Learning to Remember: End-to-End Training of Memory Agents for Long-Context Reasoning

Anonymous ACL submission

Abstract

Long-context LLMs and Retrieval-Augmented Generation (RAG) systems expand accessible context, but still defer state tracking and evidence consolidation to query time. This passive paradigm is brittle in streaming settings where facts are updated and answers depend on latent states rather than explicit spans. We introduce Unified Memory Agent (UMA), an end-to-end reinforcement-learning framework that trains a single policy to maintain external memory and answer questions. UMA updates a structured memory bank via CRUD operations and combines it with raw-context retrieval for QA. To jointly optimize memory maintenance and question answering, we propose Task-Stratified GRPO, which normalizes heterogeneous trajectories separately for more effective credit assignment. We also introduce Ledger-QA, a controlled benchmark for long-horizon state tracking over accumulated updates. Experiments on Ledger-QA and 13 TTL/AR benchmarks show that UMA improves long-horizon tracking, achieves the best average TTL/AR score among compared methods, and remains competitive on retrieval-centric benchmarks. Our results highlight learned proactive memory management as a complement to longer context windows and query-time retrieval.

1 Introduction

Modern Large Language Models (LLMs) increasingly support contexts of hundreds of thousands of tokens, with some reaching million-token windows (Liu et al., 2024; Anthropic, 2026; OpenAI, 2026; Kavukcuoglu et al., 2026; Qwen Team, 2026; Shen et al., 2025). Yet long-context modeling faces challenges beyond window size: standard attention scales quadratically with sequence length (Vaswani et al., 2017), making very long inputs costly, and empirical studies report *context rot* (Hong et al., 2025), where performance degrades as input length grows even when relevant information is present.

Retrieval-Augmented Generation (RAG) scales by retrieving relevant passages at query time, but shares the same **passive information processing** limitation with long-context approaches (Xu et al., 2024; Hu et al., 2025b; Zhang et al., 2025b): context is treated as static raw material and reasoning is deferred until a query arrives. As Figure 1 shows, RAG reprocesses retrieved passages for every query; on state tracking, contradiction resolution, and evidence aggregation, this creates both accuracy degradation and computational redundancy.

Memory-augmented agents instead consolidate information as it arrives. However, existing trainable memory agents often optimize memory maintenance and question answering separately. Memory-R1 (Yan et al., 2025), Mem-α (Wang et al., 2025c), and Mem-T (Yue et al., 2026) introduce learned memory components, but rely on staged or decoupled training, limiting credit assignment from final outcomes to earlier memory decisions.

We propose **Unified Memory Agent (UMA)**, an end-to-end reinforcement learning framework that jointly optimizes CRUD-style memory maintenance and question answering within one policy. We also introduce **Ledger-QA**, a benchmark for continuous state tracking that directly tests proactive consolidation rather than raw-history reprocessing. We evaluate on Ledger-QA and 13 TTL/AR datasets. UMA outperforms long-context and RAG baselines on dynamic reasoning tasks, remains competitive on retrieval benchmarks, and ablations support the value of end-to-end optimization.

2 Method

2.1 Problem Formulation as MDP

We formulate the long-context reasoning task as a Markov Decision Process (MDP). To handle sequences exceeding the context window, we decompose the input into a stream of chunks $C =$

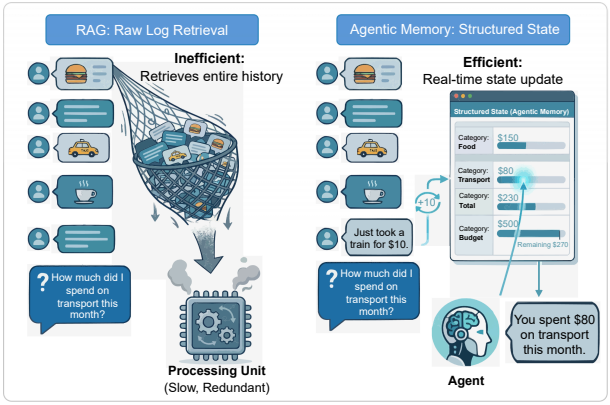

Figure 1: Expense-tracking example: RAG reprocesses logs per query, while agentic memory maintains structured state and answers from relevant fields.

$\{c_1, \ldots, c_N\}$. The MDP tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ is defined as follows:

State Space ($\mathcal{S}$). At step t, the state $s_t = (\mathcal{M}_t, x_t, h_t)$ consists of:

- $\mathcal{M}_t = (m_t^{core}, \mathcal{B}_t)$: The dual-component memory, comprising a high-level summary (m_t^{core}) and a structured key-value memory bank ($\mathcal{B}_t$).
- x_t: The current input focus. To circumvent context window limits, this is dynamically switched between a local text chunk c_k (during memory maintenance) and a specific query q (during QA).
- h_t: The immediate interaction history, which buffers recent observations and is periodically cleared to manage context length.

Action Space ($\mathcal{A}$). We unify all operations into a single action space, categorized by the phase of interaction. Crucially, read-only tools are available in both phases to ensure the agent can verify beliefs or query the memory at any time. Detailed specifications for executable tools and terminal-action parsing conventions are provided in Appendix A.

- **Memory Maintenance Operations ($\mathcal{A}_{mem}$):** Used during the chunk processing phase to maintain the state.
  - `Add`, `Update`, `Delete`: Write operations that create, revise or rename, and remove entries in $\mathcal{B}_t$.
  - `Retrieve`, `List`: Read-only operations for accessing entries and keys in $\mathcal{B}_t$.
  - `UpdateCore`: A special terminal action that commits the current progress to m_t^{core} and triggers the transition to the next chunk.
- **QA Operations ($\mathcal{A}_{qa}$):** Used during the query phase to synthesize the answer.
  - `BM25`, `Embedding`: Ranked retrieval actions for lexical and semantic search.
  - `Retrieve`, `List`: Read-only operations for accessing entries and keys in $\mathcal{B}_t$.
  - `Answer`: The terminal action to generate the final response.

Transition Dynamics ($\mathcal{P}$). The system evolves through two nested loops plus a final phase switch:

- **Inner Loop (within one session/chunk):** The agent performs multiple tool calls before committing the chunk. Each step appends interaction traces (tool calls and observations) to h_t and may update the memory bank $\mathcal{B}_t$, while the current input focus $x_t = c_k$ remains unchanged.
- **Outer Loop (across chunks):** A terminal `UpdateCore` action ends the current inner-loop episode, writes the consolidated update to m_t^{core}, advances x_t from c_k to c_{k+1}, and re-initializes h_t for the next chunk.
- **Phase Transition:** After processing c_N, the system switches from maintenance to QA, where x_t becomes a query q sampled from $P(q|C)$.

This transition structure implies that the policy should both optimize short-horizon inner-loop edits on $\mathcal{B}_t$ and optimize outer-loop consolidation into m_t^{core}, so that the final memory state $\mathcal{M}_N$ supports future queries.

2.2 Unified Memory Agent Architecture

Based on the MDP formulation, UMA employs a unified policy $\pi_\theta(a_t|s_t)$ to generate actions. The architecture is designed to handle the inputs explicitly as follows:

Input Representation. At any time step t, the input context provided to the LLM is a concatenation of four segments:

$$\text{Input}_t = [I_{sys}, m_t^{core}, x_t, h_t] \quad (1)$$

- **System Instruction (I_{sys}):** Specifies the phase-specific operational guidelines—governing either memory maintenance or question answering—and defines the available toolset ($\mathcal{A}_{mem}$ or $\mathcal{A}_{qa}$).
- **Core Memory (m_t^{core}):** The evolving high-level summary that provides global context.

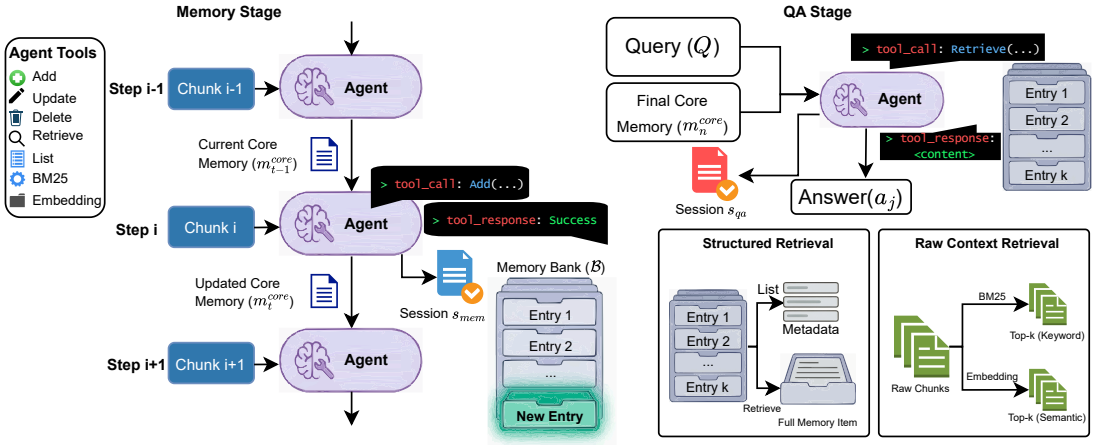

Figure 2: **Overview of UMA.** Phase I incrementally maintains a structured Memory Bank and core summary via CRUD over chunks. Phase II answers queries using both structured retrieval from the bank and raw-context retrieval.

- **Current Focus (x_t):** The raw text of the current chunk c_k (in Phase I) or the user query q (in Phase II).
- **Interaction History (h_t):** The trajectory of recent tool calls and their execution results.

The Unified Memory Agent framework contains two phases, as illustrated in Figure 2.

Phase I: Sequential Memory Maintenance. For each chunk c_k, the agent initializes $h_t = \emptyset$. It iteratively samples actions

$$a_t \sim \pi_\theta(\cdot | I_{sys}, m_t^{core}, c_k, h_t) \tag{2}$$

The environment executes the tool and appends the result to h_t. This loop continues until the agent outputs `UpdateCore`, which compresses relevant insights into m_t^{core} and advances the process to c_{k+1}.

Phase II: Hybrid Retrieval-Augmented QA. After processing the final chunk, the input focus x_t switches to the query q. The agent utilizes the fully constructed memory $\mathcal{M}_N$ and an empty history h_t. It can invoke hybrid retrieval tools (exact-title access via `Retrieve` and ranked search via `BM25/Embedding`) to gather evidence. The process terminates when the agent generates the `Answer` action.

2.3 Training: Task-Stratified GRPO

We propose a **Task-Stratified Group Relative Policy Optimization** algorithm. By viewing GRPO as a Monte Carlo (MC) advantage estimator without a value network, we adapt it to handle the hierarchical structure of our task as illustrated in Figure 3.

2.3.1 Nested Trajectory Sampling

To estimate the value of memory actions effectively, we employ a nested sampling strategy during training. For a given context C:

1. **Memory Rollout (trajectory level):** We sample G distinct trajectories $\{\tau^1, \ldots, \tau^G\}$. Each trajectory τ^i contains multiple memory-tool-calling sessions $\{s_{mem}^{i,1}, s_{mem}^{i,2}, \ldots, s_{mem}^{i,N}\}$ and yields one final memory state $\mathcal{M}_N^i$, where N is the number of chunks in C, which also equals the number of memory sessions in trajectory τ^i.
2. **Query Sampling:** For each final memory state $\mathcal{M}_N^i$, we sample M questions $\{q_1, \ldots, q_M\}$ from the dataset.
3. **QA Rollout (session level):** For each pair $(\mathcal{M}_N^i, q_j)$, we generate one multi-turn QA-tool-calling session $s_{qa}^{i,j}$ and compute its reward $R_{qa}^{i,j}$.

Thus, each update contains G memory trajectories and $G \times M$ QA sessions.

2.3.2 Reward Function

Rewards are computed at the *session* level, with different outcome assignments for QA and memory sessions. Accordingly, actions within the same tool-calling session share the same outcome-derived advantage.

For each QA session $s_{qa}^{i,j}$, the reward is

$$\begin{aligned} R_{qa}^{i,j} &= \lambda r_{tool}^{qa,i,j} + r_{outcome}^{i,j}, \\ r_{outcome}^{i,j} &= r_{outcome}(a_{ans}^{i,j}, y_j) \end{aligned} \tag{3}$$

where $r_{tool}^{qa,i,j}$ is the tool success signal of this QA session.

