FROZEN TRANSFORMERS CAN REUSE, COMPOSE AND UPDATE STATE MEMORY

Anonymous authors
Paper under double-blind review

ABSTRACT

A transformer that has read a context carries its computation on that context in every layer. We show that a frozen transformer can answer from a context it no longer has when it keeps one vector per position from a single early layer, the residual activations after its second decoder layer, and reads them back by placing them at placeholder positions of a new prompt and recomputing every layer above; we call these vectors state memory. On multi-round coreference resolution (MRCR) conversations of up to 136,577 tokens, Qwen3.8-27B discarded the text and its entire key/value cache, rebuilt the other 62 of its 64 layers from its bf16 memory and scored 75.5 against 75.0 for full history (the context read as tokens); 96 of 115 greedy answers were identical (83–84% in every range of lengths). Four-bit memory scored within 0.35 F1 of full history on all 150 MultiFieldQA-en questions of LongBench with Granite 4.2 8B and Qwen3.8-27B, and on 128 synthetic handbooks bf16 memory stayed within one percentage point of full history on Granite 4.2 8B, Qwen3.5-27B and Qwen3.5-9B, against 25.5–26.2% with neither. The same frozen model also operated on these memories. On Qwen3.8-27B, four-bit memories of DROP passages and of numerical questions, each written without the other, passed 41 of 52 tests that require the right answer under all four passage–question pairings, as many as full history. Revised ten times by regenerating its text from the memory and a counterfactual correction, a bf16 memory of SQuAD paragraphs scored the same as the corrected full history on every question, and a four-bit memory revised twenty times answered 96.5% of questions against 97.9%. Finally, the model read episodes of an environment's tool use and wrote a note; a reader holding only the note's states executed the environment's tool-call convention on 213 of 256 new requests on Granite 4.2 8B and 195 on Qwen3.8-27B, against 235 and 206 with the episodes in context and 0 from a note written without them. On Granite 4.2 8B, a policy memory written the same way decided 170 of 256 new records exactly, against 130 with the episodes in context, and policy and procedure memories acquired in different environments executed 438 of 1,024 composed workflows, against 202 with both logs in context.

1 INTRODUCTION

A model that works across many requests has to carry what it has read and learned from one to the next: a context read once may be asked about later, combined with another, corrected, or distilled from experience. We ask whether a frozen transformer can do this with its own residual states. We store the residual activations after the model's second decoder layer (layer 1, counting from 0) at every position of a context, and show that the frozen model can use these states as memory: it can read them under new questions in place of the context, combine them with memories written separately, revise them without the original text and write them from its own experience.

We call the forward pass that stores the states the writer and the pass that reads them the reader. The writer runs the model over the context alone and stores the layer-1 activations at every context position, and the reader puts one period token at each stored position of a new prompt, overwrites their layer-1 activations with the stored ones and continues with the question (Figure 1). Every layer above layer 1 recomputes from the stored states, so at the context positions the reader repeats the writer's computation and first departs from reading the context in the question's first two layers, which see periods where the context was; Section 3 measures how much that changes the answers.

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

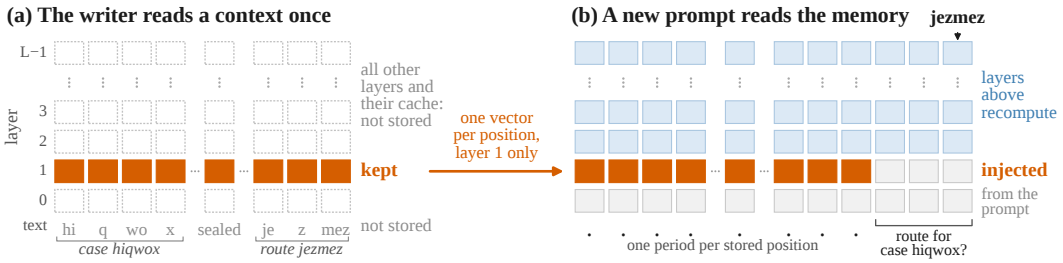

Figure 1: State memory on the grid of positions and layers. (a) The writer runs the model over a context once and keeps only the residual states after layer 1 (orange), one vector per position; the text and every other layer, cache included, are not stored. The tokens are Granite 4.2 8B tokens of the handbook in Appendix C, which records case hiqwox as sealed and sends every sealed case to route jezmez. (b) A later prompt holds one period at each stored position, the stored states overwrite layer 1 there, and every layer above recomputes over the memory and the question.

Memory read in place of its source scored within one point of full history (the context itself read as prompt tokens) on multi-round coreference resolution (MRCR), on all 150 MultiFieldQA-en questions of LongBench, on LoCoMo conversations and on the four-choice questions of synthetic operations handbooks with three checkpoints (Section 3 and Figure 2). On MRCR conversations of up to 136,577 tokens, the reader had neither the text nor any part of the model's key/value cache and rebuilt the other 62 of Qwen3.8-27B's 64 layers from the layer-1 states, and with bf16 memory it scored 75.5 against 75.0 for full history; it gave the same greedy answer in 96 of 115 conversations, and in 83–84% of the conversations in every range of lengths. Four-bit passage and question memories from DROP, written separately, passed 41 of 52 tests in which every pairing of two passages with two questions had to be answered correctly, as many as full history (Section 4). On Qwen3.8-27B, bf16 memory of SQuAD paragraphs revised through ten counterfactual corrections without their original text answered the same 935 of 1,056 questions as the corrected full history, and on the synthetic handbooks a four-bit memory revised twenty times scored 1.4 points below the corrected full history, pooled over the initial read and all twenty revisions (Section 5).

We then gave the model fourteen episodes from an environment whose tool-call convention it had to infer, asked for a reusable note and kept only the layer-1 states at the note's 224 positions, none of which is an episode position. A reader holding those states executed the convention on 213/256 unseen requests on Granite 4.2 8B, against 235/256 with every episode in context. On Granite, contiguous windows of the episodes' own layer-1 states executed at most 5/256, and a window split between the start and the end of the log worked only while the evidence sat at those two ends; after we moved the late episodes to the middle it executed 0/64 requests on the first 16 sets, against 63/64 for the note memory written from the reordered log. In environments where a hidden rule routes each record, a reader holding the states of a policy note, written in answer to a request to state that rule, decided 170/256 unseen records, more than the 130/256 it decided with the episodes in context, and a policy memory and a procedure memory (a note memory of a tool-call convention) acquired in different environments executed 438/1024 composed workflows together, against 202/1024 with both episode logs in context (Section 6).

2 Storing and reading layer-1 states

Let P be a fixed instruction prefix and H a context. The writer runs the model on P followed by H and keeps the residual activations $h_\ell(P, H)$ after decoder layer ℓ at the context positions S. The stored memory is

$$M_{\ell,S,b} = \big(Q_b(h_\ell(P, H)[S]),\ S,\ A,\ \ell,\ c\big), \tag{1}$$

where Q_b encodes the activations at precision b, A gives the positions at which the reader places them and c holds the quantiser's parameters. Positions in S are counted from the start of H, so $h_\ell(P, H)[S]$ denotes the activations at positions $|P| + S$; for a single context, S covers every position of H and $A_i = |P| + S_i$. The reader's prompt is P, then one period token for each stored position, then the question. After layer ℓ it overwrites the activations at the placeholder positions with the

decoded memory, once, during prefill, and then generates with an ordinary cache built from that pass. The overwrite is the placeholder-patching construction of Patchscopes (Ghandeharioun et al., 2024), applied at every position of a context instead of to one hidden state.

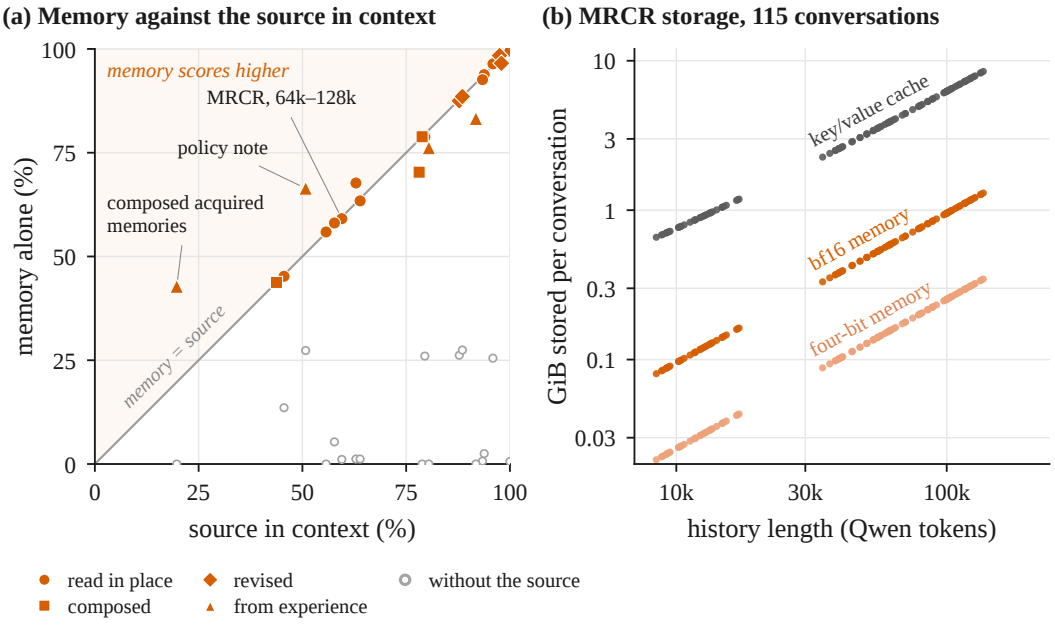

Figure 2: (a) Memory against the source in context, one point per comparison on the same questions and each benchmark's own metric: handbooks (three checkpoints), the six MRCR bins, LongBench MultiFieldQA-en (two checkpoints), LoCoMo, four-way swap tests on handbooks and DROP, revision over ten corrections (two checkpoints), SQuAD and twenty corrections, and the three tasks of Section 6. Hollow points give the same comparison without the source (no history, the final MRCR turn alone, or a memory written without the episodes). (b) GiB stored per MRCR conversation on Qwen3.8-27B for the key/value cache and for bf16 and four-bit memory of the same history.

