MEM-C: SCALING AND ALLOCATING CREDIT BEYOND DENSE REWARDS IN LONG-HORIZON AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-horizon language agents face severe temporal credit-assignment challenges, as the consequences of intermediate decisions may only emerge many steps later. Dense process supervision and tree-based training alleviate this issue by providing additional downstream feedback, but existing work largely treats credit assignment as a one-dimensional notion of reward density. This overlooks a key fact: the same credit budget can be distributed differently across decisions, and additional supervision may have very different marginal value depending on where it is assigned. We therefore ask: how does agent performance scale with credit, and how should a fixed credit budget be allocated across long-horizon decisions? To answer this question, we introduce **Credit Assignment Geometry**, which characterizes step-level credit along three dimensions: ❶ ***Quantity***, ❷ ***Heterogeneity***, and ❸ ***Utility***. We find that increasing credit quantity can yield diminishing returns, while allocations with comparable quantity and coverage can yield substantially different performance due to heterogeneous credit utility. Motivated by these findings, we propose Mem-C, which uses branch-outcome dispersion to identify high-utility decisions for importance-aware branch allocation. Under matched credit budgets, `Mem-C` outperforms the evaluated baselines on all three transfer benchmarks, including $\mathbf{12}\%\uparrow$ on HotpotQA. Code is available at Mem-C.

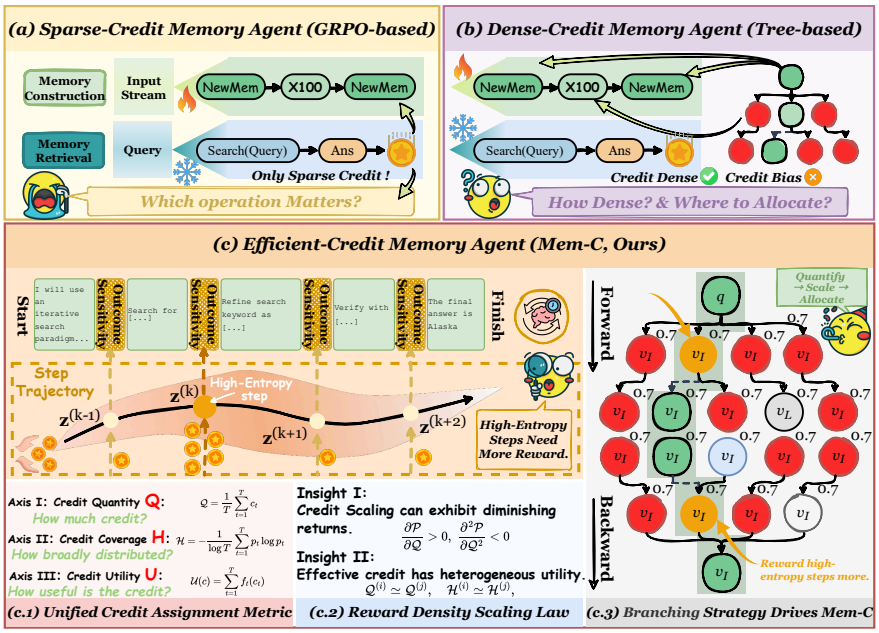

Figure 1: From GRPO-based to tree-based training, we introduce Q, H, and U metrics to characterize long-horizon credit assignment, and `Mem-C` to systematically scale and efficiently allocate downstream credit for memory agents.

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.

1 INTRODUCTION

Long-horizon memory agents must learn not only what to remember, but also how memory operations shape future outcomes (Zhou et al., 2026; Zhang et al., 2026b; Hu et al., 2025; Li et al., 2025; 2024). Constructing, revising, or retrieving memory can affect decisions many steps later, making reinforcement learning (RL) difficult (Cherepanov et al., 2026; Wang et al., 2025; Yan et al., 2026; Gupta et al., 2026; Shi et al., 2026; Xu et al., 2026) because feedback is often delayed: a trajectory contains many memory decisions, while only a few outcomes reveal whether those decisions were useful (Fang et al., 2026; Huang et al., 2025; Zhu et al., 2026). This delayed feedback creates a fundamental credit-assignment challenge, as sparse outcome signals must provide effective supervision for long sequences of memory operations (Chen et al., 2026b; Ma et al., 2026; Bull et al., 2026).

Recent methods address long-horizon credit assignment by constructing additional structure over agent trajectories, including tree-based, graph-based, and clustering-based formulations (Mao et al., 2026; Zhao et al., 2026b; Wang et al., 2026c; Yang et al., 2026a). Despite their different implementations, these approaches share a common goal: converting sparse downstream outcomes into denser supervision for intermediate decisions by exploiting relationships among multiple continuations or trajectories. For example, tree-based RL methods branch from intermediate steps of a trajectory, generate multiple downstream continuations, and use their terminal outcomes to provide additional feedback for earlier decisions. However, denser supervision does not necessarily imply more informative credit (Ji et al., 2026; Zong et al., 2026; Zhao et al., 2026a). Under comparable rollout budgets, different credit-construction strategies can induce substantially different distributions of supervision across intermediate decisions (Zou et al., 2026; Murphy, 2026). This motivates our central question:

How does memory-agent learning scale, and how should credit be allocated across decisions?

To study how credit scaling affects learning, we first make credit quantity measurable. Existing notions such as "dense rewards" often conflate the number of trajectories, terminal evaluations, and outcome signals assigned to each decision. We therefore define step-level credit density as the average number of downstream outcome signals supervising each decision. This measures the multiplicity of supervision, independent of reward magnitude and without assuming that signals are independent or equally informative. Credit density captures how much supervision is available, but not how it is allocated. The same amount of credit can be distributed broadly across many decisions or concentrated on a smaller subset, and even allocations with similar concentration may differ in value depending on which decisions receive additional supervision.

We therefore decompose credit scaling into three complementary dimensions: ❶ ***Quantity***, ❷ ***Heterogeneity***, and ❸ ***Utility***. ***Quantity*** measures how much credit each decision receives on average. ***Heterogeneity*** captures how evenly a fixed amount of credit is spread across decisions, distinguishing broad coverage from concentrated supervision. ***Utility*** asks which decisions benefit most from additional credit, separating supervision of consequential decisions from repeated credit on relatively inconsequential ones. This decomposition lets us isolate three distinct sources of improvement: having more credit, covering more decisions, and assigning credit to more useful decisions. We study these factors by varying credit quantity, comparing sparse and uniform allocations at fixed quantity, and, while matching quantity and distribution as closely as possible, varying which decisions receive additional supervision. As illustrated in Figure 1, these three dimensions provide a unified view of credit scaling and motivate our importance-aware allocation strategy. Building on this framework, we introduce `Mem-C`, an importance-aware credit allocation method based on outcome-sensitive branching. `Mem-C` estimates the sensitivity of expandable decision points from observed branch outcomes and combines this signal with a coverage-aware selector for allocate continuation branches more effectively. We compare `Mem-C` with random and coverage-based branching to test whether importance-aware targeting improves credit utility beyond what can be explained by credit quantity or distribution alone, while accounting for the additional judging cost. Our contributions are as follows:

- **Credit Assignment Geometry.** We introduce a framework that characterizes step-level credit assignment along three dimensions: *Quantity, Heterogeneity, and Utility*, providing a unified view of how much credit is available, how broadly it is distributed, and how effectively it is allocated.
- **Scaling Beyond Credit Quantity.** Through controlled credit-scaling experiments, we show that increasing credit quantity improves memory-agent performance with diminishing returns, while performance under comparable budgets also depends strongly on credit distribution and allocation, revealing an additional scaling dimension beyond reward densification.

- **Mem-C: Utility-Aware Credit Allocation.** Building on these findings, we propose `Mem-C`, an outcome-sensitive branching method that estimates decision utility and uses it to guide credit allocation. Experiments show that utility-aware branch selection can improve learning under matched search budgets and yields strong performance across long-horizon memory benchmarks.

2 Related Work

RL for Long-Horizon Memory Agents. Memory is essential for long-horizon agents to preserve task-relevant information beyond limited context windows (Hu et al., 2025; Cherepanov et al., 2026). Recent work increasingly uses reinforcement learning to learn memory management rather than relying on heuristic accumulation. One line of work learns to compress and consolidate long interaction histories into compact memories, as in MemAgent (Yu et al., 2026), MEM1 (Zhou et al., 2026), and MemPO (Li et al., 2026b); another treats memory management as explicit learnable decisions, including storage, updating, retrieval, and context reconstruction (Yan et al., 2026; Zhang et al., 2026b; Sun et al., 2025; Chen et al., 2026a). Related efforts further extend RL-based memory adaptation to online and embodied settings (Zhang et al., 2026a; Gupta et al., 2026). *However, the scaling and allocation of credit signals in memory-agent training remain underexplored.*

Credit Assignment in Long-Horizon RL. Long-horizon RL is fundamentally challenged by sparse and delayed rewards, making temporal credit assignment difficult. Existing approaches mitigate this issue through hierarchical decomposition and reward densification (Wang et al., 2026a), more refined trajectory- and transition-level credit estimation (Li et al., 2026a; Liu et al., 2026; Yang et al., 2026b), and intermediate supervision via process or memory-specific returns (Zhang et al., 2026c; Li et al., 2026b; Yan et al., 2026). Credit assignment has also been extended to multi-agent settings, with agent- and turn-specific advantage estimation and analyses of attribution at scale (Yuan et al., 2026; Geng et al., 2026). *Despite this progress, existing methods remain largely tailored to specific trajectory structures, supervision signals, or interaction regimes.*

Scaling and Adaptive Credit Allocation. With the growing emphasis on inference-time computation, test-time scaling has become an effective paradigm for improving LLM reasoning and decision making (Snell et al., 2024; Zhu et al., 2025). Increasing inference-time computation through sampling, search, revision, and verification can improve performance without enlarging model parameters (Snell et al., 2024), while extensions to language agents further highlight the importance of diversified rollouts, reflection, and verification (Zhu et al., 2025). Recent work moves beyond uniform scaling toward adaptive resource allocation: compute-optimal scaling adjusts budgets across prompts (Snell et al., 2024), constrained allocation balances accuracy and cost (Zhai et al., 2026), and DORA improves search efficiency through direction-oriented allocation (Wang et al., 2026b). A related challenge arises in long-horizon RL, where sparse outcome rewards make credit assignment difficult (Pignatelli et al., 2023; Zhang et al., 2023). Existing methods improve intermediate supervision through Monte Carlo returns (Kazemnejad et al., 2024), process-level rewards (Choudhury, 2025), transition-aware advantage estimation (Liu et al., 2026), and dense turn-level value changes (Tao et al., 2026). *However, prior work mainly optimizes* how *computation or credit is assigned, leaving its scaling behavior largely uncharacterized. In particular, how performance varies with credit density and how a fixed credit budget should be allocated across long-horizon decisions remain underexplored.*

3 Method

Existing long-horizon memory agents alleviate delayed supervision by propagating terminal feedback to intermediate memory operations. However, treating credit assignment merely as a transition from sparse to dense supervision leaves two fundamental questions unresolved: *how much credit should each decision receive, and, under a fixed computation budget, where should additional credit be allocated?* We study these questions through a unified view of long-horizon credit assignment and derive an adaptive allocation principle from its scaling behavior. As shown in Figure 2, our framework first characterizes credit assignment through its quantity, coverage, and utility, and then instantiates these principles in **Mem-C**, which uses outcome-sensitive branching to allocate a fixed continuation budget toward high-utility memory decisions.

