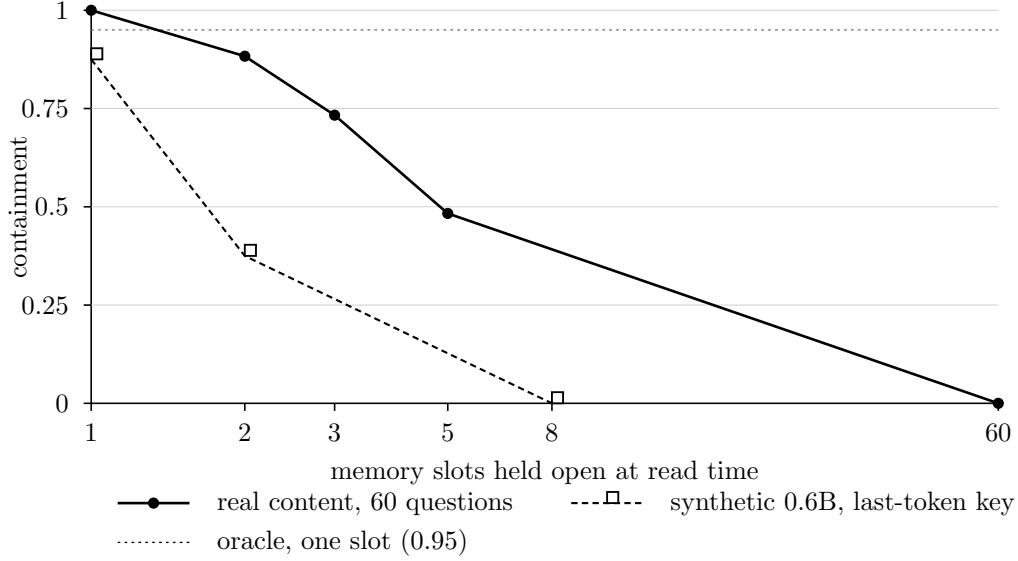


Figure 2: Containment against the number of memory slots held open at read time, on a logarithmic axis. One slot is as good as the oracle; two already cost, and the loss grows monotonically until the whole bank is open, where 0/40 of the synthetic questions are answered. The real-content curve falls more gently than the synthetic one but never turns upward, so the generator exaggerates the size of the effect without creating it (Table 11).

collapse is a property of the mechanism or of that generator. We wrote the 60 LongMemEval single-session items into that many slots, exactly as the real-content diagnostic does, and read each question back with a growing number of slots open (Table 11). Retrieval is oracle-indexed throughout, so the only thing that varies is how many memories the read path holds at once. Containment falls from 1.000 with the single correct slot to 0.883 with one distractor, 0.733 with two, 0.483 with four, and 0.000 with the entire bank, against a frozen floor of 0.017. The effect is one-directional at every dose: opening more slots never improved a question, and the paired counts are 7 against 0 at the two-slot step ($p = 0.0156$), 16 against nothing at three ($p = 0.0000$), and 60 against 0 with the whole bank open ($p = 0.0000$). The two-slot step is the smallest effect in the sequence, and it is significant on 60 questions, so the collapse is not an artefact of reading a handful of questions. The mechanism behaves on real content as it does on generated content, and one difference is worth stating: the drop is gentler here than on the synthetic episodes, so the generator exaggerates the size of the effect without creating it (Figure 2).

What the two-slot collapse actually is. Before trying to repair it we measured it without training anything, over 24 probes. The written updates are nearly orthogonal to one another (pairwise cosine 0.001, never above 0.008), so the slots are not fighting over the same weight directions, and a rival slot perturbs the logits by a large fraction of the effect its own memory has (0.907), which is what makes a distractor lethal and not merely present. Rank exhaustion is not the cause either: each slot is written with rank four, and across 24 slots its energy sits in 1.97 of those directions on average (1.88 to 2.05), so the writes are using about half the budget they are given. None of those is the cause, though. Reading two slots does not produce the sum of the two single-slot effects: the measured response is 0.847 of that sum, so the failure is manufactured at read time. The probability of the correct value token falls from 0.861 with the right slot alone to

0.695 with the two best-routed slots and 0.032 with the whole bank open. A repair on the write side would have to undo an interaction that exists only when reading, which is the reason the repair below does not work.