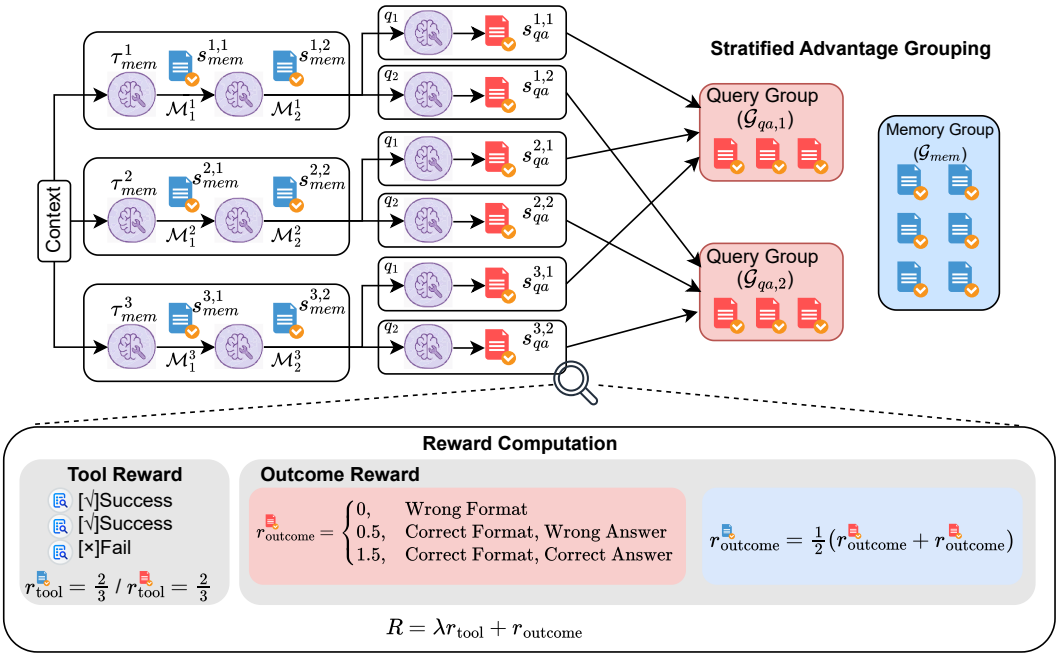

Figure 3: **Illustration of Task-Stratified GRPO.** For a given input, we first sample multiple *memory trajectories* (blue), each producing a final memory state; then for each memory state and query, we sample a *QA session* (red). **(Right)** The reward function combines immediate tool execution feedback (r_{tool}) with outcome assessments ($r_{outcome}$). Crucially, memory sessions receive a *Future Utility Signal* derived from subsequent QA rewards. **(Bottom)** Advantages are normalized within distinct groups: all memory steps are aggregated into a global pool ($\mathcal{G}_{mem}$), while QA steps are normalized strictly within their specific query groups ($\mathcal{G}_{qa,j}$).

For each memory session $s^{i,k}_{mem}$ in trajectory τ^i, we keep an independent tool reward and use the averaged QA outcome under the same trajectory as its outcome signal:

$$\begin{aligned} \bar{r}^{i}_{outcome} &= \frac{1}{M}\sum_{j=1}^{M} r^{i,j}_{outcome}, \\ R^{i,k}_{mem} &= \lambda r^{mem,i,k}_{tool} + \bar{r}^{i}_{outcome} \end{aligned} \tag{4}$$

where $r^{mem,i,k}_{tool}$ is computed from the tool-call validity of memory session $s^{i,k}_{mem}$ (e.g., valid-call ratio N_{valid}/N_{total} for that session). Here, λ controls the trade-off between tool-use quality and answer outcome, and is set to 0.5 by default.

2.3.3 Task-Stratified Advantage Estimation

Given the session-level rewards defined above, we compute GRPO advantages via group-wise normalization. Our extension uses two stratified groups for memory and QA, respectively.

Advantage for Memory Steps (A_{mem}). For memory steps, we assign advantages at the *session level*. Specifically, each memory-session reward is normalized against all memory sessions sampled in the current update:

$$\begin{aligned} A^{i,k}_{mem} &= \frac{R^{i,k}_{mem} - \bar{R}_{mem}}{\sigma(R^{\cdot,\cdot}_{mem}) + \epsilon_{std}}, \\ \bar{R}_{mem} &= \frac{1}{GN}\sum_{u=1}^{G}\sum_{v=1}^{N} R^{u,v}_{mem} \end{aligned} \tag{5}$$

This provides a session-level credit signal for each memory session.

Advantage for QA Steps (A_{qa}). For a specific question q_j, the difficulty is intrinsic. Therefore, we normalize each QA-session reward only against other sessions answering the *same* question q_j (but with different constructed memory banks).

$$A^{i,j}_{qa} = \frac{R^{i,j}_{qa} - \bar{R}_{qa,j}}{\sigma(R^{\cdot,j}_{qa}) + \epsilon_{std}}, \quad \bar{R}_{qa,j} = \frac{1}{G}\sum_{u=1}^{G} R^{u,j}_{qa} \tag{6}$$

Here, $\epsilon_{std} = 10^{-6}$ prevents division-by-zero errors. Normalizing within each question group isolates the reasoning capability from the question difficulty, ensuring fair credit assignment.

Finally, the policy π_θ is updated by maximizing a unified GRPO objective over all sampled sessions:

$$\mathcal{J}(\theta) = \frac{1}{GN + GM}\big(\mathcal{S}_{mem} + \mathcal{S}_{qa}\big) \tag{7}$$

$$\mathcal{S}_{mem} = \sum_{i=1}^{G}\sum_{k=1}^{N} \frac{1}{|s_{mem}^{i,k}|} \sum_{t \in s_{mem}^{i,k}} \ell(\rho_t, A_{mem}^{i,k}),$$
$$\mathcal{S}_{qa} = \sum_{i=1}^{G}\sum_{j=1}^{M} \frac{1}{|s_{qa}^{i,j}|} \sum_{t \in s_{qa}^{i,j}} \ell(\rho_t, A_{qa}^{i,j}) \tag{8}$$

$$\ell(\rho_t, A) = \min\big(\rho_t A,\, \text{clip}(\rho_t, 1-\epsilon, 1+\epsilon)A\big) - \beta D_{KL,t} \tag{9}$$

Here, $\rho_t = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ denotes the importance sampling ratio between the current policy π_θ and the rollout policy $\pi_{\theta_{\text{old}}}$ used for trajectory collection. In contrast, $D_{KL,t} = D_{KL}\big(\pi_\theta(\cdot \mid s_t) \,\|\, \pi_{\theta_{\text{ref}}}(\cdot \mid s_t)\big)$ penalizes divergence from the frozen reference policy $\pi_{\theta_{\text{ref}}}$.

3 The Ledger-QA Benchmark

Existing long-context benchmarks (e.g., LongMemEval) primarily focus on *retrieval-centric* tasks, where the answer is explicitly located within a specific text span. In contrast, many applications require *state tracking*: continuously updating a latent structured state from sequential, partially noisy observations. To isolate this core requirement, we introduce **Ledger-QA**, a controllable and extensible synthetic benchmark for long-horizon state updates. Rather than claiming full realism, Ledger-QA is intentionally built with simple, transparent generation rules that capture key aspects of state tracking while remaining challenging for modern long-context agents.

3.1 Dataset Construction

The construction pipeline contains three simple programmatic stages:

Long-Horizon Timeline Simulation. We sample transaction events over a continuous timeline (e.g., one year) and organize each example as a chronological stream of H sessions. Sessions are date-stamped to induce temporal dependencies across updates.

Natural Dialogue Synthesis. For each session, we use a strong LLM (e.g., Gemini 3 Pro (Pichai et al., 2025)) to verbalize sampled events into natural user-assistant dialogue. The surface form is linguistically diverse (e.g., chit-chat, clarifications, informal phrasing), while the underlying state-transition rules remain deliberately simple. This setup preserves natural language difficulty while keeping benchmark semantics explicit.

Ground Truth & Query Generation. In parallel, we maintain a structured list of `Transaction` objects as exact supervision. We then generate queries that require multi-step aggregation over accumulated updates (instead of local span matching), covering 8 question categories.

A concrete construction example and full generation details are provided in Appendix C.

3.2 Why Ledger-QA is Challenging

Ledger-QA fundamentally differs from standard RAG benchmarks by demanding **long-horizon state aggregation** rather than local information retrieval. Answers (e.g., "Total spending") are latent values derived from summing scattered transactions ($v_1 + \cdots + v_T$) across the timeline. This requires continuous belief-state maintenance: local extraction errors can propagate to final answers, and simple retrieval or approximate summarization is often insufficient. Ledger-QA should therefore be viewed as a controlled diagnostic benchmark rather than a complete proxy for real-world memory use: its simple, transparent rules isolate the prerequisite of consolidating sequential updates into a persistent state. Strong performance does not by itself guarantee robustness in noisier real-world scenarios, but poor performance reveals a weakness in this foundational capability.

4 Experiment

4.1 Experimental Setup

4.1.1 Evaluation Benchmarks

We evaluate UMA along three primary axes of memory behavior: **Dynamic State Tracking** (our Ledger-QA in Section 3), **Test-Time Learning (TTL)** (*TREC-Coarse*, *TREC-Fine*, *NLU*, *Clinic*, *Banking77*, *PubMed-RCT*), and **Accurate Retrieval (AR)** (*HotpotQA*, *LoCoMo*, *LongMemEval*, *MSC*, *PerLTQA*, *SQuAD*, *ConvoMem*) (Hu et al., 2025a; Yang et al., 2018; Maharana et al., 2024; Wu et al., 2025; Xu et al., 2022; Du et al., 2024; Rajpurkar et al., 2016; Pakhomov et al., 2025).

Task focus. Ledger-QA tests persistent state maintenance under continual updates; TTL tests storage and application of task rules from in-context supervision; AR evaluates fine-grained factual recovery from long documents or conversations. Together, they cover distinct memory capabilities rather than repeatedly testing a single aspect.

Evaluation metric. We use an LLM-as-a-Judge protocol with `DeepSeek-V3.2`. Prompt templates, scoring criteria, and dataset taxonomies are provided in Appendix D and Appendix E.

4.1.2 Baselines

We compare UMA against a diverse set of baselines. First, we evaluate standard approaches: **Concat**, which inputs the full context to test the backbone's native capacity, and **RAG**, which retrieves the top-20 chunks using a hybrid strategy combining dense embeddings and sparse BM25 via Reciprocal Rank Fusion (RRF; Cormack et al., 2009). Second, we include recurrent summarization models: **MemAgent** (Yu et al., 2025) iteratively updates a running memory state based on the query, while **MemAgent-woq** performs query-agnostic compression. Finally, we compare against advanced memory frameworks: **Mem1** (Zhou et al., 2025) interleaves chunk reading with dynamic retrieval to maintain an evolving internal state, **A-MEM** (Xu et al., 2025) is an additional external-memory baseline with explicit memory operations, **Mem-α** (Wang et al., 2025c) utilizes an RL-optimized policy to manage a composite memory system (core, episodic, and semantic), and **Mem-T** (Yue et al., 2026) is a hierarchical memory agent trained in two stages: optimizing retrieval via tree-guided RL and then refining its construction policy through hindsight-based distillation. Implementation details are provided in Appendix F.

4.1.3 Implementation Details

Training uses a mixed corpus combining multi-hop retrieval data (HotpotQA-derived), general long-context data (adapted from Mem-α), and synthetic Ledger-QA state-tracking data; statistics are in Appendix G. For dense retrieval, we use `sentence-transformers/all-MiniLM-L6-v2` served via `infinity` (Feil, 2023). We implement UMA on customized `veRL` (Sheng et al., 2024), initialize from `Qwen/Qwen3-4B-Instruct-2507`[1], and optimize with Task-Stratified GRPO on 32 NVIDIA H200 GPUs. Hyperparameters are reported in Appendix H, and default agent prompts are provided in Appendix B.