3.1 Credit Assignment Geometry

We first formalize credit assignment as a structured object rather than a scalar notion of reward density. Consider a long-horizon interaction trajectory, and we denote its downstream credit set as

$$\mathcal{T} = (\boldsymbol{\mathcal{S}}_1, a_1, \boldsymbol{\mathcal{S}}_2, a_2, \ldots, \boldsymbol{\mathcal{S}}_T, a_T), \quad \boldsymbol{\mathcal{R}}_t = \left\{ r_t^{(1)}, \ldots, r_t^{(c_t)} \right\}, \; c_t = |\boldsymbol{\mathcal{R}}_t|, \tag{1}$$

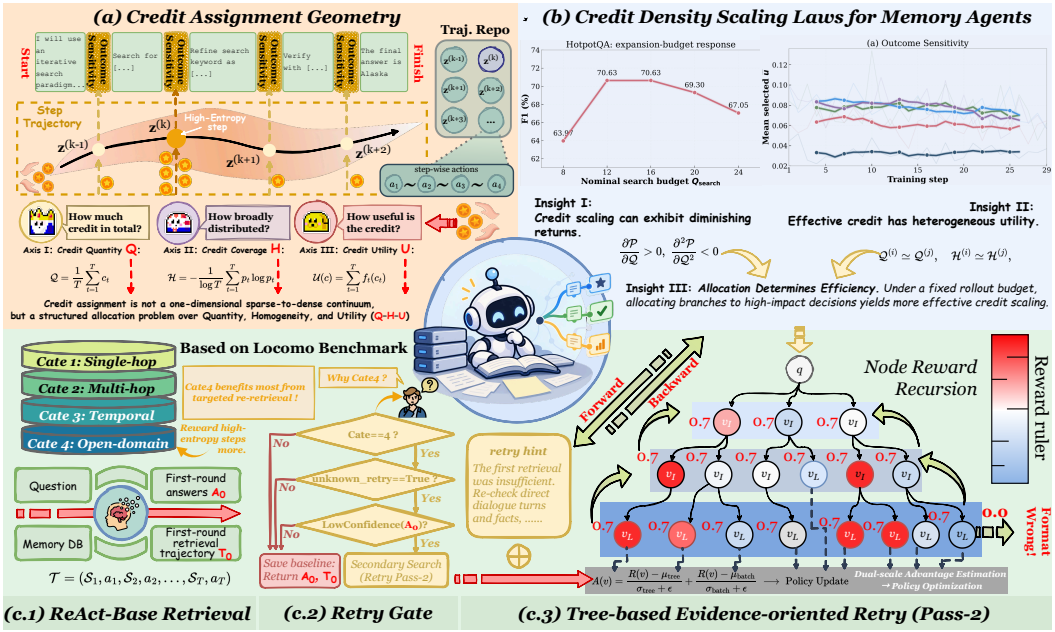

Figure 2: Overview of Mem−C. Subgraph (b): Left: HotpotQA F1 on 15 matched questions versus nominal search budget $Q_{\text{search}} = 4(1+n)$. Right: Mean selected-branch utility across β_U settings over steps 1–29 (seven-step smoothing), with nominal $Q_{\text{search}} = 14$, $H_{\text{cfg}} = 0.5$, and seed 42.

where $\boldsymbol{\mathcal{S}}_t$ denotes the agent state before the t-th memory decision, a_t is the corresponding memory operation and c_t measures how many effective downstream outcomes are assigned to decision t. Depending on the underlying credit-assignment mechanism, one decision may participate in multiple downstream branches and therefore receive multiple terminal supervision signals. Rather than collapsing $\{c_t\}_{t=1}^{T}$ into a single density statistic, we characterize a credit assignment by $\boldsymbol{\mathcal{G}}(\boldsymbol{\mathcal{T}}) = \Big(\boldsymbol{\mathcal{Q}}, \boldsymbol{\mathcal{H}}, \boldsymbol{\mathcal{U}}\Big)$, corresponding to credit quantity, credit coverage, and credit utility, respectively. These three coordinates answer fundamentally different questions: How much credit is available? How broadly is it distributed? How useful is its allocation?

$$\boldsymbol{\mathcal{Q}} = \frac{1}{T}\sum_{t=1}^{T} c_t, \quad \boldsymbol{\mathcal{H}} = -\frac{1}{\log T}\sum_{t=1}^{T} p_t \log p_t \in [0,1], \quad \boldsymbol{\mathcal{U}}(\boldsymbol{c}) = \sum_{t=1}^{T} f_t(c_t),$$
$$p_t = \frac{c_t + \epsilon}{\sum_{j=1}^{T}(c_j + \epsilon)}, \quad u_t(c) \triangleq \mathbb{E}\left[\Delta\mathcal{P} \,|\, c_t \leftarrow c_t + 1,\ \boldsymbol{\mathcal{T}}\right], \quad f_t'(c) > 0,\ f_t''(c) < 0. \tag{2}$$

Together, $\boldsymbol{\mathcal{Q}}$, $\boldsymbol{\mathcal{H}}$, and $\boldsymbol{\mathcal{U}}$ characterize three complementary aspects of downstream credit assignment. $\boldsymbol{\mathcal{Q}}$ quantifies the overall amount of supervision available along a trajectory, whereas $\boldsymbol{\mathcal{H}}$ captures how evenly this supervision is distributed across decisions. However, neither quantity nor coverage alone determines whether the available credit is assigned to decisions where it is most beneficial. We therefore use $\boldsymbol{\mathcal{U}}(\boldsymbol{c})$ to characterize the effectiveness of a particular allocation, with $u_t(c)$ representing the expected marginal performance gain from assigning one additional unit of credit to decision t. The assumptions $f_t'(c) > 0$ and $f_t''(c) < 0$ encode positive but diminishing returns, reflecting that additional supervision is useful while repeatedly allocating credit to the same decision provides progressively smaller gains. Taken together, these three coordinates distinguish how much credit is available, how broadly it is distributed, and how effectively it is allocated. The resulting geometry highlights an important distinction: How Much $\neq$ How Broad $\neq$ How Useful $\iff$ $\boldsymbol{\mathcal{Q}} \neq \boldsymbol{\mathcal{H}} \neq \boldsymbol{\mathcal{U}}$. Credit assignment is therefore not a one-dimensional "sparse-to-dense" continuum, but a structured allocation problem over quantity, coverage, and utility.

3.2 Scaling Laws of Long-Horizon Credit

The geometry above allows us to ask how agent performance changes when different dimensions of credit assignment are varied independently. Our analysis reveals two complementary insights.

♣ ***Insight I: credit scaling can exhibit diminishing returns.*** We vary the nominal search budget while approximately preserving the allocation policy. As a modeling hypothesis, increasing credit quantity may yield positive but diminishing returns, captured by the saturating form:

$$\begin{aligned} \boldsymbol{\mathcal{Q}} \uparrow \quad \Longrightarrow \quad \mathcal{P} \uparrow, \quad \frac{\partial \mathcal{P}}{\partial \boldsymbol{\mathcal{Q}}} > 0, \; \frac{\partial^2 \mathcal{P}}{\partial \boldsymbol{\mathcal{Q}}^2} < 0, \\ \mathcal{P}(\boldsymbol{\mathcal{Q}}) = \mathcal{P}_0 + \alpha \left(1 - e^{-\beta \boldsymbol{\mathcal{Q}}}\right), \; \alpha, \beta > 0, \end{aligned} \tag{3}$$

This form is illustrative rather than an empirically established scaling law: our experiments vary nominal search budget rather than directly measured credit quantity and show non-monotonic performance. Importantly, increasing raw density does not specify where the additional supervision is spent. At a fixed quantity, performance should instead be viewed as a conditional surface $\mathcal{P} = F\left(\boldsymbol{\mathcal{H}}, \boldsymbol{\mathcal{U}} \mid \boldsymbol{\mathcal{Q}}\right)$. This distinction explains why maximizing reward density alone is not a sufficient objective for long-horizon training.

➽ ***Insight II: effective credit has heterogeneous utility.*** We next consider assignments with approximately matched quantity and coverage,

$$\boldsymbol{\mathcal{Q}}^{(i)} \simeq \boldsymbol{\mathcal{Q}}^{(j)}, \qquad \boldsymbol{\mathcal{H}}^{(i)} \simeq \boldsymbol{\mathcal{H}}^{(j)}, \tag{4}$$

but different decision-selection policies. If their resulting performance differs, then $\boldsymbol{\mathcal{Q}}$ and $\boldsymbol{\mathcal{H}}$ cannot be sufficient statistics of credit assignment. Instead, the difference is explained by heterogeneous marginal utility across decisions: $u_i(c_i) \neq u_j(c_j)$ in general. This observation turns credit assignment into a constrained resource allocation problem. Given a total additional credit budget $\boldsymbol{\mathcal{B}}$, the ideal allocation satisfies

$$\boldsymbol{c}^\star = \arg\max_{\boldsymbol{c} \in \mathbb{N}^T} \quad \sum_{t=1}^{T} f_t(c_t), \quad \text{s.t.} \sum_{t=1}^{T} c_t \leq \boldsymbol{\mathcal{B}}. \tag{5}$$

Under the continuous relaxation, the optimality condition is $f_t'(c_t^\star) = \lambda^\star$ for every active decision t, where $\lambda^\star$ is the marginal value of the global budget. The equation above provides a useful interpretation: additional computation should be assigned to decisions with the largest remaining marginal credit utility, rather than being distributed uniformly across the trajectory. Combining the two insights yields the central design principle of our method: *Under a fixed branching budget, allocate additional continuations preferentially to decisions with higher estimated marginal credit utility.*

3.3 Utility-Aware Credit Allocation

Building on the allocation principle above, we instantiate utility-aware credit allocation under a fixed branching budget. We estimate the utility of an expandable decision point from the downstream variation among its observed alternative continuations, and use this signal to guide branch selection.

Consider an expandable node v with b_v scored child branches. For each child branch i, let $\mathcal{L}_i(v)$ denote the set of scored descendant leaves under that branch. We summarize its downstream value as

$$V_i(v) = \frac{1}{|\mathcal{L}_i(v)|} \sum_{\ell \in \mathcal{L}_i(v)} R(\ell), \tag{6}$$

where $R(\ell)$ denotes the terminal reward of leaf ℓ. The utility of node v is then estimated from the dispersion among alternative branch outcomes:

$$\widehat{U}(v) = \frac{2}{b_v(b_v - 1)} \sum_{i<j} \left| V_i(v) - V_j(v) \right|, \qquad b_v \geq 2. \tag{7}$$

If fewer than two scored branches are available, $\widehat{U}(v) = 0$. Intuitively, a larger $\widehat{U}(v)$ indicates that different continuations from the same decision point lead to more distinct downstream outcomes, and hence that the allocation decision at v has greater potential impact on future performance. We use $\widehat{U}(v)$ as an observable proxy for the marginal credit utility $u_t(c)$ defined in Section 3.1.

We combine this utility estimate with the coverage preference introduced above. For a candidate node v, let $\mathcal{T}(v)$ denote the set of retrieved turn identifiers along its path, and let $\mathcal{S}$ denote the union of retrieved turns from nodes already selected in the current expansion step. We define

$$O(v, \mathcal{S}) = \frac{|\mathcal{T}(v) \cap \mathcal{S}|}{\max(1, |\mathcal{T}(v) \cup \mathcal{S}|)}, \qquad N(v, \mathcal{S}) = \frac{|\mathcal{T}(v) \setminus \mathcal{S}|}{\max(1, |\mathcal{T}(v) \cup \mathcal{S}|)}, \tag{8}$$

and the corresponding coverage score $C_h(v, \mathcal{S}) = (1-h)O(v, \mathcal{S}) + hN(v, \mathcal{S})$, where $h \in [0, 1]$ controls the preference between overlapping and novel retrieval paths. The final utility-aware selection score is

$$S_U(v, \mathcal{S}) = C_h(v, \mathcal{S}) + \beta_U \widehat{U}(v), \tag{9}$$

where $\beta_U \geq 0$ controls the contribution of utility to branch selection. For each expansion round, one node is first sampled as a stochastic anchor, after which the remaining expansion nodes are selected according to Eq. 9, with random tie breaking. As β_U increases, selection increasingly favors decision points with larger downstream outcome variation. The utility-aware selector reallocates the fixed continuation budget across candidate decision points. With m initial trajectories, l expansion rounds, and n selected nodes per tree in each round, the nominal search budget is $Q_{\text{search}} = m(1+nl)$. After each expansion round, the newly scored leaves update the branch values in Eq. 6, allowing $\widehat{U}(v)$ to be refreshed. The resulting tree is then optimized with standard memory-tree advantage estimator.

4 Experiments

Experimental Setup ❶ **Evaluation and Benchmarks.** We evaluate `Mem-C` on four long-context benchmarks: LoCoMo (Maharana et al., 2024), LongMemEval (Wu et al., 2025), HotpotQA (Yang et al., 2018), and NarrativeQA (Kočiský et al., 2017). Following Memory-R1 (Yan et al., 2025), we adopt a conversation-level 1:1:8 training/validation/test split of LoCoMo, excluding adversarial questions. The other benchmarks assess transfer across memory and reading-comprehension tasks. Dataset construction is detailed in *Appendix* A.1. ❷ **Baselines and Evaluation Protocol.** We compare `Mem-C` with representative long-context and memory-augmented approaches, using official implementations and released checkpoints where available. Dataset partitioning, inference procedures, and metric definitions are detailed in *Appendix* A, while inference procedures and metric definitions are provided in *Appendix* A.6. ❸ **Implementation Details.** Our implementation uses Qwen3 backbones (Yang et al., 2025) and BGE-M3 embeddings (Chen et al., 2025). The default search configuration generates three initial trees and performs four expansion rounds, selecting three nodes per tree in each round, yielding $Q_{\text{search}} = 39$ nominal rollout slots. The full RL configuration specifies 200 optimization steps, whereas the coverage screen uses one GRPO step. Standard inference allows up to six retrieval-tool steps with top-5 retrieval. Reward construction, advantage estimation, training hyperparameters, SFT, and decoding settings are provided in *Appendix* A.

4.1 Main Results

Table 1: Performance comparison on the LoCoMo benchmark, with F1 and BLEU-1 as the evaluation metrics. †: The GAM paper recommends gpt-4o-mini; we also reproduced it using Qwen3-4B for a fair comparison. ‡: As Memory-R1 is not open-source, we faithfully report the results provided in their original paper.

