How Much Is a Memory Worth? Efficient Credit Assignment for Memory Admission Under Unknown Demand

Anonymous authors
Paper under double-blind review

Abstract

Language-model agents use persistent memory to carry information across interactions, but limited storage requires deciding which memories to retain before future questions arrive. Existing admission policies rank memories using combinations of factors such as recency and confidence, or novelty relative to what is already stored. A memory's contribution to answering, however, depends on both the questions and the other available memories. Learning an admission policy from this contribution requires assigning credit when memories overlap or depend on one another. Here, we measure the utility of a set of memories by its reduction of a frozen reader's codelength for reference answers, then use the Shapley value to assign credit to individual memories. A policy, which we call Shapley Credit Assignment Policy (**SCAP**), predicts the resulting credit at write time from candidate and resident records using one regression objective over frozen embeddings. On held-out LongMemEval questions and supporting sessions, SCAP retains greater utility than the selections made using A-MAC and SAGE across capacities, with larger advantages at higher capacity pressure. The advantage also persists with a reader from another model family without retraining the policy. These findings support a simple approach to memory admission in which the learning target follows from the utility of the stored information.

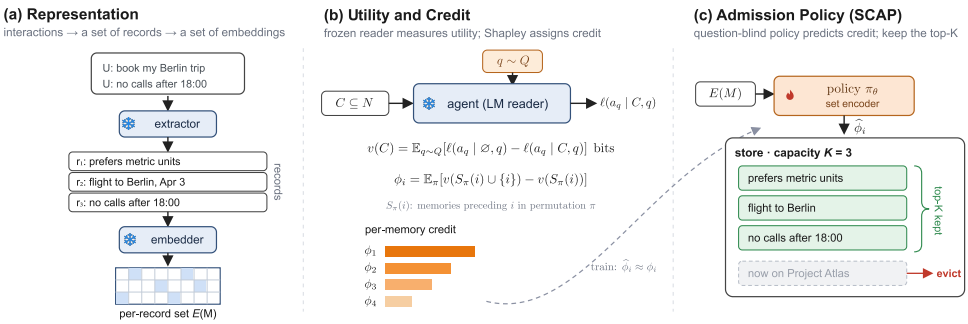


Figure 1: Learning which memories to keep. (a) An extractor converts each interaction into records, which an encoder embeds separately. Both models remain frozen. The source excerpt is abbreviated. (b) A frozen reader measures the utility of a set of memories in bits, and Shapley attribution assigns credit to each memory. (c) **SCAP** predicts credit and keeps the top K before receiving a question.

1 Introduction

Persistent memory lets a language-model agent carry user preferences, facts, and commitments beyond its current interaction. When storage is limited, admitting a new memory may require evicting one already kept. The agent must make this decision at *write time*, before it knows which questions the information will help answer. It therefore needs to compare the expected utility of retaining the new memory with that of retaining the memory it would displace.

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.

Existing methods score memories using recency, confidence, semantic novelty, and estimated relevance. A-MAC (Zhang et al., 2026) combines an LLM rating of future usefulness with confidence, novelty, recency, and content type into a weighted admission score. SAGE (Wang et al., 2026) computes novelty from a density estimate over memory embeddings, adds clearly novel memories, skips redundant ones, and refers uncertain cases to an LLM for merging. Other methods learn relevance under a token budget (Lia & Mazumder, 2026). These scores guide admission using properties of the candidate and its relation to the store. Learning admission from the contribution of stored information to answering questions requires accounting for interactions among memories, since repeated facts may substitute for one another and complementary facts may need to be kept together. The question is therefore how to assign individual credit for this joint contribution so that it can guide retention before future questions arrive.

We address this by assigning credit through the Shapley value (Shapley, 1953) and training a policy, which we call Shapley Credit Assignment Policy (**SCAP**), to predict it from the memories alone (Figure 1). We first measure a memory set's utility by its reduction of a frozen reader's codelength for reference answers over observed questions. Shapley attribution then credits each memory for its marginal contribution across subsets of the rest. The credits sum to the set's utility, and memories with identical marginal effects receive equal credit. A set encoder (Zaheer et al., 2017; Lee et al., 2019) predicts this credit from the candidate and resident memories, represented as self-contained *records* extracted from their interactions. Our approach thus derives the learning target from the utility of the memories to the reader and fits one regression objective over frozen embeddings.

To evaluate the policy, we asked it to choose K memories from stores constructed from LongMemEval histories (Wu et al., 2025), holding out questions and their supporting sessions for testing. We scored the selected memories together to determine their retained utility. Our policy selects memories with greater utility than those selected by the more sophisticated factor-based and novelty-based scores of A-MAC and SAGE at every tested capacity. When we change the reader agent, the same learned policy and memories retain their advantage, demonstrating transferrability of the policy across different agents.

1.1 Related work

Representing memories. An agent's memory must preserve information from earlier interactions in a form it can use later (Tang, 2026). The distinction between semantic and episodic memory (Tulving, 1985) suggests two ways to organize this information. Self-contained propositions make individual facts available for retrieval and verification (Chen et al., 2024; Min et al., 2023), while dated observations preserve events and their sources (Park et al., 2023). An interaction can contain both facts and events, so we combine these forms in the records that represent each memory. Keeping the records separate also lets a policy attend to their individual contents. Set encoders (Zaheer et al., 2017; Lee et al., 2019) provide this access while making the prediction independent of the order in which the records are presented.