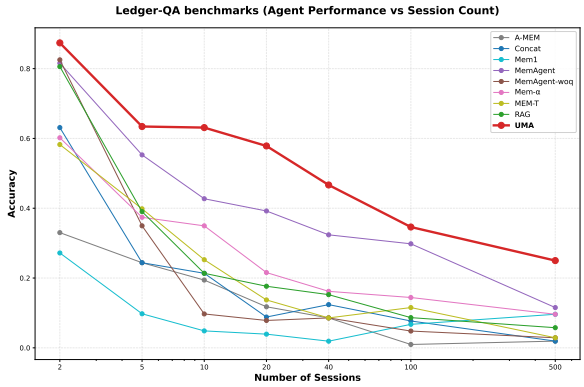

Figure 4: **Performance comparison on Ledger-QA across varying session horizons.** The x-axis represents the dialogue-session horizon H, and the y-axis denotes LLM-as-a-Judge score. For readability, we plot main baselines and UMA only; full results are in Table 13.

4.2 Main Results

Table 1 presents the comparative performance across 13 standard benchmarks (6 TTL and 7 AR tasks). We observe that:

Superiority in Test-Time Learning (TTL). Our Unified Memory Agent shows strong performance on TTL tasks (e.g., Banking77, Clinic, TREC-Fine), and remains competitive with or superior to prior memory-agent baselines. Notably, UMA performs well on Banking77 and Clinic, which are unseen during training, suggesting good generalization.

Robustness in Retrieval & Reasoning. On Accurate Retrieval (AR) tasks, our method remains competitive; efficiency and scalability analysis is provided in Appendix I.

4.3 Results on Ledger-QA

We evaluate Ledger-QA across session horizons $H \in \{2, 5, 10, 20, 30, 40, 50, 100, 500\}$. Figure 4 plots main baselines and UMA on representative horizons; Table 13 reports the full results.

Vulnerability of Baselines. At short horizons ($H \leq 5$), RAG and MemAgent-woq are competitive, suggesting retrieval or simple summarization can suffice. Performance drops as H grows: at

[1]We use the Instruct checkpoint because Qwen3-4B/8B thinking checkpoints sometimes produced overlong thinking traces, occasionally failing to stop and filling the context window, making rollout-based RL prohibitively costly.

Method	Test-Time Learning (TTL)						Accurate Retrieval (AR)							
	Bank77	**Clinic**	**NLU**	**Pub**	**T-C**	**T-F**	**Convo**	**Hotpot**	**LoCo**	**LME**	**MSC**	**PerL**	**SQuAD**	**Avg.**
Baselines														
Concat	72.00	79.00	71.00	57.80	71.00	18.00	2.46	63.28	30.01	31.60	58.40	36.00	63.38	50.30
RAG	80.00	70.00	65.00	51.20	9.00	14.00	51.12	78.12	54.03	**59.60**	54.60	**83.75**	25.35	53.52
MemAgent	25.00	13.00	22.00	49.90	61.00	37.00	59.60	51.56	**60.37**	55.40	50.20	65.25	77.02	48.25
MemAgent-woq	27.00	41.00	34.00	44.40	70.00	44.00	1.56	11.72	25.43	7.60	29.40	23.50	25.01	29.59
Mem1	0.00	6.00	0.00	9.00	7.00	0.00	1.34	5.47	21.60	6.40	8.40	7.25	15.09	6.73
A-MEM	83.00	81.00	72.00	53.70	36.00	16.00	52.46	67.97	45.87	31.80	62.80	73.00	**88.12**	58.75
Mem-α	83.00	77.00	68.00	56.50	78.00	62.00	7.14	**79.69**	44.26	51.60	56.80	74.50	81.92	63.11
Mem-T	67.00	52.00	58.00	46.80	15.00	7.00	51.12	66.41	41.04	52.40	65.80	56.75	67.03	49.72
Ablations														
UMA (w/o RL & Phase I)	26.00	27.00	23.00	39.30	13.00	12.00	79.02	64.06	50.40	38.60	36.20	57.25	82.33	42.17
UMA (w/o Phase I)	84.00	80.00	75.00	47.40	81.00	28.00	86.16	69.53	44.21	48.80	27.40	59.00	83.47	62.61
UMA (w/o RL)	77.00	72.00	61.00	38.20	69.00	20.00	84.82	64.84	44.26	48.60	63.80	51.50	81.19	59.71
UMA (w/o Raw-Context)	69.00	71.00	59.00	36.30	71.00	44.00	73.12	60.15	40.59	35.80	62.20	52.00	76.52	57.74
UMA (Global Group)	69.00	77.00	**84.00**	**72.50**	88.00	53.00	81.47	60.94	56.55	37.60	**74.40**	73.00	81.98	69.96
UMA (2 Stage)	86.00	84.00	70.00	44.60	77.00	43.00	84.82	67.19	42.75	49.20	65.60	60.75	65.59	64.65
UMA	**87.00**	**91.00**	82.00	67.20	**94.00**	**75.00**	**87.05**	76.56	46.58	51.60	66.20	74.50	87.53	**75.86**
Context-window scaling														
UMA (8k Window)	79.00	88.00	75.00	65.60	95.00	70.00	80.80	64.06	44.91	44.80	64.60	71.25	85.81	71.45
UMA (32k Window)	90.00	92.00	84.00	68.60	97.00	83.00	89.06	74.22	47.48	49.20	67.00	69.00	87.32	76.76

Table 1: Main accuracy, ablations, and context-window scaling. Bold marks the best 16k window result, and underlining compares UMA-8k/32k. **Pub**: PubMed-RCT; **Convo**: ConvoMem; **LoCo**: LoCoMo; **LME**: LongMemEval; **PerL**: PerLTQA; **T-C/T-F**: TREC-Coarse/Fine. See Section 4.4 and Appendix J for details.

$H = 500$, RAG falls to 5.77 and MemAgent to 11.54, showing that neither retrieval nor implicit compression reliably aggregates scattered updates without persistent structured storage.

Robustness of Unified Memory Agent. UMA remains more stable, especially for $H \geq 20$. At 500 sessions, it scores 25.00, matching UMA (w/o Raw-Context) and exceeding the strongest non-UMA baseline, MemAgent, at 11.54. This suggests explicit memory operations decouple state maintenance from sequence length and improve ultra-long bookkeeping.

Arithmetic Control. We further run a 100,000-example arithmetic-only control (Appendix E) where the model sums five random two-decimal numbers below 1000. Accuracy remains unchanged after UMA training (base: 25.41%; UMA: 25.14%), suggesting Ledger-QA gains come from memory maintenance and state access rather than improved standalone arithmetic. We provide a manual failure breakdown in Appendix K.

4.4 Ablation Studies

We ablate architecture and training choices in the lower block of Table 1.

Ablation Settings. **w/o Phase I** removes sequential memory maintenance and relies on QA-time retrieval only. **w/o RL** keeps the full architecture but removes RL optimization. **w/o RL & Phase I** removes both mechanisms. **w/o Raw-Context** disables `BM25/Embedding` and only allows MemoryBank retrieval at inference. **Global Group** normalizes advantages across all steps without task stratification. **2 Stage** trains memory and QA separately (no end-to-end coupling). **8k/32k Window** changes context budget from the 16k default to 8k or 32k.

Impact of Memory Maintenance (Phase I). We examine sequential memory maintenance by comparing UMA with retrieval-only variants.

- **Without RL:** Even before RL optimization, enabling explicit memory maintenance yields a clear gain over the retrieval-only variant, indicating that proactive write/update operations already improve long-horizon retention.
- **With RL:** The advantage remains after training: the full UMA still outperforms the **w/o Phase I** variant by a substantial margin (13.3 points on average), showing that retrieval alone is insufficient for sustained state tracking.

Impact of RL Training. Next, we evaluate the effectiveness of our Task-Stratified GRPO by comparing the base models with their RL-trained counterparts.

- **On Retrieval-Only Agent:** RL consistently improves the retrieval-only setting, showing that the reward design helps the policy use retrieval tools more effectively.

- **On Unified Agent:** RL also provides a large gain for the full architecture (+16.2 points on average), suggesting that training is crucial for learning stable memory lifecycles such as timely updates and accurate lookups.

Impact of Stratified Advantage Grouping. We further validate our Task-Stratified GRPO by comparing it with a **Global Group** variant that normalizes advantages across all steps indiscriminately. As shown in Table 1, the Global Group variant is worse than our stratified design (5.9 points lower on average). This is consistent with our claim that mixing heterogeneous rewards (binary QA outcomes vs. aggregated memory utility) weakens the learning signal.

Two-Stage vs. Unified End-to-End Training. We further investigate whether UMA can be learned via a decoupled two-stage pipeline that trains memory maintenance and QA separately (**UMA (2 Stage)**). This variant remains noticeably below the jointly optimized UMA (11.2 points on average), indicating a clear train–test mismatch: isolated optimization of Phase I cannot anticipate downstream QA utility, and Phase II cannot retroactively improve upstream memory decisions.

Effect of Context Window Size. We additionally evaluate UMA under different context-window budgets. Moving from 8k to 16k yields a clear improvement, while increasing further to 32k gives only modest additional gains. This indicates diminishing returns at larger windows despite higher compute and memory cost, so we use 16k as the default trade-off.

Ablation: w/o Raw-Context Retrieval. We add an ablation that disables raw-context retrieval tools (`BM25/Embedding`) and keeps only Memory-Bank access at inference. The drop is substantial on retrieval-heavy TTL/AR tasks, but minimal on Ledger-QA. This supports two points: (i) Ledger-QA mainly measures long-horizon *state tracking* rather than local span lookup; (ii) hybrid retrieval remains important for fine-grained evidence access, and key-only bank retrieval is relatively weaker for detailed factual grounding.

5 Related Work

Internal Memory and Context Compression. Several approaches extend context by modifying model internals, such as parameter-based storage (Wang et al., 2025b; Zhang et al., 2025a), KV-cache compression (Qian et al., 2025; Li et al., 2024), or soft prompts (Burtsev et al., 2020; Ge et al., 2023). These methods can be efficient, but require white-box access and face capacity bottlenecks compared to external architectures (Wang and Chen, 2025).

Prompt-based External Memory. Frameworks like MemGPT (Packer et al., 2023) and SCM (Wang et al., 2025a) manage external storage via OS-like hierarchies or graph schemas. Although compatible with black-box models, they rely on zero-shot instruction following, making smaller models unreliable for complex memory updates and retrieval in dynamic settings (Wang et al., 2025c).

Trainable Memory Agents. Recent work shifts towards *training* models for autonomous memory management. Approaches include recurrent compression via RL (Yu et al., 2025), interleaving retrieval with reasoning (Zhou et al., 2025), and utilizing composite memory structures (Yan et al., 2025; Wang et al., 2025c). Unlike our unified approach, these methods typically decouple memory maintenance from reasoning or rely on transient summaries. Our work advances this domain by unifying memory operations and reasoning into a single policy, enabling more effective cross-session state tracking.

6 Conclusion

We propose the Unified Memory Agent, an end-to-end framework that jointly optimizes memory acquisition and utilization via reinforcement learning. Task-Stratified GRPO aligns the heterogeneous objectives of long-horizon memory maintenance and downstream question answering. To address the lack of benchmarks for dynamic state tracking, we introduce **Ledger-QA**, a dataset requiring persistent updates to an evolving ledger. Experimental results show that our agent outperforms baselines on state-tracking tasks while remaining competitive on standard retrieval benchmarks. Ablation studies indicate the importance of the learned policy and explicit memory updates.

Limitations

Several limitations remain. First, we study single-agent settings with a fixed tool interface; collaborative agents and evolving APIs are left for future work. Second, although our three benchmark groups cover distinct memory capabilities, broader validation on real-world proactive state-maintenance tasks is still needed. Finally, our experiments validate session-level credit assignment for memory learning, but we have not yet established whether finer-grained or counterfactual credit assignment is feasible or would yield further gains; we leave this to future work.

References

Anthropic. 2026. Introducing Claude Opus 4.7. https://www.anthropic.com/news/claude-opus-4-7. Accessed: 2026-05-25.