We capture at $\ell = 1$ in every experiment, so all but two of the model's layers compute over the memory and the question together; memories written at layer 8 instead matched layer 1 on Qwen3.5-27B but not on Granite 4.2 8B or Qwen3.5-9B (Appendix B.3). A memory holds one vector of the model's hidden size per position, stored at the model's bfloat16 precision (bf16 memory) or in a four-bit code with one 16-bit scale for each group of 64 values (Appendix B.2).

The reference in every experiment, full history, gives the reader the context itself as prompt tokens. In MRCR it is the writer's own key/value cache continued with the question, and in revision it is the current, corrected context. Composition places memories written separately into one position layout at read time, and each writer is told at most where its own positions begin (Section 4). To revise a memory, the reader generates a revised version of the context from the current memory and a short correction. The writer then stores the layer-1 states of that text in a pass that contains neither the old memory nor the correction (Section 5).

Models and statistics. Handbook reuse uses Granite 4.2 8B (IBM Granite Team, 2026), Qwen3.5-27B and Qwen3.5-9B with bfloat16 weights; handbook composition and ten-cycle revision use Granite and Qwen3.5-27B, and LoCoMo uses Qwen3.5-27B. MRCR, DROP, the SQuAD and twenty-cycle revisions use Qwen3.8-27B (Qwen Team, 2026), LongBench uses Granite and Qwen3.8-27B, and the experiments on memory written from experience use Granite and Qwen3.8-27B. Qwen3.8-27B runs with 4-bit NormalFloat (NF4) weights and bfloat16 computation, which fits its MRCR histories on one 32 GB GPU. The weight format is separate from the memory format, which is bf16 or four-bit throughout; Appendix B.1 lists the pinned revisions. Each experiment resamples its own independent unit (a handbook, a test set of memories, a revision trajectory, a conversation or an environment set) with that unit's questions kept together, and every interval is a 95% percentile

bootstrap interval over those units (Appendix B). We fixed the prompts, the greedy decoding and the number of units in each experiment before running its evaluation.

3 READING MEMORY IN PLACE OF ITS SOURCE

3.1 LONG NATURAL CONTEXTS

Multi-round coreference resolution (MRCR; OpenAI, 2025; Vodrahalli et al., 2024) tests exact recall over long conversations. A conversation contains two or eight near-identical requests (the needles), for instance eight poems on one topic. Its last turn asks for one of them by position and gives a random string the answer must start with, and the official grader scores zero without that string and otherwise the similarity ratio of Python's `difflib.SequenceMatcher` to the reference. On Qwen3.8-27B the writer prefilled each conversation's history and stored its layer-1 states, and full history continued the writer's own key/value cache with the final turn. The memory reader has neither the history's text nor any part of that cache: it starts from an empty cache, places a period at each history position and rebuilds the other 62 of the model's 64 layers from the stored states. On the 30 eight-needle conversations of 64k–128k tokens by MRCR's count, whose histories reach 136,577 tokens under the Qwen tokenizer, bf16 and four-bit memory both scored 59.1 against 59.5, and the final turn alone scored 1.1 (Table 1). Over all 115 conversations, 96 greedy answers from bf16 memory were identical to the full-history answer, as were 93 from four-bit memory, and bf16 memory scored 75.5 against 75.0, a paired difference of $+0.5$ points $[-4.8, 5.6]$.

Needles	Tokens	n	Full history (cache)	bf16 memory	Four-bit memory	Final turn only
8	64k–128k	30	59.5	59.1	59.1	1.1
2	64k–128k	15	93.8	93.8	93.8	2.5
8	32k–64k	20	62.9	67.7	63.1	1.2
2	32k–64k	15	100.0	99.5	99.5	0.6
8	8k–16k	20	63.9	63.4	63.0	1.2
2	8k–16k	15	93.4	92.6	92.6	0.7

Table 1: MRCR score (official grader, $\times 100$) on Qwen3.8-27B with NF4 weights; n is the number of conversations. Tokens are counted with the o200k_base tokenizer that MRCR uses to bin conversations, over conversation and answer. Both memory formats keep layer-1 states at every history position.

In every range of history lengths, 83–84% of the greedy answers from bf16 memory were identical to the full-history answer, so agreement did not fall with length, and 24 of the 27 conversations of 100,000 tokens or more gave identical answers.

Four-bit memory also stayed within one point of full history on the complete 150-question MultiFieldQA-en task of LongBench (Bai et al., 2024), with sources of up to 16,746 tokens kept whole; it scored 45.22 F1 against 45.57 on Granite and 58.07 against 57.72 on Qwen3.8-27B, with paired intervals over 100 source documents of $[-1.81, 1.08]$ and $[-1.19, 1.89]$ points (Appendix E.5). On eight complete LoCoMo conversations of 21,683 to 29,724 tokens (Maharana et al., 2024), with 96 questions, Qwen3.5-27B four-bit memory scored 55.9 official F1 against 55.7, a paired interval of $[-1.3, 2.2]$ points (Appendix E.4).

3.2 SYNTHETIC HANDBOOKS

Each handbook is a synthetic operations document in which case records, a threshold routing rule with an exception, random bindings of operations to function names and a label-format convention are shuffled together. A case record might give five units and sealed status while a separate exception sends every sealed case to route `jezmez`. Finding a case's destination therefore needs retrieval, choosing its route needs the rule, and building a call needs the right function and argument bindings (Appendix C shows a complete handbook). We wrote one memory per handbook from the handbook alone and read it under ten four-choice questions, two each on retrieval, the routing rule, tool calls,

label format and combined answers, and the answer was the letter with the highest probability. Test handbooks use vocabularies and wording disjoint from the development split on which the protocol was fixed.

Across 128 test handbooks, bf16 memory answered 78.8%, 96.4% and 87.3% of 1,280 questions on Granite 4.2 8B, Qwen3.5-27B and Qwen3.5-9B. Full history answered 79.5%, 95.9% and 87.8%, and without either accuracy fell to 26.0%, 25.5% and 26.2%. Table 9 in Appendix D breaks the comparison down by question type. When the reader had to generate its answer, with exact match required on tool calls, labels and combined answers, the number of correct answers out of 192 with memory differed from full history's by -2, -1 and $+2$ (Appendix E.1). Four-bit memory stayed within 0.5 points of full history on all three checkpoints (Appendix D).

4 Combining memories written separately

Each of 32 test sets per checkpoint holds two fact memories A_0, A_1 and two procedure memories B_0, B_1, all written separately, and a fixed question q is read against every pairing,

$$y_{ij} = \mathrm{Read}(A_i \oplus B_j, q), \qquad i, j \in \{0, 1\}, \tag{2}$$

where $\oplus$ places the two memories in a shared position layout. The question and its answer choices are the same under all four pairings, but the correct answer is not. Routing questions combine a quantity from the fact memory with a branch from the procedure memory, and tool questions combine arguments from one with function bindings from the other, so an answer can change only because a memory was swapped. A four-way test passes when all four pairings are answered correctly. The procedure writer pads its memory to the longest fact span in its test set, so the only thing it learns about its partner is that length.

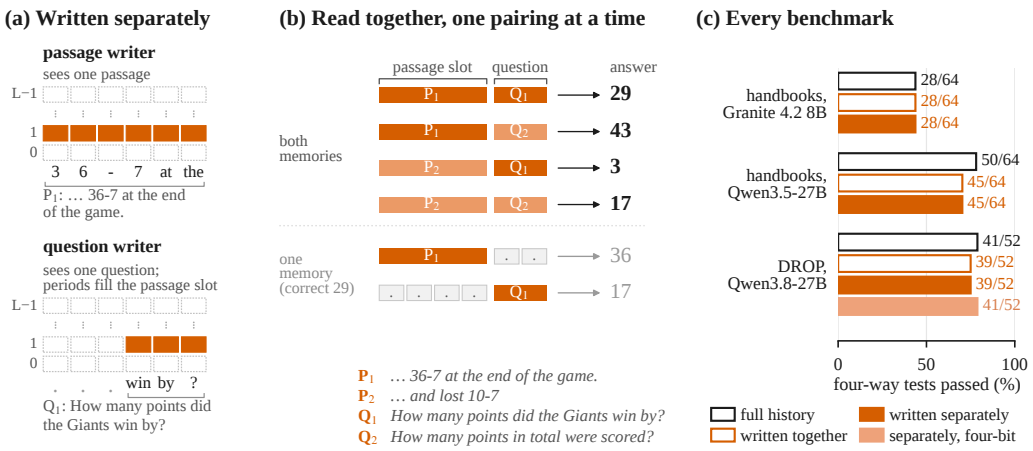

Figure 3: The four-way swap test on DROP. (a) Each writer is given one component at its fixed slot of a shared layout (2,048 passage positions, then 128 question positions) and keeps only its layer-1 states: the passage writer sees its passage, and the question writer sees periods in the passage slot and its question. Tokens are Qwen3.8-27B tokens of the example below. (b) One of the 52 test sets, read under its four pairings with four-bit memories on Qwen3.8-27B. P_1 ends "…36-7 at the end of the game", P_2 "…lost 10-7", Q_1 asks "How many points did the Giants win by?" and Q_2 "How many points in total were scored?". Each pairing has a different correct answer and the reader generated all four; from the passage memory or the question memory alone it answered 36 and 17 where the answer is 29. (c) Share of four-way tests passed, a test counting only when all four pairings of Equation 2 are answered correctly: 64 tests per handbook checkpoint with bf16 memories and 52 on DROP. "Written together" comes from one writer that saw both sources and "written separately" from two; the lighter DROP bar uses four-bit memories.

