EVOGROUP: SELF-EVOLVING MEMORY HARNESS FOR SHARED-CONTEXT GROUP AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

With the rise of Group Agents such as Claude Tag, agents increasingly serve as digital workers for long-term team collaboration. These agents operate over shared contexts that continuously accumulate through multi-user interactions. However, existing agent memory systems are mainly designed for individual users and struggle to capture group-level knowledge in shared contexts, including group relationships and collaboratively formed knowledge. Manual memory strategies further limit adaptation to diverse and evolving shared contexts. To this end, we propose **EvoGroup (EvoG)**, a self-evolving memory harness for group agents that starts from a minimal five-tool cold start and adapts through interaction experience. EvoG uses confidence and task outcomes to identify informative trajectories, while low-confidence cases provide self-reflections on uncertainty. An analysis agent then examines selected trajectories with adaptive progressive disclosure and aggregates recurring patterns into high-level findings, which guide an evolve agent to revise the Harness. Empirically, after five Harness updates, pass@1 on EverMemBench improves from 71.07% to 89.58%, surpassing both self-evolving memory methods and human-designed harnesses, including Claude Code (85.75%). For DeepSeek-V4-Flash, the evolved Harness also reduces average per-question tool calls and answer time. To test generalization, we freeze the evolved Harness and directly evaluate it on the unseen GroupMemBench, showing effective generalization while outperforming self-evolving memory baselines and Claude Code.[1]

1 INTRODUCTION

AI agents are gradually evolving from conversational assistants designed for individual users into digital workers that directly participate in multi-user interactions and team collaboration (OpenAI, 2026b; Spataro, 2025; Anthropic, 2026). In sustained multi-user collaboration, shared contexts continuously accumulate and evolve through interactions among multiple users. This makes memory essential for agents to accurately extract and apply relevant information from long-term conversations. A Group Agent must not only understand current messages but also integrate discussion topics, key events, member perspectives, and group decisions from prior interactions. Its memory system must therefore effectively model shared contexts, including group relationships, discussion processes, and collaborative knowledge.

However, most existing agent memory systems are designed for single-user settings, organizing memory around individual experiences or user-assistant interaction histories (Park et al., 2023; Zhong et al., 2024; Packer et al., 2023). Such memory can largely be structured around the history, preferences, and task states of a single entity. In group settings, however, information is distributed across multiple users, topics, and reply threads, and the same event may be supplemented or reinterpreted by different members. As a result, memory representations designed for individual users struggle to capture group relationships and collaboratively formed knowledge (Hu et al., 2026; Yang et al., 2026). More fundamentally, existing memory systems typically predefine how memory is represented, updated, and retrieved (Chhikara et al., 2025; Liu et al., 2026). For example, interaction histories may be stored in vector databases (Chhikara et al., 2025) or organized as knowledge graphs (Anokhin et al., 2024), with fixed mechanisms for updating outdated or conflicting information (Fang et al.,

[1]The code for this manuscript is available at https://anonymous.4open.science/r/37883-8DC0.

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

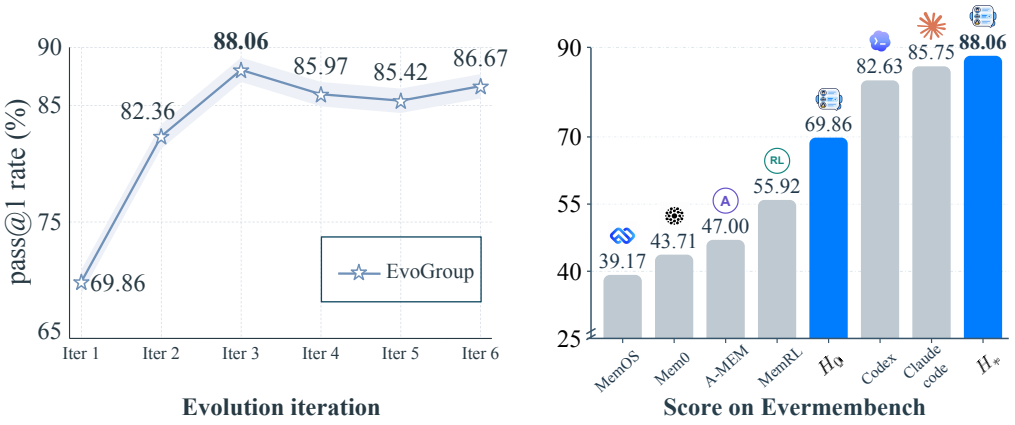


Figure 1: Without restructuring the original shared context and starting from a cold start with only five tools, EW self-evolves and, after seven iterations, surpasses all human-designed harnesses and self-evolving memory methods on EverMemBench.

2026). When group structures, shared contexts, or task requirements change, adapting these systems often requires additional rules, prompts, or retrieval modules. Such predefined designs limit their ability to autonomously adapt to diverse and evolving groups.

To address these challenges, we propose **EvoGroup (EvoG)**, a self-evolving harness framework for Group Agent memory. **EvoG** starts from a minimal modular Harness with only five tools and progressively evolves it through interaction experience. The evolution loop consists of four stages. **(1) Interaction.** The Group Agent uses the current Harness to answer queries over accumulated shared contexts, producing trajectories and confidence scores. For low-confidence queries, it additionally reflects on the source of its uncertainty without access to ground-truth answers. **(2) Trajectory Selection.** **EvoG** combines confidence with official correctness signals to identify informative trajectories, including confident-incorrect, unconfident-correct, and unconfident-incorrect cases. **(3) Analysis.** The analysis agent analyzes selected trajectories with adaptive progressive disclosure, accessing trajectory details only when needed. It attributes failures or uncertainty at the trajectory level, compresses detailed diagnoses into condensed rationales, and groups them by inferred query semantics to identify recurring patterns across trajectories. These patterns are further aggregated into high-level findings. **(4) Revision.** Finally, the evolve agent uses these findings to revise the Harness, producing H_{t+1} for the next round of interaction. Through this loop, **EvoG** enables the memory system to continuously adapt to the group and shared context from interaction experience.