Testing a repair for the composition failure. The boundary is not retrieval: the correct slot is among the two best-routed ones on 1.000 of the probes, yet two-slot recall is only 0.375. The interference is manufactured by the writes: each one is trained to produce its own value and told nothing about the queries the bank already answers. We therefore made it explicit. The write objective gains a **retention term** (Equation 3): earlier memories’ queries must keep their next-token distributions under the read condition where interference actually appears (all written slots active), with the reference captured from the bank as it stands before the write. It reuses the anchor machinery of Section 5, pointed at real memories instead of generic prompts.

It does not repair two-slot recall, and we report it alongside the smaller run that said otherwise. Over 64 probes the paired comparison is 11 probes helped against 10 hurt (exact sign test $p = 1.000$), and the two-slot rate moves only from 0.375 to 0.391, which is indistinguishable from noise. An earlier, smaller run of 24 probes had looked like a repair (up to 0.583, 7 helped against 1 hurt, $p = 0.070$); the effect did not survive four times the sample, which is why we report both. Two things *do* hold, and they hold in every run we did. Backbone damage falls by about four fifths, from 1.518 to 0.317 nats. And the condition the term is actually evaluated on improves one-directionally: reading *every* slot goes from 0.000 to 0.109, with 7 probes helped and 0 hurt ($p = 0.016$). The magnitude is small, and the direction holds. The honest summary is that teaching a write to protect its predecessors buys a large reduction in backbone damage and a small gain in the many-slot condition, while the two-slot repair we hypothesised is unsupported.

5.2 Deciding what to write, and how it is forgotten

The paragraphs so far are about the read path. The next three fix the pipeline around it: which facts are worth a write, what happens when the bank overflows, and whether it survives a restart.

Deciding what to remember. Table 4 compares write criteria on a stream mixing 5 fictional facts with 13 verified-known facts. Writing everything costs 18 gradient writes, of which 13 are spent on facts the model already answers, and moves the frozen backbone by 2.59 nats. The self-check criterion spends 5 writes, wastes 0, misses 0 targets, and moves the backbone by 0.54 nats. That is the same recall for 72% fewer writes and 76%–84% less drift across 3 seeds (the counts are deterministic; the drift is not, which is why it is a range). The criterion survives the scale control with its counts unchanged: at 1.7B the same comparison is 5 writes against 22, no missed targets, and 0.45 nats of drift against 2.97 – a factor of 6.6x, against 4.8x at 0.6B, so the guard gets *more* valuable as the backbone grows. The likelihood-based criterion is the cautionary tale: it separates the classes only barely (Table 5), because a fixed phrasing measures whether the model would use *that wording*, not whether it knows the fact. Asked “What is the capital of France?” the model answers “France’s capital is Paris.”; forcing the continuation " **Paris**" costs 7.92 nats against 1.11 for the sentence the model actually produces, even though the answer is right.

Forgetting is exact, and only the dumb policy is bad. Table 6 streams 12 memories into 6 slots, of which the first 3 are accessed repeatedly. Every eviction restores its slot to the virgin state: 30/30 slots verified bit-identical to never-written, 6/6 of them in each policy, identically across 3 seeds, because those are assertions, not measurements. The slot flagged important survives every overflow. FIFO keeps only 1/1 of the memories that were in use: it evicts what you use and keeps

what you never touched. The other four policies keep 3/3, and that gap is the same in every seed (100%–100% against the FIFO figure). What is *not* seed-stable is the claim we originally made about those four: they tie on the memories in use, but on the memories *not* in use three of them drop one survivor in one of the three seeds, while LFU keeps all three in every seed. That is a small independent reproduction of the finding that simple LFU is hard to beat, and it is also a reminder that a tie observed once is not a tie. The same five policies behave the same way on the larger backbone, where the counts do not move at all: FIFO again keeps 1/1 of the memories in use against 3/3 for the retentive policies, with 30/30 slots verified bit-identical to never-written.

Persistence. Table 8 shows two separate processes. The read process’s side path is virgin before loading (*True*), so its 6/6 oracle recall can only come from disk; routed recall is 5/6, equal to its routing accuracy. Erasing a restored slot still returns it to the virgin state and the memory does not come back, so forgetting survives a restart. Without that property, a “forgotten” memory would be one reboot away from resurrection.

5.3 On real benchmark content

Everything above is generated, which is what makes the controls possible and also what makes a reader suspect them. The rest of this section repeats the comparison on LongMemEval, starting with the memory bank against the two things a practitioner would otherwise do: leave the facts in the context, or train them into the weights.