On the handbooks, bf16 memories written separately passed 28 of 64 four-way tests on Granite, the same number as full history, and 45 of 64 on Qwen3.5-27B against 50, as many as memories written

together by one writer (Figure 3). On single questions they answered 174 and 235 of 256 against 181 and 239, the fact memory alone 97 and 94 and the procedure memory alone 69 and 65.

On natural text we used 52 test sets, each with two DROP passages (Dua et al., 2019) and two human-written numerical questions whose four combinations have four different answers (Figure 3b). A passage writer saw only its passage and a question writer only its question. Every memory went into one global layout of 2,048 passage positions and 128 question positions, so no writer needed its partner's length. On Qwen3.8-27B, bf16 and four-bit memories answered 193 and 196 of 208 questions against 194 for full history, and they passed 39 and 41 of 52 four-way tests against 41. The same memories still worked after we moved them, without rewriting, into a contiguous layout, where each format answered 83 of 96 questions against 82 for the contiguous text (Appendix F.1).

5 Revising memory without its source

Each correction has 18–26 tokens and is applied, as in Section 2, to the handbook memory the previous correction produced, and some corrections reverse earlier ones, so anything needed later has to survive every revision in between.

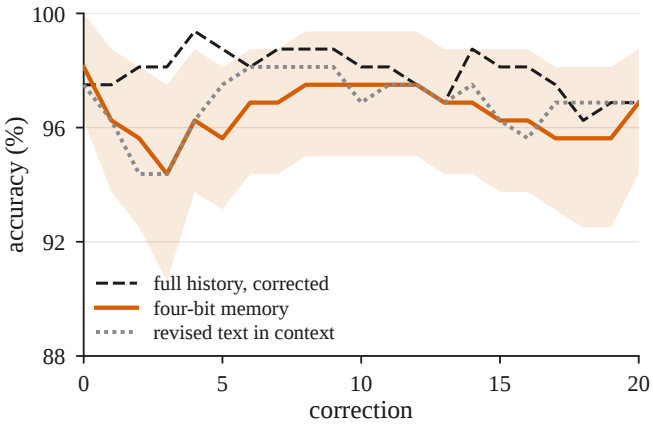

Figure 4: Accuracy of Qwen3.8-27B (NF4 weights) through twenty revisions without the source, over 16 trajectories with ten questions at the initial read and after each correction. Four-bit memory decodes its own four-bit predecessor before every correction, and the revised-text reference keeps the generated text in context instead of its states. The band is the 95% interval of four-bit memory over trajectories.

Over ten corrections on Granite, bf16 memory answered 1,540 of 1,760 questions against 1,545 for the corrected full history, a paired difference of -0.3 points $[-2.7, 2.2]$. Qwen3.5-27B bf16 memory answered 433 of 440 questions against 429.

Figure 4 extends the chain to twenty corrections on 16 new Qwen3.8-27B trajectories. Four-bit memory, where each revision reads the decoded four-bit memory from the previous correction, answered 3,244 of 3,360 questions against 3,291 for full history, and a reference that keeps the revised text itself in context answered 3,255. After the twentieth correction four-bit memory answered all 28 questions about newly changed facts, 61 of 65 questions about unchanged information in the part of the handbook the corrections edit and 63 of 64 about the rest of the handbook.

On natural text we took 24 Wikipedia paragraphs from SQuAD (Rajpurkar et al., 2016), each with four human questions, and applied ten counterfactual corrections that alternate between two of the paragraph's years, and each revision read only the previous memory and the correction. On Qwen3.8-27B, bf16 memory and the corrected full history each answered 935 of 1,056 questions across the initial read and ten revisions, with the same exact match and F1 on every paired observation. The revised memories could also still be combined with separately written question memories, and answered 334 of 384 questions after 0, 1, 5 and 10 corrections against 336 for full history (Appendix F.3).

6 WRITING MEMORY FROM EXPERIENCE

6.1 A TOOL-CALL CONVENTION WRITTEN FROM EPISODES

Each of 64 environment sets holds two environments that differ in a hidden tool-call convention, drawn from 54 candidates. A decision with fields route, amount and priority has to be issued as a `prepare` call that lists the three fields in an environment-specific order, followed by a `commit` call with an environment-specific flag and an echo of one decision field. The public interface names the tools and fields but not the convention, and fourteen episodes generated from the convention reveal it. The first four show `prepare` calls with the commit still pending, the last four show accepted `commit` calls, and the six in between rule out no candidate, so the evidence for the whole convention spans 343 tokens of the episode log.

	Tool-call convention		Policy
Reader holds	Granite 4.2 8B	Qwen3.8-27B	Granite 4.2 8B
Note memory (224 states)	213/256	195/256	170/256
Episodes in context	235/256	206/256	130/256
Memory written without the episodes	0/256	0/256	70/256
Best contiguous episode window (224 states)	5/256	41/256	113/256

Table 2: Requests executed (tool-call convention) or records decided exactly (policy) out of 256, over 64 environment sets. Episode windows hold 224 contiguous layer-1 states of the episodes, taken from the head, middle or tail; the best of the three is shown.

The writer reads the episodes followed by one fixed request, the same in every environment, for a reusable operating note, and generates 224 tokens. We keep the bf16 layer-1 states at those note positions, taken from a pass over the episodes, the request and the note, so that the note's first two layers attended to the episodes, and discard every episode position. At 224 tokens the note is shorter than the 343-token span that holds the evidence, so no contiguous stretch of the log of the note's size holds the whole convention. Each environment then gets two requests about decisions no episode contains, one of which changes a single field of a previous decision, 256 requests in all. A request succeeds only when both calls are exact and the environment accepts them (Appendix A).

On Granite, a reader holding the note memory executed 213/256 requests, against 235/256 with the episodes in context and 0/256 with a memory written without them (Table 2). The note memory therefore recovered 0.906 [0.856, 0.957] of the gain from seeing the episodes. On Qwen3.8-27B, with the same 64 sets, it executed 195/256 against 206/256, a recovery of 0.947 [0.844, 1.049]. In the 107 Granite environments whose note named the echoed field, the note memory executed 212/214 requests against 211/214 with the episodes in context, and every request it lost relative to the episodes lay in the 21 environments whose note named that field wrongly (Appendix A.6).

6.2 WHERE THE EVIDENCE SITS IN THE LOG

Contiguous windows of 224 episode states taken from the head, middle or tail executed at most 5/256 requests on Granite and 41/256 on Qwen3.8-27B. Each carried only the part of the convention whose episodes it covered (Appendix Figure 6). On the first 16 sets, a split window of the first 112 and the last 112 episode states executed 56/64 requests, close to the note memory's 60/64, because the two blocks of evidence sit at opposite ends of these logs. Moving the last four episodes to the middle, with rules and requests unchanged, dropped the split window to 0/64 and raised the middle window from 0/64 to 53/64. The note memory written from the reordered log executed 63/64 (Figure 5), while each fixed window worked only in the order that placed both blocks of evidence inside it; between the two orders, the note memory's change in success rate exceeded the split window's by 0.922 [0.781, 1.062].

6.3 POLICIES, AND MEMORIES ACQUIRED IN DIFFERENT ENVIRONMENTS

In a second family of environments a hidden rule, drawn from 81 candidates, assigns one of three routes to each combination of record class and urgency band, and fourteen episodes with the same

structure reveal it. The writer answered a second fixed request, to state the routing rule, and from the states of that note (the policy memory) the reader decided 170/256 unseen records exactly on Granite, against 130/256 with the episodes in context, a difference of 0.156 [0.062, 0.250] (Table 2).

