MEMPO: OPTIMIZING MEMORY WRITE POLICIES WITH RETRIEVAL-ATTRIBUTED CREDIT

Anonymous authors
Paper under double-blind review

ABSTRACT

Persistent memory allows LLM agents to use information from interactions that extend far beyond the context window. Existing memory systems, however, typically rely on fixed write rules that decide what to retain, summarize, merge, or discard before knowing which information future queries will need. Our analysis shows that no single retention strategy is sufficient: the utility of a write is sparse, often revealed only many steps later, and increasingly difficult to recover from trajectory-level feedback as the write-to-query horizon grows. We introduce MEMPO, a framework for optimizing memory write policies with *retrieval-attributed credit*. For each query, MEMPO attributes reward among the retrieved memory items using Shapley values, then traces each item's contribution through its provenance to the write decisions that produced it. The resulting localized returns train a policy to decide how much of each incoming segment should survive in memory. By assigning credit through the retrieved set, MEMPO ties each write's learning signal to whether it later provides useful evidence, rather than to its temporal distance from the query. Across six benchmarks spanning conversational memory, belief revision, and long-document question answering, MEMPO consistently outperforms fixed-rule and learned-memory baselines, raising LoCoMo average F1 from 39.5 to 44.7 against the best hand-designed policy over the same write operations.

1 INTRODUCTION

LLM agents often need to use information from past interactions and experiences to solve future problems, whether that is a preference a user stated weeks ago, a fact read in an earlier document, or the outcome of an attempt the agent already made [28, 35, 37, 45]. Since these interactions quickly exceed any context window, agents keep a persistent memory and retrieve from it when a query arrives [27, 44, 46]. A central question is what information should be preserved in memory for future use.

Existing approaches broadly follow two directions, each with a distinct limitation. One line of work retains the original history and leaves the selection to retrieval time [10, 18, 22]. However, as the memory store grows, relevant evidence becomes harder to retrieve reliably from an increasingly large collection, and the items that do reach the reader include more that the query does not need [23, 42]. Another line of work constructs persistent memory using fixed write rules to summarize or compress incoming information [5, 21, 27, 31, 41]. Yet these write decisions are made before future queries are known, and may therefore discard details that later become important. Figure 1a shows both failures at once, with accuracy peaking at an intermediate retention ratio and dropping on either side of it.

Considering that the right decision differs across segments rather than across streams (Figure 1b), we therefore introduce **Memory Policy Optimization** (MEMPO), which learns a write policy from downstream query feedback. We divide the incoming experience stream into small segments and make a write decision as each segment arrives. The policy adaptively determines how much information from each segment should be retained based on its expected future utility. Optimized for future query performance, MEMPO learns both what information to keep and how much detail to preserve.

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.

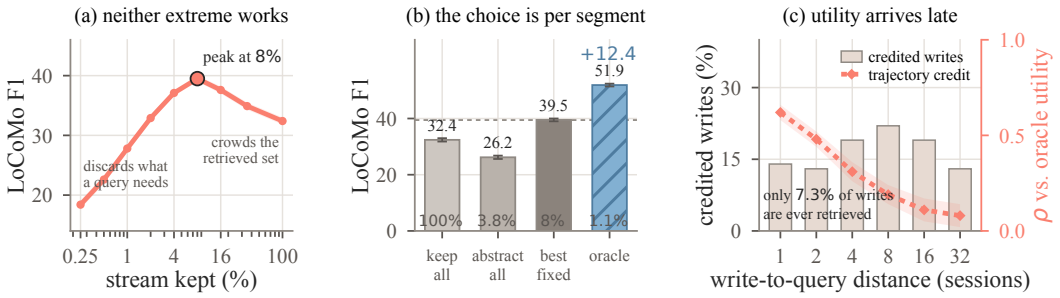


Figure 1: **A per-segment write decision recovers accuracy that no fixed rule reaches, and the feedback that identifies it arrives long after the write.** **(a)** LoCoMo F1 as a fixed write rule is swept from keeping 0.25% of the stream to keeping all of it. Accuracy peaks at 8%, because writing less discards what a query needs and writing more crowds the retrieved set. **(b)** Four ways of deciding what to write, namely keeping every segment, abstracting every segment, the best fixed retention ratio from (a), and an oracle, with the deployed store printed inside each bar. The oracle, which chooses the operation per segment with the future queries in hand, reaches 51.9, which is 12.4 points above the best fixed ratio. **(c)** Bars give the share of retrieved evidence that a query uses that many sessions after the write, and the line gives the Spearman correlation between a trajectory-level return and the true utility of a write at that distance. The utility of a write is sparse and arrives late, and a trajectory-level return stops tracking it.

Learning this policy is a long-horizon credit assignment problem. The policy acts once per segment, while reward arrives only when a later query is answered from the memory that these writes have produced. Maximizing the total query reward therefore requires attributing the reward of a query to writes that may lie thousands of segments behind it, which is the delayed-reward setting that return decomposition and hindsight methods address in reinforcement learning [1, 12]. Figure 1c shows that the utility of a write is sparse and revealed late, and that a trajectory-level return stops tracking it as the write-to-query distance grows.

To address this challenge, we design *retrieval-attributed credit*, a credit assignment mechanism that exploits the structure of retrieval-based memory. Since each query is answered using only a small retrieved subset of memory, we first attribute the query reward among the retrieved items using Shapley values [33], and then trace each item's contribution through its provenance to the write decisions that produced it. This allows us to assign reward only to the writes that could have influenced the query, rather than to the entire preceding trajectory. As a result, *retrieval-attributed credit* transforms a long-horizon learning signal into localized write-level credit, substantially reducing the dependence of credit quality on the temporal distance between writing and feedback. We then train the policy on these localized returns with a group-normalized, critic-free policy gradient [32].

Across six benchmarks spanning long-term conversational memory, belief revision, long-document question answering, and histories up to 2M tokens, MEMPO consistently improves over fixed-rule and learned-memory baselines. On LoCoMo, it reaches 44.7 average F1, compared with 39.5 for the strongest hand-designed policy over the same operations (Tables 1 and 3). The same policy transfers without retraining beyond its conversational training domain: it raises LongBench-QA from 45.9 to 53.3 and remains effective on DialSim streams roughly 38× longer than those seen during training (Section 3.2). The credit assignment is essential to these gains: replacing *retrieval-attributed credit* with a trajectory-level return lowers LoCoMo F1 by 10.1 points, and as history length grows from 8K to 2M tokens, the resulting gradient variance increases by 248×, compared with only 1.2× for *retrieval-attributed credit* (Table 2). These results show that learning what to write can produce a compact memory without sacrificing downstream utility, while retrieval-based attribution keeps the learning signal usable over long write-to-query horizons.

2 MEMPO: LEARNING A WRITE POLICY FOR AGENT MEMORY

MEMPO learns how each incoming experience should be written from its downstream use, where existing systems apply a fixed rule (Figure 2). A write policy processes the experience stream sequentially and decides how each segment updates the current memory (Section 2.1). When a query is later answered, *retrieval-attributed credit* assigns the query reward to the retrieved memory

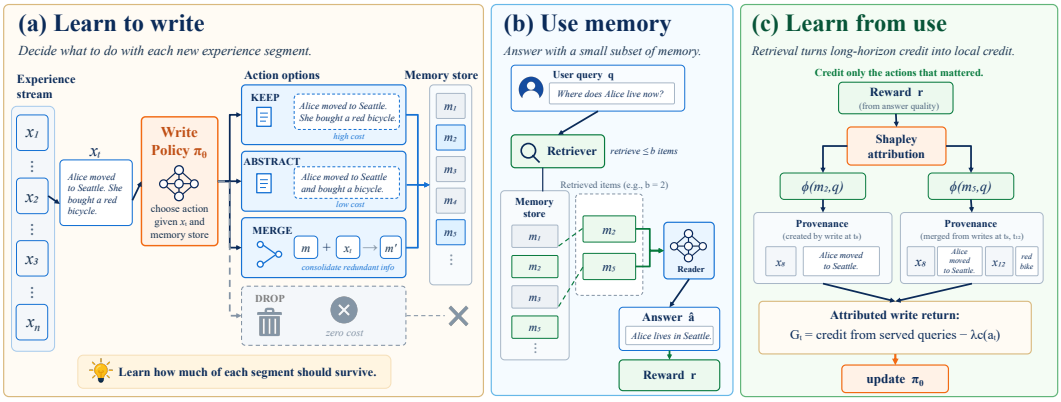


Figure 2: **Overview of MEMPO. (a)** The write policy π_θ reads each incoming segment x_t together with the current memory store, and selects one of four operations that differ in how much of the segment survives. KEEP stores x_t verbatim, ABSTRACT stores a shorter model-generated version of it, MERGE consolidates it with a nearby memory item into a single item, and DROP leaves the memory unchanged. The decision is made per segment, and before any query is seen. **(b)** A frozen retriever returns at most b memory items for a query q, and a frozen reader answers from those items alone, so the reward r grades the memory the write policy left behind. **(c)** *retrieval-attributed credit* splits r among the retrieved items by their Shapley values $\phi(m, q)$, and then traces each item's contribution through its provenance to the writes that produced it, so an item written by KEEP or ABSTRACT returns its credit to a single write, while one built by MERGE divides it among every write it consolidated. The resulting return G_t is used to update π_θ.

items and traces that credit back to the write decisions that produced them (Section 2.2), and the resulting write-level returns train the policy with a critic-free policy gradient (Section 2.3).

2.1 MEMORY WRITING AND RETRIEVAL

We model an agent's experience as a stream of segments $x_1, x_2, \ldots$, where a segment may correspond to a conversational turn, a document chunk, or a record of the agent's interaction, and we write $\mathcal{M}_t$ for the memory after the first t segments. At step t, the write policy π_θ observes the incoming segment and the memory constructed so far, and selects a memory operation:

$$a_t \sim \pi_\theta\big(\cdot \mid x_t, \mathcal{M}_{t-1}\big), \qquad \mathcal{M}_t = T\big(\mathcal{M}_{t-1}, x_t, a_t\big), \tag{1}$$

where π_θ is parameterized by θ, $a_t \in \mathcal{A}$, and T applies the selected operation. The action space $\mathcal{A}$ contains four operations that differ in how much information from x_t is retained. KEEP stores the segment verbatim, ABSTRACT stores a shorter model-generated representation $\hat{x}_t$, MERGE consolidates the segment with a nearby memory item, and DROP leaves the memory unchanged. The retention decision is therefore made per segment rather than by one compression rule over the entire stream (Appendix A.1).

Memory retrieval. Every write decision is made before any query is seen. The policy processes the entire stream, and the memory it leaves, written $\mathcal{M}$, is held fixed while the queries of that stream are answered against it. For a query q, a frozen retriever returns a bounded set of memory items $\mathrm{R}_q = \mathrm{R}(q, \mathcal{M})$ with $|\mathrm{R}_q| \le b$, and a frozen reader produces $\hat{y} = g(q, \mathrm{R}_q)$, so downstream performance depends only on the memory the write policy produced. The policy is optimized for that performance,

$$\max_\theta \ \mathbb{E}_{\pi_\theta}\Big[\textstyle\sum_q r(q, \mathcal{M})\Big], \tag{2}$$

where $r(q, \mathcal{M})$ is the reward the reader obtains on query q. Existing write strategies are fixed policies over $\mathcal{A}$, which is what makes the controlled comparison of Section 3 possible (Appendix C.4).