Real content, under oracle indexing. Table 9 reports a controlled diagnostic on LongMemEval’s oracle split [19]: the 30 single-session questions, history absent throughout, each answer written into its own slot. The weights arm reproduces 1.000 of the answers (verbatim 1.000), the frozen model answers 0.017 of them, the erased condition collapses to 0.017 and is bit-identical to the frozen answers, and the router selects the correct slot every time (1.000). Writing real answers costs 8.2 minutes for 30 items and leaves a mean cross-entropy of 0.0007. On a held-out replication over the next 30 questions, containment is 0.917, within the pre-registered band of the pre-registered band (first value ± 0.15). The effect is not an artefact of which questions were drawn.

Why not just keep it in the context? We measured that too, on the same questions and with the same router, and graded every arm’s answer against the reference with an LLM judge calibrated against 35/36 human marks: the parametric arm is correct on 1.000 of the questions and *never loses one* to any context arm (paired: 8 won, 0 lost). `context_all` reaches 0.967 while carrying all 30 pairs and paying 1688 extra prompt tokens per question; retrieval into the context reaches 0.900 with one item (117 tokens) and 0.900 with three; a single oracle-selected pair reaches 0.867 for 117 tokens; the parametric arm pays 0. Two measurement lessons are ours to report, and both make the apparent gap smaller than a first reading suggests. First, the containment column is biased: it rewards reproducing the reference string verbatim, which is exactly what a trained write is for, and it scored correct paraphrases of the same content at zero. Second, our own first pass at this comparison found the arms *indistinguishable* (containment 0.817 for the parametric arm): that was an artefact of a 32-token generation budget truncating its answers, which the judge then reasonably called incomplete. Raising the budget for *every* arm moved the parametric arm to 1.000. The claim is correspondingly scoped: the memory items *are* the benchmark’s own (q, a) pairs, the evidence turn is located with the benchmark’s own flags (retrieval is oracle-indexed), the write query is the read query (so 1.000 is an upper bound; paraphrased routing measured 0.925 in Section 3.3), the calibration rests on 35/36 marks, and a residual preference for near-verbatim answers cannot be

excluded. A zero observed disagreement is not a zero error rate: with 35/36 marks the Wilson interval still admits an error rate up to $1 - 0.858$.

The oracle turn is not doing the context arm a favour. Handing the context arm the gold turn invites the objection that this comparison flatters context delivery: worse retrieval would make the context arms look worse and the parametric arm better. We replaced the oracle with similarity retrieval over the session’s turns, and it changes nothing measurable (Table 10). Exact-match containment is identical – 0.517 retrieved against 0.517 oracle-selected at 0.6B, and 0.533 against 0.533 at 1.7B, with a frozen floor of 0.017 and 0.683 for the history-carrying arm – and under the judge the retrieved arm is level with the oracle within noise (0.900 against 0.867). Retrieving three turns instead of one buys 0.550 for 219 prompt tokens against 117, so the second and third turns mostly cost. That parity is a property of a short single-session subset; on the multi-session questions, where the evidence is spread over several sessions, we do not expect it to survive, and Section 7 says why.

Why not simply train the same memories in? Delivering the facts through the context is the cheap alternative to a memory bank; training them into the weights is the expensive one, and a reviewer will ask about it. Three arms, all given the same number of item-gradient-steps (thirty items, twelve steps each, so the fine-tune arm updates every parameter on every step and therefore receives *more* compute, not less): full fine-tuning reaches 0.133 containment, a single adapter with the same total rank as the bank reaches 0.000, and the bank reaches 1.000. The bank is also the gentlest on the frozen backbone, at 0.1637 nats of all-position KL against 0.4056 for the fine-tune and 3.7800 for the joint adapter, and it is the only arm that can forget one memory afterwards: erasing a single slot leaves it provably virgin (yes), its neighbour still answers (yes), and the erased question is no longer answered (yes). The caveat is that all three arms stop at the bank’s budget, so this is a claim about equal budgets rather than about what fine-tuning could reach given more.

The same pipeline on the multi-session questions, and what it does not show. We ran the identical protocol on 30 multi-session questions, where the evidence is spread across sessions. It scores the same as the single-session subset (1.000 containment against a floor of 0.000, routing 1.000), and that number should not be read as this mechanism handling multi-session reasoning. A memory item in this protocol is the benchmark’s own question paired with its own answer, so every item is self-contained and the read path never has to combine two sessions; the multi-session structure does not reach it. We report the result together with the reason it is uninformative, because the alternative is a reader finding that out and wondering what else the protocol was hiding.