Method	Base LLM	Single-Hop		Multi-Hop		Temporal		Open Domain		Overall	
		F1↑	B1↑	F1↑	B1↑	F1↑	B1↑	F1↑	B1↑	F1↑	B1↑
Training-free Methods											
VANILLA	Qwen3-4B	40.68	31.54	23.23	16.76	18.97	13.42	13.87	10.70	31.50	23.94
RAG	Qwen3-4B	49.45	44.94	23.50	17.13	43.07	37.35	20.23	14.94	41.59	36.45
MemGPT	Qwen3-4B	14.00	11.77	16.68	13.99	12.56	10.94	11.61	9.16	14.05	11.84
MemoryBank	Qwen3-4B	26.65	17.72	25.52	19.44	9.15	7.44	16.42	12.39	22.34	15.66
Mem0	Qwen3-4B	47.28	40.72	35.40	27.36	46.84	39.48	26.64	21.04	43.71	36.78
MemoryOS	Qwen3-4B	48.35	42.57	35.24	27.30	40.98	32.68	22.08	17.93	42.83	36.26
LightMem	Qwen3-4B	43.78	38.84	30.78	25.80	44.71	40.72	18.93	14.42	40.01	35.27
A-Mem	Qwen3-4B	44.62	38.26	27.24	21.07	43.85	35.97	15.40	12.71	39.43	33.04
GAM	Qwen3-4B	32.23	25.54	32.23	28.66	26.26	22.52	18.45	14.47	30.17	24.81
GAM†	gpt-4o-mini†	57.75	52.10	42.29	34.44	59.45	53.11	29.73	24.74	53.48	47.33
Trained Methods											
MEM1	MEM1-7B	27.48	22.10	18.98	15.56	30.52	23.48	14.21	11.43	25.68	20.50
MemAgent	MemAgent-14B	35.86	29.64	27.86	22.72	37.93	31.85	20.31	16.47	33.82	27.97
Memory-R1-PPO‡	Mem-R1-8B‡	32.52	24.47	26.86	23.47	41.57	26.11	45.30	39.18	34.08	25.54
Memory-R1-GRPO‡	Mem-R1-8B‡	35.73	27.70	35.65	30.77	49.86	38.27	**47.42**	**41.24**	39.25	31.21
Mem-T	Qwen3-4B	63.75	57.95	45.09	36.58	65.13	60.12	32.97	28.94	58.65	52.63
`Mem-C` (Ours)	Qwen3-4B	**65.57**	**59.86**	44.57	35.64	64.97	59.76	30.91	26.66	**59.41**	**53.31**

Obs. ❶: Overall Performance. As shown in Table 1, `Mem-C` achieves the best performance, reaching 59.41 F1 and 53.31 B1. Compared with Mem T, the strongest competing method under the same Qwen3 4B backbone, `Mem-C` improves the F1 and B1 by 0.76 and 0.68 points, respectively. The advantage is particularly evident on Single Hop questions, where `Mem-C` obtains 65.57 F1 and 59.86 B1, establishing the best results among all compared methods. On Multi Hop and Temporal

questions, Mem-C remains competitive with Mem T, with only marginal differences shown in Table 1. Although its performance on Open Domain questions is lower than that of Memory R1, the substantial gains on the other question types lead to the strongest overall results. These results demonstrate that Mem-C provides a more effective balance across different memory reasoning scenarios.

Obs. ❷: Cross-domain generalization. As shown in Table 2, Mem-C exhibits strong generalization across all three OOD benchmarks. It achieves 79.17 F1 on HotpotQA, 68.03 accuracy on LongMemEval, and 35.17 F1 on NarrativeQA, consistently outperforming all competing methods. Compared with the strongest trained baseline, Mem-T, Mem-C improves performance by 12.82, 2.23, and 4.88 points on the three benchmarks, respectively. This consistent advantage raises the average score from 54.15 to 60.79, demonstrating that the learned memory capability transfers effectively to previously unseen domains and tasks.

Table 2: OOD evaluation on HotpotQA, LongMemEval, and NarrativeQA. All methods use Qwen3-4B, except MEM1, which uses the original 7B model.

Method	HotpotQA F1↑	LongMemEval Acc↑	NarrativeQA F1↑	Avg.
Training-free Methods				
VANILLA	21.89	38.80	18.09	26.26
RAG	50.13	56.60	21.17	42.63
MemGPT	18.24	23.00	8.39	16.54
MemoryBank	16.90	26.20	9.65	17.58
A-Mem	30.46	61.30	25.18	38.98
Mem0	31.96	53.60	27.63	37.73
MemoryOS	26.86	46.80	23.45	32.37
LightMem	38.62	63.10	16.78	39.50
GAM	52.98	61.80	28.32	47.70
Trained Methods				
MEM1	55.36	19.00	13.49	29.28
Mem-α	58.80	52.00	28.56	46.45
Mem-T	<u>66.35</u>	<u>65.80</u>	<u>30.29</u>	<u>54.15</u>
Mem-C (Ours)	**79.17**	**68.03**	**35.17**	**60.79**

4.2 Credit Scaling and Allocation Analysis

Obs. ❶: Exp-A: Scaling Credit Quantity. Table 3 shows that the benefit of search budget depends on both its allocation and the target task. Along the tree-count path, LoCoMo reaches 59.36 at (m=4) before dropping to 58.07 at (m=5), while HotpotQA peaks at 75.97 with (m=2). Along the expansion-width path, LoCoMo improves from 59.36 at (n=1) to 59.74 at (n=4), whereas HotpotQA peaks earlier at (n=2) and (n=3) before declining. LongMemEval favors the narrower setting, reaching 65% at ((m,n)=(4,1)) and decreasing to 55∼60% with wider expansion. These task-specific optima show that larger budgets do not consistently yield better performance. Effective scaling therefore requires allocating search selectively: moderate widening benefits LoCoMo and HotpotQA but can hurt LongMemEval. This motivates viewing credit assignment as a structured allocation problem in which both supervision quantity and distribution matter.

Table 3: Exp-A: Search-budget scaling. We vary trees m and expansion nodes n with $l = k = 1$ and $Q_{\text{search}} = m(1+n)$. Results are F1 for LoCoMo and HotpotQA and judge accuracy for LongMemEval.

Config.	m	n	Q_{search}	LoCoMo F1↑	HotpotQA F1↑	LongMemEval Acc.↑
Increasing tree count ($n = 1$)						
	1	1	2	59.07	69.30	60.00
	2	1	4	59.16	75.97	60.00
	3	1	6	58.57	63.97	60.00
	4	1	8	59.36	63.97	65.00
	5	1	10	58.07	63.97	60.00
Increasing expansion width ($m = 4$)						
	4	2	12	58.80	70.63	55.00
	4	3	16	59.45	70.63	60.00
	4	4	20	59.74	69.30	55.00
	4	5	24	59.41	67.05	60.00

Obs. ❷: Exp-B: Varying Credit Coverage. Table 4 shows non-monotonic variation in rollout reward across nominal coverage settings. H_{cfg}=0.5 achieves the highest mean reward (0.1740), whereas H_{cfg}=1.0 has the shortest generation time but the lowest completion rate. These descriptive differences suggest reward-completion trade-offs, but do not establish an optimal coverage level: the results involve only one training step, and neither parameter propagation nor realized credit coverage was verified.

Table 4: Exp-B: Early-training screening. Mean ± std. over three seeds after one GRPO step ($Q_{\text{search}} = 39$).

Nominal H_{cfg}	Rollout Reward	Completion (%)	Generation Time (s)
0.00	0.1640 ±0.0211	80.56 ±5.24	1080.66 ±47.93
0.25	0.1476 ±0.0148	79.86 ±3.18	1053.22 ±11.43
0.50	0.1740 ±0.0114	82.64 ±5.24	1048.39 ±62.70
0.75	0.1531 ±0.0227	82.64 ±3.18	1027.97 ±34.65
1.00	0.1702 ±0.0247	77.78 ±1.20	1017.65 ±27.22

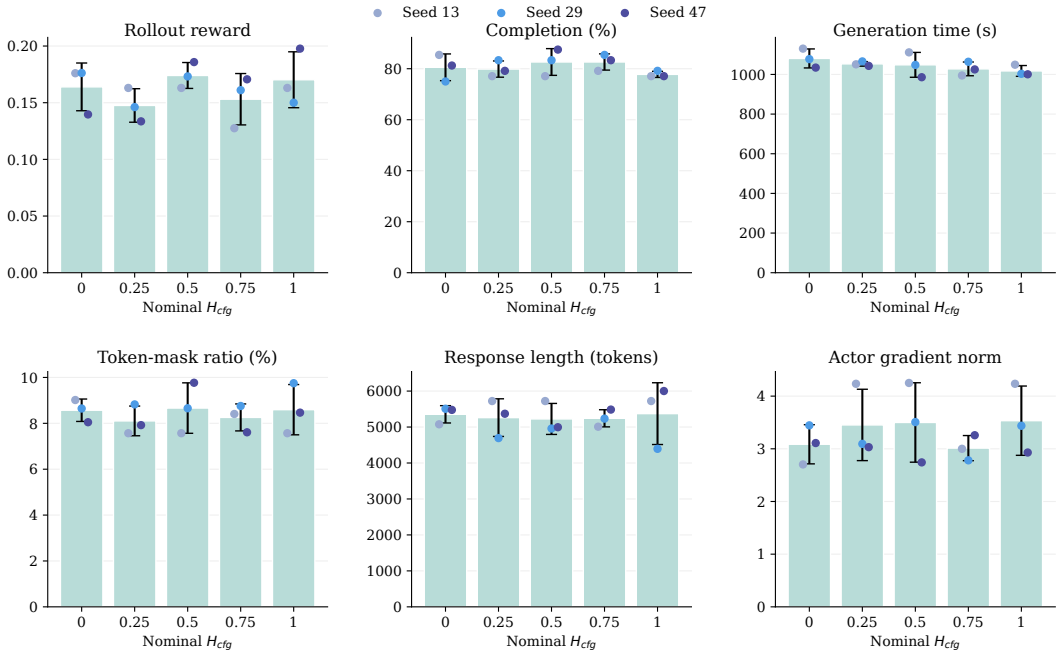

Figure 3: **Coverage-dependent training outcomes.** Reward, completion, and generation time across coverage settings at fixed $Q_{\text{search}} = 39$. Results reflect one-step GRPO screening over 3 seeds.

♣ **Coverage-Dependent Reward–Efficiency Trade-offs.** As shown in Figure 3, nominal coverage settings exhibit non-monotonic reward–efficiency trade-offs at fixed $Q_{\text{search}} = 39$. $H_{\text{cfg}} = 0.5$ achieves the highest mean rollout reward (0.1740) and ties for the highest completion rate (82.64%). By contrast, $H_{\text{cfg}} = 1$ yields shorter generation time but the lowest completion rate (77.78%), indicating that faster generation does not necessarily translate into more successful rollouts. These preliminary observations motivate studying credit allocation alongside budget, although seed variability and unverified realized coverage prevent establishing an optimal coverage level.

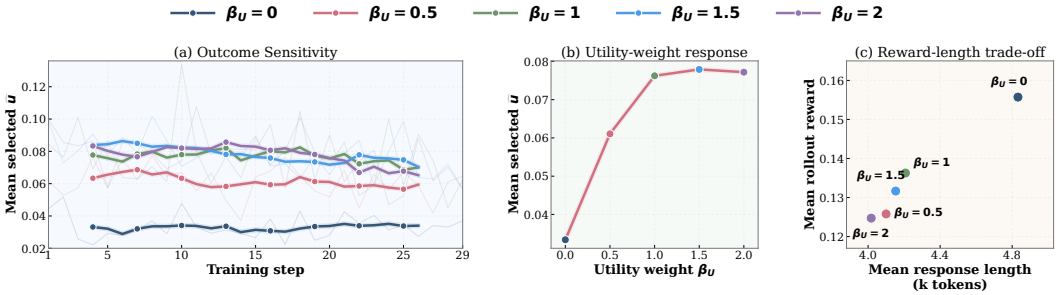

Figure 4: **Utility-weighted branch selection.** At nominal $Q_{\text{search}} = 14$ and $H_{\text{cfg}} = 0.5$, panels show utility-estimate trajectories, their response to β_U, and reward–length trade-offs. Results use seed 42 and steps 1–29, with seven-step smoothing on the left.

Obs. ❸: Exp-C: Allocating Credit by Utility. Table 5 studies the effect of utility-guided branch selection by varying β_U under a fixed nominal search budget. *Val. Score* denotes the validation aggregate computed from lexical answer matching between model predictions and reference answers. Compared with the coverage-only setting ($\beta_U = 0$), three of the four non-zero utility settings achieve higher validation scores, while $\beta_U = 1.0$ performs slightly below the baseline. The strongest performance is obtained at $\beta_U = 1.5$ and $\beta_U = 2.0$. The improvement is achieved

Table 5: Exp-C: Utility-weight scaling. All metrics are reported at the same checkpoint (step 20) under different utility coefficients β_U.

β_U	Val. Score	Completion	Token-mask	Gen. (min)	Response (k)
0.0	2119.046	87.5%	8.8%	13.09	4.25
0.5	2206.121	87.5%	6.3%	13.67	4.38
1.0	2112.997	84.4%	8.0%	12.67	4.05
1.5	**2320.754**	78.1%	7.7%	13.57	4.58
2.0	**2320.704**	**96.9%**	7.7%	12.91	**2.98**

while generation times remain within a similar range across utility settings. Several utility-aware settings achieve higher validation scores than the coverage-only baseline, while generation times remain within a similar range across settings. At step 20, $\beta_U = 2.0$ produces shorter responses than the coverage-only baseline, reducing the average response length from 4.25k to 2.98k tokens. However, response length does not decrease monotonically with β_U. These observations suggest that utility-aware selection can improve validation scores under a fixed nominal search budget.

♣ **Effect of Utility-Aware Credit Allocation.** As shown in Figure 4, increasing β_U raises the mean selected-branch utility estimate from 0.0334 at $\beta_U = 0$ to 0.0762 at $\beta_U = 1$, with little further increase at larger weights. This indicates that utility weighting changes selection toward branches with greater estimated outcome sensitivity. However, higher utility estimates do not translate into higher mean rollout reward: utility-weighted runs produce shorter responses but lower rewards than the utility-free configuration. These single-seed diagnostics therefore reveal a selection–performance distinction, rather than establish agentic superiority over random branching.