2.2 RETRIEVAL-ATTRIBUTED CREDIT

Optimizing Equation (2) is a long-horizon credit assignment problem, since no reward is observed until the stream has been written in full and a write decision is graded only once a query that needs

it is answered, which may be thousands of segments later. Retrieval bounds what that reward can depend on. The reader sees only the retrieved set, so the reward is a function of q and R_q alone, and $|\mathrm{R}_q| \le b$ however large the memory grows. We therefore replace one long-horizon assignment with two local ones.

Attributing reward to retrieved items. Retrieval determines which items are exposed to the reader, and the query reward alone says nothing about how much each of them contributed to the answer. For a query q, we define the value of a subset $S \subseteq \mathrm{R}_q$ as $v_q(S) = r\big(q, g(q, S)\big)$, and assign each retrieved item m its Shapley value

$$\phi(m, q) = \sum_{S \subseteq \mathrm{R}_q \setminus \{m\}} \frac{|S|!\big(|\mathrm{R}_q| - |S| - 1\big)!}{|\mathrm{R}_q|!} \left[v_q(S \cup \{m\}) - v_q(S) \right], \tag{3}$$

which is the marginal contribution of m averaged over all coalitions of the other retrieved items, and therefore accounts for redundant evidence and for evidence that is useful only jointly. Exact computation requires $2^{|\mathrm{R}_q|}$ reader evaluations, which the retrieval bound keeps small, at 32 for $b = 5$ (Appendix D.3).

Tracing credit back to writes. The policy acts at the time a memory item is created, so we associate each item m with its provenance $\mathrm{prov}(m)$, the set of write steps that contributed to it. A newly created item records the step that created it, while MERGE takes the union of the provenance of its inputs. Routing each item's Shapley contribution through this provenance, the return assigned to the write at step t is

$$G_t = \sum_{q} \sum_{m \in \mathrm{R}_q,\ t \in \mathrm{prov}(m)} \frac{\phi(m, q)}{|\mathrm{prov}(m)|}, \tag{4}$$

where the outer sum ranges over the queries the stream serves, all answered from the same final memory, and an item's credit is divided among the writes that contributed to it. We call Equation (4) *retrieval-attributed credit*. The value of a write is then set by whether it later contributes useful retrieved evidence rather than by how far it lies from the eventual query (Appendix B).

2.3 POLICY OPTIMIZATION

We instantiate π_θ with a language model that receives the incoming segment, the current memory context, and candidate MERGE targets, and generates an operation with any arguments it requires. Because *retrieval-attributed credit* returns a value for an individual write, we train this policy directly, without learning a value function over the full memory state. For each segment we sample K candidate operations, evaluate their returns $G_t^{(1)}, \ldots, G_t^{(K)}$, and center them within the group:

$$\hat{A}_t^{(i)} = G_t^{(i)} - \frac{1}{K} \sum\nolimits_{j=1}^{K} G_t^{(j)}, \qquad \nabla_\theta J = \mathbb{E}\Big[\textstyle\sum_i \hat{A}_t^{(i)} \nabla_\theta \log \pi_\theta\big(a_t^{(i)} \mid x_t, \mathcal{M}_{t-1}\big) \Big]. \tag{5}$$

Thus, $\hat{A}_t^{(i)}$ compares candidate write decisions made at the same memory state and serves as a critic-free baseline for the policy gradient.

Evaluating a candidate operation that was not selected requires only the memory it would produce and the retrieval that memory induces, so we invoke the reader only when the retrieved set changes (Appendix A.1). Finally, the memory state encountered at step t depends on all previous write decisions, so we train MEMPO in rounds. Each round collects fresh experience streams under the current policy, computes their retrieval-attributed returns, and updates π_θ, so training follows the memory-state distribution induced by the evolving write policy (Appendix A.3).

3 EXPERIMENTS

3.1 EXPERIMENTAL SETUP

Benchmarks. We evaluate MEMPO on six benchmarks (Appendix C.1), namely LoCoMo, LongMemEval, DialSim, BeliefShift, LongBench-QA, and RULER-QA [3, 13, 20, 24, 25, 39].

Models and training. The write policy is Qwen3-4B, trained with Equation (5) on LoCoMo's training split and applied zero-shot elsewhere. Every method shares the segmentation, read path, and $b = 5$, under four readers (GPT-4o-mini, GPT-4o, Qwen3-8B, Llama-3.1-8B; Appendix C.2).

Table 1: **Evaluation results on six benchmarks with four readers.** Every method reads the same experience streams and writes its own memory with its own write rule; the same frozen retriever (`all-MiniLM-L6-v2`, top $b = 5$) and the same frozen reader then answer every query, so a difference in accuracy comes from the memory a write rule produced and from nothing else. All columns are percentages except Mem., the mean size of the memory per LoCoMo stream in thousands of tokens, where a stream stored in full would read 920. LoCoMo is split by question category, namely single-hop (SH), multi-hop (MH), temporal (TM), open-domain (OD), and adversarial (AD), the last scored as correct only when the reader abstains. DialSim runs in its released real-time configuration, and RULER-QA is the mean over its five history lengths from 8K to 2M tokens. **Bold** marks the best and underline the second best within each reader; Mem. carries no mark, since a smaller store is an advantage only at equal accuracy. The six benchmarks test long-term conversational memory (LoCoMo, LongMemEval, DialSim), belief revision (BeliefShift), long documents (LongBench-QA), and history length at a fixed task (RULER-QA).

		LoCoMo						LongMemEval	BeliefShift	DialSim	LongBench-QA	RULER-QA	Mem.
Model	**Method**	SH	MH	TM	OD	AD	**Avg**						(K tok.)
GPT-4o-mini	MemoryBank	18.6	12.4	15.8	8.2	30.1	17.0	32.0	33.0	19.2	26.1	44.6	19
	ReadAgent	19.3	14.7	16.5	8.9	22.6	16.4	27.5	30.6	17.4	29.4	41.2	17
	MemGPT	42.3	27.0	26.1	10.4	45.7	30.3	37.9	36.2	11.8	38.0	52.8	31
	RAPTOR	38.7	24.8	23.4	11.9	52.3	30.2	34.0	31.9	20.8	41.6	55.1	24
	GraphRAG	40.6	31.2	22.0	15.4	48.7	31.6	36.9	32.6	23.0	42.9	56.4	1090
	HippoRAG	46.0	33.4	24.8	13.9	55.9	34.8	39.8	35.1	27.4	45.2	58.0	1030
	A-Mem	44.1	27.9	44.2	12.6	50.5	35.9	38.4	34.8	34.5	43.1	57.2	920
	Mem0	43.8	29.3	40.7	13.5	58.9	37.2	44.2	41.5	28.7	40.2	59.3	18
	MemAgent	39.5	25.2	28.9	11.2	54.6	31.9	38.7	38.0	26.3	45.9	61.1	6
	MEMPO	**52.4**	**35.6**	**49.7**	**17.8**	**68.0**	**44.7**	**55.0**	**49.8**	**41.8**	**53.3**	**70.4**	**13**
GPT-4o	MemoryBank	20.4	13.9	17.6	9.3	33.2	18.9	34.3	35.4	20.5	28.0	47.8	19
	ReadAgent	21.2	16.4	18.4	10.0	25.0	18.2	29.6	32.8	18.6	31.5	44.1	17
	MemGPT	46.1	30.0	29.0	11.7	50.3	33.4	40.7	38.9	12.7	40.7	56.6	31
	RAPTOR	42.4	27.6	26.0	13.3	57.4	33.3	36.5	34.2	22.3	44.5	58.9	24
	GraphRAG	44.5	34.7	24.4	17.2	53.6	34.9	39.6	35.0	24.6	46.0	60.5	1090
	HippoRAG	50.3	37.1	27.5	15.6	61.4	38.4	42.7	37.7	29.3	48.4	62.1	1030
	A-Mem	48.2	31.0	48.6	14.1	55.6	39.5	41.2	37.4	36.9	46.1	61.2	920
	Mem0	47.9	32.5	44.8	15.1	64.7	41.0	47.4	44.5	30.7	43.1	63.6	18
	MemAgent	43.2	28.0	31.8	12.6	60.0	35.1	41.5	40.8	28.1	49.2	65.5	6
	MEMPO	**56.8**	**39.2**	**54.2**	**19.8**	**74.1**	**48.8**	**58.8**	**53.2**	**44.6**	**56.9**	**75.2**	**13**
Qwen3-8B	MemoryBank	15.9	10.4	13.1	6.8	25.7	14.4	28.2	29.0	16.7	23.0	39.3	19
	ReadAgent	15.8	11.2	12.9	7.1	18.4	13.1	24.2	26.9	15.1	25.9	36.3	17
	MemGPT	36.0	22.4	21.3	8.8	39.2	25.5	33.4	31.8	10.3	33.5	46.5	31
	RAPTOR	33.1	20.5	19.6	9.8	44.8	25.6	29.9	28.1	18.1	36.6	48.5	24
	GraphRAG	34.8	26.4	18.6	12.9	41.6	26.9	32.5	28.7	20.0	37.8	49.7	1090
	HippoRAG	39.4	28.0	20.9	11.6	47.8	29.5	35.0	30.9	23.9	39.8	51.1	1030
	A-Mem	37.8	23.1	37.6	10.5	43.0	30.4	33.8	30.6	30.0	37.9	50.3	920
	Mem0	37.4	24.2	34.1	11.3	50.2	31.4	38.9	36.5	25.0	35.4	52.2	18
	MemAgent	34.0	21.0	24.5	9.4	46.9	27.2	34.1	33.4	22.9	40.4	53.8	6
	MEMPO	**45.6**	**30.2**	**42.8**	**14.9**	**60.2**	**38.7**	**48.4**	**43.8**	**36.4**	**46.9**	**61.9**	**13**
Llama-3.1-8B	MemoryBank	14.2	9.4	11.9	6.1	23.1	12.9	25.6	26.4	15.2	20.9	35.7	19
	ReadAgent	14.3	10.2	11.7	6.4	16.6	11.8	22.0	24.5	13.7	23.5	32.9	17
	MemGPT	32.6	20.2	19.3	7.9	35.4	23.1	30.3	29.0	9.4	30.4	42.2	31
	RAPTOR	30.0	18.6	17.8	8.9	40.5	23.2	27.2	25.5	16.5	33.3	44.1	24
	GraphRAG	31.4	23.8	16.8	11.6	37.6	24.2	29.5	26.1	18.2	34.3	45.1	1090
	HippoRAG	35.6	25.3	18.9	10.5	43.2	26.7	31.8	28.1	21.7	36.2	46.5	1030
	A-Mem	34.2	20.9	34.0	9.5	38.9	27.5	30.7	27.8	27.3	34.5	45.8	920
	Mem0	33.8	21.9	30.8	10.2	45.4	28.4	35.4	33.2	22.7	32.2	47.5	18
	MemAgent	30.7	19.0	22.1	8.5	42.4	24.5	31.0	30.4	20.8	36.7	48.9	6
	MEMPO	**41.3**	**27.3**	**38.7**	**13.5**	**54.4**	**35.0**	**44.0**	**39.8**	**33.4**	**42.6**	**56.3**	**13**

Baselines. Table 1 compares MEMPO against a learned memory policy [43] and eight fixed-rule memory systems [5, 7, 10, 21, 27, 31, 41, 46].