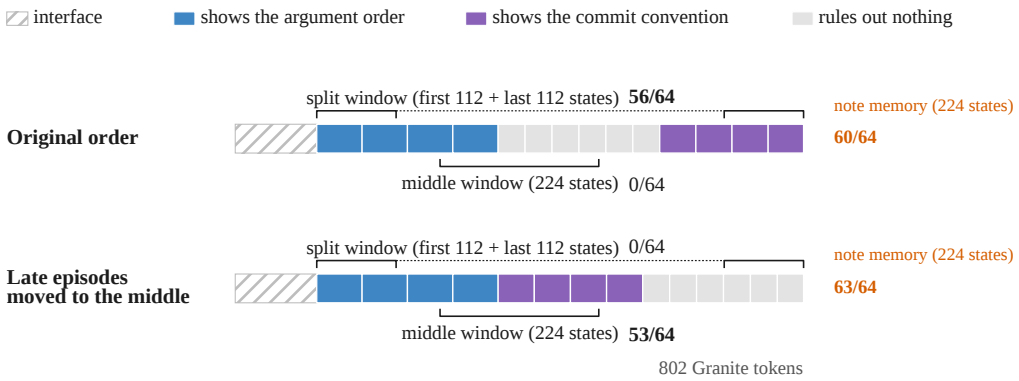

Figure 5: Where the evidence sat in the log decided which window worked. Each bar is the episode log of one environment drawn to scale in Granite 4.2 8B tokens: the interface, then fourteen episodes colored by what they show. Brackets mark the split window (the 112 states from the first episode on and the last 112 states) and the middle window (the central 224 states of the log), and the note memory reads 224 states of the note the model wrote from the same log. Counts are requests executed over the first 16 environment sets, 64 per condition. Moving the last four episodes to the middle took the split window from 56/64 to 0/64 and the middle window from 0/64 to 53/64, and the note memory executed 60/64 and 63/64 in the two orders (protocol in Appendix A).

On Granite, we then paired a policy memory with a procedure memory (a tool-call note memory, as above), the two acquired in different environments. On records that the policy had to decide and the procedure had to execute, each of the four pairings had its own correct pair of calls. The pair executed 438/1024 workflows, against 202/1024 with both episode logs in context and 0/1024 with memories written without episodes. On two prespecified pairings, the pair's success rate was higher than the procedure memory's alone by 0.189 [0.146, 0.231] (Appendix A.7).

7 Related work

Reading hidden states back into a frozen model is the basis of Patchscopes (Ghandeharioun et al., 2024), which patches one state into a placeholder position to decode what it holds, and task and function vectors (Hendel et al., 2023; Todd et al., 2024) carry a behaviour in a single added activation. Here the states of every position of a context are read back, combined and revised. CoMem (Liu et al., 2026) caches residual states at an intermediate layer and recomputes the query-conditioned upper layers over a retrieved pack of them, and HCache (Gao et al., 2025) restores key/value caches from stored activations; the swap tests, revision and memory written from experience here have no counterpart in either. Other work represents a context by a few trained vectors: gist tokens (Mu et al., 2023), AutoCompressors (Chevalier et al., 2023), the In-context Autoencoder (Ge et al., 2024), xRAG (Cheng et al., 2024) and cartridges (Eyuboglu et al., 2026) train a model or an encoder to write them, and activation aggregation (Ardoin et al., 2026) compresses a prompt's activations without training. Prompt Cache (Gim et al., 2024), CacheBlend (Yao et al., 2025) and Li (2026) combine or edit precomputed key/value caches, and MEMORYLLM (Wang et al., 2024) updates a trained latent memory pool from new text. The four-bit code follows the group-wise quantisers used for activations and caches (Ashkboos et al., 2024; Zandieh et al., 2026).

Agent memory systems keep experience as text, distilled into insights, workflows or evolving playbooks (Zhao et al., 2024; Wang et al., 2025; Suzgun et al., 2026; Zhang et al., 2026b; Ouyang et al., 2026), and latent agent memories train the writer or the memory it produces (Zhang et al., 2026a; Fu et al., 2026; Wu et al., 2026). In Section 6 the writer is the frozen model itself, and the memory is the layer-1 states of the note it writes.

8 Conclusion

Keeping only the residual activations after a frozen transformer's second decoder layer, at every position of a context, was enough for the model to answer from that context once the text and its cache were gone. On MRCR conversations of up to 136,577 tokens, Qwen3.8-27B rebuilt its other 62 layers from these states and scored 75.5 against 75.0 for its own cache. Four-bit passage and question memories written separately passed as many four-way tests on DROP as the source in context, and memories revised without their text kept pace with the corrected source. A note the model wrote from its own episodes carried a tool-call convention that contiguous windows of the episodes' own states did not, and a policy memory decided more unseen records than the episodes did in context. In these experiments the layer-1 states of a frozen model were a memory it could read, combine, revise and write, with no part of the model trained.

Reproducibility statement

The supplement contains the raw outputs of every experiment and scripts that regenerate every number and table in the paper from them. Rerunning inference needs the pinned checkpoints listed in Appendix B.1. LoCoMo is downloaded at a fixed upstream commit and checked by hash, and Appendix B documents the independent units, scoring and uncertainty.

Generative AI Disclosure

Generative AI tools assisted with manuscript preparation, implementation, and literature search. All research design, experiments, analysis, and claims are the authors' own, and the authors take full responsibility for the manuscript.

Acknowledgments

Acknowledgments are omitted for anonymous review and will be added in the final version.

References

Thibaud Ardoin, Semira Einsele, Evis Bregu, and Gerhard Wunder. Prompt compression via activation aggregation. *arXiv preprint arXiv:2607.08399*, 2026. URL https://arxiv.org/abs/2607.08399.

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L. Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. QuaRot: Outlier-free 4-bit inference in rotated LLMs. In *Advances in Neural Information Processing Systems*, 2024. URL https://papers.nips.cc/paper/2024/file/b5b939436789f76f08b9d0da5e81af7c-Paper-Conference.pdf.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3119–3137, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.172. URL https://aclanthology.org/2024.acl-long.172/.

Xin Cheng, Xun Wang, Xingxing Zhang, Tao Ge, Si-Qing Chen, Furu Wei, Huishuai Zhang, and Dongyan Zhao. xRAG: Extreme context compression for retrieval-augmented generation with one token. In *Advances in Neural Information Processing Systems*, volume 37, pp. 109487–109516. Curran Associates, Inc., 2024. doi: 10.52202/079017-3476. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/c5cf13bfd3762821ef7607e63ee90075-Abstract-Conference.html.

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. Adapting language models to compress contexts. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3829–3846. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.232. URL https://aclanthology.org/2023.emnlp-main.232/.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2368–2378, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1246. URL https://aclanthology.org/N19-1246/.

Sabri Eyuboglu, Ryan Saul Ehrlich, Simran Arora, Neel Guha, Dylan Zinsley, Emily Ruoyu Liu, Atri Rudra, James Zou, Azalia Mirhoseini, and Christopher Ré. Cartridges: Lightweight and general-purpose long context representations via self-study. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=0k5w8O0SNg.

Muxin Fu, Xiangyuan Xue, Yafu Li, Zefeng He, Siyuan Huang, Xiaoye Qu, Yu Cheng, and Yang Yang. LatentMem: Customizing latent memory for multi-agent systems. *arXiv preprint arXiv:2602.03036*, 2026. URL https://arxiv.org/abs/2602.03036v2.

Shiwei Gao, Youmin Chen, and Jiwu Shu. Fast state restoration in LLM serving with HCache. In *Proceedings of the Twentieth European Conference on Computer Systems (EuroSys '25)*, 2025. doi: 10.1145/3689031.3696072. URL https://arxiv.org/abs/2410.05004.

Tao Ge, Jing Hu, Lei Wang, Xun Wang, Si-Qing Chen, and Furu Wei. In-context autoencoder for context compression in a large language model. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/0b276510ec2d3f6613a8b60c41ff0438-Abstract-Conference.html.

Asma Ghandeharioun, Avi Caciularu, Adam Pearce, Lucas Dixon, and Mor Geva. Patchscopes: A unifying framework for inspecting hidden representations of language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, 2024. URL https://proceedings.mlr.press/v235/ghandeharioun24a.html.

In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt cache: Modular attention reuse for low-latency inference. In *Proceedings of Machine Learning and Systems*, volume 6, 2024. URL https://proceedings.mlsys.org/paper_files/paper/2024/hash/a66caa1703fe34705a4368c3014c1966-Abstract-Conference.html.

Roee Hendel, Mor Geva, and Amir Globerson. In-context learning creates task vectors. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 9318–9333. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.findings-emnlp.624. URL https://aclanthology.org/2023.findings-emnlp.624/.

IBM Granite Team. Granite 4.2 8B model card. Hugging Face model repository and model card, 2026. URL https://huggingface.co/ibm-granite/granite-4.2-8b/tree/7fce579a7fbbad4b1e7703b6850cefd517a4002b. Revision 7fce579a7fbbad4b1e7703b6850cefd517a4002b.

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.

Hanzuo Liu, Xuan Qi, Chunyu Liu, Haotian Zhong, Yulong Wang, Rayying, Key, Alex Lamb, and Mingyu Gao. Understanding is done early: A depth division of labor in large language models and its use for unbounded-context memory. *arXiv preprint arXiv:2607.28263*, 2026. URL https://arxiv.org/abs/2607.28263.

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, 2024. doi: 10.18653/v1/2024.acl-long.747. URL https://aclanthology.org/2024.acl-long.747/.

Jesse Mu, Xiang Lisa Li, and Noah Goodman. Learning to compress prompts with gist tokens. In *Advances in Neural Information Processing Systems*, volume 36, pp. 19327–19352. Curran Associates, Inc., 2023. doi: 10.52202/075280-0848. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/3d77c6dcc7f143aa2154e7f4d5e22d68-Abstract.html.

OpenAI. OpenAI MRCR: Long context multiple needle in a haystack benchmark. Hugging Face dataset openai/mrcr, 2025. URL https://huggingface.co/datasets/openai/mrcr.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T. Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. ReasoningBank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=jL7fwchScm.