4.3 Sensitivity to Utility Weight.

We examine sensitivity to the utility coefficient $\beta_U \in \{0, 0.5, 1.0, 1.5, 2.0\}$ under a fixed nominal search budget of $Q_{\text{search}} = 14$. As shown in Figure 4, comparisons are restricted to the first 29 training steps shared by all runs. Increasing β_U does not yield a consistently ordered improvement in rollout reward; instead, the settings exhibit different completion and generation-time profiles. This suggests that stronger utility weighting should not be assumed to improve all training outcomes simultaneously. These single-seed diagnostics characterize early-training sensitivity rather than establish a universally optimal coefficient or robustness across runs.

4.4 Case Study

As shown in Figure 5, answering the question requires interpreting yesterday relative to the session date, 22 December 2022. The model returns 21 December 2022, matching the reference answer despite the absence of an explicit donation date in the source utterance. This example illustrates successful use of temporal context; however, the recorded tool sequence alone does not establish whether the date was resolved during memory construction or answer generation.

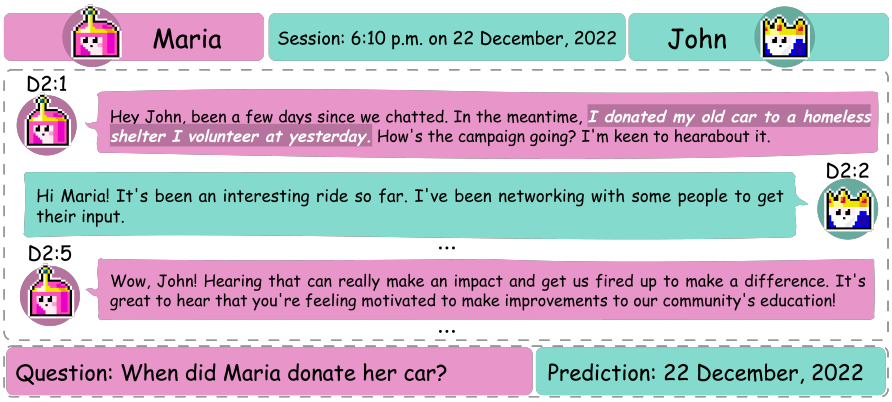

Figure 5: Example of resolving a relative temporal expression using the session date.

5 Conclusion

In this work, we study credit assignment in long-horizon agents as a structured scaling problem. We introduce **Credit Assignment Geometry**, characterized by ***Q***uantity, ***H***eterogeneity, and ***U***tility. Increasing credit yields diminishing returns, while performance under a fixed budget depends strongly on its allocation. Accordingly, we develop a utility-aware branching strategy that assigns more credit to high-impact decisions and consistently outperforms random allocation. **Limitations** Our study focuses on the scaling and allocation principles of credit assignment in long-horizon memory-agent training. Although the experiments show consistent trends across representative settings, broader evaluation across agent architectures, task domains, and credit-assignment mechanisms is needed to further assess generality. Future work may also extend our formulation beyond quantity, coverage, and utility to capture richer properties of supervision.

LLM USAGE DISCLOSURE

GPT-5.6 was used during the preparation of this manuscript as a supporting tool for editorial refinement and limited technical assistance. Its use included improving grammar, wording, sentence flow, and overall clarity of author-drafted text, as well as providing occasional support for routine coding-related tasks. The research questions, conceptual development, methodology, experimental work, data analysis, interpretation, and scientific conclusions were established and carried out by the authors. Any suggestions or outputs generated with GPT-5.6 were critically reviewed and, where appropriate, revised or verified before inclusion. The model did not independently determine the scientific content or direction of the study, and the authors retain full responsibility for the accuracy, validity, and presentation of the manuscript.

REPRODUCIBILITY STATEMENT

We provide comprehensive details to facilitate the reproduction and verification of our results. The implementation, training, and evaluation protocols are described in Appendix A, including dataset construction and splits, memory representation and retrieval, tree-search and policy optimization, credit-allocation experiments, supervised fine-tuning, and inference and evaluation settings. We report the key hyperparameters, search configurations, random seeds, and evaluation metrics used in our experiments. Code and scripts for reproducing the reported results are included in the supplementary materials.

ETHICS STATEMENT

This work investigates credit assignment for long-horizon memory agents in a controlled research setting using established benchmark datasets and released pretrained models. We do not collect new human-subject data or deploy the system on private user interactions. We acknowledge that memory-enabled agents may raise privacy and data-retention concerns in real-world deployment, and our experiments are restricted to benchmark settings intended for research and reproducible evaluation.

REFERENCES

Matthew Bull, Po-Chen Kuo, Andrew Smith, and Michael Buice. Volume transmission implements context factorization to target online credit assignment and enable compositional generalization. *Advances in Neural Information Processing Systems*, 38:148500–148547, 2026.

Guoxin Chen, Zile Qiao, Xuanzhong Chen, Donglei Yu, Haotian Xu, Xin Zhao, Ruihua Song, Wenbiao Yin, Huifeng Yin, Liwen Zhang, et al. Iterresearch: Rethinking long-horizon agents with interaction scaling. In *International Conference on Learning Representations*, volume 2026, pp. 30353–30383, 2026a.

Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. M3-embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation, 2025. URL https://arxiv.org/abs/2402.03216.

Wen-Tse Chen, Jiayu Chen, Fahim Tajwar, Hao Zhu, Xintong Duan, Ruslan Salakhutdinov, and Jeff Schneider. Retrospective in-context learning for temporal credit assignment with large language models. *Advances in Neural Information Processing Systems*, 38:71973–71998, 2026b.

Egor Cherepanov, Nikita Kachaev, Artem Zholus, Alexey Kovalev, and Aleksandr Panov. Unraveling the complexity of memory in rl agents: an approach for classification and evaluation. In *International Conference on Learning Representations*, volume 2026, pp. 42526–42545, 2026.

Sanjiban Choudhury. Process reward models for llm agents: Practical framework and directions. *arXiv preprint arXiv:2502.10325*, 2025.

Gaodan Fang, Vatche Isahagian, KR Jayaram, Ritesh Kumar, Vinod Muthusamy, Punleuk Oum, and Gegi Thomas. Trajectory-informed memory generation for self-improving agent systems. *arXiv preprint arXiv:2603.10600*, 2026.

Minghong Geng, Shubham Pateria, Budhitama Subagdja, and Ah-Hwee Tan. Scaling up multi-agent reinforcement learning for large agent teams and long-horizon tasks: A survey. *ACM Computing Surveys*, 58(14):1–37, 2026.

Gunshi Gupta, Karmesh Yadav, Zsolt Kira, Yarin Gal, and Rahaf Aljundi. Memo: Training memory-efficient embodied agents with reinforcement learning. *Advances in Neural Information Processing Systems*, 38:104392–104414, 2026.

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

Tenghao Huang, Kinjal Basu, Ibrahim Abdelaziz, Pavan Kapanipathi, Jonathan May, and Muhao Chen. R2d2: Remembering, replaying and dynamic decision making with a reflective agentic memory. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 30318–30330, 2025.

Yuxiang Ji, Ziyu Ma, Yong Wang, Guanhua Chen, Xiangxiang Chu, and Liaoni Wu. Tree search for llm agent reinforcement learning. In *International Conference on Learning Representations*, volume 2026, pp. 87362–87388, 2026.

Amirhossein Kazemnejad, Milad Aghajohari, Eva Portelance, Alessandro Sordoni, Siva Reddy, Aaron Courville, and Nicolas Le Roux. Vineppo: Refining credit assignment in rl training of llms. *arXiv preprint arXiv:2410.01679*, 2024.

Tomáš Kočiský, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. The narrativeqa reading comprehension challenge, 2017. URL https://arxiv.org/abs/1712.07040.

Daixun Li, Yusi Zhang, Mingxiang Cao, Donglai Liu, Weiying Xie, Tianlin Hui, Lunkai Lin, Zhiqiang Xie, and Yunsong Li. Towards long-horizon vision-language-action system: Reasoning, acting and memory. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 6839–6848. IEEE, 2025.

Jiazheng Li, Yawei Wang, Qiaojing Yan, Yijun Tian, Zhichao Xu, Huan Song, Panpan Xu, and Lin Lee Cheong. Salt: Step-level advantage assignment for long-horizon agents via trajectory graph. In *Findings of the Association for Computational Linguistics: EACL 2026*, pp. 4709–4725, 2026a.

Ruoran Li, Xinghua Zhang, Haiyang Yu, Shitong Duan, Xiang Li, Wenxin Xiang, Chonghua Liao, Xudong Guo, Yongbin Li, and Jinli Suo. Mempo: Self-memory policy optimization for long-horizon agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 23286–23301, 2026b.

Zaijing Li, Yuquan Xie, Rui Shao, Gongwei Chen, Dongmei Jiang, and Liqiang Nie. Optimus-1: Hybrid multimodal memory empowered agents excel in long-horizon tasks. *Advances in neural information processing systems*, 37:49881–49913, 2024.

Yuze Liu, Chaochao Lu, and Chao Yang. Shadow: Dynamic-aware credit assignment against long-horizon tasks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 23935–23944, 2026.

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, 2026.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of LLM agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13851–13870, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.747. URL https://aclanthology.org/2024.acl-long.747/.

Marina Mao, Alexandr Liu, Pengbo Li, Siheng Li, Bo Zhou, and Xiang Wang. Tree-based credit assignment for multi-agent memory system. *arXiv preprint arXiv:2605.04811*, 2026.

Kevin Murphy. Model discovery agent: Llm-assisted bayesian experiment design for data-efficient discovery of mechanistic world models. *arXiv preprint arXiv:2608.09696*, 2026.

Eduardo Pignatelli, Johan Ferret, Matthieu Geist, Thomas Mesnard, Hado van Hasselt, Olivier Pietquin, and Laura Toni. A survey of temporal credit assignment in deep reinforcement learning. *arXiv preprint arXiv:2312.01072*, 2023.

Yaorui Shi, Yuxin Chen, Siyuan Wang, Sihang Li, Hengxing Cai, Qi Gu, Xiang Wang, and An Zhang. Look back to reason forward: Revisitable memory for long-context llm agents. In *International Conference on Learning Representations*, volume 2026, pp. 78055–78082, 2026.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Weiwei Sun, Miao Lu, Zhan Ling, Kang Liu, Xuesong Yao, Yiming Yang, and Jiecao Chen. Scaling long-horizon llm agent via context-folding. *arXiv preprint arXiv:2510.11967*, 2025.

Leitian Tao, Baolin Peng, Wenlin Yao, Tao Ge, Hao Cheng, Mike Hang Wang, Jianfeng Gao, and Sharon Li. Trace: Turn-level reward assignment via credit estimation for long-horizon agents. *arXiv preprint arXiv:2607.13988*, 2026.

Jichao Wang, Liuyang Bian, Yufeng Zhou, Han Xiao, Yue Pan, Guozhi Wang, Hao Wang, Zhaoxiong Wang, Yafei Wen, Xiaoxin Chen, et al. Solar-rl: Semi-online long-horizon assignment reinforcement learning. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 24419–24432, 2026a.

Xinglin Wang, Yiwei Li, Shaoxiong Feng, Peiwen Yuan, Yueqi Zhang, Jiayi Shi, Chuyi Tan, Boyuan Pan, and Yao Hu. Every rollout counts: Optimal resource allocation for efficient test-time scaling. *Advances in Neural Information Processing Systems*, 38:102312–102338, 2026b.

Yaxuan Wang, Yuhao Liu, Quan Liu, Jinlong Pang, Wei Wei, Yujia Bao, and Yang Liu. Dragon: Guard llm unlearning in context via negative detection and reasoning. In *International Conference on Learning Representations*, volume 2026, pp. 145078–145126, 2026c.

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

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. Longmemeval: Benchmarking chat assistants on long-term interactive memory. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=pZiyCaVuti.

Derong Xu, Shuochen Liu, Pengfei Luo, Pengyue Jia, Yingyi Zhang, Yi Wen, Yimin Deng, Wenlin Zhang, Enhong Chen, Xiangyu Zhao, et al. Learning how and what to memorize: Cognition-inspired two-stage optimization for evolving memory. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 44987–45011, 2026.

Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Hinrich Schütze, Volker Tresp, and Yunpu Ma. Memory-R1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. *arXiv preprint arXiv:2508.19828*, 2025.

Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Jinhe Bi, Kristian Kersting, Jeff Z Pan, et al. 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, 2026.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Chang Yang, Chuang Zhou, Yilin Xiao, Su Dong, Luyao Zhuang, Yujing Zhang, Zhu Wang, Zijin Hong, Zheng Yuan, Zhishang Xiang, et al. Graph-based agent memory: Taxonomy, techniques, and applications. *arXiv preprint arXiv:2602.05665*, 2026a.

Haojin Yang, Ai Jian, Yiwei Wang, Xinyue Huang, Weipeng Zhang, Ke Zeng, Xunliang Cai, and Jingqing Ruan. Harmonizing dense and sparse signals in multi-turn rl: Dual-horizon credit assignment for industrial sales agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (ACL 2026)*, pp. 1067–1076, 2026b.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering, 2018. URL https://arxiv.org/abs/1809.09600.

Hongli Yu, Tinghong Chen, Jiangtao Feng, Jiangjie Chen, Weinan Dai, Qiying Yu, Ya-Qin Zhang, Wei-Ying Ma, Jingjing Liu, Mingxuan Wang, et al. Memagent: Reshaping long-context llm with multi-conv rl-based memory agent. In *International Conference on Learning Representations*, volume 2026, pp. 39458–39486, 2026.