Mikhail S Burtsev, Yuri Kuratov, Anton Peganov, and Grigory V Sapunov. 2020. Memory transformer. *arXiv preprint arXiv:2006.11527*.

Gordon V. Cormack, Charles L. A. Clarke, and Stefan Buettcher. 2009. Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 758–759.

Yiming Du, Hongru Wang, Zhengyi Zhao, Bin Liang, Baojun Wang, Wanjun Zhong, Zezhong Wang, and Kam-Fai Wong. 2024. PerLTQA: A personal long-term memory dataset for memory classification, retrieval, and synthesis in question answering. *arXiv preprint arXiv:2402.16288*.

Michael Feil. 2023. Infinity - to embeddings and beyond.

Tao Ge, Jing Hu, Lei Wang, Xun Wang, Si-Qing Chen, and Furu Wei. 2023. In-context autoencoder for context compression in a large language model. *arXiv preprint arXiv:2307.06945*.

Kelly Hong, Anton Troynikov, and Jeff Huber. 2025. Context Rot: How increasing input tokens impacts LLM performance. Technical report, Chroma.

Yuanzhe Hu, Yu Wang, and Julian McAuley. 2025a. Evaluating memory in llm agents via incremental multi-turn interactions. *arXiv preprint arXiv:2507.05257*.

Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, and 1 others. 2025b. Memory in the age of ai agents. *arXiv preprint arXiv:2512.13564*.

Koray Kavukcuoglu, Jeff Dean, Oriol Vinyals, and Noam Shazeer. 2026. Gemini 3.5: Frontier intelligence with action. https://blog.google/innovation-and-ai/models-and-research/gemini-models/gemini-3-5. Accessed: 2026-05-25.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024. Snapkv: Llm knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37:22947–22970.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, and 1 others. 2024. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*.

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

OpenAI. 2026. Introducing GPT-5.5. https://openai.com/index/introducing-gpt-5-5. Accessed: 2026-05-25.

Charles Packer, Vivian Fang, Shishir G Patil, Kevin Lin, Sarah Wooders, and Joseph E Gonzalez. 2023. Memgpt: Towards llms as operating systems. *CoRR*.

Egor Pakhomov, Erik Nijkamp, and Caiming Xiong. 2025. Convomem benchmark: Why your first 150 conversations don't need rag. *arXiv preprint arXiv:2511.10523*.

Sundar Pichai, Demis Hassabis, and Koray Kavukcuoglu. 2025. A new era of intelligence with Gemini 3. https://blog.google/products-and-platforms/products/gemini/gemini-3. Accessed: 2026-05-25.

Hongjin Qian, Zheng Liu, Peitian Zhang, Kelong Mao, Defu Lian, Zhicheng Dou, and Tiejun Huang. 2025. Memorag: Boosting long context processing with global memory-enhanced retrieval augmentation. In *Proceedings of the ACM on Web Conference 2025*, pages 2366–2377.

Qwen Team. 2026. Qwen3.7: The agent frontier. https://qwen.ai/blog?id=qwen3.7. Accessed: 2026-05-25.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392.

Weizhou Shen, Ziyi Yang, Chenliang Li, Zhiyuan Lu, Miao Peng, Huashan Sun, Yingcheng Shi, Shengyi Liao, Shaopeng Lai, Bo Zhang, and 1 others. 2025. QwenLong-L1.5: Post-training recipe for long-context reasoning and memory management. *arXiv preprint arXiv:2512.12967*.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2024. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv:2409.19256*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Bing Wang, Xinnian Liang, Jian Yang, Hui Huang, Zhenhe Wu, ShuangZhi Wu, Zejun Ma, and Zhoujun Li. 2025a. Scm: Enhancing large language model with self-controlled memory framework. In *International Conference on Database Systems for Advanced Applications*, pages 188–203. Springer.

Yu Wang and Xi Chen. 2025. Mirix: Multi-agent memory system for llm-based agents. *arXiv preprint arXiv:2507.07957*.

Yu Wang, Xinshuang Liu, Xiusi Chen, Sean O'Brien, Junda Wu, and Julian McAuley. 2025b. Self-updatable large language models by integrating context into model parameters. In *The Thirteenth International Conference on Learning Representations*.

Yu Wang, Ryuichi Takanobu, Zhiqi Liang, Yuzhen Mao, Yuanzhe Hu, Julian McAuley, and Xiaojian Wu. 2025c. Mem-{\alpha}: Learning memory construction via reinforcement learning. *arXiv preprint arXiv:2509.25911*.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. 2025. Longmemeval: Benchmarking chat assistants on long-term interactive memory. In *The Thirteenth International Conference on Learning Representations*.

Jing Xu, Arthur Szlam, and Jason Weston. 2022. Beyond goldfish memory: Long-term open-domain conversation. In *Proceedings of the 60th annual meeting of the association for computational linguistics (volume 1: long papers)*, pages 5180–5197.

Peng Xu, Wei Ping, Xianchao Wu, and 1 others. 2024. Retrieval meets long context large language models. *arXiv preprint arXiv:2310.03025*.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. 2025. A-mem: Agentic memory for llm agents. *arXiv preprint arXiv:2502.12110*.

Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Kristian Kersting, Jeff Z Pan, Hinrich Schütze, and 1 others. 2025. Memory-r1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. *arXiv preprint arXiv:2508.19828*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pages 2369–2380.

Hongli Yu, Tinghong Chen, Jiangtao Feng, Jiangjie Chen, Weinan Dai, Qiying Yu, Ya-Qin Zhang, Wei-Ying Ma, Jingjing Liu, Mingxuan Wang, and 1 others. 2025. Memagent: Reshaping long-context llm with multi-conv rl-based memory agent. *arXiv e-prints*, pages arXiv–2507.

Yanwei Yue, Boci Peng, Xuanbo Fan, Jiaxin Guo, Qiankun Li, and Yan Zhang. 2026. Mem-t: Densifying rewards for long-horizon memory agents. *arXiv preprint arXiv:2601.23014*.

Guibin Zhang, Muxin Fu, and Shuicheng Yan. 2025a. Memgen: Weaving generative latent memory for self-evolving agents. *arXiv preprint arXiv:2509.24704*.

Zeyu Zhang, Quanyu Dai, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. 2025b. A survey on the memory mechanism of large language model-based agents. *ACM Transactions on Information Systems*, 43(6):1–47.

Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Jinhua Zhao, Bryan Kian Hsiang Low, and Paul Pu Liang. 2025. Mem1: Learning to synergize memory and reasoning for efficient long-horizon agents. *arXiv preprint arXiv:2506.15841*.

A Tool Definitions

Our agent interacts with the Memory Bank and raw context through a structured API plus controller-level terminal-action conventions. Below we provide the detailed specifications for each executable tool available in the **Memory Toolset** ($\mathcal{T}_{mem}$) and the **Retrieval Toolset** ($\mathcal{T}_{qa}$), followed by the parsing rules for terminal actions. Each item also states the concise action alias used in Section 2.1.

A.1 Memory Toolset ($\mathcal{T}_{mem}$)

This toolset is used during the *Sequential Memory Maintenance* phase to manage the persistent Memory Bank ($\mathcal{B}_t$).

- `Add: memory_add(title: str, content: str)` $\rightarrow$ `str`: Creates a new entry in the memory bank. **Input:** A unique `title` and the `content`. **Output:** Returns "Success" if the title is new; returns an error message if it already exists.

- `Update: memory_update(title: str, content: str[, new_title: str])` → `str`: Modifies or renames an existing entry. **Input:** The target `title`, replacement `content`, and optional `new_title`. **Output:** Returns "Success" if the update is applied; returns an error if the title is not found.

- `Delete: memory_delete(title: str)` → `str`: Removes an entry. **Input:** The target `title`. **Output:** Returns "Success" upon deletion; returns an error if the title does not exist.

- `Retrieve: memory_key_retrieve(title: str)` → `str`: Reads the full content of a specific entry. **Input:** The target `title`. **Output:** Returns the stored `content` string; returns an error if the title does not exist.

- `List: memory_list([filter: str])` → `List[str]`: Scans the memory bank. **Input:** An optional `filter` string. **Output:** Returns matching keys (titles), enabling the agent to survey stored topics before performing operations.

A.2 Retrieval Toolset ($\mathcal{T}_{qa}$)

Used during the *Hybrid Retrieval-Augmented QA* phase to fetch information from the retrieval store. Note that `Retrieve` and `List` are also available in this phase to access the Memory Bank.

- `Embedding: memory_embedding_retrieve(query: str[, top_k: int])` → `List[str]`: Performs dense retrieval over memory documents. **Input:** A search `query` and optional `top_k` (default: 20). **Mechanism:** Retrieves the top-k documents by semantic similarity. **Output:** Ranked results in JSON format.

- `BM25: memory_bm25_retrieve(query: str[, top_k: int])` → `List[str]`: Performs sparse retrieval over memory documents. **Input:** A search `query` and optional `top_k` (default: 20). **Mechanism:** Uses the BM25 algorithm to match keywords. **Output:** Ranked results in JSON format.

A.3 Terminal Actions and Parsing Convention

`UpdateCore` and `Answer` are terminal actions in the MDP formulation, but they are not implemented as explicit executable tool calls in the tool executor. Instead, the controller interprets the model's final plain-text response according to the current phase:

- **Memory maintenance phase.** If the agent emits a structured memory-tool call, the executor runs the requested tool and returns its observation. If the agent emits no structured tool call and instead outputs plain text, the controller treats that text as an implicit `UpdateCore` action: the text is committed as the updated core memory m_t^{core}, the current chunk-level session terminates, and the environment advances from c_k to c_{k+1}.

- **QA phase.** If the agent emits a retrieval or memory-read tool call, the executor runs the tool and returns its observation. If the agent emits no structured tool call and instead outputs plain text, the controller treats that text as the implicit `Answer(text)` action and terminates the QA session.

Thus, `UpdateCore` and `Answer(text)` define phase-specific stopping semantics rather than callable API endpoints. Tool-validity rewards are computed over explicit structured tool calls, while these terminal text outputs are evaluated through the corresponding memory-update or answer-outcome signal.

B Agent Prompt Templates

We provide the complete prompt templates used by the Generalist Agent for memory maintenance and final question answering. Each prompt includes the phase-specific tool-calling system message followed by the corresponding task instruction. The task instruction uses *episodic* and *semantic* memory as instruction-level categories for deciding what information to store; in implementation, both are uniformly stored as key-value entries in the same Memory Bank, without separate memory modules. The full memory-maintenance and final-QA prompts are shown in Tables 2 and 3, respectively.

C Details of Ledger-QA Construction

The construction of the Ledger-QA benchmark employs a three-stage pipeline designed to simulate realistic, long-horizon bookkeeping scenarios. In our implementation, we adopt a **joint synthesis strategy**: we provide the LLM with high-level user intents (e.g., "wants to buy coffee"), and the LLM simultaneously generates both the natural dialogue and the corresponding structured transaction records. This design helps ensure that details such

as amounts and descriptions are contextually appropriate and naturally woven into the conversation.

C.1 Consumption Scene Taxonomy

To simulate a diverse personal finance environment, we define a hierarchical taxonomy of consumption scenes. The dataset covers 8 major categories, each with multiple sub-scenes:

- **Dining:** Fast Food, Restaurant, Coffee, Bubble Tea, BBQ, Hot Pot, Snacks, Takeout.
- **Transportation:** Subway, Bus, Taxi, Gas, Parking, Train, Flight.
- **Shopping:** Clothing, Electronics, Daily Necessities, Cosmetics, Books, Groceries, Furniture.
- **Entertainment:** Movie, KTV, Gaming, Gym, Travel, Concert, Escape Room.
- **Utilities:** Water & Electricity, Property Fee, Phone Bill, Internet, Gas Bill, Rent.
- **Medical:** Medicine, Doctor Visit, Health Checkup, Dental, Glasses.
- **Education:** Training Course, Books & Materials, Online Course, Exam Registration, Tuition.
- **Other:** Transfer, Red Envelope, Donation, Pet, Beauty & Salon.