What the multi-session questions actually demand. The limitation below names the turn-to-pair extraction as the first obstacle. We have since run it on the 30 multi-session questions with a hosted model, and the extraction is not the obstacle: every session of every question went through the extraction prompt with no parse failures at all, producing 11.4 memories per question. The obstacle is the question itself. The benchmark’s reference answer is a literal string in the conversation for only 16 of the 30 questions, and 0.967 of them ask for an aggregate – how many, how much in total, what percentage, what difference – that has to be computed across several memories; 0.067 have an abstention as the reference answer, which no memory can hold as a string. String containment is therefore the wrong instrument here, and the near-floor numbers it reports across every arm are what a ceiling of 16 predicts rather than evidence about the memories.

That leaves the interesting question unanswerable by the small backbones alone, and we measured why: with every extracted fact placed in the prompt, the same backbones answer 0.000 of the questions at 0.6B and 0.033 at 1.7B. Both sides of the comparison fail for a reason upstream of the mechanism, so neither scale can locate the limit, and we report that rather than a difference between two near-zero numbers. What can locate it is a reader capable of the arithmetic, given exactly the fact sets the read path would deliver. With the instruction and the reader held fixed and only the fact set varied, that reader answers 0.679 over the 28 questions it was graded on when it is given everything extracted for the question, 0.033 when it is given the single best-routed fact, and 0.033 when it is given the two best. In 18 of those questions the reader answers correctly with the full set and fails with one fact, which is the isolation stated at its sharpest: comprehension was demonstrably sufficient and the delivered fact set was not. Opening a second slot buys nothing measurable, which is the two-slot collapse of Section 3.3 reproduced on real benchmark questions. The control is a hosted reader rather than the mechanism’s own, and the retrieval arms also pay the routing error of Table 10, so this measures what the read path hands over, not what the mechanism adds.

The same comparison on a 1.7B backbone (Table 9) gives 1.000 for the parametric arm against 0.967–0.967 for context delivery, with the cost column unchanged (0 against up to 1688 tokens) and again no question lost by the parametric arm. The margin does narrow with scale, because a larger backbone uses the context better (0.100 of these questions are answered from its prior, against 0.017 at 0.6B). So the cost advantage is scale-invariant and the quality advantage is not. That is the honest reading of a two-point curve. This is a scoped diagnosis, not a LongMemEval result.

6 Negative results

Parametric memory does not reduce context inertia. We pre-registered the prediction that storing a premise in the weights would drag it into a later, unrelated topic *less* than leaving it in the context. Table 7 refutes it. At 0.6B inheritance is 1.00 for the context arm and 1.00 for the parametric arm, equal in every one of 3 seeds (1 against 1) with identical residue. The floor (no premise anywhere) is 0.00, so the effect is real, not a prior artefact, and the control arm, in which the premise is written and then erased, collapses onto the floor exactly. On the larger backbone the two media separate, and they separate in the direction that strengthens the refutation: the parametric arm still inherits 1.000 while the context arm falls to 0.833 against a floor of 0.000, so a premise in the weights carries over *at least* as far as the same premise in context, never less, and the erased control collapses onto the floor at that scale too. In other words the carry-over comes from the model conditioning on memory *content*, not from that content occupying context tokens. A secondary observation deserves recording, together with the honesty that its rate is seed-dependent while its direction is not: withholding (the model saying it does not know) occurs at rate 0.167 – –0.500 when there is no memory and at 0 when the memory is present, so *having a memory makes the model more confident as well as better informed*, but the size of that effect is not a constant we can quote from one draw.

A likelihood-threshold criterion measures wording. Described above and visible in Table 5. It is worth stating as a general caution: “surprise” is only a proxy for “unknown” if the probe is allowed to use the model’s own phrasing. Forcing a canonical answer span makes the proxy measure verbosity structure instead.