Qwen Team. Qwen3.8-27B model card. Hugging Face model repository and model card, 2026. URL https://huggingface.co/Qwen/Qwen3.8-27B/blob/1d4bf0f2ff6012fd82039f2fa52739d0dd7c60c0/README.md. Revision 1d4bf0f2ff6012fd82039f2fa52739d0dd7c60c0.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In Jian Su, Kevin Duh, and Xavier Carreras (eds.), *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 2383–2392, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1264. URL https://aclanthology.org/D16-1264/.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7080–7106. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.eacl-long.333. URL https://aclanthology.org/2026.eacl-long.333/.

Eric Todd, Millicent L. Li, Arnab Sen Sharma, Aaron Mueller, Byron C. Wallace, and David Bau. Function vectors in large language models. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=AwyxtyMwaG.

Kiran Vodrahalli, Santiago Ontanon, Nilesh Tripuraneni, Kelvin Xu, et al. Michelangelo: Long context evaluations beyond haystacks via latent structure queries, 2024. URL https://arxiv.org/abs/2409.12640.

Yu Wang, Yifan Gao, Xiusi Chen, Haoming Jiang, Shiyang Li, Jingfeng Yang, Qingyu Yin, Zheng Li, Xian Li, Bing Yin, Jingbo Shang, and Julian McAuley. MEMORYLLM: Towards self-updatable large language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 50453–50466. PMLR, 2024. URL https://proceedings.mlr.press/v235/wang24s.html.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 63897–63911. PMLR, 2025. URL https://proceedings.mlr.press/v267/wang25bx.html.

Zijun Wu, Yongchang Hao, and Lili Mou. TokMem: One-token procedural memory for large language models. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=RWjEf9PdiJ.

Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. CacheBlend: Fast large language model serving for RAG with cached knowledge fusion. In *Proceedings of the Twentieth European Conference on Computer Systems (EuroSys '25)*, 2025. doi: 10.1145/3689031.3696098.

Amir Zandieh, Majid Daliri, Majid Hadian, and Vahab Mirrokni. TurboQuant: Online vector quantization with near-optimal distortion rate. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=tO3ASKZlok. Preprint arXiv:2504.19874.

Guibin Zhang, Muxin Fu, and Shuicheng Yan. MemGen: Weaving generative latent memory for self-evolving agents. In *International Conference on Learning Representations*, 2026a. URL https://openreview.net/forum?id=vI56m4Iu4e.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, 2026b. URL https://openreview.net/forum?id=eC4ygDs02R.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024. doi: 10.1609/aaai.v38i17.29936. URL https://ojs.aaai.org/index.php/AAAI/article/view/29936.

A Acquisition from experience: protocol and complete results

A.1 Environments and requests

Each environment draws its rule from 54 candidates, which differ in the order of the arguments to `prepare`, the flag passed to `commit` and the decision field that `commit` echoes. Fourteen episodes generated from the rule are rendered in a fixed order, with the first four showing `prepare` calls whose commit arguments were unresolved, the middle six showing rejected decisions without calls and the last four showing accepted `commit` calls. The environment's oracle confirms that the fourteen episodes identify the rule uniquely, that the early block alone identifies only the argument order and the late block alone only the commit convention (the flag and the echoed field), and that the middle block eliminates no candidate. We call the argument order and the commit convention the rule's two components, and in every one of the 128 evaluation environments the shortest span of the episode log that holds evidence for both is 343 Granite tokens long.

The 64 environment sets used for evaluation were generated and their hashes recorded before any model output was read. Each set contributes two environments, each environment receives two execution requests on inputs that no episode contains, one of them on a decision with a single field edited, and every condition answers the same requests.

A.2 Writer, reader and controls

The writer's request, the same in every environment, is

> Write a reusable procedure note for this environment that another instance of you could follow with no access to these interactions. State, in this order: (1) the exact call template for each tool, written with the literal argument names and placeholders in the exact order this environment requires; (2) every required field and the exact literal value of any required constant; (3) the corrections that were needed. Copy the surface form of the calls exactly as it appears. Be terse. Do not answer any other question.

After the episodes and this request the writer generates 224 tokens, and a second forward pass over the episodes, the request and the generated note captures the layer-1 residual states; the note memory keeps the states at the note positions. The memory written without the episodes comes from the same request with the episodes removed. Each episode window keeps 224 contiguous layer-1 states from a

separate forward pass over the episode log alone (the interface and the fourteen episodes, n tokens), starting at offset 0, $\lfloor (n-224)/2 \rfloor$ or $n-224$; its states are captured at their positions in the episodes and injected at the reader positions that hold the note states. Decoding is greedy, in batches of up to 16 with left padding, and the positions that receive the memory are set separately for each request. The reader may write a short reasoning prefix, and we score the last complete JSON array it emits.

A.3 Complete results

	Granite 4.2 8B	Qwen3.8-27B
episodes in context	235/256	206/256
note memory	213/256	195/256
no-episode memory	0/256	0/256
episode window, head	0/256	0/256
episode window, middle	0/256	0/256
episode window, tail	5/256	41/256
memory − best window	0.812 [0.750, 0.875]	0.602 [0.512, 0.688]
recovery	0.906 [0.856, 0.957]	0.947 [0.844, 1.049]

Table 3: Execution requests on the 64 environment sets, reading layer-1 states at 224 positions. Counts are requests that succeeded, recovery is the note memory's share of the gain from seeing the episodes, and intervals are 95% percentile intervals from 10,000 bootstrap resamples of environment sets. Qwen3.8-27B runs with NF4 weights and bfloat16 computation and decodes one request at a time.

Qwen3.8-27B notes. Where the Qwen3.8-27B note stated the argument order, in 99 of the 128 environments, the note memory succeeded on 191/198 requests, more than the 164/198 with the episodes in context.

A.4 Independent verification

We parsed the 1,536 scored Granite outputs of the execution requests with a separately written parser and replayed each through the environment's own transition system against expected actions re-derived by a second implementation of the oracle. The replay reproduced their counts in Table 3 with 0 disagreements, and in a mutation test on correct outputs it rejected all 1,812 single-field alterations.

A.5 Components carried by each window

Figure 6 breaks the results down by component on the 208 requests whose argument order differs from the order in which the interface lists the fields. The head and middle windows, which contain early episodes but no late ones, reproduced the argument order on 208/208 and 206/208 of these requests and the commit convention on none, while the tail window, which contains only late episodes, reproduced the commit convention on 200/208 and the argument order on 23/208. The note memory reproduced the argument order on 208/208 and the commit convention on 170/208.

A.6 What the notes state

The notes gave the correct argument order in all 128 environments and named the echoed field in 107, and in those the note memory succeeded on 212/214 requests against 211/214 with the episodes in context. In the other 21, all of which echo priority, the note described the echo as an assigned reference number or index, and there the note memory succeeded on 1/42 requests against 24/42.

Placement comparison. Figure 5 uses the first 16 environment sets in each order. The notes for the original and the reordered logs were generated on CPU in bfloat16 from the same fixed request, and the same reader read every memory on CPU with greedy decoding, in batches of 8 and at most 256 generated tokens; the one exception is the original-order middle window, which comes from the 64-set GPU evaluation of the same requests. On 24 development environments, CPU reads of notes

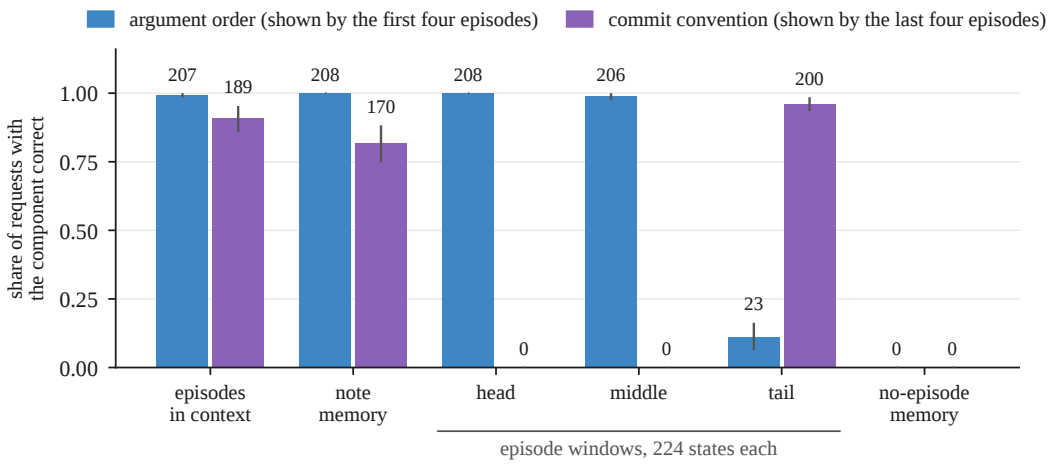

Figure 6: Read at the note memory's positions, each episode window carried the rule component whose episodes it contains, and the note memory carried both. Component 1 is the `prepare` argument order, shown by the first four episodes; component 2 is the `commit` flag and echoed field, shown by the last four. Every memory holds 224 layer-1 states read with the same prompt. Results are for Granite 4.2 8B on 64 environment sets and the 208 requests whose argument order differs from the order in which the public interface lists the fields; labels give the number of correct requests, and bars show 95% intervals over sets.

written on GPU reproduced the GPU outcome on 48/48 requests. All episode windows are cut from one pass over the episodes, and the split window joins the 112 states that follow the interface header to the last 112 states.