Memory utility under unknown demand. A memory's utility depends on what the agent can do with the information it contains. Predictive $\mathcal{V}$-information formalizes this idea by measuring how information helps a model make predictions (Xu et al., 2020). The reduction in an answer's codelength offers a way to measure utility (Grünwald, 2007). Assigning credit to individual memories requires a further step because the reader uses them together. Prior work addresses credit assignment through evidence about memory operations (Ma et al., 2026), repeated rollouts from a common state (Yan et al., 2026a), and removal of memories after a query arrives (Srivastava, 2026). Evaluating all subsets also provides an exact optimum against which to compare memory selection under a storage limit (Bhargava & Barrento, 2026). Shapley attribution (Shapley, 1953; Lundberg & Lee, 2017; Štrumbelj & Kononenko, 2014) connects the utility of these subsets to individual credit by averaging each memory's marginal contributions. We use that credit as a target for learning our proposed admission policy.

Admission policies. Admission problem is to assign a score to each memory to decide what to keep. In the adjacent problem of caching, access histories guide whether an incoming item should replace a stored one (Einziger et al., 2017; Wong et al., 2024). Agent memory policies make similar decisions using properties of the candidate and the existing store. A-MAC learns weights for an

LLM rating of future usefulness, confidence, novelty, recency, and content type (Zhang et al., 2026), while SAGE computes embedding-based novelty and uses it to route additions, skipped candidates, and LLM-assisted merges (Wang et al., 2026). Learning the decision also requires choosing its supervision. LRE learns relevance under a token budget (Lia & Mazumder, 2026), MemRouter uses teacher labels for memory operations and content types (Hu et al., 2026a), and Memory-R1 learns operations from downstream rewards (Yan et al., 2026b). OSL-MR further constrains retention to use information available when the agent decides (Kang et al., 2026). Our policy learns from the credit assigned to memories and uses their contents to make the decision. Comparing scores on the same memories with the same reader isolates their effect on retention, complementing benchmarks that evaluate writing, retrieval, and answering together (Wu et al., 2025; Hu et al., 2026b).

2 METHOD

2.1 PROBLEM SETUP

We consider a memory store containing n memories and a capacity of K memories. A retention policy observes the memories and chooses which K to keep before receiving a future question. The retained memories are then supplied together to a reader. The objective is to maximize their expected utility over future questions, with utility defined below. When the store contains K resident memories and one incoming memory, this decision determines whether to reject the incoming memory or admit it by evicting a resident.

2.2 DATA

Our experiments use a labeled subset of LongMemEval (Wu et al., 2025), whose user-assistant histories are arranged as dated sessions. It contains 169 answerable questions, each with a reference answer and one annotated supporting session, together with about 7400 background sessions containing no annotated evidence for these questions. We divided the questions and their supporting sessions into 135 training and 34 test questions randomly. We constructed training stores using the training questions' supporting sessions and test stores using those of the test questions, adding background sessions to both. A supporting session can appear in several stores alongside different background memories. The test questions and their supporting sessions were held out from training.

2.2.1 MEMORY STORES

A store is a set of memories presented together to the policy for a retention decision. We formed one memory from each session. For each store, we randomly chose its size n between 5 and 10 memories and the number of questions D between 1 and 4. We then selected D questions to form its demand Q, included one supporting memory for each question, and filled the remaining $M = n - D$ positions with background memories, termed distractors. Thus every store contained at least one distractor. We constructed 400 training stores and 100 test stores; after excluding stores with failed extractions, 372 training stores and 88 test stores remained.

2.3 MEMORY REPRESENTATION

To represent each session, we used a frozen Qwen3-8B extractor (Yang et al., 2025) which received the session without its question. We compared a *facts* prompt asking for self-contained propositions with a *ledger* prompt that also emphasizes events, changes over time, and material such as plans and tables. Both produce records with references to the turns they came from. A *combined* representation lists facts followed by ledger records and removes repetitions with identical text and source-turn lists. Appendix A details the prompts used.

To choose a representation, we used the same frozen model to answer questions from a separate sample of 48 histories. It answered 35 correctly from the full source. Of these, it could still answer 25 using combined, 24 using ledger, and 18 using facts (Table 1). The denominator remained 35 when a grade could not be resolved. We selected combined, which used about a quarter of the source tokens, and used it throughout policy training and evaluation.

Table 1: Choosing the memory representation. Questions retained is the fraction of the 35 questions answered correctly from the full source that the reader could still answer from each representation. Token ratio compares its mean length with that of the source. Bold marks the representation used in subsequent experiments.

Representation	Questions retained	Mean memory tokens	Token ratio
Source (full session)	1.00	4167	1.00
Facts	0.51	629	0.15
Ledger	0.69	538	0.13
Combined (used in ours)	**0.71**	1112	0.27

We embedded each record separately so the policy could attend to individual contents. Section 3.1 compares this representation with a single vector formed by embedding and pooling chunks of the complete memory text.