Huining Yuan, Zelai Xu, Zheyue Tan, Xiangmin Yi, Mo Guang, Kaiwen Long, Haojia Hui, Boxun Li, Xinlei Chen, Bo Zhao, et al. Marshal: Incentivizing multi-agent reasoning via self-play with strategic llms. In *International Conference on Learning Representations*, volume 2026, pp. 6209–6236, 2026.

Zhiyuan Zhai, Bingcong Li, Bingnan Xiao, Ming Li, and Xin Wang. Adaptive test-time compute allocation for reasoning llms via constrained policy optimization. *arXiv preprint arXiv:2604.14853*, 2026.

Shengtao Zhang, Jiaqian Wang, Ruiwen Zhou, Junwei Liao, Yuchen Feng, Zhuo Li, Yujie Zheng, Weinan Zhang, Ying Wen, Zhiyu Li, et al. Memrl: Self-evolving agents via runtime reinforcement learning on episodic memory. *arXiv preprint arXiv:2601.03192*, 2026a.

Yudi Zhang, Yali Du, Biwei Huang, Ziyan Wang, Jun Wang, Meng Fang, and Mykola Pechenizkiy. Interpretable reward redistribution in reinforcement learning: A causal approach. *Advances in Neural Information Processing Systems*, 36:20208–20229, 2023.

Yuxiang Zhang, Jiangming Shu, Ye Ma, Xueyuan Lin, Shangxi Wu, and Jitao Sang. Memory as action: Autonomous context curation for long-horizon agentic tasks. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 19149–19164, 2026b.

Zijing Zhang, Ziyang Chen, Mingxiao Li, Zhaopeng Tu, and Xiaolong Li. Rlvmr: Reinforcement learning with verifiable meta-reasoning rewards for robust long-horizon agents. In *International Conference on Learning Representations*, volume 2026, pp. 114676–114700, 2026c.

Jihao Zhao, Ding Chen, Zhaoxin Fan, Kerun Xu, Mengting Hu, Bo Tang, Feiyu Xiong, and Zhiyu Li. Inside out: Evolving user-centric core memory trees for long-term personalized dialogue systems. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13429–13446, 2026a.

Xinyu Zhao, Junpeng Wang, Yuzhong Chen, Menghai Pan, Chin-Chia Michael Yeh, Jiarui Sun, Yan Zheng, Mahashweta Das, and Tianlong Chen. Demystify the role of memory in machine learning engineering agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 10811–10826, 2026b.

Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Bryan Kian Hsiang Low, and Paul Liang. Mem1: Learning to synergize memory and reasoning for efficient long-horizon agents. In *International Conference on Learning Representations*, volume 2026, pp. 58413–58438, 2026.

King Zhu, Hanhao Li, Siwei Wu, Tianshun Xing, Dehua Ma, Xiangru Tang, Minghao Liu, Jian Yang, Jiaheng Liu, Yuchen Eleanor Jiang, et al. Scaling test-time compute for llm agents. *arXiv preprint arXiv:2506.12928*, 2025.

Wei Zhu, Zhiwen Tang, and Kun Yue. Symphony: Synergistic multi-agent planning with heterogeneous language model assembly. *Advances in Neural Information Processing Systems*, 38: 57690–57724, 2026.

Zefang Zong, Dingwei Chen, Yang Li, Qi Yi, Bo Zhou, Chengming Li, Bo Qian, Peng Chen, and Jie Jiang. At^{2}po: Agentic turn-based policy optimization via tree search. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 24120–24143, 2026.

Heming Zou, Qi Wang, Yun Qu, Yuhang Jiang, Lizhou Cai, Yixiu Mao, Ru Peng, Xin Xu, Weijie Liu, Kai Yang, et al. Trace: A unified rollout budget allocation framework for efficient agentic reinforcement learning. *arXiv preprint arXiv:2606.11119*, 2026.

APPENDIX CONTENTS

Appendix A. More Implementation Details 16
Appendix B. More Results of Exp-A 20
Appendix C. More Results of Exp-B 22
Appendix D. More Results of Exp-C 29
Appendix E. Qualitative Case Studies 30

A More Implementation Details

A.1 Implementation Organization and Data Preparation

❶ **Implementation scope.** The implementation separates memory-agent inference from policy optimization. The inference stack contains modules for memory formation, updating, retrieval, and trajectory logging. The training stack builds on the local Tree-GRPO/verl implementation, with separate workspaces for the budget, coverage, and selection experiments. We distinguish launcher configurations from completed experimental records: the full-run configuration described below is not the protocol used for the one-step coverage screen. ❷ **LoCoMo partition.** The versioned preprocessing pipeline partitions LoCoMo at the conversation level rather than randomly splitting questions from the same conversation. Conversation `conv-26` is used for training and `conv-30` for validation. The remaining eight conversations, `conv-41`, `conv-42`, `conv-43`, `conv-44`, `conv-47`, `conv-48`, `conv-49`, and `conv-50`, form the test split. Questions with category 5 or an `adversarial_answer` field are excluded. The saved manifest contains 152 training, 81 validation, and 1,307 test questions. The preprocessing script checks conversation identifiers against the expected source ordering and records SHA-256 hashes of the source and derived files. ❸ **Training records.** Each Parquet record contains a category-conditioned retrieval prompt, a sample identifier, the question, answer, category, and reward annotations. The latter include the reference answer and supporting evidence identifiers. The sample identifier is retained for routing retrieval requests to the appropriate memory collections. Prompt construction does not require running the language model. The full split and the smaller screening subsets are distinct data artifacts and should not be pooled.

A.2 Memory Representation and Retrieval Interface

❶ **Sample-scoped storage.** The inspected inference implementation uses Chroma and a BGE-M3 embedding function. It first looks for the local `models/bge-m3` checkpoint and otherwise falls back to its model identifier. Each sample has five named collections: turns, facts, experiences, personas, and summary. Collection names include the sample identifier to avoid retrieving another sample's memory. Memory and QA trajectories preserve the sample identifier and relevant operation metadata for subsequent auditing. ❷ **Construction and updates.** The default construction context groups four turns per batch. The inference pipeline instantiates formation, update, and retrieval modules with the same local model client. The configuration specifies retrieval of three related items for memory updates and five for question answering, with a maximum of six retrieval-tool steps. These are separate settings from generation-token limits. The code additionally supports a category-4 uncertainty retry with retrieval top-$k = 6$ and at most seven steps; its activation must be kept consistent when comparing inference conditions. ❸ **Reuse versus reconstruction.** With `from_scratch=true`, the builder deletes the sample's collections before creating them. With this flag disabled, the inspected builder checks for the expected turn-batch identifiers and skips sessions marked as built. Consequently, invoking a benchmark script is not necessarily a read-only evaluation: it still calls the builder before answering questions. The turn-batch check alone does not certify that all derived memories or all benchmark samples are complete.

❹ **RL retrieval service.** The reproduction launcher accesses a pre-existing persistent LoCoMo memory store through an HTTP retrieval service. Requests contain the tool name, arguments, sample identifier, and previously seen turn identifiers. Responses return the observation, memory metadata, and updated seen-turn information. The service executes retrieval tools rather than constructing a new database for every rollout. This distinction separates retrieval-policy optimization from offline memory-construction training.

A.3 Tree Rollouts and Policy Optimization

❶ **Nominal search budget.** The generator first produces m initial action chains. It then performs l expansion rounds, selecting n expandable nodes per tree in each round and generating a continuation from each selected node. The final sampling stage retains k leaves per tree. For the $k = 1$ configurations considered here, we report the nominal number of generated chain/continuation slots as

$$Q_{\text{search}} = m(1 + nl). \tag{10}$$

This quantity is not the number of unique decision nodes, generated tokens, or independent credit signals. Continuations can have different lengths and share prefixes; equal Q_{search} therefore does not imply equal runtime or realized credit density. ❷ **Node rewards.** The memory-tree reward implementation propagates terminal performance through the tree. For a terminal node, performance

is the score supplied by the leaf reward function; for an internal node, it is the arithmetic mean of its immediate children's performance values. This recursive average need not equal an unweighted average over all descendant leaves in an unbalanced tree. The node reward is implemented as

$$R(v) = I_{\text{fmt}}(v)\,[\alpha E(v) + P(v)]\,, \qquad \alpha = 0.5, \tag{11}$$

where I_{fmt} is a format-validity indicator. For nonterminal nodes, $E(v)$ is the fraction of reference evidence turn identifiers retrieved by that node; at terminal nodes, retrievals are accumulated along the path. Evidence reward is zero when reference evidence is empty. Reference evidence is used for training rewards, not by the coverage selector described below. ❸ **Dual-scale advantages.** The local memory-tree estimator combines within-tree and batch-global standardization of node rewards:

$$A(v) = \frac{R(v) - \mu_{t(v)}}{\sigma_{t(v)} + \epsilon} + \frac{R(v) - \mu_{\mathcal{B}}}{\sigma_{\mathcal{B}} + \epsilon}, \qquad \epsilon = 10^{-6}. \tag{12}$$

The inspected routine uses population standard deviations over the supplied node-reward lists, with mean zero and scale one when a group has at most one entry. Shared-prefix rewards can occur more than once in these path-based lists. Advantages are assigned to response segments and masked before policy optimization. Accordingly, the global term should not be described as a verified per-question statistic without checking the actual batching scope. ❹ **Full-run launcher settings.** The reproduction launcher initializes the policy from the local Mem-T-4B checkpoint and uses eight GPUs on one node, Ray-based workers, FSDP, bfloat16, gradient checkpointing, and padding removal. Actor parameters, gradients, and optimizer states are configured for offloading; the reference model also uses parameter offloading. Rollouts use vLLM with tensor parallelism one, temperature 1.0, chunked prefill, and a GPU-memory utilization setting of 0.5. The launcher selects the XFormers attention backend. Table 6 lists the explicit full-run settings, not a claim that every experiment reached the requested number of updates.

Table 6: Explicit RL launcher settings and coverage-screen overrides.

Setting	Full-run launcher	Exp-B screen
Training / validation batch	32 / 32	16 / 16
Requested optimizer steps	200	1
Learning rate	5×10^{-6}	Same
Warmup ratio	0.285	Same
PPO mini-/micro-batch knobs	8 / 8	Same
Actor/reference log-prob. micro-batch	8 / 8	Same
KL coefficient	0.001	Same
KL estimator	`low_var_kl`	Same
Maximum prompt tokens	40,960	Same
Maximum response tokens	1,024	Same
Maximum initial-context tokens	15,360	Same
Maximum observation tokens	20,480	Same
Retrieval top-k / maximum turns	5 / 6	Same
Pre-training validation	Enabled	Disabled
Periodic save / test frequency	50 / 50	0 / 999999

The KL term is enabled in the actor loss and disabled as a reward penalty. The batch entries above are global configuration knobs, not per-device batch sizes; distributed normalization must be respected when changing GPU count. The full launcher also sets a 20-epoch limit, whereas the screen sets one epoch. The screening save/test values describe periodic scheduling and should not be interpreted as proof that no end-of-run validation or saving occurs.

A.4 Budget, Coverage, and Selection Experiments

❶ **Exp-A: budget configurations.** The inspected evaluation launcher enumerates $(m, n, l, k) = (m, 1, 1, 1)$ for $m \in \{1, 2, 3, 4, 5\}$ and $(4, n, 1, 1)$ for $n \in \{2, 3, 4\}$. Their nominal budgets are $2, 4, 6, 8, 10, 12, 16, 20$, respectively. These values describe the registered configurations; completed training and evaluation must be established from each run's artifacts rather than from its presence in the launcher. ❷ **Exp-B: coverage-preference selector.** For candidate node v, let $T(v)$ denote the union of retrieved turn IDs along its path, and let S denote the union for nodes already selected in the

current selection call. The selector computes

$$O(v, S) = \frac{|T(v) \cap S|}{\max(1, |T(v) \cup S|)}, \tag{13}$$

$$N(v, S) = \frac{|T(v) \setminus S|}{\max(1, |T(v) \cup S|)}, \tag{14}$$

$$C_h(v, S) = (1 - h)O(v, S) + hN(v, S). \tag{15}$$

The first node is a random anchor. Subsequent nodes maximize C_h, with random tie breaking. Selected nodes are removed from the remaining pool; the full candidate pool is reused when it is exhausted. Candidates include the root and nonterminal descendants. In contrast, random mode samples with replacement. Thus, zero nominal coverage in coverage mode is not identical to the random-mode baseline. Empty retrieved-ID sets and ties can weaken the effective intervention. The first-anchor rule also makes the coverage term inactive when only one node is selected per call. **Screening protocol and diagnostics.** The screen fixes $(m, n, l, k) = (3, 3, 4, 1)$, giving $Q_{\text{search}} = 39$, and tests $h \in \{0, 0.25, 0.5, 0.75, 1\}$ with seeds 13, 29, and 47. Each run uses 16 fixed training and 16 held-out validation questions and one GRPO update, with training-data shuffling disabled. The pilot is separate: it contains three settings and overlapping training/validation examples. Neither study establishes convergence-level performance. The logged `state_tokens/coverage` is $\sum$ `loss_mask`/$\sum$ `response_mask`, not measured credit entropy. Selection logs can record selected node IDs and path-overlap diagnostics, but nominal h alone does not establish matched realized coverage across runs. ❸ **Exp-C: implemented outcome-sensitivity proxy.** The inspected counterfactual selector estimates utility from already scored leaves. For each child branch, it averages the available descendant leaf scores to obtain V_i. With $b \geq 2$ scored branches, it computes

$$\widehat{U}(v) = \frac{2}{b(b-1)} \sum_{i<j} |V_i - V_j|. \tag{16}$$

With fewer than two scored branches, utility is zero and marked unavailable. After the initial random anchor, selection uses $C_h(v, S) + \beta_U \widehat{U}(v)$. This proxy reuses observed outcomes rather than requesting additional continuation rollouts, but its scoring work may still incur overhead. It is an outcome-sensitivity heuristic, not automatically an agentic judge or a causal counterfactual estimator. Equal nominal h does not ensure equal realized coverage after adding the utility term; comparisons should also match budget, observed training horizon, and evaluation protocol.