A.7 Policy acquisition and composition of acquired memories

Each environment set also contains two policy environments, whose public interface gives the amount and priority formulas and whose hidden rule, drawn from 81 candidates, assigns one of the routes `std`, `exp` and `hold` to each combination of record class and urgency band. The fourteen episodes are laid out as in the procedure environments: the first four resolve the low-band routes, the last four resolve the high-band routes, and the six between them are rejected decisions that rule out no candidate. The writer receives a second fixed request, the same in every policy environment:

> Write a reusable policy note for this environment that another instance of you could follow with no access to these interactions. State, in this order: (1) the exact rule that selects the route, including the exact threshold value and the behaviour exactly at it; (2) every exception and the exact condition that triggers it; (3) the exact formula for the amount and for the priority; (4) the corrections that were needed. Be terse and literal. Do not answer any other question.

We write, capture and read the note exactly as in Appendix A, keeping the layer-1 states at 224 note positions, and we left-pad writer requests with placeholders to a common length so that each note occupies a fixed slot. Each environment receives two decision requests on records that no episode contains, one of them an edit of a single field, and a decision succeeds only when route, amount and priority are all exact (the amount depends on the route). Table 4 gives every condition.

For composition, each of the 64 sets supplies four workflow units, and each unit pairs its two policy environments with its two procedure environments in four ways. The reader receives the public interfaces of the policy and the procedure environment and one canonical record and has to emit the `prepare` and `commit` calls that the paired policy and procedure determine together; the request text is the same under all four pairings, but every pairing has a different correct pair of calls. No policy episode contains a workflow record, and no procedure episode contains the decision for one. With separate memories the reader holds the policy memory and the procedure memory, each at its own slot; with the episodes in context it sees both episode logs; with no-episode memories both slots hold memories written without the episodes; and the two single-component controls, run at

two prespecified opposite pairings, replace one memory with its no-episode counterpart. Paired differences are taken over the same units and pairings.

Policy acquisition (256 decisions)	Count	Rate [95% CI]
Policy memory	170/256	0.664 [0.602, 0.727]
Episodes in context	130/256	0.508 [0.445, 0.570]
No-episode memory	70/256	0.273 [0.230, 0.316]
Episode window, head	89/256	0.348 [0.293, 0.402]
Episode window, middle	92/256	0.359 [0.297, 0.426]
Episode window, tail	113/256	0.441 [0.387, 0.500]
Memory − episodes in context		0.156 [0.062, 0.250]
Memory − best window		0.223 [0.141, 0.301]
Composition (1,024 workflows)		
Separate policy and procedure memories	438/1024	0.428 [0.380, 0.478]
Episodes in context, both records	202/1024	0.197 [0.165, 0.230]
No-episode memories	0/1024	0.000 [0.000, 0.000]
Without policy memory	128/512	0.250 [0.213, 0.289]
Without procedure memory	0/512	0.000 [0.000, 0.000]
Separate − episodes in context		0.230 [0.194, 0.267]
Separate − without policy		0.189 [0.146, 0.231]
Separate − without procedure		0.439 [0.399, 0.480]

Table 4: Granite 4.2 8B on the 64 environment sets. Intervals are 95% percentile intervals from 10,000 bootstrap resamples of environment sets.

B Implementation and statistical details

B.1 Checkpoints and execution

Evaluations on Granite 4.2 8B, Qwen3.5-27B and Qwen3.5-9B use these repositories at pinned revisions:

- `ibm-granite/granite-4.2-8b`, revision `7fce579a7fbbad4b1e7703b6850cefd517a4002b`.
- `Qwen/Qwen3.5-9B`, revision `c202236235762e1c871ad0ccb60c8ee5ba337b9a`.
- `Qwen/Qwen3.5-27B`, revision `fc05daec18b0a78c049392ed2e771dde82bdf654`.

MRCR, LongBench, DROP, SQuAD, the twenty-cycle revision and the evaluations of memory written from experience on Qwen3.8 use `Qwen/Qwen3.8-27B`, revision `1d4bf0f2ff6012fd82039f2fa52739d0dd7c60c0`. The Qwen model cards are at `https://huggingface.co/Qwen/Qwen3.5-9B` and `https://huggingface.co/Qwen/Qwen3.5-27B`, and the supplement records the pinned revisions and tokenizer hashes. For Qwen we run only the text decoder, with the gated-delta recurrent blocks and full-attention blocks that the checkpoint's implementation supplies. Runs on these three checkpoints use bfloat16 weights, scaled dot-product attention where it applies and the recorded native fallback implementations.

The writer tokenizes the fixed prefix and the history separately, with the same segment tokenization in every condition, and then captures one forward pass over prefix and history with no question suffix. To read a multiple-choice answer we sum, for each letter, the next-token probability of its bare and leading-space forms, counting only forms that are a single token and each token ID once, and choose the letter with the largest sum. In batched reading we verify, for every example that carries a memory, that the activations at its placeholder positions are overwritten exactly once. When the reader generates, it overwrites the activations with the memory during prefill only and then decodes without overwriting, from a fresh generation cache.

The fixed reference header is:

```
Use the reference memory to answer future requests.  Apply
its facts, procedures, and output conventions exactly.

Reference memory:
```

Two newline characters separate the two displayed lines, and the header ends with one newline. We render one user message holding this header, a boundary marker and `\n\nQuestion:\n` through the checkpoint's chat template with `add_generation_prompt=True` and `enable_thinking=False`, split the result at the marker and take the rendered part before it as the fixed prefix; each segment is tokenized with `add_special_tokens=False`. The question suffix inserts this template at the rendered question boundary:

```
{request}
A. {choice_A}
B. {choice_B}
C. {choice_C}
D. {choice_D}
Reply with only the correct option letter, A, B, C, or D.
```

Its last line ends with one newline before the checkpoint's closing and assistant-generation delimiters, and if the chat template strips trailing whitespace from a message, the implementation restores the newline after `Question:`.

B.2 Four-bit code

The four-bit code works on groups of 64 coordinates. A group's scale is its maximum absolute value divided by seven, floored at 10^{-7} and cast to float16, and we divide each coordinate by this stored scale, round it to the nearest integer with ties to even, clip it to $[-7, 7]$ and add seven, packing consecutive codes into the low and high nibbles of each byte.

B.3 Capture depth

Every experiment captures at layer 1, and for this comparison we also wrote bf16 memories at layer 8, at every position of the same 128 test handbooks, and read them with the same 1,280 four-choice questions as the layer-1 memories of Section 3.2 (Table 5).

bf16 memory, every position	Granite 4.2 8B	Qwen3.5-27B	Qwen3.5-9B
Full history	79.5 [77.5, 81.6]	95.9 [94.8, 97.0]	87.8 [85.9, 89.7]
Layer 1	78.8 [76.8, 80.9]	96.4 [95.3, 97.4]	87.3 [85.5, 89.2]
Layer 8	65.0 [62.6, 67.5]	95.8 [94.5, 97.0]	32.6 [30.2, 35.1]
Layer 8 − layer 1 (points)	−13.8 [−16.0, −11.6]	−0.6 [−1.6, +0.2]	−54.8 [−57.4, −52.1]

Table 5: Accuracy (%) of bf16 memory written at layer 1 and at layer 8, at every position of the 128 test handbooks, with 95% bootstrap intervals over handbooks. The last row is the paired difference in percentage points.

B.4 Sampling and uncertainty

We resample each experiment's independent units with their questions kept together: 32 test sets for the composition of fact and procedure memories, each with two question types and four pairings; 16 trajectories on Granite and four on Qwen3.5-27B for ten-cycle revision and 16 for the twenty-cycle revision on Qwen3.8-27B, with all cycles of a trajectory kept together; and eight complete conversations for LoCoMo.

Unless an evaluation specifies otherwise, bootstrap intervals use 50,000 resamples with seed 20260918.

The conditions and analysis of every evaluation were fixed on development data before the evaluation data were run. The released manifests give the membership of each sample and the source assigned

to each condition, every sampling unit contributes all of its scheduled questions, and hashes of the saved memories verify that the same bytes were read for every question and every pairing.

C A COMPLETE HANDBOOK WITH ITS QUESTIONS

The first test handbook shows the procedural structure every handbook shares and its own randomized bindings, with its blocks in their sampled order; the writer receives it without the questions in Table 6.

> This is the current operations handbook for a fictional service. All names, codes, functions, and routing assignments are local conventions. Use these records for later requests. Each case has its own record. Unmentioned real-world practices do not change this handbook. The rules below apply to new cases as well as registered cases.
>
> Worked routing examples: unsealed with 1 units gives pivmez; unsealed with 10 units gives woxhiq; sealed with 1 units gives jezmez. These examples illustrate the policy. They do not override the registered case records.
>
> Case kufmez: destination rizqor; permit jezqor; 6 units; status sealed. The destination and permit are distinct fields. Use the permit, not the destination, to authorize a function call.
>
> Case jezsud: destination kufwox; permit qorriz; 2 units; status unsealed. The destination and permit are distinct fields. Use the permit, not the destination, to authorize a function call.
>
> Routing exception: every sealed request takes route jezmez, regardless of its unit count. Check the sealed status before applying the numerical threshold. Route hiqpiv is reserved and is not assigned by the current rules.
>
> Function table: inspect uses nopqor; archive uses jezlaz; dispatch uses qorpiv; cancel uses kufpiv. Every function takes exactly these named arguments in this written order: sudriz="the case value"; then rizmez="the permit value". Put double quotes around each value and separate the arguments with a comma and one space. Use the requested operation, case identifier, and permit value. Do not substitute a destination for a permit.
>
> Case hiqwox: destination lazsud; permit rizhiq; 5 units; status sealed. The destination and permit are distinct fields. Use the permit, not the destination, to authorize a function call.
>
> Case sudlaz: destination qorwox; permit lazkuf; 7 units; status unsealed. The destination and permit are distinct fields. Use the permit, not the destination, to authorize a function call.
>
> Routing policy: an unsealed request with fewer than 5 units takes route pivmez. An unsealed request with 5 or more units takes route woxhiq. The threshold is inclusive for the second route. Choose exactly one route.
>
> Output preference: when asked for a label, the exact template is SECOND|FIRST. FIRST and SECOND are placeholders for the supplied strings. Copy each supplied value exactly, preserve the stated field order, and use the displayed separator. Do not add spaces, quotation marks, a heading, or explanatory prose to the label.