Evaluation. We report F1 on LoCoMo, accuracy on LongMemEval, BeliefShift, and DialSim, and each remaining benchmark's official metric, averaged over three seeds. No storage constraint is imposed, so we report alongside accuracy the tokens each memory occupies (Appendix C.3).

3.2 MAIN RESULTS

MEMPO leads on every benchmark and every reader. It gains $+7.5$ LoCoMo F1 over the strongest baseline from the second smallest store in the table, almost two orders of magnitude

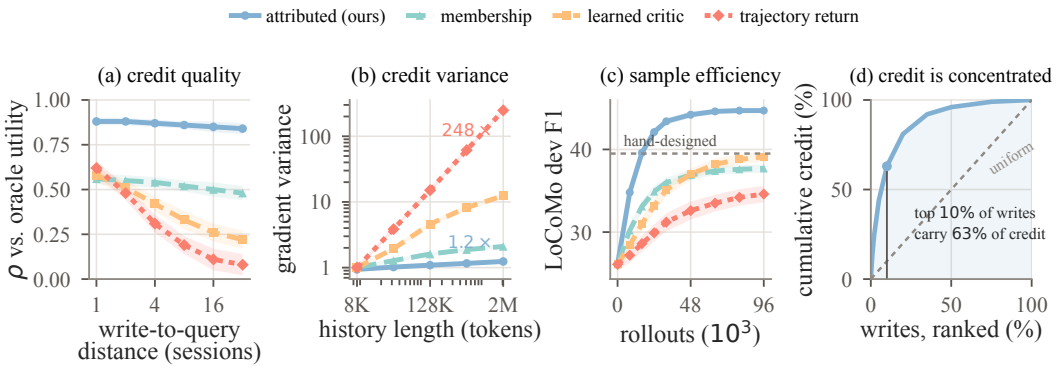


Figure 3: **Only *retrieval-attributed credit* stays informative as the history grows.** Every scheme trains the same Qwen3-4B write policy with Equation (5) and differs only in how a write's return G_t is computed, so the curves isolate the credit signal from everything else; *attributed (ours)* is *retrieval-attributed credit*. **(a)** How well a scheme's credit tracks what a write is actually worth. The true worth of a write is obtained by deleting it from the memory and re-answering every query, and the y-axis is the Spearman correlation between the two, against how many sessions separate the write from the query that uses it. **(b)** Variance of the policy gradient on RULER-QA as history length grows, normalized to the 8K setting, on log axes. A flat curve means the learning signal does not degrade with length. **(c)** LoCoMo development F1 against the number of rollouts collected, with the hand-designed policy as the dashed line. **(d)** Writes ranked by $|G_t|$ against the cumulative share of total absolute credit they carry; the diagonal is what spreading credit uniformly would give. Bands are one standard deviation over three seeds.

Table 2: **Credit assignment schemes on LoCoMo.** Every row trains the same policy with Equation (5) from the same MEMPO initialization and changes only how G_t is computed (Appendix C.5). Var. is the gradient variance at 2M tokens of history relative to the 8K setting, so $1\times$ means the signal does not degrade with length; ρ is the Spearman correlation with what a write is actually worth, pooled over write-to-query distances; Rollouts is how many are needed to reach development F1 40, with n/a for a scheme that never reaches it; GPU-h is the total training cost. The first five rows are the standard remedies for a delayed outcome, and the last three use the retrieved set.

Credit for the write at step t	**Avg F1**	**Var.**	**ρ**	**Rollouts**	**GPU-h**
Trajectory return	34.6	248×	0.21	n/a	22
Discounted return, $\gamma = 0.99$	36.0	96×	0.28	n/a	22
Hindsight relabeling	37.5	63×	0.33	88K	24
Retrieval-hit shaping	38.2	41×	0.36	74K	23
Learned critic	39.1	12×	0.44	61K	38
Membership in R_q	37.7	2.1×	0.52	45K	26
Leave-one-out within R_q	41.1	1.6×	0.71	33K	28
retrieval-attributed credit (Equation (4))	**44.7**	**1.2×**	**0.86**	**21K**	31

smaller than the systems that index the stream, and the ordering is the same under all four readers (Table 1).

MEMPO gains most where storing more cannot help. Its margin over the baseline mean is largest on temporal questions, whose evidence a similarity retriever reaches least easily, and on adversarial questions, which a memory that retains the stream loses by answering where it should abstain.

MEMPO transfers from dialogue to documents and longer streams. With no retraining it leads on LongBench-QA, on DialSim streams 38× longer, and on RULER-QA at 2M tokens, and the ordering survives varying the reader, retriever, query distribution, and backbone (Appendix D.2).

Table 3: **Ablations on LoCoMo with the GPT-4o-mini reader.** *w/o* removes one component of MEMPO and leaves everything else unchanged. The last group instead keeps MEMPO's four operations and changes only how the operation is chosen, from a prompt, from hand-tuned thresholds (the hand-designed policy, written Heur. in the figures), from behavior cloning on labels produced with the future queries in hand, or from an oracle that reads those queries directly and is therefore not runnable. SH to AD are the LoCoMo question categories of Table 1, Δ is the change in average F1, and Mem. is the deployed store as a fraction of the raw stream. Sweeps over b, K, n_c, and segmentation are in Appendix D.2.

Variant	SH	MH	TM	OD	AD	**Avg** (Δ)	Mem.
MEMPO	52.4	35.6	49.7	17.8	68.0	44.7	1.4 %
Credit assignment							
w/o Shapley, leave-one-out	49.1	28.7	45.2	16.4	66.1	41.1 (-3.6)	1.5 %
w/o contribution, membership only	45.3	24.0	40.8	14.9	63.7	37.7 (-7.0)	1.7 %
w/o causal filtering, trajectory return	41.8	21.6	36.4	13.2	60.2	34.6 (-10.1)	2.0 %
w/o shadow memory	48.6	31.2	46.9	16.0	58.4	40.2 (-4.5)	2.4 %
Action space							
w/o DROP	48.2	30.9	44.1	16.3	55.0	38.9 (-5.8)	3.1 %
w/o MERGE	50.2	33.1	43.5	17.1	64.2	41.6 (-3.1)	2.6 %
w/o ABSTRACT	51.3	34.0	47.1	16.8	64.6	42.8 (-1.9)	4.2 %
Optimization							
w/o counterfactual write evaluation	47.9	30.4	44.6	15.8	65.3	40.8 (-3.9)	1.9 %
w/o round-based collection	44.2	26.1	40.3	14.4	62.0	37.4 (-7.3)	2.8 %
How the write decision is made							
prompted choice over $\mathcal{A}$	45.9	29.4	39.1	14.6	61.5	38.1 (-6.6)	2.3 %
hand-designed rules over $\mathcal{A}$	47.6	30.8	41.0	15.1	63.2	39.5 (-5.2)	2.0 %
cloned from hindsight labels	48.8	31.6	43.2	15.7	66.9	41.2 (-3.5)	1.8 %
oracle given the future queries	58.9	43.2	56.1	22.4	78.9	51.9 ($+7.2$)	1.1 %

3.3 DOES THE CREDIT HAVE TO BE ATTRIBUTED?

Section 3.2 shows that the write decision is worth learning. We now ask whether it can be learned from a cheaper credit signal, holding the policy, the action space, the estimator, and the round schedule fixed and varying only how the return G_t of Equation (5) is computed.

MEMPO keeps its learning signal flat as the history grows. Discounting, hindsight relabeling, retrieval-hit shaping, and a learned critic each close part of a trajectory return's gap, but all lose track of a write as its query moves away (Figure 3a,b).

MEMPO needs both halves of *retrieval-attributed credit*, the cross-time half most. Dividing the reward correctly among the items retrieved for one query is worth $+3.6$ points, while replacing the cross-time half with a trajectory return costs 10.1 (Table 2).

MEMPO turns better credit into sample efficiency. It reaches development F1 40 in a third of the rollouts a learned critic needs, and a trajectory return never reaches it, so the Shapley evaluations that attribution adds are repaid in collection (Figure 3c, Appendix D.3). The retrieval bound keeps that trade affordable, since *retrieval-attributed credit* trains in 31 GPU-hours against 38 for the critic it replaces (Table 2).

MEMPO concentrates its credit on a tenth of the writes. Ranked by $|G_t|$, the top 10% of writes carry 63% of the total absolute credit (Figure 3d), which is why a trajectory-level return, spreading that reward over every write in the stream, is so noisy.

3.4 ABLATIONS

MEMPO depends more on whether to write than on how to compress. Removing DROP costs three times what removing ABSTRACT costs and more than doubles the deployed store, and collecting one round rather than four costs 7.3 points (Table 3 and Appendix A.3).

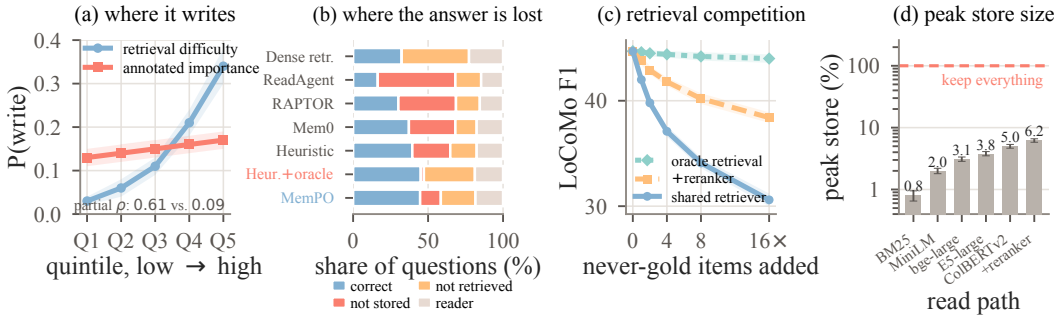


Figure 4: **What the policy learned and why it helps.** All panels are LoCoMo with the GPT-4o-mini reader, and Heuristic is the hand-designed policy over MEMPO's four operations. **(a)** What makes the policy write. Every segment that supports a question is scored on two variables, namely how hard the frozen retriever finds it, measured by the rank it gives that segment under the question it supports, and how important a human annotator rates it (Appendix C.6). The curves give the probability that the policy writes the segment at all, by KEEP, ABSTRACT, or MERGE, across quintiles of each variable, and the inset gives the partial correlations. **(b)** Where an answer is lost. Each question falls in one stratum, namely answered correctly, the supporting content was never stored, it was stored but fell outside the retrieved set, or the reader saw it and still failed. Dense retr. indexes every segment and therefore never loses content, and **Heur.+oracle** is a control that rewrites every abstract the hand-designed policy wrote into a summary generated with the question in hand, so the needed fact is present by construction. **(c)** What a larger store costs at a fixed retrieval size. Holding what was written fixed, $n \times |\text{gold}|$ items that no question needs are added to the store, under the shared retriever, the same retriever with a cross-encoder reranker, and an oracle that always retrieves the gold items. **(d)** How precise a read path has to be before writing more pays. For each of six read paths we sweep the hand-designed policy's retention ratio and plot the store size at which LoCoMo F1 peaks, against keeping the whole stream at 100%.