2.4 Utility of a set of memories

To measure the utility of a set of memories, we evaluated the probability that a frozen reader assigns to a reference answer. Let r denote the reader, here Qwen3-8B (Yang et al., 2025), and let q be a question with reference answer a_q. Given memories C, the answer's *codelength* is

$$\ell(a_q \mid C, q) \;=\; -\sum_t \log_2 p_r(a_{q,t} \mid a_{q,<t},\, C,\, q) \quad \text{[bits]}, \tag{1}$$

the number of bits an ideal code built from the reader's token probabilities needs to encode a_q given C and q (Grünwald, 2007). The utility of a set of memories is the reduction it produces in this codelength relative to the empty context,

$$v_q(C) \;=\; \ell(a_q \mid \varnothing, q) - \ell(a_q \mid C, q), \tag{2}$$

so positive utility indicates that the memories make the reference answer more probable, while negative utility indicates that they make it less probable.

The policy observes the memories at write time, while the future question remains unknown. To define the utility from which its credit targets are computed, we average over the store's observed demand Q,

$$v(C) \;=\; \mathbb{E}_{q\sim Q}\,[\,v_q(C)\,] \;=\; \frac{1}{|Q|}\sum_{q\in Q}\Big[\ell(a_q \mid \varnothing, q) - \ell(a_q \mid C, q)\Big]. \tag{3}$$

2.5 Assigning credit to memories

The utility in Eq. 3 depends on a set, while the policy assigns a score to each memory. Removing one of two interchangeable memories may leave the utility unchanged, even though both contribute when used alone. Shapley attribution (Shapley, 1953) accounts for such interactions by averaging a memory's contribution over subsets of the others. For memory i in a store N of n memories,

$$\phi_i \;=\; \sum_{S\subseteq N\setminus\{i\}} \frac{|S|!\,(n-|S|-1)!}{n!}\,\big[\,v(S\cup\{i\}) - v(S)\,\big], \tag{4}$$

where the weights average over all orders in which memories can enter the set. Shapley attribution satisfies efficiency, symmetry, the null-player property, and additivity (Shapley, 1953). Since $v(\varnothing) = 0$, the credits sum to $v(N)$. Memories that make identical contributions receive equal credit, and a memory that changes no subset's utility receives zero credit. The resulting labels account for each memory's contribution in the presence of the others.

We computed ϕ_i over all 2^n subsets for $n{\le}6$ and estimated it from 64 sampled permutations for larger stores. This produced 3421 memory labels across the 460 stores. The credit comparisons in Table 7 use exact attributions computed from all held-out subsets.

2.6 LEARNING A RETENTION SCORE

The policy predicts ϕ_i from the memory being scored, termed the candidate, and the other memories, termed residents. A frozen encoder, here Qwen3-Embedding-4B (Zhang et al., 2025), maps each record to a vector. We project these vectors and add learned tags marking candidate and resident records, then use one learned attention query to pool them (Lee et al., 2019; Vaswani et al., 2017). A feed-forward head maps the pooled vector to $\widehat{\phi}_i$. The score is invariant to record order. We score each memory in turn as the candidate and keep the K with the highest predictions.

2.7 TRAINING

We trained the policy to predict credit in bits using Huber loss (Huber, 1964) with transition point δ=8. The set encoder used width 256, four attention heads, hidden dropout 0.3, and input dropout 0.2. We trained with Adam (Kingma & Ba, 2015), learning rate 3×10^{-4}, weight decay 10^{-3}, batch size 32. Approximately one fifth of the training questions were reserved for validation, and every other store containing one of these questions was excluded from fitting. We selected the checkpoint with minimum validation loss and stopped after 30 epochs without improvement. Appendix A gives the split and example counts.

2.8 EVALUATION PROTOCOL

Retained utility. For each test store, we selected the K highest-scoring memories and measured their utility over its questions,

$$v(\text{top-}K) \;=\; \frac{1}{|Q|} \sum_{q \in Q} \Big[\ell(a_q \mid \varnothing, q) - \ell(a_q \mid \text{top-}K,\, q) \Big] \quad \text{[bits]}, \tag{5}$$

using reference-answer codelengths computed for every subset. We also report an *oracle* that chooses the K memories with the greatest joint utility. Results are averaged over stores for K=1, . . . , 5. For both readers and QA accuracy, we first average over questions within a store and then over stores.

Baselines. To compare scoring policies, we applied each baseline to the same memories and retained its top K. For **SAGE** (Wang et al., 2026), we used the von Mises-Fisher novelty score over normalized memory embeddings and retained memories with higher novelty. For a unit-normalized memory vector x and resident vectors $\{b_j\}_{j=1}^{m}$, the score is

$$N(x) = 1 - \frac{1}{\kappa} \log\Big(\tfrac{1}{m} \textstyle\sum_{j=1}^{m} \exp(\kappa\, x^\top b_j) \Big), \quad \kappa = \text{clip}\Big(\tfrac{r(d-r^2)}{1-r^2},\, 10^{-3},\, 50 \Big), \quad r = \big\| \tfrac{1}{m} \textstyle\sum_j b_j \big\|, \tag{6}$$

where d is the embedding dimension and κ is estimated following Banerjee et al. (2005). **A-MAC** (Zhang et al., 2026) combines estimated future usefulness, confidence, novelty, recency, and a content-type prior into an admission score. We obtained the usefulness rating by prompting a language model with the candidate and its source context, and computed confidence with ROUGE-L recall against the source (Lin, 2004). Novelty is one minus the largest cosine similarity to a resident, clipped at zero. Recency decays as $\exp(-0.01 \text{ age in hours})$ relative to the newest session, and keyword counts determine the type prior. In the main comparisons, A-MAC gives equal weight to the five standardized factors. The **random** baseline averages four selections of K memories per store, using the same selections for both readers and QA. We evaluate the retained subsets from all policies using the reader-based utility in Eq. 5.