Request	Exact answer
Consult the case records: jezsud has which destination?	`kufwox`
Process a fresh request: units=9, status=unsealed. Select the route dictated by the handbook.	`woxhiq`
Execute cancel using case identifier woxmez and permit value tevhiq. Which recorded call syntax applies?	`kufpiv( sudriz="woxmez", rizmez="tevhiq")`
Construct the prescribed label with the two inputs FIRST=kufriz; SECOND=76.	`76|kufriz`
Resolve the route for registered case hiqwox by combining its record with the handbook's rule.	`jezmez`

Table 6: One question of each of the five types for the example handbook, with the solver's exact answer. The four-choice evaluation presents each answer among fixed alternatives.

The test split's nonce vocabularies and question templates are disjoint from those of every split used during development, while the same kinds of routing rules, tool bindings and label operations occur in all splits.

D REUSE RESULTS ON THE HANDBOOKS

Both bf16 and four-bit memory keep the layer-1 states at every source position, and each memory is written from its handbook alone and read unchanged for all ten of that handbook's questions. Tables 7 and 8 give aggregate accuracy and paired differences from full history, and Table 9 breaks accuracy down by the five question types.

Memory	Granite 4.2 8B	Qwen3.5-27B	Qwen3.5-9B
Full history	79.5	95.9	87.8
No history	26.0	25.5	26.2
bf16 memory	78.8	96.4	87.3
Four-bit memory	79.1	96.3	87.3

Table 7: Accuracy of bf16 and four-bit layer-1 memory on the test handbooks, in percent over 128 handbooks and 1,280 questions per checkpoint. The full-history and no-history controls answer the same questions.

Difference from full history	Granite 4.2 8B	Qwen3.5-27B	Qwen3.5-9B
bf16 memory	-0.7 [-1.6, +0.2]	+0.5 [+0.2, +0.9]	-0.5 [-1.8, +0.9]
Four-bit memory	-0.4 [-1.3, +0.5]	+0.4 [-0.1, +0.9]	-0.5 [-1.8, +0.9]

Table 8: Paired differences from full history for bf16 and four-bit layer-1 memory, in percentage points, with 95% bootstrap intervals over handbooks.

Model	Question type	Full history	No history	bf16 memory	Four-bit memory
Granite 4.2 8B	retrieval	99.6	30.1	99.6	99.2
Granite 4.2 8B	rule	98.8	26.6	98.8	98.8
Granite 4.2 8B	tool	73.8	25.4	73.4	75.0
Granite 4.2 8B	format	50.0	23.8	47.7	48.0
Granite 4.2 8B	composition	75.4	24.2	74.6	74.6
Qwen3.5-27B	retrieval	100.0	26.2	100.0	99.6
Qwen3.5-27B	rule	100.0	23.0	100.0	100.0
Qwen3.5-27B	tool	99.2	27.7	99.2	99.2
Qwen3.5-27B	format	82.4	26.2	84.8	84.8
Qwen3.5-27B	composition	98.0	24.2	98.0	98.0
Qwen3.5-9B	retrieval	100.0	22.3	99.6	99.2
Qwen3.5-9B	rule	99.6	24.2	99.6	99.2
Qwen3.5-9B	tool	89.8	32.0	90.2	90.2
Qwen3.5-9B	format	64.1	24.2	66.0	65.2
Qwen3.5-9B	composition	85.5	28.5	81.2	82.8

Table 9: Accuracy by question type, in percent, over 128 handbooks and 256 questions for each checkpoint and question type.

E GENERATION, COMPOSITION, REVISION AND CONVERSATIONS

E.1 GENERATION SCORING

The generation test uses 32 of the test handbooks, six questions for each, and fixed examples of individual tool operations and label formats. Decoding is greedy with a 64-token cap, and an answer counts as correct when it equals the solver's answer exactly after whitespace is collapsed; no semantic judge is used.

Every emitted output stays in the results, including any that reached the token cap.

Memory	Granite 4.2 8B	Qwen3.5-27B	Qwen3.5-9B
Full history	165/192	186/192	178/192
No history	0/192	0/192	0/192
bf16 memory	163/192	185/192	180/192

Table 10: Correct answers under every generation condition, counted by whitespace-normalized exact match over 192 questions per checkpoint.

E.2 CONTROLS FOR SEPARATELY WRITTEN FACT AND PROCEDURE MEMORIES

Fact and procedure memories share one position layout, which fixes the positions each memory occupies when it is read. The procedure writer receives period-token padding in place of the fact text, as many tokens as the longest fact span in its test set, so it learns that length, which is shared layout metadata, and none of the factual content. The fact and procedure memories are therefore independent in content, conditional on this layout, and we compare each saved memory's hash across pairings and questions. The reference conditions supply the full text, write both memories together, or keep only one of the two.

Memory	Granite answers	Qwen answers	Granite four-way	Qwen four-way
Full history	181/256	239/256	28/64	50/64
Written together (bf16)	176/256	234/256	28/64	45/64
Written separately (bf16)	174/256	235/256	28/64	45/64
Fact memory only	97/256	94/256	0/64	0/64
Procedure memory only	69/256	65/256	0/64	0/64

Table 11: Correct answers and four-way tests passed (all four pairings answered correctly), with 256 answers and 64 four-way tests per condition clustered in 32 independent test sets; Qwen here is Qwen3.5-27B.

E.3 REVISION TRAJECTORIES

Each correction contains 18–26 tokens and changes a short part of the handbook, and some corrections reverse an earlier one. Both state memory and the text-rewriting reference have a 1,024-position memory cap and a 768-token generation cap. For state memory the model reads the previous memory and the correction and generates the replacement text, whose states the writer then stores; the text-rewriting reference reads its own previously generated text and the same correction. No input is truncated, and rewrites that reach the cap are kept.

Model	Full history	Revised text	Memory
Granite 4.2 8B	143/160	141/160	142/160
Qwen3.5-27B	39/40	39/40	39/40

Table 12: Correct answers after ten revisions through text. The full-history, text-rewriting and state-memory conditions use the same questions and the same fixed rewrite instructions.

The twenty-cycle revision on Qwen3.8-27B uses 16 new trajectories from the fixed handbook generator and asks ten questions at the initial state and after every correction. The four-bit memory and text-rewriting conditions each revise their own predecessor, and the four-bit update starts from the decoded four-bit memory it stored at the previous cycle; no revision or read sees the original history. Intervals come from 50,000 resamples of whole trajectories with seed 20260920.

	All cycles	Cycle 20			
Condition	Correct/3,360	All/160	Changed/28	Unmentioned/65	Other/64
Current full history	3,291	155	28	62	62
Four-bit memory	3,244	155	28	61	63
Revised text	3,255	155	28	62	62

Table 13: Revision of procedural handbooks on Qwen3.8 over 16 independent trajectories and twenty corrections. Counts over all cycles include cycle zero, and the last three columns count the cycle-20 answers about newly changed facts, unmentioned information in the edited part and the other part.

E.4 Complete conversation histories

The held-out LoCoMo evaluation uses eight complete conversations with 12 questions each. We took the questions from official categories 1, 2 and 4 (multi-hop, temporal and single-hop) to target recall that depends on the conversation, and within each category we selected the four questions with the smallest SHA-256 values of a fixed salted question identifier, whose salt the supplement records. The official scorer computes token-overlap F1 after normalization and stemming. For category 1 it splits comma-separated answers and averages each reference item's best match to a predicted item, while categories 2 and 4 use token F1 over the whole answer. The conversations were generated by agents and verified by humans.

Conversations are 21,683–29,724 Qwen3.5-27B tokens long (mean 27,130.75). No conversation is truncated, bf16 and four-bit memories keep every position, and generation is capped at 64 tokens with every output scored as generated.

Memory	Official F1 $\times 100$	Difference, 95% interval
Full history	55.7	–
No history	0.0	-55.7 [-62.6, -47.9]
bf16 memory	55.4	-0.3 [-1.0, +0.5]
Four-bit memory	55.9	+0.2 [-1.3, +2.2]

Table 14: Official LoCoMo score of Qwen3.5-27B under its category-specific rules, multiplied by 100, and paired differences from full history. Confidence intervals resample the eight conversations and keep all 12 questions within each.

E.5 Complete MultiFieldQA-en evaluation

We evaluated all 150 official MultiFieldQA-en questions from LongBench (Bai et al., 2024) with the complete source contexts, the official English QA prompt and the 64-token greedy generation cap. The 150 questions draw on 112 distinct contexts, and the writer stored each context once, before any evaluation question was read, so every question about that exact context read the same saved memory; 29 contexts are asked about more than once, by 67 questions in all. No history, full history, bf16 memory and four-bit memory all received the same question suffix.