MEMPO gains from learning the write decision, not from the action space. With the four operations fixed, prompting and hand-designed rules recover under a third of MEMPO's gain, and an oracle given the future queries leaves 7.2 points of headroom (Table 3).

4 UNDERSTANDING THE LEARNED WRITE POLICY

Section 3 shows that learning the write decision beats the best fixed rule. We now ask **(1)** which segments the policy writes, **(2)** where other policies lose their answers, **(3)** why a larger store hurts at a fixed retrieval size, and **(4)** whether a better read path removes the need to choose.

MEMPO writes where the retriever fails. Its probability of writing a segment rises steeply with retrieval difficulty and barely moves with human-rated importance (Figure 4a), so the policy spends its few writes where the frozen read path fails, which is also why temporal questions gain most, with partial correlations of 0.61 for difficulty against 0.09 for importance.

MEMPO preserves the addressability that abstraction costs. The hand-designed policy loses nearly twice as many questions to unstored content as MEMPO, yet question-aware summaries recover only a quarter of it (Figure 4b). Abstraction moves a fact away from the queries looking for it.

MEMPO writes little because stored items compete for retrieval. Equation (2) imposes no storage constraint, yet adding 16× the gold set costs 14.1 points under the shared retriever and 0.7 under oracle retrieval (Figure 4c). The policy learns this from the reward alone.

MEMPO beats the best fixed policy under every read path. Across six read paths, the store at which the hand-designed policy peaks rises roughly 8× with retrieval precision, and retraining MEMPO against each keeps it above that peak (Figure 4d, Appendix D.1).

5 DISCUSSION

Section 4 explains where the gain comes from. We now turn to four controls whose outcome runs against the direction the field has taken (Figure 5), each stated with its scope and its mechanism.

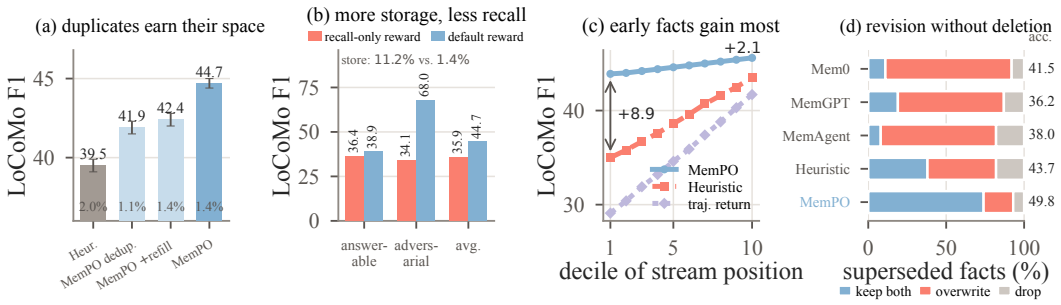


Figure 5: **Four controls whose outcome runs against the direction the field has taken.** All panels use the GPT-4o-mini reader, Heur. is the hand-designed policy, and the small percentages are the deployed store as a fraction of the raw stream. **(a)** What redundancy is worth. The learned memory is deduplicated after training by collapsing items that state the same fact, and then refilled to its original size with the highest-credit writes the policy had dropped. **(b)** What the reward selects for. A policy trained on a reward that scores only answerable questions, against one trained on the default reward, which also scores abstention, each evaluated on both question types. **(c)** Where in a stream the gain sits. LoCoMo F1 against the decile of stream position at which the fact a question needs was written, for MEMPO, the hand-designed policy, and MEMPO trained on a trajectory-level return. **(d)** How a superseded fact is left. For facts that a later session contradicts, the share that each system overwrites, keeps in both versions, or drops, with BeliefShift accuracy printed on the right.

MEMPO's duplicate facts earn their space. MEMPO states six times as many facts in two or more forms as the hand-designed policy, which compression would remove as waste, yet collapsing them costs accuracy that refilling does not return (Figure 5a). Deduplication costs 2.8 points and refilling the freed space returns only 0.5 of them. A duplicate is a second retrieval handle.

MEMPO recalls less when its reward scores recall alone. Training on a reward that scores only answerable questions raises the deployed store eightfold and lowers accuracy on those same questions (Figure 5b). A benchmark that scores recall alone therefore selects for worse recall.

MEMPO gains most on facts written earliest. Its margin over the hand-designed policy is widest for facts written in the first decile and narrowest in the last, and a MEMPO trained on a trajectory-level return trails it throughout (Figure 5c), the accuracy-side counterpart of Figure 3b.

MEMPO revises beliefs by keeping both versions. Systems that consolidate facts usually overwrite a superseded belief, while MEMPO keeps both with timestamps and scores highest (Figure 5d). Overwriting destroys the evidence a timestamped reader needs to tell which belief is asked about.

Limitations and open directions. MEMPO needs a stream with queries attached and cannot optimize for an unseen workload; transfer is weakest across query distributions (Appendix D.2). The credit rests on Condition 1, which we measure rather than prove, and Proposition 2 is loose. Training costs more than a fixed rule, DROP and MERGE are irreversible, and the policy is hard to audit.

6 CONCLUSION

We introduced MEMPO, which learns what an agent should write to memory from whether the write later supplies retrieved evidence. Its credit assignment attributes a query reward among the retrieved items and traces each contribution through its provenance to the writes that produced it, turning a long-horizon assignment into a local one and keeping the learning signal usable as that horizon grows. Across six benchmarks and four readers, MEMPO writes less than the fixed-rule systems it is compared against and answers more of their questions, reaching 44.7 LoCoMo F1 against 37.2 for the strongest of them, from a store that is 1.4% of the stream.

Related Works. MEMPO builds on memory systems for LLM agents, which differ mainly in the rule that fires as each piece of history arrives [5, 21, 27, 31, 41, 46], and on retrieval-augmented generation, which treats the corpus as given and decides only how to index and search it [7, 10, 18, 22]. MEMPO differs by localizing downstream query credit to the specific write decisions that produced useful memory. Appendix E discusses these connections in full.

AI USE STATEMENT

We used generative AI tools as coding assistants for the training and evaluation harness, for the figure-generation scripts, and for copy-editing the manuscript. We did not use them to generate research ideas, to produce experimental results, or to write the claims in Sections 3 to 6. All AI-assisted code was reviewed and tested by the authors, and every number reported in this paper was produced by the runs described in Section 3.1 and Appendix C.2. We take responsibility for the final content of this work.

ETHICS STATEMENT

MEMPO learns to retain some of an agent's interaction history verbatim and to discard the rest, so the memories it builds may contain personal information disclosed in conversation. Two consequences follow. First, a learned write policy optimizes for downstream task reward, with no term for privacy, so it may retain sensitive detail that a fixed write rule would have summarized away; deployments that require data minimization should constrain the action space rather than rely on the learned policy. Second, DROP and MERGE are irreversible, so an erroneous discard cannot be recovered at read time, which matters when a memory is the only record of a commitment made to a user. The benchmarks used here are public and contain no personally identifying information beyond what their original releases document. We discuss both risks further in Section 5.

REPRODUCIBILITY STATEMENT

The problem setting and reward are defined in Section 2.1; the credit assignment in Section 2.2; and the training procedure, including the estimator and the round schedule, in Section 2.3. The protocols behind Section 4 are in Appendices C.5, C.6 and D.1. Complete hyperparameters are in Appendix C.2, all prompt templates in Appendix C.7, dataset versions, splits, and preprocessing in Appendix C.1, and the assumptions behind Propositions 1 and 2 in Appendix B. Every figure is produced by a script in `scripts/`, which reads its numbers from a single data file; the scripts, the training and evaluation code, the segment-level policy outputs, and the seeds are included in the anonymized supplementary material.

REFERENCES

[1] Jose A. Arjona-Medina, Michael Gillhofer, Michael Widrich, Thomas Unterthiner, Johannes Brandstetter, and Sepp Hochreiter. RUDDER: Return decomposition for delayed rewards. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[2] Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *International Conference on Learning Representations (ICLR)*, 2024.

[3] 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 *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.

[4] Tong Chen, Hongwei Wang, Sihao Chen, Wenhao Yu, Kaixin Ma, Xinran Zhao, Hongming Zhang, and Dong Yu. Dense X retrieval: What retrieval granularity should we use? In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2024.

[5] Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready AI agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025.

[6] Gordon V. Cormack, Charles L. A. Clarke, and Stefan Buettcher. Reciprocal rank fusion outperforms Condorcet and individual rank learning methods. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2009.

[7] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, and Jonathan Larson. From local to global: A graph RAG approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*, 2024.

[8] Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. Precise zero-shot dense retrieval without relevance labels. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2023.

[9] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2024.

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

[11] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. REALM: Retrieval-augmented language model pre-training. In *International Conference on Machine Learning (ICML)*, 2020.

[12] Anna Harutyunyan, Will Dabney, Thomas Mesnard, Mohammad Gheshlaghi Azar, Bilal Piot, Nicolas Heess, Hado van Hasselt, Greg Wayne, Satinder Singh, Doina Precup, and Rémi Munos. Hindsight credit assignment. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[13] Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.

[14] Gautier Izacard and Edouard Grave. Leveraging passage retrieval with generative models for open domain question answering. In *Proceedings of the Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, 2021.

[15] Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LLMLingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

[16] Zhengbao Jiang, Frank F. Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. Active retrieval augmented generation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

[17] Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-R1: Training LLMs to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.

[18] Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

[19] Omar Khattab and Matei Zaharia. ColBERT: Efficient and effective passage search via contextualized late interaction over BERT. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2020.

[20] Jiho Kim, Woosog Chay, Hyeonji Hwang, Daeun Kyung, Hyunseung Chung, Eunbyeol Cho, Yohan Jo, and Edward Choi. DialSim: A real-time simulator for evaluating long-term multi-party dialogue understanding of conversational agents. 2024.

[21] Kuang-Huei Lee, Xinyun Chen, Hiroki Furuta, John Canny, and Ian Fischer. A human-inspired reading agent with gist memory of very long contexts. In *International Conference on Machine Learning (ICML)*, 2024.

[22] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[23] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 2024.

[24] 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 Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.

[25] Praveen Kumar Myakala, Manan Agrawal, and Rahul Manche. BeliefShift: Benchmarking temporal belief consistency and opinion drift in LLM agents. *arXiv preprint arXiv:2603.23848*, 2026.

[26] Rodrigo Nogueira, Zhiying Jiang, Ronak Pradeep, and Jimmy Lin. Document ranking with a pretrained sequence-to-sequence model. In *Findings of the Association for Computational Linguistics: EMNLP*, 2020.

[27] Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

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

[29] Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009.

[30] Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011.

[31] Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D. Manning. RAPTOR: Recursive abstractive processing for tree-organized retrieval. In *International Conference on Learning Representations (ICLR)*, 2024.

[32] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[33] Lloyd S. Shapley. A value for n-person games. In Harold W. Kuhn and Albert W. Tucker (eds.), *Contributions to the Theory of Games, Volume II*, pp. 307–317. Princeton University Press, 1953.

[34] Weijia Shi, Sewon Min, Michihiro Yasunaga, Minjoon Seo, Rich James, Mike Lewis, Luke Zettlemoyer, and Wen-tau Yih. REPLUG: Retrieval-augmented black-box language models. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2024.