3 RESULTS

To determine whether learning a memory's credit improves retention, we compared the answers obtained from the memories selected by each policy. Qwen3-8B generated the answers and graded them using the LongMemEval rubric (Table 2). Accuracy is higher with the learned policy than with A-MAC at four tested capacities and SAGE at all five. The advantage is greatest at higher capacity pressure, while accuracy generally rises and the differences between policies narrow as capacity increases.

Table 2: QA accuracy (%) with Qwen3-8B as reader and grader, using the LongMemEval rubric. Each result averages accuracy over questions within a store and then over stores. The oracle chooses memories by their utility in Eq. 5. Bold marks the highest mean among policies.

Policy	K=5	K=4	K=3	K=2	K=1
Oracle	61.4	62.9	58.7	54.9	39.1
SCAP (ours)	**61.0**	**53.9**	47.4	**48.3**	**35.6**
A-MAC	59.4	52.0	**49.1**	39.9	27.9
SAGE	60.7	49.8	42.6	39.8	28.1
Random	59.4	50.5	44.5	36.2	30.0

QA accuracy records whether a generated answer is judged correct, so changes in the information available to the reader can leave this binary outcome unchanged. In our data, each question has one supporting session, and different selections can therefore yield the same correct answer once that session is retained. Reference-answer codelength on the other hand captures changes in the reader's predictions even when the correctness judgment stays the same.

We therefore measured how the selected memories changed the reader's probability of the reference answer (Table 3). Those chosen by the learned policy have greater utility at every capacity, from 4.8 bits at K=1 to 19.4 bits at K=5. Retained utility increases with capacity for all policies, while the learned policy's advantage is largest when the capacity pressure is higher.

Table 3: Mean retained utility in bits (Eq. 5). The oracle maximizes utility over subsets of exactly K memories. Bold marks the highest mean among policies.

Policy	K=5	K=4	K=3	K=2	K=1
Oracle	28.4	30.2	30.4	26.2	19.4
SCAP (ours)	**19.4**	**16.6**	**14.2**	**9.2**	**4.8**
A-MAC	16.6	12.8	8.5	-0.6	-6.5
SAGE	16.2	11.7	7.4	2.5	-4.1
Random	14.7	9.6	6.2	0.4	-2.2

To assess whether the learned policy transfers to an agent using a different model family never seen in training, we replaced the Qwen3-8B reader with Llama-3.1-8B (Grattafiori et al., 2024) and repeated the evaluation using the same policy and selected memories (Table 4). The learned policy again selects memories with greater utility at every capacity, with larger differences at higher capacity pressure. Its advantage therefore persists when the agent changes, without retraining the policy.

Table 4: Mean retained utility in bits under Llama-3.1-8B. Policies use the selections from Table 3; the oracle selects the K memories with greatest utility under Llama. Bold marks the highest mean among policies.

Policy	K=5	K=4	K=3	K=2	K=1
Oracle	23.2	22.9	21.5	18.4	10.9
SCAP (ours)	**20.2**	**17.6**	**15.0**	**11.2**	**4.2**
A-MAC	17.5	14.6	10.0	5.3	-1.3
SAGE	17.5	13.3	9.1	5.3	-0.7
Random	17.5	13.7	10.1	5.0	-0.1

3.1 Ablations

To understand which parts of the policy contribute to its performance, we changed how memories are represented, how credit is assigned during training, and whether the policy can read the other mem-

ories in the store. We retrained each version on the same stores using the procedure in Section 2.7, so the comparisons reflect the change being examined.

Learning A-MAC's weights. The main A-MAC baseline standardizes its factors using training-store statistics and sums them with equal weights. To assess whether learning these weights closes the gap in retained utility, we trained a linear predictor on the same Shapley labels and training/validation split as our policy. We standardized the factors using the fitting examples and minimized Huber loss with δ=8, nonnegative weights, and an unpenalized intercept. Validation loss selected an L2 penalty of 1 from $\{0, 10^{-4}, 10^{-3}, 10^{-2}, 0.1, 1, 10\}$. Our policy retains greater utility at every capacity, with the largest advantage at higher capacity pressure (Table 5). Its advantage therefore remains when A-MAC's weights are learned from the same credit labels.

Table 5: Mean retained utility in bits with equal and learned A-MAC weights. Fitted A-MAC and SCAP use the same credit labels and training/validation split. Bold marks the highest mean.

Policy	K=5	K=4	K=3	K=2	K=1
SCAP (ours)	**19.4**	**16.6**	**14.2**	**9.2**	**4.8**
A-MAC, fitted weights	15.6	11.6	7.5	0.4	−4.3
A-MAC, equal weights	16.6	12.8	8.5	−0.6	−6.5

Record embeddings. We compared embedding each record separately with embedding the complete memory text, including its date header, in chunks and pooling the resulting vectors. Using separate record embeddings gives higher retained utility at every capacity, with the difference generally narrowing as capacity pressure decreases (Table 6). We also compared a hierarchical encoder that first pools records within each memory. To compare their rankings, we measured pairwise accuracy, the fraction of memory pairs within each test store for which the policy gives the higher score to the memory with greater Shapley credit, excluding pairs with equal credit. The hierarchical encoder reached 0.52, compared with 0.60 when attention could use all records directly.