Memory Maintenance Prompt

```
<|im_start|>system
# Tools

You may call one or more functions to assist with the user query.

You are provided with function signatures within <tools></tools> XML tags:
<tools>
{"type": "native", "function": {"name": "memory_add", "description": "Store new information in memory with a unique title for later retrieval",
↪  "parameters": {"type": "object", "properties": {"title": {"type": "string", "description": "Human-friendly title used to reference this memory
↪  entry"}, "content": {"type": "string", "description": "Content to store in memory"}}, "required": ["title", "content"]}}}
{"type": "native", "function": {"name": "memory_update", "description": "Update or rename existing memory entries by title, replacing their content
↪  completely", "parameters": {"type": "object", "properties": {"title": {"type": "string", "description": "Existing memory title to update"},
↪  "content": {"type": "string", "description": "New content to persist"}, "new_title": {"type": "string", "description": "Optional new title for
↪  the memory entry"}}, "required": ["title", "content"]}}}
{"type": "native", "function": {"name": "memory_delete", "description": "Permanently delete memory entries by their exact title", "parameters":
↪  {"type": "object", "properties": {"title": {"type": "string", "description": "Memory title to delete"}}, "required": ["title"]}}}
{"type": "native", "function": {"name": "memory_key_retrieve", "description": "Retrieve the full content of a memory entry using its exact title",
↪  "parameters": {"type": "object", "properties": {"title": {"type": "string", "description": "Memory title to retrieve"}}, "required": ["title"]}}}
{"type": "native", "function": {"name": "memory_list", "description": "List stored memory keys. HIGHLY RECOMMENDED: Use the 'filter' parameter to
↪  narrow down results.", "parameters": {"type": "object", "properties": {"filter": {"type": "string", "description": "Optional keyword to filter
↪  keys. Only keys containing this string will be returned (e.g., 'Transportation', '2024-04')."}}, "required": []}}}
</tools>

For each function call, return a json object with function name and arguments within <tool_call></tool_call> XML tags:
<tool_call>
{"name": <function-name>, "arguments": <args-json-object>}
</tool_call><|im_end|>
<|im_start|>user
You are an expert assistant for memory management.

### Step 1: External Memory Storage (Tool Calls)
- Episodic Memory: record specific events, user actions, friend actions, or assistant actions with timestamps.
- Semantic Memory: record detailed facts, definitions, relationships, labels.
- Always store these details externally via tools like `memory_add`.
- Do NOT store the Core Memory externally -- it is for internal topic reference only.

### Step 2: Core Memory Update
- Core memory should store only high-level themes and main topics of the context.
- Do NOT include detailed facts, numerical data, timestamps, or full event descriptions.
- Summarize in 1-3 short sentences what the context is about:
  - Main subject / theme
  - Overall purpose or intent
  - Any broad category or domain (e.g., technology, sports, finance, etc.)
- Keep it concise, clear, and relevant to understanding "what this context is generally about".
- If the new section changes or adds themes, update the core memory accordingly.
- Avoid redundancy; merge overlapping topics.

<core_memory>
{memory}
</core_memory>

<section>
{chunk}
</section>

Important:
- Final answer must be the updated concise core memory only (plain text, no tool syntax).
- Max length: 1-3 sentences.
- Focus only on WHAT the context is generally about, not details.

Updated concise core memory:
<|im_end|>
```

Table 2: Complete prompt template for UMA memory maintenance. The table spans both columns and includes both the tool-calling system message and task instruction.

C.2 Data Generation Pipeline

Stage 1: Timeline & Scenario Sampling. For each sample, we first generate a chronological timeline spanning a full year. We randomly sample H dates (sessions) from this range. For each session, we stochastically select a subset of consumption scenes (e.g., *Dining-Coffee*, *Transportation-Taxi*) to form the user's "intent" for that day.

Stage 2: Joint Dialogue & Transaction Synthesis. We feed the selected date and scene intents into an advanced LLM (e.g., Gemini 3 Pro (Pichai et al., 2025)) to obtain natural language realizations while keeping generation controls explicit. The model is instructed to act as a user-assistant pair and generate a JSON object containing two aligned fields:

1. **Dialogue:** The natural conversation turns.
2. **Transactions:** The structured ground truth (Amount, Description, etc.) corresponding to the dialogue.

Crucially, we do not pre-determine the amounts. The LLM decides reasonable amounts based on the context (e.g., $5 for coffee vs. $500 for a flight), ensuring semantic consistency. To validate the dialogue–transaction alignment, we manually audited 100 transactions randomly sampled from the Stage-2 outputs. All 100 sampled transactions were accurately supported by the corresponding dialogue, and their scenes and amounts were judged contextually reasonable, indicating that the LLM-generated dialogues and structured transactions are reliable for this synthesis step.

Example Multi-Session Sample from Stage 2 (Excerpt). Table 5 shows a shortened multi-session sample generated by the Stage-2 joint synthesis pipeline. We include two sessions from different dates, aligned transaction records with explicit date fields, and key metadata, to illustrate the temporal structure used by Ledger-QA.

Final QA Prompt

```
<|im_start|>system
# Tools

You may call one or more functions to assist with the user query.

You are provided with function signatures within <tools></tools> XML tags:
<tools>
{"type": "native", "function": {"name": "memory_key_retrieve", "description": "Retrieve the full content of a memory entry using its exact title",
↪  "parameters": {"type": "object", "properties": {"title": {"type": "string", "description": "Memory title to retrieve"}}, "required": ["title"]}}}
{"type": "native", "function": {"name": "memory_list", "description": "List stored memory keys. HIGHLY RECOMMENDED: Use the 'filter' parameter to
↪  narrow down results.", "parameters": {"type": "object", "properties": {"filter": {"type": "string", "description": "Optional keyword to filter
↪  keys. Only keys containing this string will be returned (e.g., 'Transportation', '2024-04')."}}, "required": []}}}
{"type": "native", "function": {"name": "memory_bm25_retrieve", "description": "Search memory using keyword matching (BM25). Best for finding
↪  documents with specific terms or exact phrases. Returns JSON with ranked results.", "parameters": {"type": "object", "properties": {"query":
↪  {"type": "string", "description": "Natural language query used to retrieve relevant memory chunks."}, "top_k": {"type": "integer", "description":
↪  "Maximum number of retrieved documents to return. Defaults to 20 if not specified."}}, "required": ["query"]}}}
{"type": "native", "function": {"name": "memory_embedding_retrieve", "description": "Search memory using semantic similarity (embeddings). Best for
↪  finding conceptually related content when exact keywords may vary. Returns JSON with ranked results.", "parameters": {"type": "object",
↪  "properties": {"query": {"type": "string", "description": "Natural language query used to retrieve relevant memory documents."}, "top_k": {"type":
↪  "integer", "description": "Maximum number of documents to return. Defaults to 20 if not specified."}}, "required": ["query"]}}}
</tools>

For each function call, return a json object with function name and arguments within <tool_call></tool_call> XML tags:
<tool_call>
{"name": <function-name>, "arguments": <args-json-object>}
</tool_call><|im_end|>
<|im_start|>user
You are presented with a problem and a previous memory.

You have access to tools that can help you solve the problem. Use them if necessary. In particular, if you can't find the answer in the memory, you
↪  should use the memory_list, memory_key_retrieve, memory_embedding_retrieve and memory_bm25_retrieve tools to find relevant information. If you
↪  can not find the answer using one of these tools, you can also use other tools to help you. Try as hard as you can to find the answer using the
↪  tools available to you until you find the answer or you are sure that the answer is not in the memory. Do not decide that the answer is not in the
↪  memory unless you have used all relevant tools to search for the answer. Before deciding that the information is unavailable, make sure you have
↪  used all relevant memory and retrieval tools.


Please answer the problem based on the previous memory and the tools. Put your final answer in \boxed{{}}.

<problem>
{promptext}
</problem>

<memory>
{memory}
</memory>

Your answer:
<|im_end|>
```

Table 3: Complete prompt template for UMA final question answering. The table spans both columns and includes both the tool-calling system message and task instruction.

Prompt for Joint Synthesis (Simplified)
Please generate a natural conversation between a user and an AI assistant about expense tracking. **Context:** - Date: {date} - The user wants to record: {selected_scenes} - *[Instruction: First Session vs. Continuation]* **Requirements:** - The dialogue should be natural, containing {min_turns}-{max_turns} turns. - Each consumption scene needs a specific amount (**you decide a reasonable amount**). - Return JSON with two fields: 1. "dialogue": List of turns. 2. "transactions": List of records (scene, amount, description, date).

Table 4: Prompt template used for the joint generation of dialogue and structured data.

Stage 3: Question Generation. We extract the ground-truth transaction ledger generated in Stage 2 and programmatically generate questions across 8 distinct templates.

The question types include:

1. **Time-Range Scene Amount:** "How much was spent on Dining from Jan to Mar?"
2. **Time-Range Multi-Scene:** "Total spending on Dining and Travel in Q1?"
3. **Global Total:** "What was the total spending across all records?"
4. **Max Scene:** "Which category had the highest spending?"
5. **Max Frequency Date:** "On which date were there the most transactions?"
6. **Max Single Amount:** "What was the largest single transaction?"
7. **Point Query:** "How much was spent on Coffee on 2024-05-01?"
8. **Single Date Scene Amount:** "How much was spent on Dining on 2024-05-01?"

C.3 Dataset Statistics

To evaluate the agent's robustness across varying temporal horizons, we constructed a comprehensive suite of datasets ranging from short-term interactions (2 sessions) to ultra-long histories (500 sessions).

Session Dates: 2024-02-23 and 2024-04-29
Dialogue Excerpts (cross-date) `[2024-02-23][User]` Hi there! I'd like to start using this chat to track my daily expenses. `[2024-02-23][Assistant]` Great initiative. I can help log each expense with category and amount. `[2024-02-23][User]` I paid my monthly Internet bill. It cost 75. `[2024-02-23][Assistant]` Got it. Should I categorize this under Utilities? `[2024-02-23][User]` Yes, please put it under Utilities. `[2024-02-23][Assistant]` Recorded: Utilities–Internet, 75. Do you have other expenses? . . . `[2024-04-29][User]` I have a few more receipts. First, I took a cab to the city center. `[2024-04-29][Assistant]` Transportation–Taxi noted. How much was the fare? `[2024-04-29][User]` 25.5. Also, a clinic co-pay (50), a gas bill (85.20), and skincare products (120). `[2024-04-29][Assistant]` Done. I've saved all four transactions for April 29th. . . .
Aligned Structured Transactions (date, scene, subscene, amount) 1) (2024-02-23, Utilities, Internet, 75.00) 2) (2024-02-23, Transportation, Subway, 2.90) 3) (2024-02-23, Entertainment, Escape Room, 40.00) 4) (2024-02-23, Education, Tuition, 1500.00) 5) (2024-02-23, Transportation, Bus, 1.50) 6) (2024-04-29, Transportation, Taxi, 25.50) 7) (2024-04-29, Medical, Doctor Visit, 50.00) 8) (2024-04-29, Utilities, Gas Bill, 85.20) 9) (2024-04-29, Shopping, Cosmetics, 120.00)
Sample Metadata num_sessions=2; num_turns=48; num_transactions=9; date_range=[2024-02-23, 2024-04-29]; total_amount=1900.10.

Table 5: Shortened multi-session Stage-2 generation example. The excerpt preserves cross-date dialogue evidence, date-aware transaction records, and core metadata.

Training Set. We utilize the $H = 10$ configuration for training, consisting of 200 samples with over 6,000 questions. This provides a balanced difficulty level for the agent to learn state-tracking policies.

Test Set. The test set is stratified by session horizon ($H \in \{2, 5, 10, 20, 30, 40, 50, 100, 500\}$). Detailed statistics for each configuration are presented in Table 6.

C.4 Generation Algorithm

The core logic of our joint synthesis pipeline is formalized in Algorithm 1.

D Evaluation Prompts

To ensure consistent and fair evaluation, we employ an LLM-as-a-Judge protocol with two independent evaluators: `DeepSeek-V3.2` (main results) and `Qwen3-30B-A3B-Instruct-2507` (supplementary comparison). The specific prompts used for different benchmarks are detailed below.