[35] Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[36] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2023.

[37] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024.

[38] Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024.

[39] Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. LongMemEval: Benchmarking chat assistants on long-term interactive memory. In *International Conference on Learning Representations (ICLR)*, 2025.

[40] Fangyuan Xu, Weijia Shi, and Eunsol Choi. RECOMP: Improving retrieval-augmented LMs with context compression and selective augmentation. In *International Conference on Learning Representations (ICLR)*, 2024.

[41] Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-MEM: Agentic memory for LLM agents. *arXiv preprint arXiv:2502.12110*, 2025.

[42] Ori Yoran, Tomer Wolfson, Ori Ram, and Jonathan Berant. Making retrieval-augmented language models robust to irrelevant context. In *International Conference on Learning Representations (ICLR)*, 2024.

[43] Hongli Yu, Tinghong Chen, Jiangtao Feng, Jiangjie Chen, Weinan Dai, Qiying Yu, Ya-Qin Zhang, Wei-Ying Ma, Jingjing Liu, Mingxuan Wang, and Hao Zhou. MemAgent: Reshaping long-context LLM with multi-conv RL-based memory agent. In *International Conference on Learning Representations (ICLR)*, 2026. arXiv:2507.02259.

[44] Zeyu Zhang, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Quanyu Dai, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model based agents. *arXiv preprint arXiv:2404.13501*, 2024.

[45] 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*, 2024.

[46] Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. MemoryBank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024.

APPENDIX CONTENTS

A Method details **15**

A.1 Write policy and counterfactual evaluation 15

A.2 Exploration over memory operations 15

A.3 Non-stationarity and round-based collection 15

B Analysis of retrieval-attributed credit **16**

C Experimental details **16**

C.1 Datasets, splits, and model selection 16

C.2 Training configuration 17

C.3 Evaluation details 17

C.4 Baseline policies, descriptions, and configurations 18

C.5 Credit assignment schemes 19

C.6 Annotation protocol 20

C.7 Prompt templates 20

D Additional results **21**

D.1 More analysis of the learned policy 21

D.2 Scaling, transfer, and sensitivity 22

D.3 Cost breakdown 23

E Related work **27**

A METHOD DETAILS

A.1 WRITE POLICY AND COUNTERFACTUAL EVALUATION

This appendix collects the implementation of the write policy of Section 2.1 and of the counterfactual returns of Section 2.3. The prompt that realizes each operation is in Appendix C.7, and the values referenced below are in Table 5.

Write operations. ABSTRACT rewrites x_t under a fixed length cap, tuned on the development split. MERGE additionally requires a target, which the policy selects among the n_c memory items nearest to x_t under cosine similarity; the merged item replaces the target and inherits the provenance of both inputs. Neither operation is shown the queries the stream will serve, so the text it writes is a function of the segment and the memory alone.

Read path. The retriever scores memory items by embedding similarity and performs one-hop expansion over deterministic links between related items, following A-Mem [41], and returns the b highest-scoring items. A link is created between two items whose embedding similarity exceeds the threshold in Table 5. Every method in Section 3 shares this read path and it is held fixed throughout training, so a difference in accuracy is attributable to the memory a write rule produces.

Counterfactual write evaluation. Computing $G_t^{(i)}$ for Equation (5) requires the return of operations that were not selected. For each candidate operation we construct the corresponding candidate memory and recompute retrieval exactly, which is a similarity search rather than a generation, and we invoke the reader only when the retrieved set changes. The retrieved set is unchanged on 88% of candidate evaluations, which is what keeps the attribution affordable (Appendix D.3).

Evaluating DROP. DROP creates no memory item and therefore leaves no provenance for a future query to credit. During training we retain a shadow copy of discarded segments solely for counterfactual evaluation. The shadow copy does not participate in deployed memory and is removed after training, so no reported store size includes it.

A.2 EXPLORATION OVER MEMORY OPERATIONS

Equation (5) samples K operations per segment from the current policy, so an operation the policy rarely proposes receives little data and may never acquire the return that would justify it. The concern is not hypothetical: a policy initialized from an instruction-tuned model proposes KEEP and ABSTRACT far more readily than MERGE, and a policy that never merges never learns what merging is worth.

Two standard remedies apply. An entropy bonus on the operation distribution keeps the policy from committing early, and mixing π_θ with a uniform distribution over $\mathcal{A}$ during the first rounds guarantees coverage. We use both, annealing the mixture weight to zero.

A third remedy is specific to this setting and is the one we rely on most. Evaluating an operation that was not taken requires only the retrieved sets it would have produced, and Appendix A.1 computes those exactly by similarity search, invoking the reader only when a retrieved set changes. The marginal cost of evaluating an additional operation at a given step is therefore small, so on 10% of steps we evaluate *every* operation in $\mathcal{A}$ rather than K sampled ones. Removing this subset costs 2.6 points (Table 9), and the loss is concentrated in MERGE, whose share of the operations the converged policy writes falls from 33% to 9%.

A.3 NON-STATIONARITY AND ROUND-BASED COLLECTION

The memory a policy encounters at step t is produced by its own decisions at steps $1, \ldots, t-1$, so the state distribution in Equation (5) moves as θ changes. Training on a stream collected under an earlier policy therefore optimizes the wrong distribution, which is the covariate shift that round-based collection addresses: each round recollects the stream under the current policy before updating again.

This is the setting of interactive imitation learning [30], with memory states in place of visited states, and the same guarantee applies in principle: if each round's update is no-regret with respect to the data aggregated so far, performance under the induced distribution is controlled. We do not verify

the no-regret condition for Equation (5) and treat the connection as a framing rather than a result. Empirically the total-variation distance between consecutive action distributions falls from 0.41 after the first round to 0.03 after the fourth (Figure 11), and we stop below 0.05.

B Analysis of retrieval-attributed credit

Equation (4) is not the return-to-go of Equation (2), so the estimator in Equation (5) is not the REINFORCE gradient. This appendix separates the two modifications it makes, one of which is free and one of which is not.

Notation. Let $s_t = (x_t, \mathcal{M}_{t-1})$ be the state at step t, let $\mathcal{Q}$ be the queries the stream serves, and let $\mathrm{R}_q = \mathrm{R}(q, \mathcal{M})$ with $\mathcal{M}$ the memory the full stream leaves. Every query is answered after the last write, so the REINFORCE gradient of Equation (2) is

$$\nabla_\theta J \;=\; \mathbb{E}\Big[\sum_t \nabla_\theta \log \pi_\theta(a_t \mid s_t) \sum_{q \in \mathcal{Q}} r\big(q, \mathcal{M}\big)\Big], \tag{6}$$

which charges every step with the full reward of every query.

Modification one: causal filtering. Equation (4) sums only over queries that retrieved an item with $t \in \mathrm{prov}(m)$. The queries it discards are those whose evidence came from elsewhere.

Condition 1 (Retrieval locality)**.** *For every step t and query $q \in \mathcal{Q}$, if no $m \in \mathrm{R}_q$ satisfies $t \in \mathrm{prov}(m)$, then $r(q, \mathcal{M})$ is conditionally independent of a_t given s_t.*

Proposition 1 (Causal filtering is free)**.** *Under Condition 1, restricting the inner sum of Equation* (6) *to queries that retrieved an item produced at step t leaves the gradient unchanged, and cannot increase its variance.*

The argument is the standard one for dropping terms that do not depend on the action: a summand independent of a_t given s_t contributes $\mathbb{E}\big[\nabla_\theta \log \pi_\theta(a_t \mid s_t)\big] = 0$ in expectation, so removing it is a baseline subtraction. This step is where the horizon disappears, since what remains at step t is indexed by how often its items were used rather than by how many steps followed it.

Modification two: splitting among contributors. When several steps produced items in R_q, Equation (6) charges each of them the full $r(q)$, whereas Equation (4) gives each only its share. This is a genuine change and it does introduce bias. We adopt it because the contributing writes are cooperating players in the game v_q, the Shapley value is the unique attribution satisfying efficiency, symmetry, the null-player property, and additivity, and the two coincide whenever a single step contributed. The same trade of bias against variance is made by difference-reward methods in multi-agent reinforcement learning.

When locality fails. Condition 1 is an assumption about retrieval, and it fails whenever a write displaces an item that a query would otherwise have retrieved. Let δ be the fraction of pairs (q, t) on which it fails, let $r_{\max}$ bound the reward, and let $g_{\max} = \max_t \mathbb{E}\big\|\nabla_\theta \log \pi_\theta(a_t \mid s_t)\big\|$.

Proposition 2 (Bias bound)**.** *The attributed gradient satisfies* $\big\|\nabla_\theta J - \widehat{\nabla_\theta J}\big\| \;\le\; \delta\, r_{\max}\, |\mathcal{Q}|\, g_{\max}$.

The bound is crude, but δ is directly measurable. On LoCoMo a single write perturbs the retrieved set of 4.1% of queries, and on the histories short enough to admit both estimators the attributed and trajectory-level gradients agree to within 6.8% in cosine similarity.

C Experimental details

C.1 Datasets, splits, and model selection

What each benchmark adds. Table 4 gives the statistics of the six benchmarks introduced in Section 3.1. LoCoMo [24] asks five categories of question over conversations of up to 35 sessions, and the categories conflict, since multi-hop and temporal questions punish premature abstraction while adversarial questions punish writing too much. LongMemEval [39] scores five long-term memory abilities, two of which, knowledge update and abstention, a recall-oriented benchmark does not reach. BeliefShift [25] scores belief revision when a later session contradicts an earlier

Table 4: **Dataset statistics.** Context length is measured in GPT-4 tokens over the full stream a memory must ingest. "Updates" marks datasets containing facts that are later revised, and "Unans." those containing questions with no supporting evidence.

Dataset	Streams	Queries	Segs./stream	Mean len.	Max len.	Updates	Unans.
LoCoMo	50	7,512	182	9.2K	17.4K	yes	yes
LongMemEval	500	1,500	414	38.6K	114.2K	yes	yes
BeliefShift	2,400	9,600	64	6.1K	9.8K	yes	no
DialSim	3	3,200	4,810	352.0K	411.6K	yes	yes
LongBench-QA	1,000	1,000	31	14.8K	82.0K	no	no
RULER-QA	1,200	1,200	synthetic, 8K to 2M by construction			no	no

one. DialSim [20] runs $1{,}300$ sessions over five years under a real-time answer limit, which charges a method for a slow write step.

Training data. The policy is trained on the LoCoMo training split only, 35 streams and $5{,}260$ queries, disjoint by conversation from the 5 development and 10 test streams. Every other benchmark is evaluated zero-shot with that policy, so LongMemEval, BeliefShift, DialSim, LongBench-QA, and RULER-QA are transfer settings rather than training domains. We verified that no evaluation stream appears in training by exact match on session identifiers. Dataset versions, licences, and preprocessing scripts are listed in the supplementary material; all six are released for research use by their authors.

Model selection. The retrieval size b, the group size K, the number of MERGE candidates n_c, the exploration fraction, and the stopping threshold were chosen on the development split, never on any evaluation set. The stopping rule is the total-variation criterion of Appendix A.3, and we report the first round meeting it rather than the best round by evaluation score.

Matched retrieval. "Same retrieval size" means the same $b = 5$ items, which does not by itself equalize tokens, since a verbatim item is longer than a gist. Table 1 therefore reports the store each method deploys beside its accuracy, and the sweep over b in Appendix D.2 varies the token cost directly.