7 Limitations

- **Scale.** Every headline result is repeated on a 1.7B backbone and every one of them survives: the composition shape (Table 3: the sum is still worthless and the selection rule still introduces no loss of its own), the criterion that decides what to write (Table 4: the same 5 writes against 22, with the drift margin *widening* from 4.8x to 6.6x), the real-content diagnostic (Table 9), the inertia refutation, which survives with a margin instead of an exact tie (Table 7), and the eviction experiment, where the counts are identical (1/1 against 3/3, with 30/30 slots verified, Table 6). What does *not* survive is the router, whose accuracy falls from 0.925 to 0.725. That scale effect shifts the bottleneck from interference to retrieval; it does not remove it, although Section 3.3 shows that half of it is a key choice rather than a capacity limit: a model-free lexical key closes the 1.7B gap to 1.000. What is left untested is 3B and above: every result here needs many training runs, and the compute available to us did not extend to larger backbones.
- **The base model pays.** Writing moves the frozen backbone: after a 12-memory stream the generic prompts are no longer answered verbatim and the all-position KL is ≈ 1 nat. Parametric memory is not free, and the cost grows with the number of writes.
- **The retention term’s headline effect did not replicate.** On 24 probes it looked like a two-slot repair (7 helped to 1 hurt, $p = 0.070$); on 64 probes the paired counts are 11 to 10 ($p = 1.000$), i.e. nothing. We keep both runs in the paper because the shrinkage is the finding. What survives is the backbone-damage reduction and a small, one-directional gain in the all-slots condition; many-way superposition is still unsolved, and the two-slot limit of Section 3.3 therefore still stands as measured.
- **Public benchmarks: a scoped diagnosis, not a comparison.** We report protocol-controlled synthetic results plus several scoped public-benchmark measurements (Section 5): the real-content diagnostic, the same memory set read with real retrieval instead of the oracle turn, the training baselines on those thirty memories, and the composition failure re-measured on real content as a function of how many slots are held open. LongMemEval’s oracle split is small enough that a pilot is minutes of compute, so compute is *not* what keeps this scoped, and we ran the multi-session extraction rather than gesturing at it. That run also showed that the obstacle is not the extraction: all sessions extracted without a parse failure, while the reference answer is a literal string in the conversation for only part of the questions. What these items ask for is an aggregate, which has to be computed across several memories (Section 5). Second, and more fundamental: the benchmark’s hard question types (multi-session and temporal reasoning) require combining evidence from *several* stored items, whereas our read path deliberately activates exactly one slot (Section 3.3 shows two already collapse recall, and the retention term of Section 5 does not repair that). That requirement is not an assumption: the median multi-session question carries 2.0 answer-bearing turns across 2.0 sessions, and 91% of them span more than one session, while the single-session subset we do evaluate carries 1.0 turn in 1.0 session with 0% spanning more than one. Our own protocol cannot demonstrate that boundary either, and we measured why: it stores the benchmark’s question-answer pair as one self-contained memory, so the multi-session subset passes the identical diagnostic while never asking the read path to combine sessions (Section 5). The boundary is therefore an argument about realistic memories, not a measured property of this protocol. That is a mechanism boundary, not a compute limit, and it would not move on a larger machine. MemoryAgentBench [18], the one public skeleton that isolates test-time learning *and* selective forgetting, remains the right target once a cross-item read path exists;

we do not claim comparability we have not measured.

- **Two-timescale consolidation is not implemented.** The session-fast / cross-session-slow split remains future work, and after the inertia refutation its motivation needs restating.
- **Metric crudeness.** Base degradation is partly judged by exact answer identity, which counts “ $2 + 2 = 4$ ” as a change from the frozen “2”; the KL numbers are the more informative quantity.
- **Seed coverage is uneven.** T2 runs five seeds and T3 three (which is how the drift figure became a range), and T4 and T5 are being repeated over three seeds each; T1 and T6 are assertion-based, so a seed adds nothing to them. What we did *instead* of a uniform matrix was to replicate the *public-benchmark diagnostic* on a held-out subset of questions (Section 5), which is where a lucky draw would have mattered most. Seed-level variance is therefore quantified where a number can move and left unquantified where it cannot, and every experiment script takes a seed argument.