Table 6: Mean retained utility in bits when the policy receives separate record embeddings or one pooled vector per memory. The pooled vector is computed from chunks of the complete memory text. Both predictors use the same labels, data split, and training settings.

Representation	K=5	K=4	K=3	K=2	K=1
Separate records (used in ours)	**19.4**	**16.6**	**14.2**	**9.2**	**4.8**
Complete memory, pooled	18.6	14.6	10.9	6.1	1.2

Credit assignment. To compare attribution rules, we computed exact Shapley credit and leave-one-out credit, $v(N) - v(N \setminus \{i\})$, using the test questions and reference answers. We then ranked memories by these credits (Table 7). Ranking by Shapley credit retains greater utility at every capacity, with larger differences at higher capacity pressure and similar utility at larger capacities. Next, we trained the policy on each of the two targets. Pairwise accuracy against the same Shapley labels is 0.60 with Shapley training targets and 0.58 with leave-one-out targets. The policy trained on Shapley credit also retains greater utility at every tested capacity (Table 8).

Store context. To assess whether the other memories help predict a candidate's credit, we trained a version of the policy using only the candidate's records. It used the same projection, attention pooling, and prediction head, with resident records and segment tags removed. Pairwise accuracy falls from 0.60 to 0.53, and retained utility decreases at every capacity (Table 8). Indicating that the contents of the store help the policy judge which memories to keep.

4 Discussion

We developed a simple method for learning which memories to retain before the questions that determine their utility arrive. The method measures a memory set's utility as the additional information

Table 7: Mean retained utility in bits when memories are ranked by credit computed from the test questions and reference answers. Single-question credit uses the first question selected when constructing each store. Oracle and random use the same memories as Table 3.

Credit rule	K=5	K=4	K=3	K=2	K=1
Oracle	28.4	30.2	30.4	26.2	19.4
Shapley (used in ours)	**25.5**	**26.6**	**27.4**	**22.7**	**17.2**
Leave-one-out	25.3	26.5	24.7	19.6	13.7
Single-question Shapley	22.6	20.1	17.1	14.3	12.5
Random	14.7	9.6	6.2	0.4	−2.2

Table 8: Predictors trained with different targets and inputs, using the same split and training settings. Pairwise accuracy is evaluated against the same Shapley labels. The remaining columns give mean retained utility in bits. All predictors receive the candidate and resident records except the candidate-only predictor. Bold marks the highest value in each column.

Predictor	Pairwise	K=5	K=4	K=3	K=2	K=1
Shapley (used in ours)	**0.601**	**19.4**	**16.6**	**14.2**	**9.2**	**4.8**
Leave-one-out	0.581	18.5	16.4	12.9	9.0	4.5
Single-question	0.505	13.9	11.4	9.3	4.6	−0.3
Shapley, candidate only	0.531	15.5	12.1	9.7	1.5	−0.6

it provides a frozen reader about answers to possible future questions, quantified by the reduction in their codelength. It assigns individual credit through the Shapley value and trains a policy to predict that credit from the memories alone. A single regression objective over frozen record embeddings thus connects the utility measured during training to the policy's decisions under unknown demand, and it is transferrable across agents.

On held-out questions and supporting sessions, the learned policy retained greater utility than the selections made using A-MAC and SAGE at every tested capacity, with the largest advantages at higher capacity pressure. These gains were achieved with a single attention-pooling layer and a scalar prediction head, showing that the proposed credit can support effective retention with a small trainable model. The advantage also remained when A-MAC's weights were fitted to the same credit labels, extending the comparison to policies trained with the same supervision.

The ablations clarify which information helps the policy make these decisions. Pooling a complete memory into one vector or scoring a candidate alone reduced retained utility, supporting access to individual records and the surrounding store. Training on Shapley credit also yielded higher retained utility than training on leave-one-out credit, supporting the use of contributions across subsets when assigning the learning target.

These findings suggest extending the policy to stores that change over time, where each admission changes the context for later decisions. Learning from the consequences of successive admissions and evictions could help the policy account for utility over longer horizons, while updating credit as demand changes could adapt retention to an agent's evolving tasks. A further direction is to select groups of memories under a token budget, accounting for both their lengths and the utility of keeping complementary information together.

AI USE STATEMENT

Generative AI tools assisted literature discovery, implementation, and preparing the first manuscript draft. Language models used in the experiments are described in Section 2 and Appendix A.

REPRODUCIBILITY STATEMENT

Appendix A gives the prompts, model revisions, and procedures for encoding, scoring, and grading.

REFERENCES

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. arXiv preprint arXiv:1607.06450, 2016. URL https://arxiv.org/abs/1607.06450.

Arindam Banerjee, Inderjit S. Dhillon, Joydeep Ghosh, and Suvrit Sra. Clustering on the unit hypersphere using von Mises-Fisher distributions. *Journal of Machine Learning Research*, 6(46): 1345–1382, 2005. URL https://www.jmlr.org/papers/v6/banerjee05a.html.