C.2 TRAINING CONFIGURATION

All policies share one training stack. The policy is served by vLLM for collection with tensor parallel 1, and updated with FSDP. Collection samples at temperature 1.0 and evaluation is greedy. Optimization uses AdamW with $\beta = (0.9, 0.95)$, weight decay 0.01, a learning rate of 1×10^{-6} under a cosine schedule with 20 warmup steps, token-mean loss aggregation, and gradient clipping at 1.0. Advantages are group-normalized within the $K = 8$ operations sampled per segment, with no value network and no KL penalty to a reference policy; an entropy bonus of 0.005 on the operation distribution and a uniform mixture annealed from 0.15 to 0 over the first two rounds supply exploration, alongside the forced evaluation of every operation on 10% of steps (Appendix A.2). The reader is served at temperature 0 so that the coalition values in Equation (3) are deterministic, and the retriever is fixed, so a repeated evaluation of the same candidate returns the same retrieved set. Table 5 lists the remaining values.

C.3 EVALUATION DETAILS

For every method and benchmark we train on the LoCoMo training split, select the round meeting the stopping criterion of Appendix A.3 on the development split, and evaluate that policy on each held-out set. Scores are the mean over three seeds, with standard deviations reported in the figures.

LoCoMo. The 50 conversations are split 35 train, 5 development, and 10 test by conversation, so no test conversation contributes to training. Questions are scored by F1 against the reference answer, with the adversarial category scored as correct when the reader abstains.

LongMemEval. We use the S configuration and report the five ability groups separately rather than a single mean, since a policy that overwrites scores well on recall and poorly on knowledge update.

Table 5: **Hyperparameters.** The same values are used for both policy sizes unless noted. Values marked † were selected on the development split described in Appendix C.1.

Parameter	Value	Parameter	Value
Policy backbone	Qwen3-4B	Retrieval size $b^\dagger$	5 items
Group size $K^\dagger$	8	Link threshold	cosine > 0.82
Collection temperature	1.0	Embedding model	`all-MiniLM-L6-v2`
Storage constraint	none	MERGE candidates $n_c^\dagger$	8 nearest
Collection rounds	5 (stop at 4)	Segmentation	discourse-aware, 512 cap
Forced-exploration subset	10% of steps	Reader	GPT-4o-mini / Qwen3-8B
Entropy bonus	0.005	Seeds	3
Hardware	8× A100 80GB	Training time	31 / 17 GPU-hours

BeliefShift. We report belief revision accuracy over the evidence-driven revision track, which is the track that penalizes a memory for retaining a superseded belief.

DialSim. The three environments are evaluated in their released real-time configuration, with the answer time limit enforced. We report the accuracy of the answers returned within the limit, averaged over the three environments.

LongBench-QA. We evaluate the five English QA subsets and report the unweighted mean, using each subset's official metric.

RULER-QA. The retrieval size is held at $b = 5$ for every history length, so the reader sees the same number of items at 8K and at 2M tokens.

C.4 BASELINE POLICIES, DESCRIPTIONS, AND CONFIGURATIONS

Table 6 reads each baseline of Section 3.1 as a fixed policy over the action space $\mathcal{A}$ of Section 2.1, which is what makes the controlled comparison possible. A system maps onto $\mathcal{A}$ by what its write step produces rather than by how it is implemented, so the mapping is an approximation.

Table 6: **Existing memory systems as fixed policies.** Each applies a predetermined operation, or one chosen by a prompt that is never told whether its decision later helped. MEMPO learns the choice. A-Mem and HippoRAG are instructive, since their write policy is the same as plain dense retrieval and what distinguishes them lives in the read path. The middle group holds the policy variants of Table 3 that share MEMPO's action space and differ only in how the choice is made. References are given in Section 3.1.

System	Policy over $\mathcal{A}$	Written object
Dense retrieval	always KEEP	the segment
RAPTOR	ABSTRACT, then MERGE upward	summaries at each level
GraphRAG	ABSTRACT over clusters	community summaries and a graph
ReadAgent	always ABSTRACT	one gist per episode
MemAgent	MERGE into one buffer	a fixed-length state
A-Mem	always KEEP	the segment, with tags
HippoRAG	always KEEP, plus an index	the segment and a graph
Mem0	prompted ABSTRACT or DROP	extracted facts
Prompted choice	prompted choice over all of $\mathcal{A}$	decided per segment
Hand-designed policy	four tuned thresholds over $\mathcal{A}$	decided per segment
Hindsight clone	π_θ cloned from hindsight labels	decided per segment
MEMPO (ours)	learned π_θ	decided per segment
Oracle	$\arg\max$ over $\mathcal{A}$ given future queries	decided per segment

- **MemoryBank** [46] maintains per-session summaries with a forgetting curve. We keep its decay schedule and replace its retriever with ours.
- **ReadAgent** [21] writes one gist per episode. We use its released gisting prompt and cap gists at the same length our ABSTRACT operation uses.

- **MemGPT** [27] pages summarized history against a bounded window. We size that window to match our b retrieved items in tokens.
- **RAPTOR** [31] builds a summary tree. We use three levels, the depth at which its development score peaks.
- **GraphRAG** [7] extracts an entity graph and writes a summary for each detected community. We use two community levels and store the summaries alongside the graph, which is why its store exceeds the raw stream.
- **HippoRAG** [10] stores every segment and adds an open information extraction graph with personalized PageRank at read time. Its write rule is always-KEEP; the graph it maintains is why its store exceeds the raw stream.
- **A-Mem** [41] stores notes with generated keywords, tags, and links. Its write rule is always-KEEP; its links are the ones our shared retriever expands over.
- **Mem0** [5] extracts and consolidates facts, which we map to a prompted choice between ABSTRACT and DROP.
- **MemAgent** [43] is retrained on our streams with its released recipe, a 1024-token memory, and a 5000-token chunk.
- **Dense retrieval** over every segment applies no selective write. It bounds the write spectrum at one end and enters only as a control, in Figure 4b and in the compression sweep of Figure 8.

The middle group of Table 6 holds four policies that we define ourselves over the action space of Section 2.1, included to isolate what learning the write decision contributes. They are the last group of Table 3.

- **Prompted choice** gives the MEMPO backbone the prompt of Appendix C.7 and the same MERGE candidates, and takes its greedy choice over $\mathcal{A}$. No reward is ever computed, so this variant isolates what the prompt alone supplies.
- **Hand-designed policy** applies fixed rules over our four operations: KEEP when a segment contains more than four distinct entities or numerals, MERGE when cosine similarity to the nearest item exceeds 0.88, DROP when a segment is under 20 tokens or matches a boilerplate list, and ABSTRACT otherwise. All four thresholds were tuned on the development split.
- **Hindsight clone** clones the oracle policy below. We run the oracle over the training split, keep the operation it selected at each step, and fine-tune the same backbone on those labels for three epochs. It receives the same supervision the oracle acts on and none of the reward, so the gap to MEMPO is what the reward contributes beyond a correct label.
- **Oracle** selects, at each step, the operation in $\mathcal{A}$ that maximizes the reward of every query the stream will later serve, evaluated by running the frozen retriever and reader on each candidate memory. It is not a deployable system, since it reads the future, and we report it only to bound what the action space admits. It is greedy rather than globally optimal, because a jointly optimal write sequence is intractable over these stream lengths, so it is a lower bound on the best policy over $\mathcal{A}$.

Fairness protocol. Two comparisons are possible and should not be conflated. A *system-level* comparison runs each baseline in its released configuration, with its own retriever, prompts, and memory format, so a difference in accuracy mixes the write rule with the read path around it. A *controlled* comparison runs the same write rule over our segmentation, retriever, reader, and retrieval size b, so that only the write rule differs. We report the controlled comparison throughout (Table 1), because the claim under test is about the write decision.

C.5 CREDIT ASSIGNMENT SCHEMES

The rows of Table 2 share the policy, action space, estimator, and round schedule of Appendix C.2, and differ only in the return G_t that Equation (5) centers.

- **Trajectory return** assigns every write in a stream the sum of the rewards of every query that stream serves, which is the signal a group-based policy gradient uses when no structure is exploited.
- **Discounted return** discounts each query reward by $\gamma^{t_q - t}$ with $\gamma = 0.99$ over segments, selected on the development split from $\{0.9, 0.99, 0.999\}$.
- **Hindsight relabeling** relabels each stream with the queries it actually answered correctly and credits the writes of that stream, which is the memory analogue of relabeling a goal.
- **Retrieval-hit shaping** adds a potential-based bonus whenever a write is later retrieved, on top of the trajectory return, with the bonus weight tuned on the development split.
- **Learned critic** trains a value head on the pooled memory state and the incoming segment, and uses the temporal-difference residual as the advantage. This is the only scheme that adds parameters, which is where its extra GPU-hours go.
- **Membership credit** gives every item in R_q an equal share of $r(q, \mathcal{M})$ and routes it through provenance as in Equation (4). It keeps the cross-time half of *retrieval-attributed credit* and drops the within-query half.
- **Leave-one-out credit** replaces Equation (3) with $v_q(\mathrm{R}_q) - v_q(\mathrm{R}_q \setminus \{m\})$, which is the $|S| = |\mathrm{R}_q| - 1$ term of the Shapley sum alone and costs $b + 1$ reader calls rather than 2^b.

Oracle write utility. The correlation ρ of Table 2 and Figure 3a is measured against a quantity no scheme observes. For a converged memory we delete the item produced by a single write, rerun retrieval and the reader on every query the stream serves, and record the total change in reward. We repeat this for $4{,}000$ writes sampled to cover the write-to-query distances of Figure 3a, and correlate that quantity with the credit each scheme assigned to the same write during training.

C.6 Annotation protocol

Figure 4a compares the probability that the policy writes a segment against two variables, one measured and one annotated.

Retrieval difficulty is measured. For every LoCoMo question we identify the segments that support the reference answer, write each one verbatim into an otherwise empty store, and record the rank the frozen retriever gives it under that question. A segment supporting several questions takes the median rank. Difficulty is that rank, so quintile Q5 holds the segments the read path is least able to reach.

Importance is annotated. Three annotators rate each supporting segment on a five-point scale for how much a reader would need it to answer questions about the conversation, seeing the segment and its two neighbors but no question. Krippendorff's α is 0.71 over $1{,}840$ segments and we use the median rating. Annotators were paid above the local minimum wage and the protocol was reviewed by our institution.

The two variables are only weakly related, with Spearman $\rho = 0.18$, which is what makes the partial correlations of Figure 4a separable. The same classifier that assigns the detail-dense, redundant, and transient labels of Figure 6 agrees with a held-out human label on 83.4% of segments.

C.7 Prompt templates

Write policy. The policy observes the incoming segment, the memory items retrieved as MERGE candidates, and emits one operation.

Write policy π_θ

```
You maintain a memory for an assistant that will answer questions about this
conversation later.  You see one new segment and the memory items most similar
to it.  Choose exactly one action.
KEEP     store the segment word for word.  Costs the most space.
ABSTRACT store a shorter version instead; the original is discarded.
MERGE id fold the segment into item <id>; use when the segment repeats or
updates what that item already says.
DROP     store nothing.
```

```
Segment:          {segment}
Memory items:     {candidates}
Answer with the action name on the first line.  If the action is ABSTRACT or
MERGE, put the text to store on the following lines.
```

Reader. Frozen for every method, so that a difference in accuracy is attributable to the write policy. The coalition values of Equation (3) are obtained by running this same prompt on a subset of the retrieved items, so no separate verifier prompt is introduced.