Algorithm 1 Ledger-QA Data Generation Pipeline

Require: Date range $[D_{start}, D_{end}]$, Session horizon H, Diversify flag $diversify$
Ensure: Context chunks $\mathcal{C}$, Question set $\mathcal{Q}$, Ground-truth Ledger $\mathcal{L}$

```
D ← SampleSortedDates(D_start, D_end, H)
L ← ∅                                          ▷ Initialize global transaction ledger
C ← ∅                                          ▷ Initialize dialogue history
for i ← 1 to H do
    d_i ← D[i]
    S_intent ← SampleScenes(Taxonomy)          ▷ Randomly select user intents
    P ← FormatPrompt(d_i, S_intent, is_first = (i == 1))   ▷ Construct prompt for joint synthesis
    Response ← LLM_gen(P)                      ▷ Generate dialogue and transaction simultaneously
    Dialogue_i, Trans_i ← ParseJSON(Response)
    C.append(FormatChunk(d_i, Dialogue_i))
    L.extend(Trans_i)
end for
Q_raw ← ApplyTemplates(L, Types = {1, ..., 8})   ▷ Generate questions based on the full ledger
if diversify then                              ▷ Optional: Diversify phrasing
    Q ← LLM_paraphrase(Q_raw)
else
    Q ← Q_raw
end if
return C, Q, L
```

General Prompt. For all benchmarks except LongMemEval, we utilize a unified evaluation template designed to assess semantic correctness:

> "I will give you a question, a correct answer, and a response from a model. Please answer yes if the response contains the correct answer. Otherwise, answer no. If the response is equivalent to the correct answer or contains all the intermediate steps to get the correct answer, you should also answer yes. If the response only contains a subset of the information required by the answer, answer no.
>
> **Question:** {question}
>
> **Correct Answer:** {answer}
>
> **Model Response:** {prediction}
>
> Is the model response correct? Answer yes or no only."

LongMemEval Prompts. Following the official implementation (Wu et al., 2025), we apply task-specific prompts to handle the nuances of different reasoning types (e.g., temporal reasoning, knowledge updates).

- **Default / Fact Retrieval:** Same as the General Prompt above.
- **Temporal Reasoning:** Adds tolerance for off-by-one errors:

  > "...In addition, do not penalize off-by-one errors for the number of days. If the question asks for the number of days/weeks/months, etc., and the model makes off-by-one errors (e.g., predicting 19 days when the answer is 18), the model's response is still correct..."

- **Knowledge Update:** Allows inclusion of previous information if the update is correct:

  > "...If the response contains some previous information along with an updated answer, the response should be considered as correct as long as the updated answer is the required answer..."

- **Single-Session Preference:** Evaluates personalization based on a rubric:

  > "...Please answer yes if the response satisfies the desired response... The response is correct as long as it recalls and utilizes the user's personal information correctly.
  > **Rubric:** {rubric}..."

- **Abstention:** Checks for correct identification of unanswerable questions:

  > "...Please answer yes if the model correctly identifies the question as unanswerable. The model could say that the information is incomplete...
  > **Explanation:** {explanation}
  > Does the model correctly identify the question as unanswerable? Answer yes or no only."

E Evaluation Dataset Details

We employ a diverse set of benchmarks to evaluate the agent's capabilities. Detailed statistics for

Category	Dataset	Samples	Avg. Qs	Avg. Len (Tok)	Avg. Chunks
State Tracking	Ledger-QA ($H = 2$)	5	20.6	1,693	2.0
	Ledger-QA ($H = 5$)	4	30.8	4,204	5.0
	Ledger-QA ($H = 10$)	3	34.3	8,192	10.0
	Ledger-QA ($H = 20$)	3	34.0	16,487	20.0
	Ledger-QA ($H = 30$)	3	34.3	24,772	30.0
	Ledger-QA ($H = 40$)	3	35.0	32,792	40.0
	Ledger-QA ($H = 50$)	3	35.0	40,610	50.0
	Ledger-QA ($H = 100$)	3	34.7	81,587	100.0
	Ledger-QA ($H = 500$)	3	34.7	405,990	500.0
Test Time Learning	Banking77	1	100.0	127,568	111.0
	Clinic150	1	100.0	130,702	38.0
	NLU	1	100.0	134,060	115.0
	PubMed-RCT	10	100.0	16,724	10.0
	TREC-Coarse	1	100.0	123,606	111.0
	TREC-Fine	1	100.0	125,531	108.0
Accurate Retrieval	ConvoMem	7	64.0	**522,602**	300.0
	HotpotQA	128	1.0	27,945	200.0
	LoCoMo	10	198.6	22,161	27.2
	LongMemEval	500	1.0	107,863	47.8
	MSC (Batch=5)	100	5.0	9,853	25.0
	PerLTQA	4	100.0	13,038	23.0
	SQuAD	30	96.8	10,561	10.0

Table 6: Detailed statistics of all evaluation datasets. **Avg. Len (Tok)** denotes the average total context length in tokens. **Avg. Chunks** denotes the average number of context chunks per sample processed by the agent.

all evaluation datasets are summarized in Table 6. Below we detail the preprocessing and construction logic for each dataset, corresponding to the implementation in our codebase.

Artifact Licenses and Intended Use. We use existing datasets, model checkpoints, and code artifacts in accordance with their released licenses, terms of use, and accompanying documentation. All third-party artifacts and derived/processed versions are used only for research purposes consistent with their intended use. The synthetic artifacts we create, including Ledger-QA and the arithmetic-control set, are intended for research on long-context memory and state tracking, and should be used in research contexts under the access conditions of the paper release.

E.1 Dynamic State Tracking

Ledger-QA (Synthetic). As detailed in Appendix C, we generate synthetic bookkeeping data. For evaluation, we use a stratified test set across session horizons $H \in \{2, 5, 10, 20, 30, 40, 50, 100, 500\}$, as summarized in Table 6.

Arithmetic-Control Set. We construct 100,000 arithmetic-only examples to isolate numerical computation: each asks for the sum of five randomly sampled numbers below 1000 rounded to two decimals. Exact-match accuracy is computed against the generated sum. The set contains no long-context stream, memory updates, or retrieval requirement.

E.2 Test-Time Learning (TTL)

For TTL tasks, we source data from **MemoryAgentBench** (Hu et al., 2025a). To ensure the agent understands the classification nature of these tasks, we prepend a specific instruction hint to each query if not already present:

> *"Sentence: {original_query}*
> *What are the labels for the above sentence?"*

E.3 Accurate Retrieval (AR)

HotpotQA. Following MemAgent (Yu et al., 2025), we ensure strict data isolation by using non-overlapping training and validation splits. We inject ground-truth documents into 200 shuffled distractors, requiring the agent to retrieve evidence from the chunked stream to solve multi-hop queries without any risk of data leakage.

LoCoMo. We adapt prompts for specific question categories to ensure valid evaluation:

- **Temporal (Cat 2):** We append *"Use DATE of CONVERSATION to answer with an approximate date"* to guide relative time reasoning.
- **Adversarial (Cat 5):** We convert these into a multiple-choice format *"(A) {distractor} (B) Not mentioned"* (with randomized order). The agent must select *"Not mentioned"* to demonstrate correct rejection of unanswerable queries.

LongMemEval. We utilize the `longmemeval_s` subset. Questions are timestamped (e.g., `[2024-01-01] Question. . .`) to provide temporal context.

MSC (Multi-Session Chat). We use a modified version of MSC introduced by MemGPT (Packer et al., 2023) provided at https://huggingface.co/datasets/MemGPT/MSC-Self-Instruct. To increase context length, we batch 5 base samples into a single evaluation instance (`batch_size=5`). Each instance thus contains 25 sessions (5 samples $\times$ 5 sessions), and the agent is queried about specific details from any of the batched conversations.

ConvoMem. We curate evaluation samples from the `pre_mixed_testcases` split provided at https://huggingface.co/datasets/Salesforce/ConvoMem/tree/main, retaining only conversations whose total length exceeds 2 million characters to ensure they represent genuine long-context tasks. Each conversation turn is formatted as `"{Speaker}: {Text}"`. We use the `user_evidence` category for evaluation.

F Baseline Implementation Details

We implement the baseline methods **Mem1** and **Mem-α** using the codebase provided at https://github.com/wangyu-ustc/Mem-alpha. Since **MemAgent** follows a simple recurrent summarization procedure, we implement it ourselves while using the official prompt templates. We implement **Mem-T** using the official open-source model `EdwinYue/Mem-T-4B` and the core implementation from https://github.com/yanweiyue/Mem-T/tree/main, with only interface-level adaptations to integrate all baselines into our chunked-stream evaluation pipeline and unified evaluator. We keep released memory-formation checkpoints, prompts, and default inference hyperparameters whenever available, and do not tune baseline hyperparameters on the test sets. For all experiments, we enforce a strict context window limit of 16k tokens. When the total input length (including memory/context, query, and instructions) exceeds this limit, we apply left-truncation to the memory or context sequence, retaining only the most recent tokens to fit within the 16k window.

For a fair comparison, all embedding-based retrieval components use the same dense encoder, `sentence-transformers/all-MiniLM-L6-v2`. This applies to RAG and to memory-agent baselines with embedding retrieval components, including Mem-α and Mem-T.

Backbone Models. To isolate the contribution of the memory mechanism, we standardize the backbone model where possible:

- **Standard Backbone:** Unless otherwise specified (e.g., for Concat, RAG, MemAgent), we utilize `Qwen/Qwen3-4B-Instruct-2507` for both the context processing and final QA phases.
- **Specialized Memory Models:** For baselines that require specific pre-trained weights for memory operations, we follow their original configurations:
  - **Mem1:** Uses `Mem-Lab/Qwen2.5-7B-RL-RAG-Q2-EM-Release` for the memory formation phase.
  - **Mem-α:** Uses `YuWangX/Memalpha-4B` for the memory formation phase.
  - **Mem-T:** Uses the official `EdwinYue/Mem-T-4B` checkpoint. Since Mem-T designs speaker-aware memories for dialogue, for benchmarks that are not dialogue-formatted, we treat each complete chunk as a single turn and set the speaker to `context`.

Crucially, for the final Question Answering phase, both Mem1 and Mem-α use the standard `Qwen/Qwen3-4B-Instruct-2507` backbone. The original Mem-α repository uses a larger Qwen3-32B QA model; for fair comparison under our evaluation setting, we replace only this QA model and keep the released Mem-α memory-formation model unchanged.

G Training Data Details

Table 7 summarizes the detailed statistics of the datasets utilized in our training phase.

Dataset	Samples	Avg. Qs	Avg. Len (Tok)	Avg. Chunks
HotpotQA	8,192	1.0	25,667	61
SQuAD	264	10.0	10,780	10.0
PerLTQA	27	10.0	12,046	23.3
NLU	180	10.0	6,100	10.0
TREC-Coarse	180	10.0	3,900	10.0
PubMed-RCT	90	10.0	16,760	10.0
BookSum	1,387	1.0	15,328	8.0
Ledger-QA	200	34.0	8,254	10.0

Table 7: Detailed statistics of the training datasets. **Avg. Len (Tok)** denotes the average total context length in tokens. **Avg. Chunks** denotes the average number of context chunks per sample processed by the agent.

Data Sources. Our training data is curated from three main sources: (1) **HotpotQA (Modified):** We construct a retrieval-intensive version of HotpotQA following the protocol in MemAgent (Yu et al., 2025), augmenting each sample with 200 documents. (2) **Ledger-QA (Synthetic):** We generate a specialized dataset for dynamic state tracking as described in Section 3. (3) **Mem-α Corpus:** For all other datasets (SQuAD, PerLTQA, NLU, TREC, PubMed, BookSum), we directly incorporate the high-quality training corpus provided by Mem-α (Wang et al., 2025c) to ensure broad coverage of reasoning tasks (provided at `https://huggingface.co/datasets/YuWangX/Memalpha-full`).

Data Isolation. We strictly enforce data isolation to prevent leakage. For **HotpotQA**, we follow the MemAgent protocol, constructing our training and evaluation sets from the original HotpotQA training and validation splits, respectively. For **Ledger-QA**, the training and test samples are synthesized independently with distinct seed parameters. For the **Mem-α Corpus**, we start from the official training split provided by the authors and remove the LongMemEval portion to avoid evaluation-data leakage. We also remove the HotpotQA portion, both to avoid leakage and because our training data already includes the MemAgent-style modified HotpotQA data described above.