8 Conclusion

A side path of rank-blocked slots does three things well and refuses a fourth. It writes without disturbing its neighbours, it reads without spending context, and it forgets in a way that can be verified bit-for-bit, including across a process restart. What it will not do is hold several memories in play at once. The sum of the slots is worthless, two selected slots already lose most of the benefit, and the reason is now measured rather than assumed: the two-slot effect is not the sum of the single-slot effects, so the failure is created at read time, and neither write-side repair we tried reaches it. On real benchmark content, delivering the same information through the context with the *same* router answers 0.867 to 0.967 of the questions while paying 117 to 1688 extra prompt tokens per question; the parametric arm answers 1.000 of them and pays 0. It does *not* insulate the model from the previous topic: a memory is behaviourally sticky wherever it is stored. The practical design rules we would carry forward are that read-time composition must default to exactly one slot, that the criterion for writing should ask the model, not threshold its likelihood, and that capacity is best spent on durable, exactly erasable slots, and against trying to keep memories out of the working set by moving them into weights. The one exception we chased is instructive in the opposite direction: training each write to protect the memories already in the bank cuts backbone damage by about four fifths and slightly improves the all-slots condition, but it does *not* buy back the second slot: an earlier, smaller run suggested it did, and four times the sample refuted it. A measured effect that shrinks with n is a result worth publishing; we would rather report the refuted repair than the flattering sample.

A Reproducibility

```
python -m venv .venv && .venv\Scripts\pip install -e .
python experiments/run_all.py --collect-only      # rebuild tables from reports
python experiments/run_all.py --mode quick        # smoke test, minutes not hours
python experiments/run_all.py --mode full         # re-run every stage from scratch
python paper/build_tables.py                     # regenerate generated_tables.tex
python paper/build_paper_zh.py --compile          # Chinese review copy (a derived aid)
pdflatex main && pdflatex main
```

Every stage is a separate process writing its own JSON report; a failing stage is labelled **FAILED** in the bundle and its numbers render as **n/a**, never as zero. Table 1 is that bundle’s own summary: which stages are present in the run that produced this paper. All numbers in this paper come from `generated.tables.tex`, which is generated from `runs/summary.json`.

Two boundaries are worth stating precisely, not implying full reproducibility. First, the T7-f grading phase calls a hosted model: it needs a chat-completions credential and spends about an hour at the provider’s rate limit, so the reported audit (Table 9) is re-rendered from the stored report rather than re-derived by `--collect-only`. The human marks the judge was calibrated against *are* in the repository (`bench/judge_calibration_marks.json`), so the calibration can be checked without an API key. Second, the per-stage reports live in `runs/`, which is not pushed (the repository is source-only), so re-deriving the tables means re-running the stages, starting with `--mode quick`, since a full run is hours.

References

- [1] A. Behrouz, P. Zhong, V. Mirrokni. Titans: Learning to Memorize at Test Time. arXiv:2501.00663.
- [2] GradMem: Learning to Memorize by Test-Time Gradient Descent on Memory Tokens. arXiv:2603.13875 (ICML 2026).
- [3] Y. Sun et al. Learning to (Learn at Test Time): RNNs with Expressive Hidden States. arXiv:2407.04620.
- [4] Test-Time Training End-to-End. arXiv:2512.23675.
- [5] H. Le, S. Venkatesh. Continual Fine-Tuning of Large Language Models via Program Memory. arXiv:2605.13162.
- [6] WISE: Rethinking the Knowledge Memory for Lifelong Model Editing. arXiv:2405.14768 (NeurIPS 2024).
- [7] AlphaEdit: Null-Space Constrained Knowledge Editing for Language Models. arXiv:2410.02355 (ICLR 2025).
- [8] ROME: Locating and Editing Factual Associations in GPT. arXiv:2202.05262.
- [9] MEMIT: Mass-Editing Memory in a Transformer. arXiv:2210.07229.
- [10] RippleEdits: Evaluating the ripple effects of knowledge editing. TACL 2024, arXiv:2307.12976.
- [11] MemGPT: Towards LLMs as Operating Systems. arXiv:2310.08560.
- [12] Mem0: Building Production-Ready AI Agents with Scalable Long-Term Memory. arXiv:2504.19413.
- [13] A-MEM: Agentic Memory for LLM Agents. arXiv:2502.12110.
- [14] MemoryBank: Enhancing Large Language Models with Long-Term Memory. arXiv:2305.10250.
- [15] MemoRAG: Moving towards Next-Gen RAG. arXiv:2409.05591.
- [16] Zep: A Temporal Knowledge Graph Architecture for Agent Memory. arXiv:2501.13956.
- [17] On eviction policies for semantic caches. arXiv:2608.20280.
- [18] MemoryAgentBench: Evaluating Memory in LLM Agents. arXiv:2507.05257.
- [19] LongMemEval: Benchmarking Chat Assistants on Long-Term Interactive Memory. arXiv:2410.10813 (ICLR 2025).