Scoring used the official English QA normalization unchanged and took the maximum token F1 across references, and the reported task score is the mean over questions multiplied by 100. Before estimating intervals, we grouped the 112 context strings into 100 source documents by exact context hash and by overlapping five-word shingles, and these documents are the sampling units. The paired bootstrap resamples source documents and in each draw divides the summed question scores by the number of questions drawn, which keeps the estimate on the question-weighted task mean.

Every output was scored as generated, including those that reached the generation cap, and all 150 questions stayed in every condition. The intervals use 50,000 bootstrap draws of source

Memory	Granite 4.2 8B	Qwen3.8-27B
Full history	45.57	57.72
bf16 memory	45.55	57.94
Four-bit memory	45.22	58.07
No history	13.55	5.33

Table 15: Official MultiFieldQA-en F1 over all 150 questions on each checkpoint. Sources are kept whole and span 1,344–15,790 Granite tokens and 1,320–16,746 Qwen3.8-27B tokens. Qwen3.8-27B runs with NF4 model weights and bfloat16 computation.

documents with seed 20260920. We pinned the source data to `zai-org/LongBench` revision `5e628be450b7e67fb7ae6e201bd6d8f7056f7672`, and the supplement includes the exact prompt, the grouping of contexts into source documents and the official scoring functions.

E.6 MRCR PROTOCOL

We used the two- and eight-needle files of the public `openai/mrcr` release (snapshot `f4c69fae`) and the benchmark's length bins, which count the tokens of the conversation and the answer with the o200k_base tokenizer. Within each needle count and bin we took the first conversations in file order, which gave 30 eight-needle and 15 two-needle conversations in the 64k–128k bin and 20 eight-needle and 15 two-needle conversations in each of the 32k–64k and 8k–16k bins. The history is every message but the last, rendered with the model's chat template with thinking disabled, and the final turn is rendered as its continuation; all conditions share these token sequences, except that the final-turn-only condition renders the last message alone.

The writer prefills the history alone and stores its layer-1 states at every history position. Prefill runs in chunks of 4,096 tokens, falling to 2,048 above 112,000 tokens and to 1,024 above 125,000 tokens; every condition of a conversation uses the same chunk boundaries, and the final turn is a single chunk. Full history continues the writer's own key/value cache with the final turn, whereas the memory readers start from a fresh cache with a period token at each history position and overwrite the layer-1 activations at those positions with the stored states during prefill. Decoding is greedy and stops at the end-of-turn token or after the reference length plus 64 tokens, and in every condition leading whitespace is removed before the official grader scores the answer.

A memory answer counts as identical to the full-history answer when the two greedy responses match as strings after leading and trailing whitespace is removed. The paired interval resamples conversations 10,000 times with seed 20260920.

F COMPOSITION AND REVISION ON NATURAL TEXT

F.1 PASSAGE AND QUESTION MEMORIES ON DROP

The test takes its passages from DROP unchanged and keeps the human wording of the questions. We selected pairs of passages that share two questions, kept a pair only if its four combinations of passage and question have four distinct integer answers, and made the pairs passage-disjoint, all before any model was run; four training pairs were used for development. The evaluation uses 24 pairs from the official development split, which keep their official labels, and 28 disjoint pairs from the official training split, whose integer labels were adjudicated directly against the source passage before inference; the pairs from the two splits stay identifiable, and the combined interval resamples pairs separately within each split. All four combinations of passage and question are scored, with greedy decoding and a 24-token cap. An answer counts as correct when the whole output is the correct integer, whether bare, with thousands separators, as an integer-valued decimal or with a final period, and a correct number inside an explanation is not extracted.

Every memory goes into one global position layout that reserves 2,048 positions for the passage and 128 for the request. Each writer receives only the fixed prefix, its own component and that component's fixed offset, with period tokens occupying the positions before the component, and the reader puts placeholders in both slots, also when one component is absent. A component's hash is the

same with every partner it is read with. All memories hold layer-1 states, and confidence intervals use 10,000 bootstrap draws over pairs with seed 20260920.

Memory	Answers	Four-way
Full history	194/208	41/52
Written together (bf16)	192/208	39/52
Written separately (bf16)	193/208	39/52
Written separately (four-bit)	196/208	41/52
Passage memory only	14/208	0/52
Question memory only	5/208	0/52
No history	0/208	0/52

Table 16: Numerical answers generated by Qwen3.8-27B when DROP passage and question memories are recombined, over 52 passage pairs and 208 answers per condition. Four-way counts the tests that pass when all four pairings are answered correctly. Qwen3.8 runs with NF4 weights and bfloat16 computation, and the memory format is varied separately.

Contiguous position layout. We also read the Qwen3.8-27B memories of the 24 pairs from the official development split, unchanged, in a contiguous position layout, where the passage starts immediately after the fixed prefix, the request follows the passage and the query follows the request, so each component's positions, its partner's positions and the query positions all move together. The contiguous full-text reference uses the same layout, and the 166 stored memories have the same hashes in the fixed and contiguous reads. Intervals use 10,000 paired resamples of the 24 passage pairs.

Read condition	Answers	Four-way
Contiguous full text	82/96	14/24
Written separately (bf16), contiguous	83/96	14/24
Written separately (four-bit), contiguous	83/96	14/24
Passage memory only, contiguous	7/96	0/24
Question memory only, contiguous	4/96	0/24
No history	0/96	0/24

Table 17: The same separately written Qwen3.8-27B memories read in the contiguous position layout, over 24 pairs and 96 answers.

F.2 Revising SQuAD paragraphs

The passages and questions come from SQuAD v1.1, and the corrections are controlled edits to that natural text. A fixed hash order selects one paragraph per article, with the four development and 24 evaluation articles kept disjoint, and each selected paragraph has two editable four-digit years among its answers and two other non-overlapping answer spans. The ten corrections alternate between the two editable years and include reversions. Each replacement keeps the year's numerical order relative to every integer in the passage, differs from every other value in the source and leaves the question wording unchanged, and the current reference passage is built by exact span substitution.

At each update the reader receives its own previous memory and one correction, with no evaluation question, and generates a revised paragraph whose layer-1 states the writer stores as the next memory. Every condition caps memory at 1,024 positions, the rewrite at 512 tokens and the answer at 32 tokens, and all four questions are asked at the initial state and after every correction. The bf16, four-bit and text-rewrite trajectories each keep their own predecessors, so the four-bit trajectory packs and decodes its own states before each update. Full history supplies the current reference passage, while the no-history condition generates each greedy answer once and scores that unchanged output against the current answer at every cycle. We score exact match and token F1 with the official SQuAD functions, and intervals use 50,000 resamples of articles with seed 20260920, keeping all cycles and questions of an article together.

Memory	All cycles	F1	Cycle 10	Edited	Untouched
Current full history	935/1056	95.37	85/96	48/48	37/48
bf16 memory	935/1056	95.37	85/96	48/48	37/48
Four-bit memory	925/1056	95.20	84/96	48/48	36/48
Revised text	935/1056	95.37	85/96	48/48	37/48
No history	290/1056	31.12	38/96	22/48	16/48

Table 18: Revision of SQuAD paragraphs on Qwen3.8-27B across 24 independent articles, with NF4 weights independently of the memory format. All-cycle counts cover cycle zero and the ten updates, F1 is multiplied by 100, and at cycle 10 "Edited" and "Untouched" separate the questions about the two revised facts from the two questions whose answers never changed.

F.3 Composing revised memories

We read the bf16 and four-bit memories that the 24-article revision experiment produced at cycles 0, 1, 5 and 10, loading each fact memory from its own revision chain without rewriting it or capturing it again. Each of an article's four human questions was written once into a separate question memory for each format, with no passage, correction or current answer, so the 192 question memories are identical across all four cycles. The reader reserves 1,024 positions for the fact memory and 128 for the question memory, adds a fixed instruction to execute the stored request and caps the answer at 32 tokens, and when both components are memories no source passage, correction text or question text is in context.

All conditions share the layout, the questions and the current answers. The conditions that give the question as text test the revised fact memory without composing it with a question memory, and the single-component controls keep both slot widths. Every article contributes 16 answers per condition, newly corrected questions are counted at cycles 1, 5 and 10, and exact match and F1 are scored as in the preceding experiment.

Memory	All reads	Changed	Cycle 10	Edited	Untouched
Full history	336/384	72/72	84/96	48/48	36/48
bf16 fact memory, question text	336/384	72/72	84/96	48/48	36/48
Four-bit fact memory, question text	336/384	72/72	84/96	48/48	36/48
bf16 fact and question memories	334/384	72/72	83/96	48/48	35/48
Four-bit fact and question memories	332/384	72/72	83/96	48/48	35/48
bf16 fact memory only	26/384	8/72	6/96	3/48	3/48
Four-bit fact memory only	32/384	8/72	8/96	4/48	4/48
bf16 question memory only	102/384	20/72	30/96	19/48	11/48
Four-bit question memory only	109/384	21/72	32/96	20/48	12/48

Table 19: Revised fact memories combined with separately written question memories on Qwen3.8-27B, over 24 articles, four questions and four evaluated cycles. "Changed" counts newly corrected facts at cycles 1, 5 and 10, and "Edited" and "Untouched" separate revised from never-changed facts at cycle 10. Weights are NF4, independently of the memory format.