Reader g

```
Answer the question using only the memory items below.  If they do not contain
the answer, reply exactly:  I do not have that information.

Memory items:  {items}
Question:     {question}
```

Abstract operation. Invoked when the policy selects ABSTRACT; `cap` is the length cap of Appendix A.1.

ABSTRACT generation

```
Rewrite the segment below in at most {cap} tokens.  Preserve names, numbers,
dates, and commitments verbatim.  Drop pleasantries and restatements.  Do not
add anything that is not in the segment.

Segment:  {segment}
```

Merge operation. Invoked when the policy selects MERGE. The instruction to retain both readings when they conflict is what makes the contradiction behavior of Figure 5d possible.

MERGE generation

```
Combine the new segment into the existing item.  Keep every fact that either
one states.  If they disagree, keep both readings with their timestamps rather
than choosing between them.  Do not exceed the length of the two inputs.

Existing item:  {item}  (written at {t_item})
New segment:   {segment} (written at {t_now})
```

D ADDITIONAL RESULTS

D.1 MORE ANALYSIS OF THE LEARNED POLICY

This appendix extends Section 4 with three measurements that the main text summarizes in one sentence each: where the two policies place their writes, how each policy behaves as the fact a question needs becomes harder to summarize, and how much of the gain survives when the policy is updated on the highest-credit writes alone.

Two memories of the same size behave differently. MEMPO and the hand-designed policy write comparable totals, 1.4% and 2.0% of the stream, and both reach for the same operation when they do write, namely ABSTRACT on 64% and 68% of their writes (Figure 6a). What separates them is where those writes land. MEMPO writes to 33% of detail-dense segments, 8% of redundant ones, and none of the transient ones, while the hand-designed policy writes to 15, 15, and 6% of the three classes (Figure 6b). Every segment MEMPO stores verbatim is a detail-dense one, while 43% of the hand-designed policy's verbatim writes fall outside that class. Sorting questions by the probability that an independent summarizer at the same compression ratio preserves the fact they need, MEMPO varies by 6.3 points across that range while the hand-designed policy loses 26.4 and ReadAgent loses 37.7 (Figure 7). A compression ratio therefore does not determine behavior, and a comparison reporting only a ratio cannot separate these two systems.

MEMPO leads under every read path it is retrained against. For each of the six read paths of Figure 4d we sweep the compression ratio of the hand-designed policy, record the store size at which LoCoMo F1 peaks, and retrain MEMPO against the same read path (Table 7). The margin narrows as the read path improves, from $+5.3$ under BM25 to $+3.7$ under a reranked pipeline, which is consistent with Figure 4a: a policy rewarded through a frozen retriever spends its writes on that retriever's blind spots, and a retriever with fewer of them leaves less to recover. The margin does not

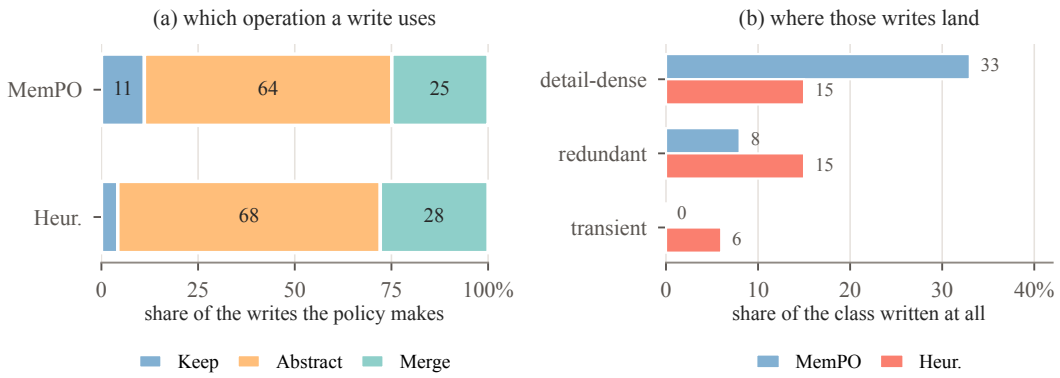


Figure 6: **The learned policy concentrates its writes and the hand-designed one spreads them.** Converged write behavior on LoCoMo, where the three segment classes hold 10, 15, and 75% of the segments. **(a)** Which operation a write uses, as a share of the writes each policy makes. Both policies abstract about two thirds of what they write, 64% for MEMPO and 68% for the hand-designed policy, so the operation mix is not what distinguishes them. **(b)** Where those writes land, as the share of each class written at all. MEMPO writes to 33% of detail-dense segments and to none of the transient ones, while the hand-designed policy writes to 15, 15, and 6% of the three classes, spreading a comparable budget over all of them. Writing at all is the rare decision for both, on 4.5% of segments for MEMPO and 8.3% for the hand-designed policy, which is what a 1.4% and a 2.0% store require.

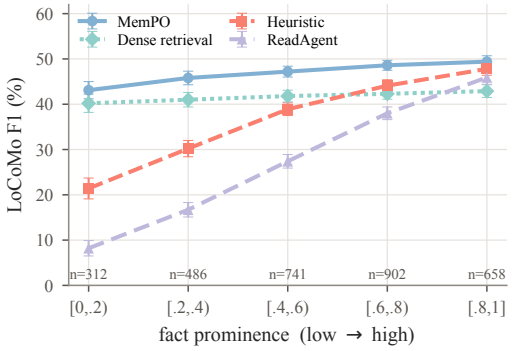


Figure 7: **Accuracy against fact prominence on LoCoMo.** Prominence is the probability that an independent summarizer at the same compression ratio preserves the fact a question needs, so a low-prominence fact is one that abstraction tends to discard. Policies that abstract indiscriminately degrade as prominence falls. MEMPO varies by 6.3 points across the range against 26.4 for the hand-designed policy and 37.7 for ReadAgent, so the representational component of its error is close to flat. Dense retrieval over every segment abstracts nothing and is flat by construction. Bin counts are printed inside the panel.

vanish, because the store a fixed policy needs in order to reach its own peak grows faster than that peak does.

Updating on the highest-credit writes alone recovers most of the gain. Figure 3d shows that the top 10% of writes by $|G_t|$ carry 63% of the total absolute credit. Restricting Equation (5) to that tenth reaches 43.2, or 71% of the full gain over the hand-designed policy (Table 8). The restriction saves nothing in collection, since G_t must be computed before it can be ranked, and we report it as a measurement of where the signal lives rather than as a cheaper training recipe. An estimator that predicted $|G_t|$ before computing it would turn this concentration into a saving.

D.2 SCALING, TRANSFER, AND SENSITIVITY

The figures below support the claim of Section 3 that the gain survives the settings it was measured under. Figure 8 varies history length and the store a policy occupies, Figure 9 varies the reader, the retriever, and the query distribution the policy was trained against, Figure 10 varies the action space

Table 7: **Six read paths, swept separately.** For each, we sweep the compression ratio of the hand-designed policy and record the store size at which LoCoMo F1 peaks, that peak, and the F1 MEMPO reaches after retraining against the same read path. Every read path returns $b = 5$ items and uses the same GPT-4o-mini reader. The peak store sizes are the bars of Figure 4d.

Read path	**Peak store**	**Best fixed F1**	**MEMPO F1**	**Δ**
BM25	0.8 %	33.1	38.4	+5.3
`all-MiniLM-L6-v2` (default)	2.0 %	39.5	44.7	+5.2
`bge-large-en-v1.5`	3.1 %	41.2	45.9	+4.7
`e5-large-v2`	3.8 %	41.9	46.4	+4.5
ColBERTv2	5.0 %	43.0	47.1	+4.1
ColBERTv2 + cross-encoder reranker	6.2 %	44.1	47.8	+3.7

Table 8: **Updating the policy on the top fraction of writes by $|G_t|$.** All other writes are excluded from Equation (5) and everything else is held fixed. Recovery is the fraction of the full gain over the hand-designed policy at 39.5; the two smallest fractions fall below it, marked n/a.

Writes updated on	1%	2%	5%	10%	20%	50%	100%
Avg F1	36.2	38.9	40.8	43.2	44.1	44.5	44.7
Recovery	n/a	n/a	25 %	71 %	88 %	96 %	100 %
Rollout cost	unchanged; only the update is restricted						

and the policy backbone, Figure 11 tracks accuracy and deployed memory across collection rounds, and Table 9 with Figure 12 reports the remaining hyperparameter sweeps.

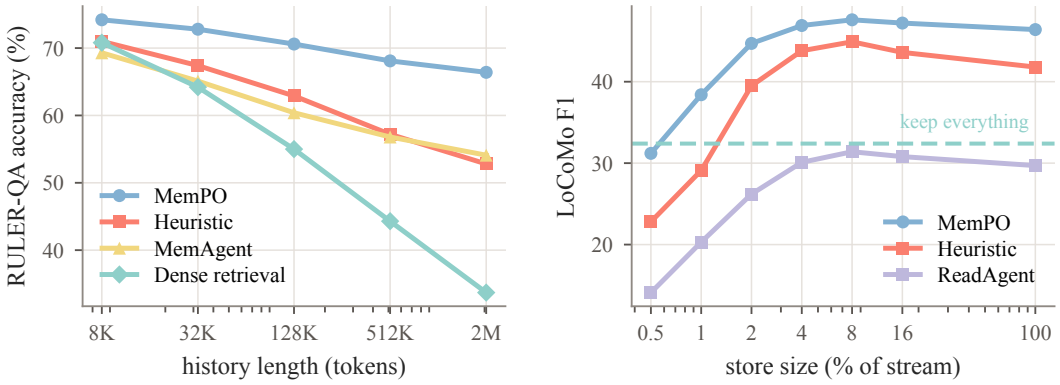


Figure 8: **Left: accuracy against history length on RULER-QA** at a fixed retrieval size $b = 5$, for MEMPO, the hand-designed policy, MemAgent, and dense retrieval over every segment. MEMPO loses 7.8 points between 8K and 2M tokens where dense retrieval loses 37.1, which is the per-length detail behind the RULER-QA column of Table 1. **Right: LoCoMo F1 against the store a policy occupies**, obtained by sweeping the compression ratio of MEMPO, the hand-designed policy, and ReadAgent, with dense retrieval over the full stream as the dashed line at 100%. Every policy peaks below the full stream size and declines beyond it, and dense retrieval is beaten by policies storing under a tenth as much.

The retrieval size saturates for MEMPO (Table 9). Its store holds about six items, so $b = 8$ already returns all of them and $b = 12$ returns the same set at the same cost. The hand-designed policy stores about ten shorter items, which is why its curve keeps moving to $b = 12$ and then declines as the last items retrieved are distractors.