To maintain training efficiency, for Mem-α corpus containing a large number of queries per instance (e.g., SQuAD, NLU), we randomly sample up to 10 queries per training sample.

H Training Configuration

We implement our training framework based on a customized version of `veRL` (Sheng et al., 2024), optimized for multi-conversation and multi-turn agent rollouts. The policy model is initialized from `Qwen/Qwen3-4B-Instruct-2507` and optimized using AdamW with a learning rate of 1×10^{-6}. We employ the Task-Stratified Group Relative Policy Optimization (GRPO) algorithm with a group size of $G = 16$ samples per context and a KL divergence penalty coefficient of $\beta = 0.001$.

To balance generalization across diverse tasks with the specific demands of intensive state tracking, we adopt a dual-track training strategy:

- **Generalist Agent (for TTL & AR):** To ensure robust performance across broad retrieval and reasoning tasks, this model is trained on the combined HotpotQA data and Mem-α Corpus for 2 epochs.
- **Specialist Agent (for Ledger-QA):** Given the unique requirement for continuous numerical updates and long-horizon consistency in Ledger-QA, we train a separate model specifically on the Ledger-QA training set. This model is trained for 6 epochs to fully adapt to the state-tracking dynamics, allowing us to probe the architectural upper bound for this specialized domain.

All experiments are conducted on a cluster of 32 NVIDIA H200 GPUs (4 nodes $\times$ 8 GPUs). To manage memory efficiency during long-horizon rollouts, we utilize vLLM for inference with a tensor parallelism size of 1 and set the GPU memory utilization to 0.5. The maximum sequence length for both prompts and responses is set to 8,192 tokens. The total training time spans approximately 2 days for the Generalist Agent and 0.5 days for the Specialist Agent.

I Inference Efficiency and Resource Scalability

We benchmark inference latency on a single ConvoMem sample (528k-token context with 10

Method	Latency (s)	Relative Speed
UMA	**107.32**	**1.0x**
Mem1	368.46	3.4x Slower
A-MEM	3251.75	30.3x Slower
Mem-α	1503.87	14.0x Slower
MemAgent-woq	2718.88	25.3x Slower
MemAgent	4753.26	44.3x Slower
Mem-T	8611.18	80.2x Slower
Concat	2.58	41.6x Faster
Concat-256k	61.16	1.8x Faster
RAG	58.01	1.9x Faster

Table 8: Inference latency comparison on a single ConvoMem sample (528k tokens context, 10 queries) using 8 NVIDIA A800 GPUs.

queries) using 8 NVIDIA A800 GPUs. Results are shown in Table 8. UMA reaches a practical sweet spot: it is substantially faster than recurrent baselines while preserving strong long-horizon reasoning capability.

The results indicate that UMA achieves a favorable efficiency–capability balance. Compared with MemAgent, UMA processes the full context only once during memory construction and then answers subsequent queries using the maintained memory together with lightweight retrieval. In contrast, MemAgent must traverse the entire context separately for each query, which leads to substantially higher latency when multiple queries are asked over the same long context. Mem-T is the slowest method in this comparison, which may be attributed to its more complex memory architecture, workflow, and retrieval strategy.

Resource Scalability. Beyond large-cluster training, we verify that the training pipeline is compatible with smaller setups and can run on a single node with 8 NVIDIA A800 GPUs (80GB HBM each). Moreover, parameter-efficient fine-tuning methods (e.g., LoRA) offer a practical path to further reduce memory requirements, improving accessibility for broader research groups.

J Detailed Evaluation Metrics

In addition to the accuracy metric reported in the main text, we provide the Exact Match (EM), F1, ROUGE, and supplementary LLM-as-a-Judge scores to offer a more granular view of model performance. Table 9 presents the Exact Match scores, which measure the percentage of predictions that match the ground truth verbatim. Table 10 presents the F1 scores, which account for partial overlaps between the prediction and the reference answer. Table 11 presents the ROUGE-L scores, evaluating the lexical overlap and fluency of the generated responses. For ROUGE-L, we use the `rouge_score` package's `RougeScorer` with metric `rougeL` and `use_stemmer=True`, and report the returned `fmeasure` after scoring normalized ground-truth and prediction strings. Table 12 further reports LLM-as-a-Judge summary scores using `Qwen3-30B-A3B-Instruct-2507` as the evaluator.

K Manual Error Analysis

To better characterize the remaining failure modes, we manually inspect incorrect UMA predictions together with the corresponding memory-bank snapshots, retrieval outputs, and final answers. We assign each case to the earliest failing stage: memory formation, retrieval/access, or final evidence use.

Ledger-QA. On the $H = 50$ split of Ledger-QA, which contains 105 queries, UMA achieves 45.71% accuracy and makes 57 errors. Among them, 37 cases (64.9%) are memory-formation errors, including incorrect writes, incomplete aggregation, or stale/conflicting state entries, while 20 cases (35.1%) are QA-time reasoning errors. A typical memory error is an incorrect numeric update: two same-day Shopping expenses should be aggregated as $145.50 + 25.99 = 171.49$, but the memory bank stores 166.49, causing later range queries to fail even when the relevant memory entry is accessed. This confirms that Ledger-QA primarily stresses durable state maintenance rather than local span retrieval.

LoCoMo. We further sample 50 incorrect LoCoMo cases to analyze failures in a dialogue-centric retrieval setting. We find 12 missing-memory cases (24%), where the required fact is

Method	Test-Time Learning (TTL)						Accurate Retrieval (AR)							
	Bank77	**Clinic**	**NLU**	**Pub**	**T-C**	**T-F**	**Convo**	**Hotpot**	**LoCo**	**LME**	**MSC**	**PerL**	**SQuAD**	**Avg.**
Baselines														
Concat	66.00	79.00	71.00	57.80	70.00	18.00	0.45	30.47	13.95	18.20	27.00	5.50	40.68	38.31
RAG ($k = 20$)	4.00	12.00	2.00	17.50	0.00	0.00	3.79	0.78	0.20	0.40	1.40	1.00	0.34	3.34
MemAgent	24.00	13.00	23.00	11.70	61.00	37.00	2.90	14.06	2.92	8.80	5.20	1.00	3.31	15.99
MemAgent-woq	27.00	41.00	34.00	0.00	69.00	42.00	0.00	0.00	0.05	0.00	0.40	0.00	0.17	16.43
Mem1	0.00	4.00	0.00	0.00	7.00	0.00	0.00	0.00	0.05	0.00	0.00	0.00	0.00	0.85
A-MEM	71.00	77.00	71.00	53.50	32.00	13.00	13.39	44.53	9.42	15.60	24.00	9.50	34.62	36.04
Mem-α	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Mem-T	58.00	35.00	46.00	21.10	0.00	3.00	15.40	41.41	15.61	17.80	**33.00**	11.50	39.79	25.97
Ablations														
UMA (w/o RL & Phase I)	26.00	27.00	22.00	39.00	12.00	7.00	18.08	34.38	16.52	20.20	15.60	10.50	46.68	22.69
UMA (w/o Phase I)	84.00	79.00	75.00	47.30	81.00	25.00	19.42	43.75	15.51	25.00	12.60	11.25	48.40	43.63
UMA (w/o RL)	76.00	72.00	61.00	37.30	69.00	20.00	19.42	41.41	16.72	25.00	23.80	9.00	44.47	39.62
UMA (w/o Raw-Context)	69.00	71.00	59.00	36.30	71.00	44.00	18.74	40.46	15.38	26.46	21.20	8.25	**59.52**	41.56
UMA (Global Group)	69.00	77.00	**82.00**	**69.10**	87.00	47.00	19.20	40.62	**19.54**	14.80	28.40	6.50	30.73	45.45
UMA (2 Stage)	84.00	84.00	69.00	44.30	77.00	43.00	19.20	43.75	13.75	24.40	28.00	11.00	37.62	44.54
UMA	**87.00**	**91.00**	**82.00**	67.20	**95.00**	**75.00**	**20.98**	**48.44**	17.62	**30.60**	29.20	**12.25**	56.32	**54.82**
Context-window scaling														
UMA (8k Window)	79.00	88.00	75.00	65.70	95.00	70.00	19.87	42.97	16.16	27.80	29.40	13.25	55.60	52.13
UMA (32k Window)	90.00	92.00	84.00	68.60	97.00	83.00	21.65	47.66	18.23	31.20	31.20	12.25	56.46	56.40

Table 9: Exact Match (EM) scores across all benchmarks. Unless otherwise specified, all methods use a 16k context budget. UMA-8k and UMA-32k are reported separately as context-window scaling variants. Bold numbers in the default-budget block indicate the best result among methods using the 16k budget; underlined numbers in the scaling block indicate the better value between UMA-8k and UMA-32k.

never written or is overwritten by an incomplete memory; 19 retrieval/access failures (38%), where the relevant memory exists but is not selected; and 19 final evidence-use errors (38%), where correct evidence is retrieved but ignored or polluted by noisy raw-context retrieval. These results suggest complementary bottlenecks: Ledger-QA is dominated by memory-write accuracy, whereas LoCoMo additionally requires stronger memory addressing and final evidence selection.

L Numerical Results on Ledger-QA

Table 13 reports full Ledger-QA LLM-as-a-Judge scores across all session horizons, ablations, and context-window variants. Table 14 further reports the corresponding Sub-EM scores. Sub-EM counts a prediction as correct if the model output and ground-truth answer satisfy a containment match, i.e., either one contains the other.

Method	Test-Time Learning (TTL)						Accurate Retrieval (AR)							
	Bank77	**Clinic**	**NLU**	**Pub**	**T-C**	**T-F**	**Convo**	**Hotpot**	**LoCo**	**LME**	**MSC**	**PerL**	**SQuAD**	**Avg.**
Baselines														
Concat	70.00	79.00	71.00	57.80	70.00	18.00	8.65	49.27	25.89	27.84	48.21	18.00	53.57	45.94
RAG ($k = 20$)	25.12	30.82	20.90	18.92	3.63	2.04	38.66	15.57	16.90	17.62	23.33	13.92	5.29	17.90
MemAgent	24.40	13.00	23.00	26.01	61.00	37.00	33.55	26.34	21.96	27.11	25.11	8.42	14.88	26.29
MemAgent-woq	27.00	41.00	34.00	0.82	69.67	42.04	12.13	2.71	2.96	5.62	8.77	2.22	3.38	19.41
Mem1	0.00	4.00	0.00	0.44	7.00	0.00	14.22	1.66	3.37	5.60	5.69	1.44	1.48	3.45
A-MEM	79.00	79.00	71.67	53.57	32.00	13.00	37.70	61.99	28.61	28.84	49.10	30.05	57.46	47.87
Mem-α	0.00	0.00	2.61	0.53	0.02	0.00	16.18	8.51	5.50	9.33	9.58	4.37	5.18	4.75
Mem-T	61.86	42.94	52.16	32.97	7.09	5.14	31.21	59.06	35.90	33.88	**56.48**	**32.95**	53.62	38.87
Ablations														
UMA (w/o RL & Phase I)	26.00	27.50	22.00	40.20	12.00	7.00	40.94	52.73	36.70	28.82	28.36	26.01	62.08	31.56
UMA (w/o Phase I)	84.67	79.25	75.00	47.35	81.00	25.00	44.21	59.53	34.57	36.66	21.08	28.22	63.18	52.28
UMA (w/o RL)	78.12	72.00	61.67	39.64	69.00	20.00	43.38	56.81	32.96	36.32	46.73	24.01	60.38	49.31
UMA (w/o Raw-Context)	69.00	71.00	59.00	36.30	71.00	44.00	43.72	56.64	30.78	33.86	40.20	29.39	59.52	49.57
UMA (Global Group)	69.01	77.00	**82.01**	**69.11**	87.01	47.03	44.01	56.67	**40.18**	27.25	52.04	30.12	50.82	56.33
UMA (2 Stage)	85.07	84.67	70.17	45.47	77.00	43.00	44.36	58.73	33.03	37.53	49.98	28.86	49.78	54.43
UMA	**87.00**	**91.00**	82.00	67.20	**95.00**	**75.00**	**46.33**	**65.42**	39.39	**41.73**	52.59	32.88	**71.37**	**65.15**
Context-window scaling														
UMA (8k Window)	79.00	88.00	75.00	65.70	95.00	70.00	44.16	56.51	37.05	37.34	51.03	32.61	70.64	61.69
UMA (32k Window)	90.00	92.00	84.00	68.60	97.00	83.00	47.34	65.36	40.08	41.11	54.24	32.49	71.54	66.67

Table 10: F1 scores across all benchmarks. Unless otherwise specified, all methods use a 16k context budget. UMA-8k and UMA-32k are reported separately as context-window scaling variants. Bold numbers in the default-budget block indicate the best result among methods using the 16k budget; underlined numbers in the scaling block indicate the better value between UMA-8k and UMA-32k.