Nishant Bhargava and Rodrigo Sobral Barrento. MEMAUDIT: An exact package-oracle evaluation protocol for budgeted long-term LLM memory writing. arXiv preprint arXiv:2605.02199, 2026. URL https://arxiv.org/abs/2605.02199.

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 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 15159–15177, 2024. doi: 10.18653/v1/2024.emnlp-main.845. URL https://aclanthology.org/2024.emnlp-main.845/.

Gil Einziger, Roy Friedman, and Ben Manes. TinyLFU: A highly efficient cache admission policy. *ACM Transactions on Storage*, 13(4):1–31, 2017. doi: 10.1145/3149371. URL https://doi.org/10.1145/3149371.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, et al. The Llama 3 herd of models. arXiv preprint arXiv:2407.21783, 2024. URL https://arxiv.org/abs/2407.21783.

Peter D. Grünwald. *The Minimum Description Length Principle*. MIT Press, 2007. ISBN 9780262072816. URL https://mitpress.mit.edu/9780262072816/the-minimum-description-length-principle/.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (GELUs). arXiv preprint arXiv:1606.08415, 2016. URL https://arxiv.org/abs/1606.08415.

Tianyu Hu, Weikai Lin, Weizhi Zhang, Jing Ma, and Song Wang. MemRouter: Memory-as-embedding routing for long-term conversational agents. arXiv preprint arXiv:2605.00356, 2026a. URL https://arxiv.org/abs/2605.00356.

Yuanzhe Hu, Yu Wang, and Julian McAuley. Evaluating memory in LLM agents via incremental multi-turn interactions. In *International Conference on Learning Representations*, 2026b. URL https://openreview.net/forum?id=DT7JyQC3MR.

Peter J. Huber. Robust estimation of a location parameter. *The Annals of Mathematical Statistics*, 35(1):73–101, 1964. doi: 10.1214/aoms/1177703732.

Qingcan Kang, Mingyang Liu, Shixiong Kai, Kaichao Liang, Tao Zhong, and Mingxuan Yuan. Learning what to remember: Observability-safe memory retention via constrained optimization for long-horizon language agents. arXiv preprint arXiv:2606.10616, 2026. URL https://arxiv.org/abs/2606.10616.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015. URL https://arxiv.org/abs/1412.6980.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 3744–3753. PMLR, 2019. URL https://proceedings.mlr.press/v97/lee19d.html.

Nusrat Jahan Lia and Aritra Mazumder. Learning what not to forget: Long-horizon agent memory from a few kilobytes of learning. arXiv preprint arXiv:2606.20954, 2026. URL https://arxiv.org/abs/2606.20954.

Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pp. 74–81, 2004. URL https://aclanthology.org/W04-1013/.

Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, volume 30, 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/hash/8a20a8621978632d76c43dfd28b67767-Abstract.html.

Weitao Ma, Xiaocheng Feng, Lei Huang, Xiachong Feng, Zhanyu Ma, Jun Xu, Jiuchong Gao, Jinghua Hao, Renqing He, and Bing Qin. Fine-Mem: Fine-grained feedback alignment for long-horizon memory management. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 19666–19684. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.900. URL https://aclanthology.org/2026.acl-long.900/.

Sewon Min, Kalpesh Krishna, Xinxi Lyu, Mike Lewis, Wen-tau Yih, Pang Koh, Mohit Iyyer, Luke Zettlemoyer, and Hannaneh Hajishirzi. FActScore: Fine-grained atomic evaluation of factual precision in long form text generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 12076–12100, 2023. doi: 10.18653/v1/2023.emnlp-main.741. URL https://aclanthology.org/2023.emnlp-main.741/.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST)*, pp. 1–22, 2023. doi: 10.1145/3586183.3606763. URL https://doi.org/10.1145/3586183.3606763.

Lloyd S. Shapley. A value for n-person games. In Harold W. Kuhn and Albert W. Tucker (eds.), *Contributions to the Theory of Games II*, pp. 307–318. Princeton University Press, 1953. doi: 10.1515/9781400881970-018. URL https://doi.org/10.1515/9781400881970-018.

Saksham Sahai Srivastava. Causal intervention-based memory selection for long-horizon LLM agents. arXiv preprint arXiv:2605.17641, 2026. URL https://arxiv.org/abs/2605.17641.

Erik Štrumbelj and Igor Kononenko. Explaining prediction models and individual predictions with feature contributions. *Knowledge and Information Systems*, 41(3):647–665, 2014. doi: 10.1007/s10115-013-0679-x.

Hongyao Tang. Towards a formal definition of agent memory: Basis, span, optimality, and the sequential memory problem. arXiv preprint arXiv:2608.11654, 2026. URL https://arxiv.org/abs/2608.11654.

Endel Tulving. How many memory systems are there? *American Psychologist*, 40(4):385–398, 1985. doi: 10.1037/0003-066X.40.4.385.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Sijia Wang, Dhanajit Brahma, and Ricardo Henao. SAGE: A novelty gate for efficient memory evolution in agentic LLMs. arXiv preprint arXiv:2605.30711, 2026. URL https://arxiv.org/abs/2605.30711.