A.5 Offline Supervised Fine-Tuning

❶ **Trajectory conversion.** The inspected SFT preparation script reads eight historical LoCoMo memory trajectory shards and four LongMemEval-oracle shards. It converts their messages to a ShareGPT-compatible representation and excludes examples whose tokenized chat-template length exceeds 6,144 tokens. Source paths and hashes are written to a separate manifest. This preparation script does not itself select the highest-reward operations from the current RL run; that provenance must not be assumed from the intended training order. ❷ **SFT configuration.** The dedicated reproduction launcher uses LLaMA-Factory 0.9.3, the Qwen3 template, and full-parameter SFT initialized from Qwen3-4B-Instruct-2507, rather than automatically loading the preceding RL checkpoint. It configures eight processes, bfloat16, FlashAttention 2, and DeepSpeed ZeRO-3. The per-device batch size is two with two accumulation steps, giving an effective batch size of 32 across eight devices. Other explicit settings are learning rate 5×10^{-6}, cosine scheduling, warmup ratio 0.1, a 200-step limit, a 10% validation split, evaluation every 20 steps, and checkpoint saving every 50 steps. These settings describe the dedicated reproduction script, not the separate legacy 8B SFT script.

A.6 Inference and Evaluation

❶ **Backend and decoding.** The current Exp-A benchmark entry point explicitly instantiates the Transformers-based `LocalClient`; it is not a verified automatic fallback from vLLM. The client loads the supplied model in bfloat16, applies its chat template, enables the KV cache, and limits each generation to 1,024 new tokens. Although temperature 0.7 is passed, the call does not explicitly set `do_sample`; the checkpoint generation configuration must therefore be checked before claiming stochastic or greedy decoding. The separate vLLM client configures a 65,536-token model context and 1,024 output tokens, with temperature 0.7 and minimum probability 0.05. The context limit is not the maximum generated answer length. ❷ **Sharding and dataset variants.** The Exp-A launcher evaluates

checkpoints sequentially and uses eight independent single-GPU processes per checkpoint. Samples are assigned by strided indexing, $D_s = D[s :: 8]$, rather than eight-way tensor parallelism. The benchmark entry point uses the final eight LoCoMo conversations, `eval_400.json` for HotpotQA, and selectable LongMemEval variants. NarrativeQA supports either summaries or document text; the latter uses paragraph grouping with defaults of 32 paragraphs and 24,000 characters per session. These input variants are not interchangeable protocols. Dataset filenames alone do not certify the number of evaluated examples. ❸ **Answer metrics.** The local evaluator lowercases text, replaces punctuation with spaces, removes English articles, and normalizes whitespace. Token F1 is computed from multiset overlap between prediction and reference tokens. With multiple references, the maximum reference-wise F1 is used. The evaluator also implements unigram BLEU with clipped precision and a brevity penalty. Both-empty strings receive F1 one; a single empty string receives zero. These lexical metrics must not be presented as an official category-specific LoCoMo evaluation without establishing protocol equivalence. Optional LLM-judge outputs are separate from lexical scores and require their own judge-model and prompt specification. ❹ **Completion and reproducibility checks.** QA trajectories retain predictions, references, evidence, categories, and tool traces. Evaluation requires reconciling expected sample/question IDs with completed output records across shards. A nonempty database or a successful process launch does not establish benchmark completion. In particular, the audited historical HotpotQA stores do not currently provide verified complete benchmark coverage. Likewise, the screen's raw `val/test_score/locomo` aggregate is not identified as benchmark F1. The 15-run screen reports descriptive seed variation; pilot data, incomplete runs, and repeated metrics are not treated as independent generalization evidence.

B MORE RESULTS OF EXP-A

Budget response across datasets. The two sweeps also expose different response patterns. Along the tree-count path, LoCoMo rises from 59.07 to 59.36 between $m = 1$ and $m = 4$, then falls to 58.07 at $m = 5$. HotpotQA peaks earlier, at 75.97 with $m = 2$, before decreasing to 63.97 for $m \geq 3$. Along the expansion-width path, LoCoMo reaches 59.74 at $n = 4$, a gain of 0.39 percentage points over $n = 1$. HotpotQA instead favors $n = 2$ or $n = 3$ within this sweep, improving from 63.97 to 70.63, while every wider setting lowers LongMemEval accuracy from its 65.00 starting value to 55.00 or 60.00. **Additional search budget therefore has a task-dependent return.** These results make the choice of expansion structure central to budget selection and motivate treating credit allocation as a separate design question from increasing the nominal quantity of search.

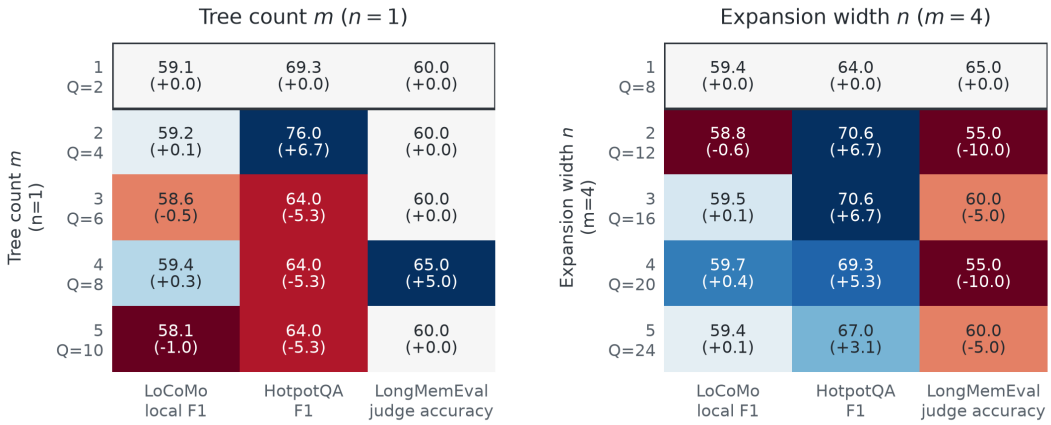

Figure 6: **Task-dependent responses to search-budget allocation.** The tree-count sweep fixes $n = 1$; the expansion-width sweep fixes $m = 4$. Cells report scores in percent and changes in percentage points relative to the outlined first row of the corresponding panel. Blue and red indicate increases and decreases. Color intensity is normalized separately within each dataset and panel; the printed values give absolute effect sizes. Each configuration uses the same 1,307 LoCoMo, 15 HotpotQA, and 20 LongMemEval questions. Metrics follow Table 3: recorded token F1 for LoCoMo and HotpotQA, and judge accuracy for LongMemEval.

Conversation-level variation localizes the aggregate change. Figure 7 reveals gains of 4.68 and 5.62 points on C48 and C50, alongside a 22.56-point decrease on C49. These conversations contain 25, 18, and 10 paired questions, respectively. The two largest groups, C41 and C44, account for 275 of the 386 questions and change by only -0.42 and -0.52 points. At the question level, 285 F1 scores remain unchanged, 53 increase, and 48 decrease. The small overall change therefore reflects localized gains and regressions, rather than a uniform shift across conversations.

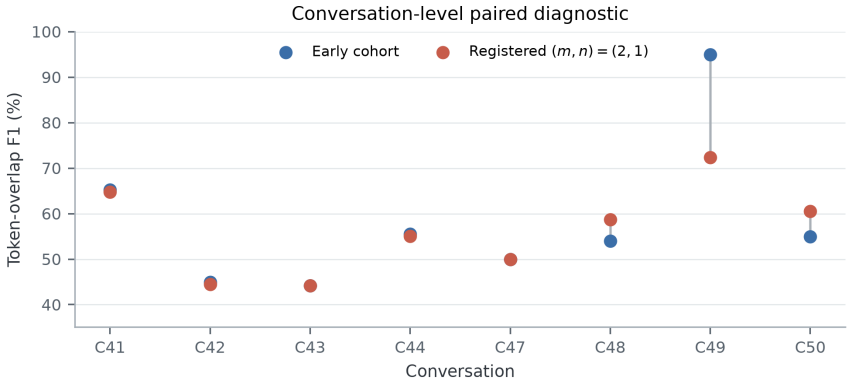

Figure 7: **Paired performance within eight conversations.** Each connector joins Early and Registered mean lexical F1 computed on the same questions within a conversation. Conversation sizes are 152, 23, 33, 123, 2, 25, 10, and 18, in plotted order. The overall paired score weights questions equally, so the conversations contribute in proportion to these sizes.

Longer interaction does not ensure better answers. Figure 8 shows that both historical cohorts are dominated by short traces: the median is two recorded steps, and two-step traces account for 80.57% and 81.18% of the Early and Registered records. Their mean lexical F1 is nearly identical, at 63.62 and 63.67. By comparison, seven-step traces score only 22.67 and 24.37, based on 50 and 143 questions. Longer traces have substantially lower mean answer quality in both cohorts.

Longer interactions often terminate with low lexical F1, motivating closer examination of continued retrieval and stopping decisions. This association describes observed trajectories; it does not establish that extra steps cause lower scores. Together, the budget and trajectory results favor evaluating the return from additional search at the task and question levels, rather than using a larger nominal budget or a longer trace as a proxy for improvement.

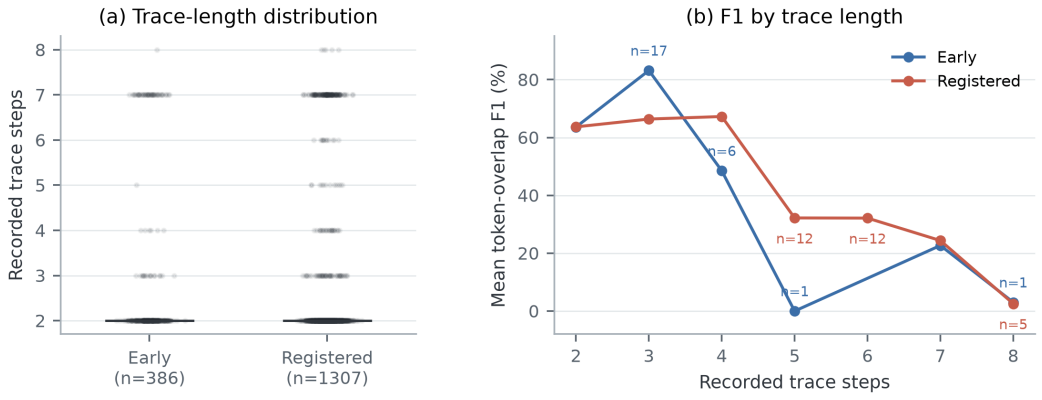

Figure 8: **Interaction length and answer quality in the historical cohorts.** Left: trace-length distributions over 386 Early and 1,307 Registered questions, with horizontally jittered observations. Right: mean lexical F1 conditional on recorded trace length, with sample counts annotated for groups smaller than 20 questions. Steps count saved QA trace entries, including finish or fallback, separately from Q_{search}. Curves compare length-conditioned groups, not successive stages of the same answers.

C MORE RESULTS OF EXP-B

Additional diagnostics of coverage allocation. Under a fixed nominal search budget ($Q_{\text{search}} = 39$), we examine how coverage settings relate to training outcomes, behavior, and computational cost. Figure 10 summarizes joint metric variation, while Figure 11 reveals seed-dependent contrasts against $H_{\text{cfg}} = 0$. Figure 12 presents descriptive associations between outcomes and logged diagnostics. For transparency, Figures 13 and 14 provide the complete optimization, reward, interaction, and runtime records. Together, these one-step results motivate further investigation of allocation alongside budget, but do not establish an optimal coverage setting or a coverage scaling law: realized credit coverage remains unverified, and nominally equal search budgets do not guarantee equal computational cost.

Coverage Preferences and Outcome Trade-offs. With $Q_{\text{search}} = 39$ fixed, different coverage settings exhibit different reward–completion trade-offs (Figure 9). $H_{\text{cfg}} = 0.5$ yields the highest mean rollout reward (0.1740), whereas $H_{\text{cfg}} = 1$ has relatively high reward but the lowest mean completion rate. These preliminary observations motivate studying allocation alongside budget; seed-dependent differences and unverified realized coverage preclude concluding an optimal H.