Method	Test-Time Learning (TTL)						Accurate Retrieval (AR)							
	Bank77	**Clinic**	**NLU**	**Pub**	**T-C**	**T-F**	**Convo**	**Hotpot**	**LoCo**	**LME**	**MSC**	**PerL**	**SQuAD**	**Avg.**
Baselines														
Concat	70.00	79.00	71.00	57.80	70.00	18.00	8.63	50.44	25.97	27.76	48.79	18.36	54.00	46.13
RAG ($k = 20$)	25.13	30.82	20.90	18.92	3.63	2.04	36.55	15.47	16.55	16.30	23.17	13.92	5.40	17.60
MemAgent	24.40	13.00	23.00	26.04	61.00	37.00	32.31	25.71	21.51	26.59	25.17	8.35	15.15	26.10
MemAgent-woq	27.00	41.00	34.00	0.83	69.67	42.04	10.80	2.74	2.88	4.81	8.78	2.33	3.44	19.25
Mem1	0.00	4.00	0.00	0.45	7.00	0.00	11.63	1.77	3.28	4.70	5.36	1.54	1.54	3.17
A-MEM	79.00	79.00	71.67	53.57	32.00	13.00	37.56	61.99	28.62	28.80	50.01	30.53	57.80	47.97
Mem-α	0.00	0.00	2.61	0.54	0.02	0.00	14.33	8.49	5.26	8.43	9.71	4.29	5.22	4.53
Mem-T	61.87	42.98	52.16	32.97	7.01	5.14	31.17	59.43	35.75	33.42	**57.49**	**34.06**	54.18	39.05
Ablations														
UMA (w/o RL & Phase I)	26.00	27.50	22.00	40.20	12.00	7.00	40.86	52.78	36.34	28.76	29.18	26.24	63.03	31.68
UMA (w/o Phase I)	84.67	79.25	75.00	47.35	81.00	25.00	44.16	59.34	34.15	36.51	21.29	28.57	64.20	52.34
UMA (w/o RL)	78.12	72.00	61.67	39.64	69.00	20.00	43.28	56.79	32.87	36.17	48.05	24.14	61.38	49.47
UMA (w/o Raw-Context)	69.00	71.00	59.00	36.30	71.00	44.00	43.42	56.44	31.68	33.80	41.20	28.19	59.25	49.56
UMA (Global Group)	69.01	77.00	**82.01**	**69.11**	87.01	47.03	43.91	57.61	**39.85**	27.33	52.41	29.97	51.05	56.37
UMA (2 Stage)	85.07	84.67	70.17	45.47	77.00	43.00	44.40	59.47	32.72	37.40	50.89	29.36	50.59	54.63
UMA	**87.00**	**91.00**	82.00	67.20	**95.00**	**75.00**	**46.39**	**66.16**	39.13	**41.52**	53.28	33.14	**72.04**	**65.30**
Context-window scaling														
UMA (8k Window)	79.00	88.00	75.00	65.70	95.00	70.00	44.30	57.25	36.79	37.20	51.90	32.88	71.28	61.87
UMA (32k Window)	90.00	92.00	84.00	68.60	97.00	83.00	47.44	66.01	39.85	40.81	54.97	32.93	72.34	66.84

Table 11: ROUGE-L scores across all benchmarks. Unless otherwise specified, all methods use a 16k context budget. UMA-8k and UMA-32k are reported separately as context-window scaling variants. Bold numbers in the default-budget block indicate the best result among methods using the 16k budget; underlined numbers in the scaling block indicate the better value between UMA-8k and UMA-32k.

Method	Test-Time Learning (TTL)						Accurate Retrieval (AR)							
	Bank77	**Clinic**	**NLU**	**Pub**	**T-C**	**T-F**	**Convo**	**Hotpot**	**LoCo**	**LME**	**MSC**	**PerL**	**SQuAD**	**Avg.**
Baselines														
Concat	72.00	79.00	71.00	57.80	71.00	18.00	5.36	65.62	33.43	33.00	61.20	42.50	64.42	51.87
RAG ($k = 20$)	81.00	70.00	65.00	53.30	10.00	10.00	51.34	77.34	57.35	**61.20**	55.80	**85.25**	25.56	54.09
MemAgent	26.00	14.00	24.00	51.20	61.00	37.00	60.27	51.56	**64.15**	59.40	53.00	71.50	78.06	50.09
MemAgent-woq	27.00	41.00	34.00	47.00	70.00	44.00	2.68	14.84	29.71	9.80	29.60	26.75	25.80	30.94
Mem1	0.00	4.00	0.00	45.40	7.00	0.00	1.56	6.25	22.41	8.00	9.60	7.75	18.43	10.03
A-MEM	83.00	80.00	72.00	53.70	32.00	15.00	53.35	72.22	50.25	51.61	64.21	80.00	**89.46**	61.29
Mem-α	83.00	78.00	71.00	57.40	78.00	60.00	8.48	**78.91**	48.64	55.60	56.60	79.50	83.40	64.50
Mem-T	69.00	53.00	59.00	54.90	23.00	12.00	51.34	69.53	45.52	55.00	73.00	67.75	67.76	53.91
Ablations														
UMA (w/o RL & Phase I)	34.00	28.00	24.00	40.20	23.00	15.00	79.91	62.50	54.88	41.00	38.20	65.50	82.23	45.26
UMA (w/o Phase I)	84.00	79.00	74.00	47.70	81.00	25.00	86.61	71.09	47.43	51.40	28.80	69.75	83.71	63.81
UMA (w/o RL)	76.00	72.00	61.00	38.80	70.00	20.00	85.49	66.41	47.99	50.80	66.20	62.50	80.71	61.38
UMA (w/o Raw-Context)	71.00	73.00	63.00	40.30	73.00	45.00	71.43	62.50	44.86	37.60	63.00	51.25	71.97	59.07
UMA (Global Group)	72.00	79.00	**82.00**	**74.00**	89.00	55.00	83.71	76.56	61.98	44.00	**75.40**	83.00	80.92	73.58
UMA (2 Stage)	84.00	84.00	69.00	45.00	78.00	43.00	85.49	69.53	45.97	52.20	67.60	71.50	66.45	66.29
UMA	**87.00**	**91.00**	**82.00**	67.20	**95.00**	**75.00**	**88.17**	77.34	50.10	54.80	68.00	82.50	87.32	**77.34**
Context-window scaling														
UMA (8k Window)	79.00	88.00	75.00	65.70	95.00	70.00	83.04	66.41	48.04	46.80	65.40	79.75	86.53	72.97
UMA (32k Window)	90.00	92.00	84.00	68.60	97.00	83.00	90.18	74.22	50.86	51.80	69.60	82.00	87.87	78.55

Table 12: LLM-as-a-Judge summary scores using `Qwen3-30B-A3B-Instruct-2507` as the evaluator across all benchmarks. Unless otherwise specified, all methods use a 16k context budget. UMA-8k and UMA-32k are reported separately as context-window scaling variants. Bold numbers in the default-budget block indicate the best result among methods using the 16k budget; underlined numbers in the scaling block indicate the better value between UMA-8k and UMA-32k.

Method	Session Horizon H									
	2	**5**	**10**	**20**	**30**	**40**	**50**	**100**	**500**	**Avg.**
Baselines										
Concat	63.11	24.39	21.36	8.82	7.77	12.38	12.38	7.69	1.92	17.76
RAG	80.58	39.02	21.36	17.65	13.59	15.24	10.48	8.65	5.77	23.59
MemAgent	81.55	55.28	42.72	39.22	39.81	32.38	32.38	29.81	11.54	40.52
MemAgent-woq	82.52	34.96	9.71	7.84	12.62	8.57	7.62	4.81	2.88	19.06
Mem1	27.18	9.76	4.85	3.92	9.71	1.90	1.90	6.73	9.62	8.40
A-MEM	33.01	24.39	19.42	11.76	10.68	8.57	8.57	0.96	1.92	13.25
Mem-α	60.19	37.40	34.95	21.57	18.45	16.19	22.86	14.42	9.62	26.18
Mem-T	58.25	39.84	25.24	13.73	10.68	8.57	13.33	11.54	2.88	20.45
Ablations										
UMA (w/o RL & Phase I)	18.45	11.38	8.74	8.82	7.77	8.57	5.71	24.04	14.42	11.99
UMA (w/o RL)	58.25	37.40	29.13	24.51	31.07	26.67	32.38	25.00	9.62	30.45
UMA (w/o Raw-Context)	83.50	61.79	59.22	**60.78**	52.43	**46.67**	**48.57**	**34.62**	**25.00**	52.51
UMA	**87.38**	**63.41**	**63.11**	57.84	**53.40**	**46.67**	45.71	**34.62**	**25.00**	**53.02**
Context-window scaling										
UMA (8k Window)	83.50	59.35	54.37	50.00	39.81	40.00	33.33	22.12	20.19	44.74
UMA (32k Window)	85.44	67.48	61.17	54.90	48.54	50.48	47.62	38.46	25.96	53.34

Table 13: Detailed LLM-as-a-Judge scores (%) on Ledger-QA across varying session horizons H. Columns represent the number of sessions in the input stream. Bold numbers in the default-budget block indicate the best result among methods using the 16k budget; underlined numbers in the scaling block indicate the better value between UMA-8k and UMA-32k.

Method	Session Horizon H									
	2	**5**	**10**	**20**	**30**	**40**	**50**	**100**	**500**	**Avg.**
Baselines										
Concat	65.05	32.52	31.07	20.59	18.45	21.90	16.19	8.65	4.81	24.36
RAG	46.60	34.96	32.04	26.47	23.30	24.76	24.76	14.42	14.42	26.86
MemAgent	78.64	58.54	46.60	45.10	43.69	32.38	32.38	**46.15**	16.35	44.43
MemAgent-woq	83.50	39.84	6.80	16.67	15.53	11.43	7.62	8.65	15.38	22.82
Mem1	20.39	8.94	6.80	8.82	7.77	3.81	3.81	4.81	12.50	8.63
A-MEM	30.10	28.46	33.98	23.53	15.53	20.00	13.33	13.46	12.50	21.21
Mem-α	32.04	41.46	43.69	27.45	27.18	25.71	20.00	19.23	9.62	27.38
Mem-T	48.54	44.72	36.89	21.57	18.45	15.24	22.86	10.58	0.96	24.42
Ablations										
UMA (w/o RL & Phase I)	63.11	52.03	49.51	45.10	39.81	40.95	40.00	22.12	19.23	41.32
UMA (w/o RL)	67.96	47.15	42.72	40.20	39.81	35.24	38.10	40.38	**31.73**	42.59
UMA (w/o Raw-Context)	**84.47**	64.23	61.17	**61.76**	57.28	**52.38**	**57.14**	45.19	25.96	**56.62**
UMA	74.76	**65.04**	**66.02**	58.82	**58.25**	**52.38**	53.33	42.31	21.15	54.67
Context-window scaling										
UMA (8k Window)	84.47	62.60	58.25	52.94	51.46	55.24	52.38	27.88	22.12	51.93
UMA (32k Window)	84.47	69.92	63.11	55.88	53.40	57.14	55.24	45.19	24.04	56.49

Table 14: Detailed Sub-EM scores (%) on Ledger-QA across varying session horizons H. Columns represent the number of sessions in the input stream. Bold numbers in the default-budget block indicate the best result among methods using the 16k budget; underlined numbers in the scaling block indicate the better value between UMA-8k and UMA-32k.