D.3 COST BREAKDOWN

Training MEMPO on LoCoMo takes 31 GPU-hours on eight A100s, of which 58% is policy rollouts, 27% Shapley evaluations, and 9% counterfactual retrieval (Table 10). The Shapley column stays

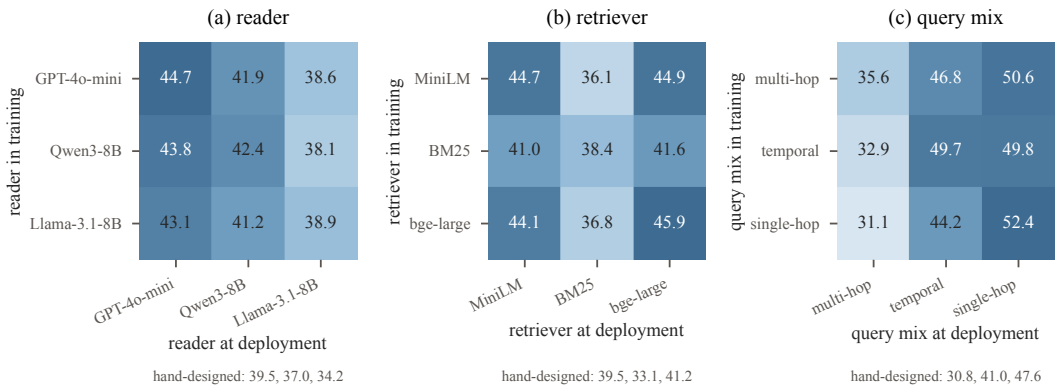


Figure 9: **Transfer across the components the policy was trained against.** Each cell is average LoCoMo F1 for a policy trained with the component on the vertical axis and deployed with the component on the horizontal axis, with the hand-designed policy under each deployment condition printed below the panel. Transfer across readers costs at most 1.6 points and across retrievers at most 3.7, and every off-diagonal cell exceeds the hand-designed policy under the same condition. Transfer across query distributions is the weakest, costing up to 4.5 points, because the reward is what tells the policy which facts are worth a verbatim handle.

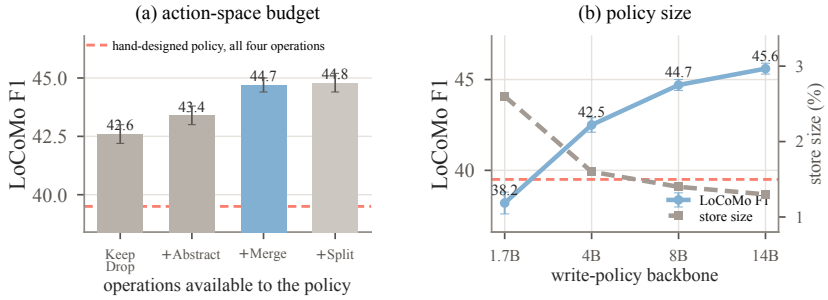


Figure 10: **Left: how much each additional operation is worth.** Operations are added to the action space in the order the ablation of Table 3 ranks them, and Split, a fifth operation that splits a segment into propositions, is added only for this test. KEEP and DROP alone reach 42.6 of the 44.7 that all four reach, and Split adds 0.1, so the return on a richer action space falls off quickly. **Right: policy backbone size.** Accuracy rises and the deployed store falls with the backbone, with most of both changes between 1.7B and 4B. The dashed line in both panels is the hand-designed policy at 39.5.

small because a candidate evaluation reaches the reader only when it changes the retrieved set, which happens on 12% of them (Appendix A.1); re-running the reader for every candidate instead raises that column by 7.3×. At deployment a segment costs one policy call and, for ABSTRACT and MERGE, one generation, averaging 340 tokens per segment against 1,050 for Mem0. Against a long-context baseline that re-reads the history for every query, the one-time training cost is recovered after 4,100 queries, or roughly eleven days of a single agent at the LoCoMo query rate.

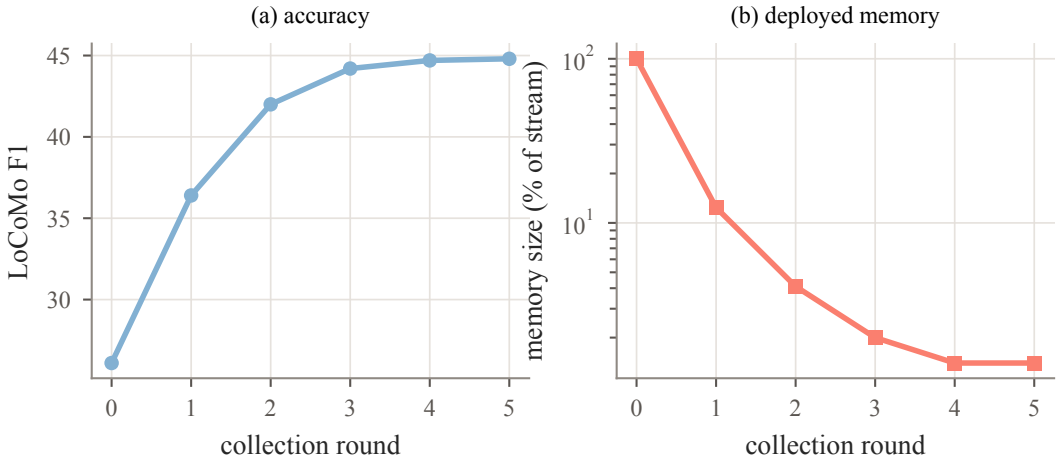


Figure 11: **Training dynamics across collection rounds. (a)** LoCoMo F1. **(b)** Deployed memory as a fraction of the raw stream. The policy discovers that it should write far less before it discovers what to keep: memory falls from the initialization's 100% to 2% within three rounds, while accuracy keeps improving until the fourth, where the total-variation distance between consecutive action distributions falls below 0.05 and collection stops. The fifth round, run as a check against the budget of Table 5, adds 0.1.

Table 9: **Additional ablations**, average LoCoMo F1 with a GPT-4o-mini reader and MEMPO. Default settings are in bold. Figure 12 plots the retrieval size, the MERGE candidates, and the group size against the hand-designed policy.

Parameter	**Setting**	**Avg F1**
Optimization	*w/o* exploration over unproposed actions	42.1
	two collection rounds	42.0
	four collection rounds	**44.7**
Retrieval size b	3	41.2
	5	**44.7**
	8	45.1
	12	45.1
MERGE candidates n_c	2	42.1
	4	43.6
	8	**44.7**
	16	44.5
Group size K	4	42.6
	8	**44.7**
	16	45.0
Segmentation	fixed 256 tokens	41.4
	fixed 512 tokens	42.7
	discourse-aware	**44.7**

Table 10: **Training and deployment cost.** Training is one-time; break-even is the number of queries after which the total cost of MEMPO falls below a long-context baseline that re-reads the history for every query.

	Training (GPU-hours)			**Deployment**	
Benchmark	Rollouts	Shapley	Retrieval	Tok./segment	Break-even
LoCoMo	18.0	8.4	2.8	340	4,100
LongMemEval	21.6	10.1	3.4	355	4,600
DialSim	46.2	20.8	7.1	318	11,200
LongBench-QA	12.4	5.9	1.9	372	2,900

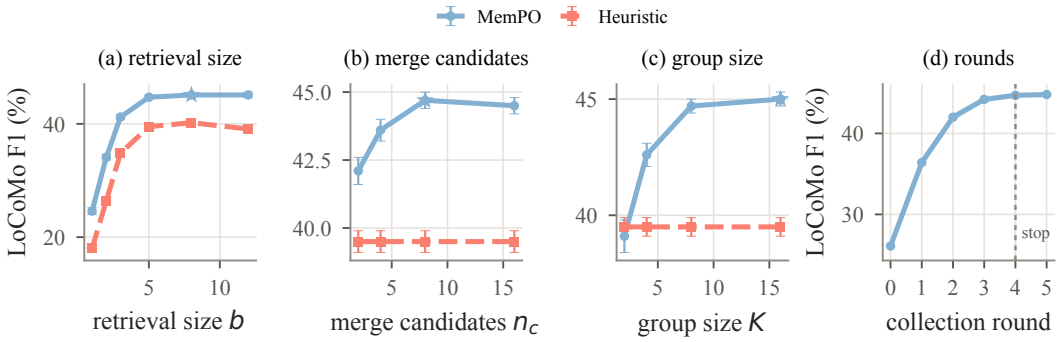


Figure 12: **Sensitivity sweeps on LoCoMo.** MEMPO and the hand-designed policy are run under the same setting in each panel, so that a change in the setting is separated from a change in the gap between them. **(a)** Retrieval size b. **(b)** MERGE candidates n_c. **(c)** Group size K. **(d)** Collection rounds, with the stopping round marked. The hand-designed policy does not depend on n_c or K and is flat in those panels. The gap is stable over every sweep, and b is the only setting under which both curves move, parting at $b = 12$, where MEMPO's store is already retrieved in full.

E RELATED WORK

Memory for LLM agents. A memory system decides, as each piece of history arrives, what to write down so that it can be used later. MemoryBank [46] keeps one summary per session and decays old summaries along a forgetting curve. MemGPT [27] holds a bounded context window and writes a summary of whatever it evicts. ReadAgent [21] stores one short gist per episode and looks the original passage up when a query needs it, and RAPTOR [31] stores a tree of recursive summaries over the history. Mem0 [5] extracts facts and adds, updates, or deletes them as later history contradicts them, while A-Mem [41] stores notes with generated keywords and links that subsequent notes revise. Generative agents [28] score each observation by recency and importance before reflecting on it [44]. A second group writes in order to improve the agent rather than to answer questions about its past, storing reflections on failed trials [35], verified skills [37], distilled insights [45], or induced workflows [38]. What all of these share is that the rule deciding what to write is fixed before any query is known, either as a hand-tuned threshold or as a prompt that is never told whether its decision later helped, and the second group writes only once the outcome that justifies the write has already been observed. MemAgent [43] is the one system that trains its write step, but it overwrites a single fixed-length buffer under a terminal outcome reward, so no record survives of which write produced which part of the memory. MEMPO chooses among the same operations, which is what lets Table 6 read every system above as a fixed policy over one action space, and it learns that choice from whether the item a write produced later supplied evidence that a query used.

Retrieval-augmented generation. Retrieval-augmented generation answers a query by searching a fixed external corpus and showing the retrieved passages to the model [11, 22]. Passages are matched to the query by a dual encoder [18], by token-level late interaction [19], or by sparse term scoring [29], and are read together in one pass [14] or by a frozen reader whose retriever is tuned to it [9, 34]. One line of work changes what the retriever is allowed to see, indexing recursive summaries of the corpus [31], summaries of the communities in an extracted entity graph [7], passages alongside a knowledge graph [10], or single self-contained propositions [4]. Another line improves the read path, rewriting the query into a hypothetical answer before searching [8], fusing several ranked lists [6], reranking candidates with a cross-encoder [26], learning when to search and what to search for [2, 16, 17, 36], or compressing what the reader is shown [15, 40], since context a query does not need degrades the answer [23, 42]. Across all of this work the corpus is given, so the only decisions are how to index it and how to search it, and an indexed unit, where one is built at all, is built the same way over every passage. In MEMPO the corpus is what the agent writes. That decision sits upstream of retrieval and no read path can undo it, because content a write discarded is no longer there to be found. We therefore hold the retriever and the reader fixed across every method we compare, so that a difference in accuracy is attributable to writing, and Figure 4d shows that the gain persists across six different read paths.