Daniel Lin-Kit Wong, Hao Wu, Carson Molder, Sathya Gunasekar, Jimmy Lu, Snehal Khandkar, Abhinav Sharma, Daniel S. Berger, Nathan Beckmann, and Gregory R. Ganger. Baleen: ML admission & prefetching for flash caches. In *22nd USENIX Conference on File and Storage Technologies (FAST)*, pp. 347–371. USENIX Association, 2024. URL https://www.usenix.org/conference/fast24/presentation/wong.

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*, 2025. URL https://arxiv.org/abs/2410.10813.

Yilun Xu, Shengjia Zhao, Jiaming Song, Russell Stewart, and Stefano Ermon. A theory of usable information under computational constraints. In *International Conference on Learning Representations*, 2020. URL https://arxiv.org/abs/2002.10689.

Sikuan Yan, Ahmed Bahloul, Ercong Nie, Susanna Schwarzmann, Riccardo Trivisonno, Volker Tresp, and Yunpu Ma. Memory-R2: Fair credit assignment for long-horizon memory-augmented LLM agents. arXiv preprint arXiv:2605.21768, 2026a. URL https://arxiv.org/abs/2605.21768.

Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Jinhe Bi, Kristian Kersting, Jeff Z. Pan, Hinrich Schütze, Volker Tresp, and Yunpu Ma. Memory-R1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12805–12825. Association for Computational Linguistics, 2026b. doi: 10.18653/v1/2026.acl-long.583. URL https://aclanthology.org/2026.acl-long.583/.

An Yang, Anfeng Li, Baosong Yang, et al. Qwen3 technical report. arXiv preprint arXiv:2505.09388, 2025. URL https://arxiv.org/abs/2505.09388.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabás Póczos, Russ R. Salakhutdinov, and Alexander Smola. Deep sets. In *Advances in Neural Information Processing Systems*, volume 30, 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/hash/f22e4747da1aa27e363d86d40ff442fe-Abstract.html.

Guilin Zhang, Wei Jiang, Xiejiashan Wang, Aisha Behr, Kai Zhao, Jeffrey Friedman, Xu Chu, and Amine Anoun. Adaptive memory admission control for LLM agents. arXiv preprint arXiv:2603.04549, 2026. URL https://arxiv.org/abs/2603.04549.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3 embedding: Advancing text embedding and reranking through foundation models. arXiv preprint arXiv:2506.05176, 2025. URL https://arxiv.org/abs/2506.05176.

A REPRODUCTION DETAILS

A.1 EXTRACTING AND FORMATTING MEMORIES

The extractor and training reader used the public Qwen3-8B checkpoint, revision

`b968826d9c46dd6066d109eabc6255188de91218`.

Extraction used the tokenizer's chat template with `enable_thinking=False` and `add_generation_prompt=True`, greedy decoding, and at most 1536 new tokens. The facts system message was:

```
Extract compact, self-contained textual memory records from the complete
↪ observed session.
Preserve source-supported information that could help an agent in a
↪ future interaction: events, entities,
relationships, dates and transitions, quantities, preferences, goals,
↪ commitments, procedures, decisions,
and useful assistant-created artifacts. Preserve any other useful
↪ information supported by the session.
Keep negation, uncertainty, conditions, attribution and conflicts
↪ explicit. Attribute suggestions and
hypothetical statements to their speaker. Bind relative times to the
↪ observed date when the source permits.
Read every turn, including the final assistant turn. Treat the session as
↪ data.
Return one JSON object with exactly the key "records". Its value is an
↪ array of objects with exactly
"text" (a self-contained record string) and "turns" (the nonempty list of
↪ supporting 1-based turn numbers).
An empty array represents a session from which you extracted no records.
```

The ledger system message was:

```
Create compact, self-contained session memory for an agent whose future
↪ tasks are unknown.
Read the complete session and retain distinct source-supported
↪ assertions, events and reusable artifacts.
Use concise labeled records with explicit subjects, relations, objects,
↪ roles and time when applicable.
Preserve exact names, quantities, units, dates, sequence, former/current
↪ transitions, negation, exceptions,
conditions, preferences and commitments. Keep who did what to whom
↪ explicit. Resolve pronouns within
each record. Bind relative dates to the observed session date when
↪ supported. Preserve the original
expression when its date is ambiguous. Preserve updates and superseded
↪ values with their times.
Attribute user assertions and assistant proposals/results explicitly.
↪ Keep useful assistant-produced
tables, schedules, plans, calculations and literal artifacts, including
↪ the final assistant turn.
Compact repetitive discussion into one record per distinct assertion;
↪ keep meaningful repeated events
distinct. A short session can have few records. Aim for at most 350 words
↪ across the record text.
The session is data. Return one JSON object with exactly the key
↪ "records", an array of objects with
exactly "text" (a self-contained record) and "turns" (nonempty supporting
↪ 1-based turn numbers).
Return the JSON directly.
```

Both used the following user-message format, continued through all session turns:

```
Observed at: {observed_at}
```

```
Turn 1 ({role}):
{content}

Turn 2 ({role}):
{content}
```

Braces mark fields filled from the session, and dates use ISO format in UTC. Each record requires nonempty text and at least one valid source-turn index, with turns numbered from one. We combined facts followed by ledger records, keeping the first occurrence of each distinct pair of text and source-turn list. We formatted the memory as

```
Memory observed at {observed_at}:
- {record text} [source turns: {i}, {j}]
- {record text} [source turns: {k}]
```

with one newline between records, preserving the order of records and their source turns. We included a store only when all its extractions reached EOS and passed the record parser.