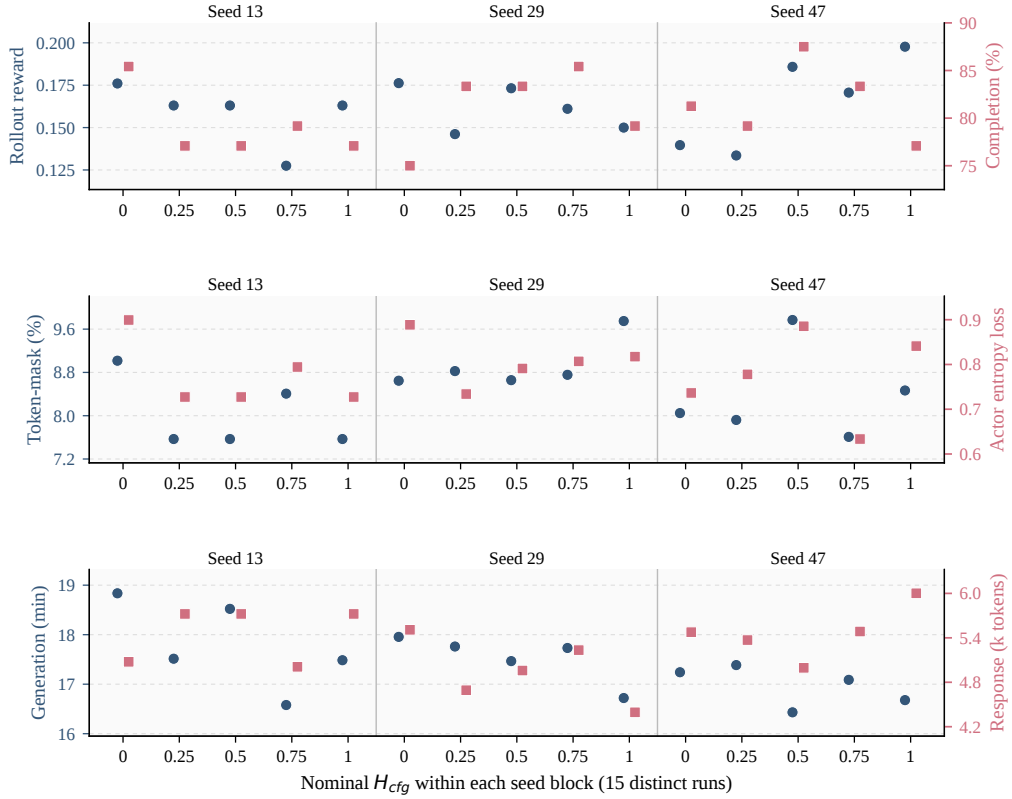

Figure 9: **Coverage preferences under a fixed nominal search budget.** Results from 15 one-step GRPO runs across five coverage settings and three seeds. Blue circles and pink squares use the left and right axes, respectively. Token-mask fraction is not a measurement of realized credit coverage.

Joint metric diagnostics. Figure 10 provides a joint view of training outcomes, agent behavior, optimization, and resource use. Standardizing each metric makes relative variation visible despite differences in units, without making the metrics interchangeable. The heterogeneous patterns encourage interpreting reward together with completion behavior and computational cost, rather than ranking configurations by a single metric or counting positive cells.

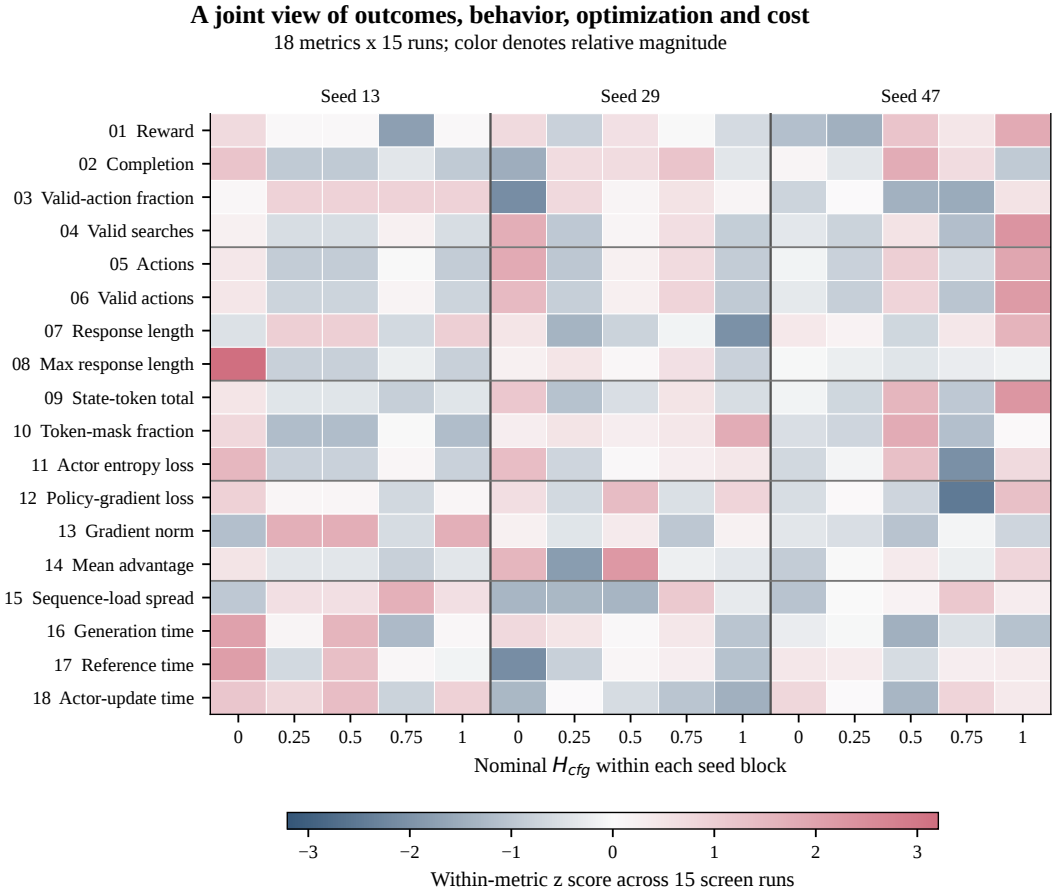

Figure 10: Joint diagnostics across nominal coverage settings. Rows represent 18 logged metrics and columns represent 15 screen runs, grouped by seed and ordered by H_{cfg}. Colors encode within-metric z-scores computed using the sample standard deviation across all runs. Positive values indicate larger measurements, not necessarily better performance; no composite score is constructed.

♣ **Seed-Matched Coverage Contrasts.** We investigate nominal coverage preferences while holding the search configuration fixed at $(m, n, l, k) = (3, 3, 4, 1)$. Each setting is evaluated with three seeds using 16 training questions and 16 held-out validation questions. Figure 11 reports seed-matched changes in training outcomes, behavior, optimization, and runtime. The observed differences are seed-dependent. Although $H_{\text{cfg}} = 0.5$ yields the highest mean rollout reward, its improvement over $H_{\text{cfg}} = 0$ is driven by one seed, while the other two show decreases. Thus, the apparent aggregate advantage does not establish a consistently optimal intermediate coverage setting. These results motivate further investigation of credit allocation, but do not yet establish a coverage scaling law: the nominal coverage parameter is not a measurement of realized credit coverage, and a fixed search budget does not guarantee equal computational cost.

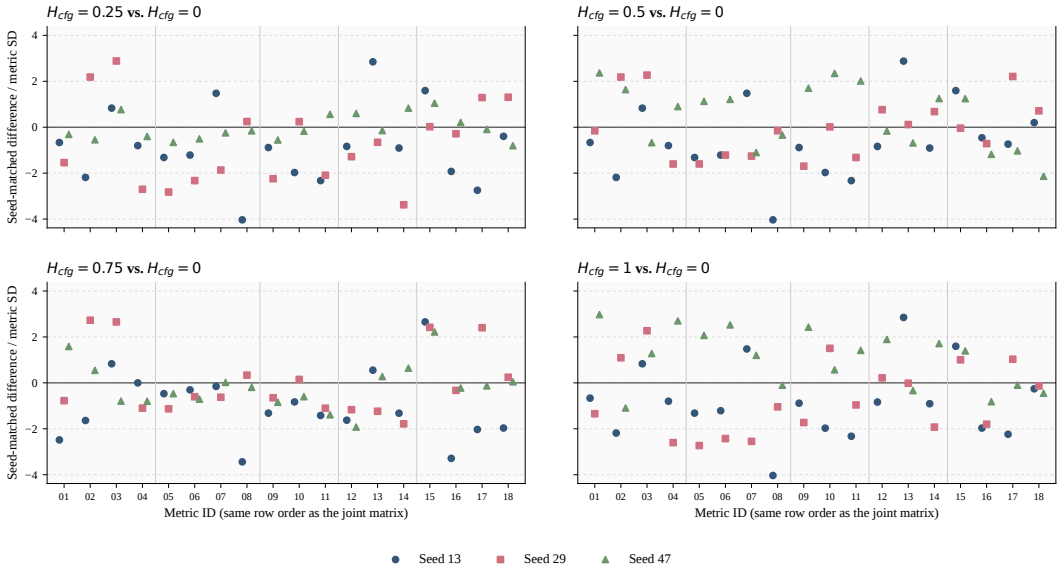

Figure 11: **Seed-matched contrasts across nominal coverage settings.** Under a fixed nominal search budget ($Q_{\text{search}} = 39$), each panel compares a coverage setting $H_{\text{cfg}} \in \{0.25, 0.5, 0.75, 1.0\}$ against $H_{\text{cfg}} = 0$ using the same seed. Differences are normalized by the corresponding metric's sample standard deviation across all 15 runs. Markers represent seeds 13, 29, and 47. Positive values indicate larger metric values, not necessarily better performance. These are one-step GRPO screening results, not converged benchmark evaluations.

Observed relationships. Figure 12 explores whether outcome variation accompanies changes in logged allocation diagnostics and runtime. The panels offer complementary views of the same observations, not additional independent experiments. Token-mask fraction is not a measurement of credit coverage, and actor entropy loss is not credit-allocation entropy. Accordingly, these associations cannot establish an optimal H or the causal benefit of utility-aware branching. Generation time also captures only part of computational cost.

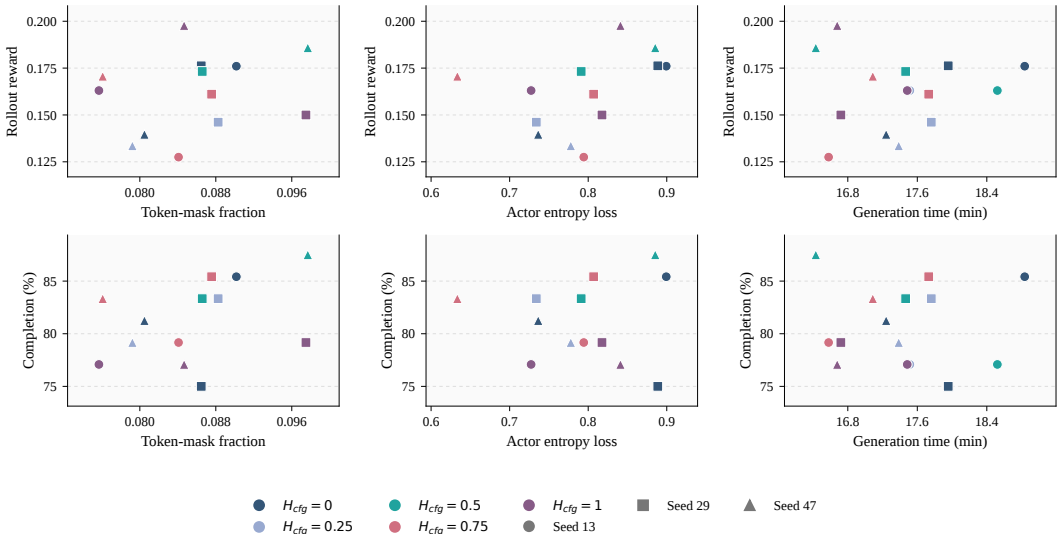

Figure 12: **Observed relationships between outcomes and diagnostics.** Rollout reward (top) and completion rate (bottom) are plotted against token-mask fraction, actor entropy loss, and generation time. Colors identify coverage settings and marker shapes identify seeds. Each panel displays the same 15 screen runs; overlapping points may obscure individual observations. The relationships are descriptive, with no fitted scaling law.

Optimization and advantages. Figure 13 documents entropy, gradient norms, KL-related quantities, learning rates, policy losses, and advantage statistics. These records support checks of logged optimization settings and unusual numerical behavior. Loss and advantage magnitudes should not be interpreted directly as task performance, and zero-valued diagnostics alone do not establish that no learning occurred. Repeated metric values are retained for transparency rather than treated as independent corroboration.

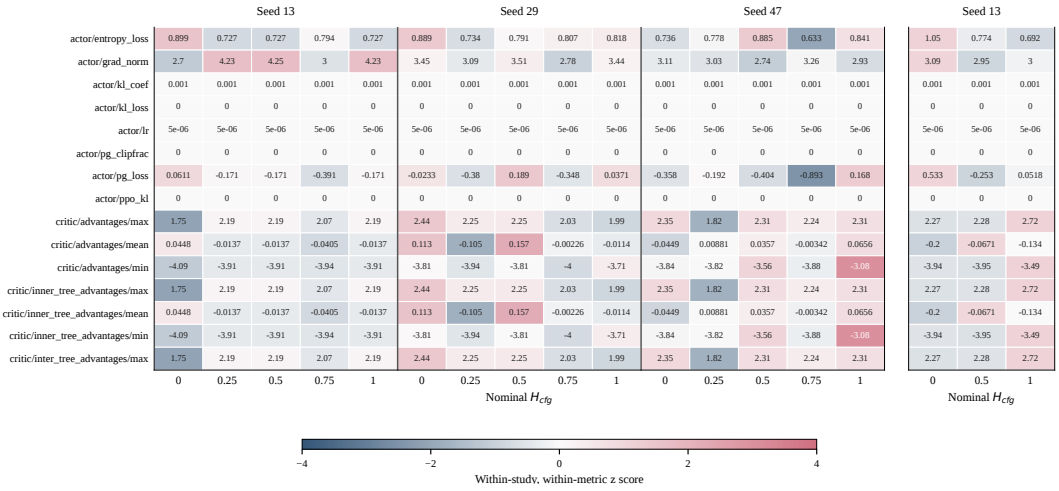

Figure 13: **Complete metric ledger I: optimization and advantages.** The first page reports actor optimization statistics and advantage summaries for the screen (left) and pilot (right). Cell labels show rounded raw values; colors standardize each metric separately within each study. Constant rows are neutral and missing entries are gray. The two studies are not pooled.

Runtime and validation records. Figure 14 documents resource variation despite the fixed nominal search configuration. Generation time, per-token timing, and total stage timings describe different aspects of cost and should not be conflated. The intermediate setting has a higher mean raw validation aggregate, but its metric definition and identical values across seeds require verification before interpreting this as a generalization gain. Pilot validation overlaps training and therefore cannot provide independent held-out evidence.

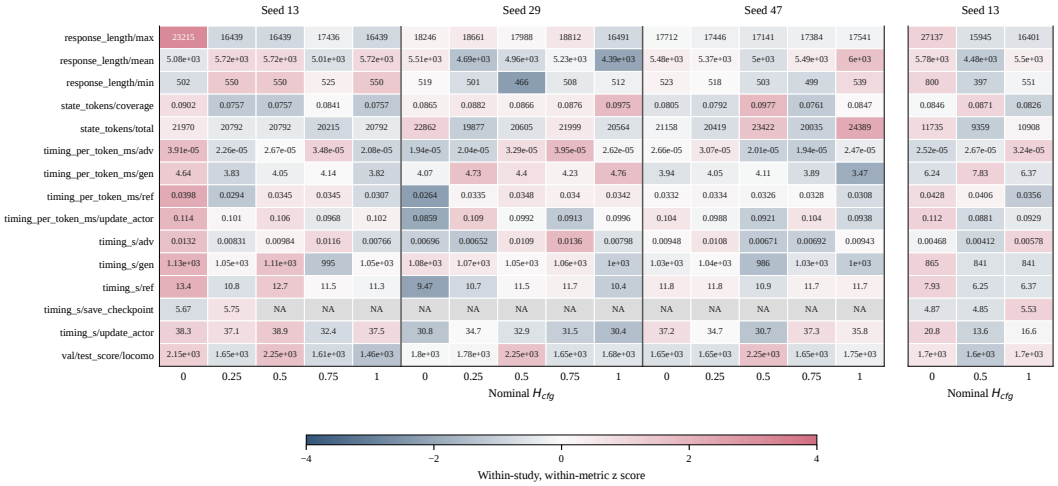

Figure 14: **Complete metric ledger IV: runtime and validation records.** The final page reports response lengths, state-token diagnostics, stage-level and per-token timings, checkpoint time, and the raw `val/test_score/locomo` aggregate. Labels show raw values; colors use within-study standardization. Missing timings are not zeros, and the validation aggregate is not identified as benchmark F1 or accuracy.

D MORE RESULTS OF EXP-C

Experimental protocol. Exp-C investigates utility-aware branch allocation using Mem-T-4B as the base policy on LoCoMo. We fix $(m, n, l, k) = (2, 2, 3, 1)$, corresponding to two initial trees per prompt, two expansion nodes per tree in each round, three expansion rounds, and one retained sample per tree. The nominal search budget per prompt is therefore $Q_{\text{search}} = m(1 + nl) = 14$. We set the coverage-preference parameter to $h = H_{\text{cfg}} = 0.5$ and the selection seed to 42, while varying $\beta_U \in \{0, 0.5, 1.0, 1.5, 2.0\}$. Because the runs have different recorded lengths, step-level diagnostics are compared over their common training window, steps 1–29.

Utility estimation. For an expandable node v, let $V_i(v)$ denote the mean reward of the scored descendant leaves under its i-th child branch. Given $b_v \geq 2$ scored child branches, utility is estimated as

$$\widehat{U}(v) = \frac{2}{b_v(b_v - 1)} \sum_{i<j} |V_i(v) - V_j(v)|. \tag{17}$$

If fewer than two scored branches are available, $\widehat{U}(v) = 0$ and the utility is marked as not ready. This quantity measures observed downstream outcome dispersion and serves as a practical proxy for decision utility. Newly generated leaves are scored between expansion rounds, allowing their outcomes to guide subsequent selections without increasing the rollout count; this introduces additional reward-evaluation overhead.

Branch-selection rule. Let $\mathcal{T}(v)$ contain the retrieved memory-turn identifiers along the path to v, and let $\mathcal{S}$ contain identifiers covered by nodes already selected in the current selection call. We compute

$$O(v, \mathcal{S}) = \frac{|\mathcal{T}(v) \cap \mathcal{S}|}{\max(1, |\mathcal{T}(v) \cup \mathcal{S}|)}, \tag{18}$$

$$N(v, \mathcal{S}) = \frac{|\mathcal{T}(v) \setminus \mathcal{S}|}{\max(1, |\mathcal{T}(v) \cup \mathcal{S}|)}, \tag{19}$$

$$C_h(v, \mathcal{S}) = (1 - h)O(v, \mathcal{S}) + hN(v, \mathcal{S}), \tag{20}$$

$$S_U(v, \mathcal{S}) = C_h(v, \mathcal{S}) + \beta_U \widehat{U}(v). \tag{21}$$

The first node is sampled as a stochastic anchor; subsequent nodes maximize $S_U(v, \mathcal{S})$, with seeded random tie breaking. Thus, $\beta_U = 0$ corresponds to the coverage-only selector rather than the separate random-with-replacement baseline. Fixing H_{cfg} does not guarantee identical realized coverage across runs.

Training configuration. The launcher uses eight GPUs, a training batch size of 32, a learning rate of 5×10^{-6}, a warmup ratio of 0.285, and a KL coefficient of 0.001. Generation uses temperature 1, up to six interaction turns, and top-5 retrieval. The configured prompt and per-generation response limits are 40,960 and 1,024 tokens, respectively; the latter should not be confused with the logged length of the complete multi-turn response.

Observations and scope. Utility weighting changes the reward–completion–length trade-off rather than improving every metric simultaneously. The coverage-only control achieves the highest mean rollout reward, whereas $\beta_U = 2$ yields the highest completion rate. All positive utility weights produce shorter mean responses than the coverage-only control. These single-seed, early-training diagnostics characterize sensitivity to utility-guided allocation; they do not establish a universally optimal coefficient, convergence-level gains, or superiority over a budget-matched random baseline.

E QUALITATIVE CASE STUDIES

As shown in Figure 15 and Figure 16, we present two additional selected successful examples. Evidence excerpts come from the source conversations; tool sequences and answers come from recorded QA trajectories. Omissions are marked with [...]. These examples illustrate agent behavior rather than establish aggregate superiority.

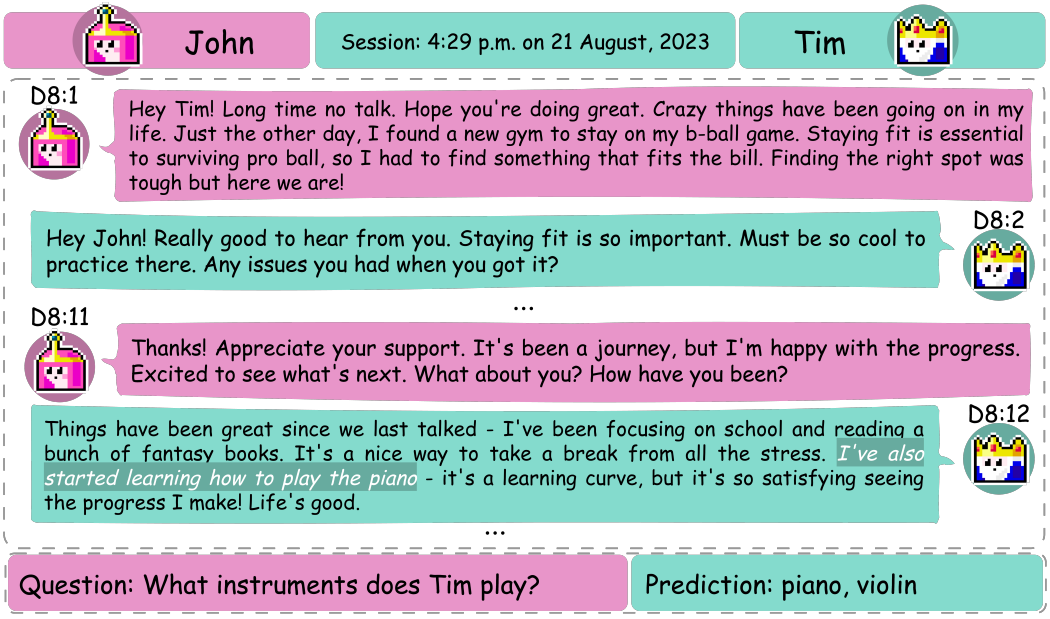

Figure 15: Example of resolving a relative temporal expression using the session date.

As shown in Figure 15, this case evaluates the model's ability to aggregate complementary information distributed across multiple sessions. The evidence relevant to the question appears in two temporally distant conversations: Tim mentions learning the piano in Session 8 and later reports learning the violin in Session 21. Answering correctly therefore requires more than retrieving the most recent relevant fact; the model must preserve and combine information from both sessions. The recorded tool sequence shows repeated fact retrieval before producing the final response, suggesting that the model successfully searched across the available interaction history and integrated multiple pieces of evidence. Its answer, "piano, violin," exactly matches the reference answer, demonstrating successful cross-session information aggregation without overwriting the earlier piano-related fact with the more recent violin-related update. This example highlights the importance of cumulative memory for questions whose answers depend on complementary facts acquired at different points in time.

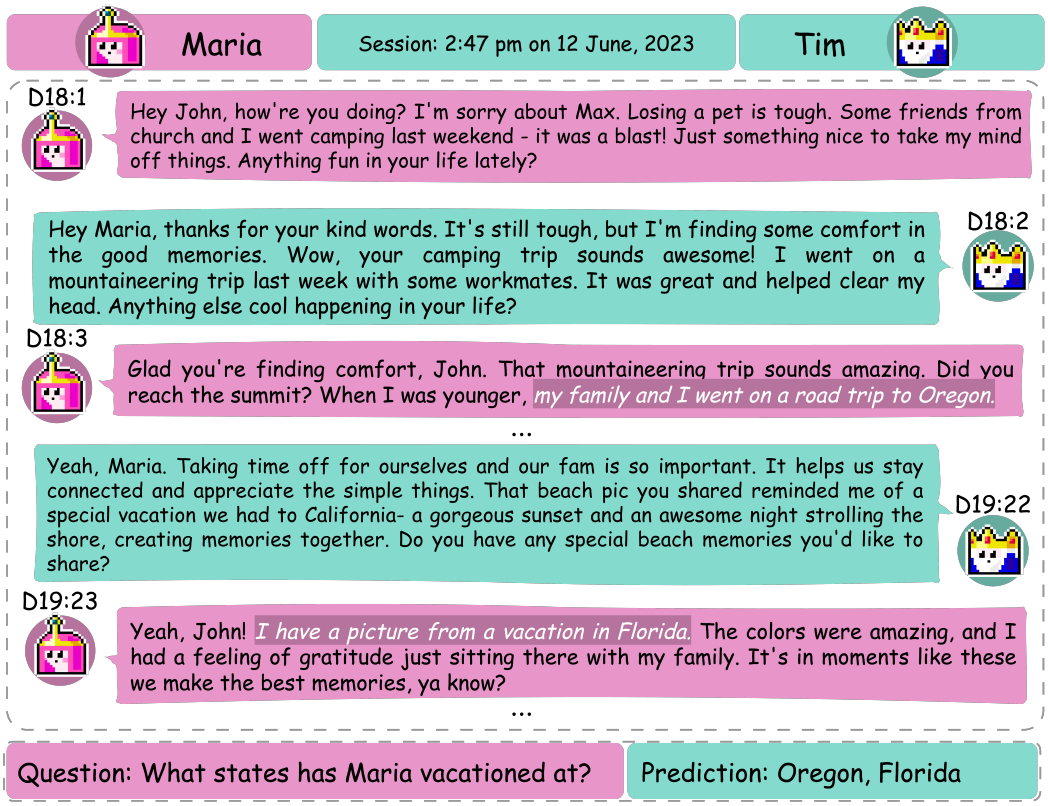

Figure 16: Example of resolving a relative temporal expression using the session date.

As shown in Figure 16, this case examines whether the model can aggregate multiple location entities mentioned across separate conversations. The relevant evidence is distributed over two sessions: Maria recalls a road trip to Oregon in Session 18 and mentions a vacation in Florida in Session 19. To answer the question completely, the model must retrieve both location facts and treat them as members of the same answer set, rather than selecting only one salient or more recent destination. The recorded tool sequence indicates that the model performs multiple fact searches before producing the final response. Its answer, "Oregon, Florida," exactly matches the reference answer, showing that it successfully combines geographically distinct pieces of information from different sessions into a complete set. This case demonstrates the model's ability to perform cross-session entity aggregation while avoiding partial-answer errors caused by retrieving only a single relevant location.