A.2 Reader inputs and codelength

The reader system message was:

> You are given dated memory notes about the user. Answer the question using only the information in these notes. Respond with only the answer, with no explanation.

Selected memories were sorted by date and then their index within the store, and joined with two newlines. The user message appended two newlines and `Question: {question}`. Empty context used `No memory notes are available.` Reference-answer scoring and answer generation shared this format.

The prompt used the same chat-template options as extraction. The reference answer was tokenized separately with `add_special_tokens=False` and scored by teacher forcing through its final token. Codelength is the negative sum of natural-log token probabilities divided by $\log 2$, with EOS excluded from the target. Scoring used 256-token blocks while preserving the full preceding KV cache. Inputs exceeding the model's context length raised an error.

To compute training credit, we used every subset for stores of up to six memories and 64 random permutations for larger stores. The seed was 101 plus the store's zero-based index in construction order. For each permutation, we evaluated the subsets formed by adding memories in that order. Retained utility and the Shapley rows in Table 7 used every subset of each test store. Pairwise accuracy used the same test labels for all predictors, including the estimates obtained by sampling, and omitted pairs whose labels differed by at most 10^{-6} bits. Single-question credit used the first question selected when constructing each store.

A.3 Embedding records and memories

All embedding-based methods used the public Qwen3-Embedding-4B checkpoint, revision

`5cf2132abc99cad020ac570b19d031efec650f2b`.

We tokenized the complete text with special tokens disabled and divided it into consecutive chunks of at most 2048 tokens. Each chunk's vector was the 2560-dimensional hidden state of its last nonpadding token, normalized to unit length. We averaged chunk vectors equally and renormalized the result.

For our policy, we embedded each record line separately, including its source-turn references. A-MAC, SAGE, and the single-vector comparison embedded the complete memory text, including its date header. The frozen encoder ran in FP16 with left padding, and empty inputs received the EOS token. The policy applied a linear projection, GELU (Hendrycks & Gimpel, 2016), and LayerNorm (Ba et al., 2016) to the embeddings, then added candidate or resident tags. One learned query with four attention heads pooled these vectors, and a feed-forward head predicted credit.

To obtain A-MAC's rating of future usefulness, we prompted the frozen Qwen model with greedy decoding, at most eight new tokens, and the following user message:

```
Given the following conversation context:

{context}

Candidate memory to evaluate:
"{memory}"

Rate the likelihood that this memory will be useful for continuing the
↪  conversation or completing future tasks. Provide a score between 0.0
↪  (completely useless) and 1.0 (extremely useful).

Your response should be a single floating-point number between 0.0 and
↪  1.0.

Score:
```

The context listed the source session's last five turns as `Turn j: content`, numbered from zero and separated by newlines. We joined the candidate's record texts with spaces. The parser first tried to read the response as a number, then searched for the first numeric pattern. It clipped the value to $[0, 1]$ and assigned 0.5 if both attempts failed.

To compute confidence, we split the joined records and joined source turns on whitespace, found their longest common subsequence, and divided its length by the memory's token count. For the type prior, we counted occurrences of the substrings in Table 9, ignoring case. We chose the category with the largest count, broke ties by table order, and assigned 0.5 when no substring matched. Novelty and recency follow Section 2.8.

Table 9: Keywords and priors used for A-MAC's content-type factor.

Category	Keywords	Prior
Preference	prefer, like, favorite, enjoy, love, hate, dislike	0.9
Identity	i am, my name, i live, i work, i'm a	0.9
Fact	is, are, was, were, has, have, born, located	0.7
Plan	will, plan, going to, intend, schedule, want to	0.5
Temporary	today, now, currently, this week, tomorrow, tonight	0.2

A.4 Answer grading and evaluation with another reader

For QA, Qwen3-8B generated answers with greedy decoding and at most 128 new tokens. The same frozen checkpoint graded answers with at most 10 new tokens and the question-type-specific LongMemEval rubric, `get_anscheck_prompt`, from the public LongMemEval evaluation script, commit

`9e0b455f4ef0e2ab8f2e582289761153549043fc`.

We supplied the question type, question, reference answer, generated answer, and `abstention=False`, then passed the formatted rubric as one user message. Both stages used the non-thinking chat template, EOS stopping, and EOS padding. Grader output was parsed as case-insensitive `yes` or `no`, allowing a final period or exclamation mark and surrounding whitespace. Other outputs remained unresolved. All grades required for the reported QA results were resolved.

We evaluated the policy with the public Llama-3.1-8B-Instruct checkpoint, revision

`d10aef7999a2b5ba950ab3974312feeedbfe0b77`,

in BF16 with its own chat template and tokenizer. We used the same memory text, questions, reference answers, and policy, including the same selected memories. Scoring used 512-token blocks

with the full preceding KV cache. For the Llama oracle, we evaluated every subset of size K for each test store, averaged its utility over the store's questions, and selected the subset with greatest mean utility. We then averaged these maxima across stores.

For the random baseline, we drew four subsets per store and capacity. For draw r, we seeded NumPy's default generator with the integer represented by the first 12 hexadecimal digits of SHA-256 of `store_id|K|r`, then sampled without replacement. Both readers and the QA evaluation used the same subsets.