We conduct extensive experiments to evaluate the effectiveness of **EvoG**. After six evolution iterations, the pass@1 of DeepSeek-V4-Flash on EverMemBench improves from 71.07% to 89.58%, outperforming the human-designed Harnesses Codex (82.63%) and Claude Code (85.75%), as well as the strongest self-evolving memory baseline (55.92%). Across two additional backbone models, **EvoG** consistently improves pass@1 by 15.4–16.1 percentage points. To evaluate generalization beyond the evolution distribution, we freeze the evolved Harness and directly apply it to GroupMemBench without further adaptation, where it continues to outperform self-evolving memory baselines and Claude Code. We further conduct component- and process-level ablations and analyze capability transitions, confidence, self-reflection, progressive disclosure, and Harness revisions to better understand how **EvoG** evolves. Together with our findings, analyses, and open-source release, we hope **EvoG** inspires future efforts toward more capable and adaptive Group Agents.

2 EvoGroup

In this section, we propose **EvoGroup (EvoG)**, a self-evolving Harness framework for Group Agent Memory over shared contexts, as illustrated in Figure 2.

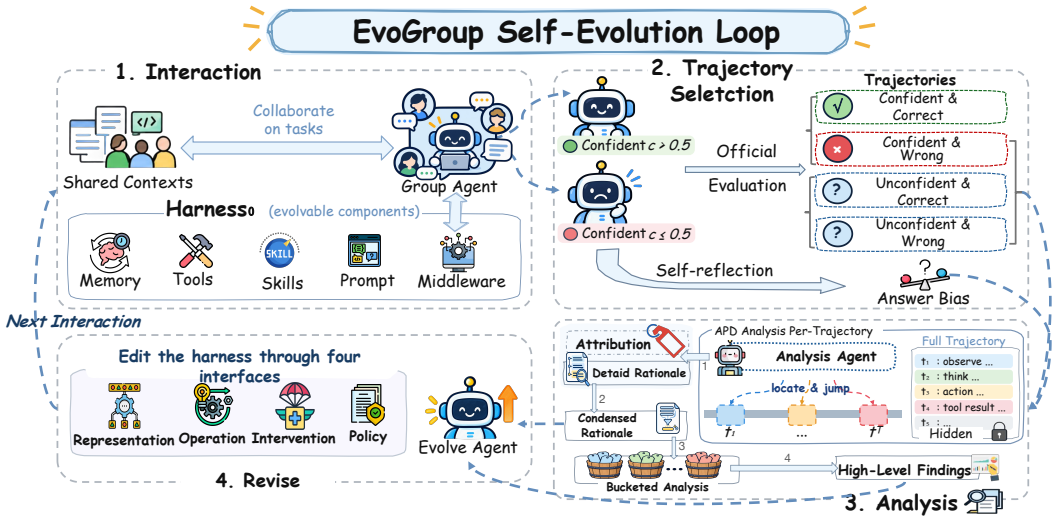


Figure 2: Overview of EvoGroup. It forms a self-evolution loop through four stages: interaction, trajectory selection, analysis, and harness revision.

2.1 Evolvable Harness over Shared Contexts

In multi-user shared contexts, fixed memory organization and usage strategies struggle to adapt to evolving group information and task requirements. Rather than updating the parameters of the underlying Agent, EvoGroup treats the surrounding *Harness* as the object of evolution.

Specifically, EvoGroup starts from a minimal modular Harness H_0. The initial Harness is equipped with only five basic tools: `list_files`, `read_file`, `grep_search`, `write_file`, and `create_file`, while its skill library and memory are both initialized as empty. From this cold start, EvoGroup repeatedly cycles through four stages—*interaction*, *trajectory selection*, *analysis*, and *revision*—allowing the Harness to progressively accumulate memories and skills from interaction experience and thereby producing a sequence of increasingly capable Harnesses:

$$H_0, H_1, \ldots, H_T. \tag{1}$$

At iteration t, the Group Agent interacts with a shared group context and solves collaborative tasks using the current harness H_t. For a query q_i, the interaction produces an answer a_i and a trajectory

$$\tau_i = \left(q_i, o_i^1, u_i^1, \ldots, o_i^{L_i}, u_i^{L_i}, a_i\right), \tag{2}$$

where o_i^ℓ denotes the information observed at step ℓ and u_i^ℓ denotes the corresponding reasoning, communication, or tool-use action. The trajectory therefore records not only the final response, but also the intermediate decisions through which the Group Agent coordinates its members and uses the components of H_t.

Along with its answer, the Group Agent reports a subjective confidence score $c_i \in [0, 1]$. We use a threshold γ (set to 0.5 in Figure 2) to distinguish confident from unconfident responses. If $c_i \leq \gamma$, EvoGroup asks the Group Agent to revisit its reasoning and explain the source of its uncertainty. This self-reflection is generated before any supervision is revealed: neither the ground-truth answer nor the official evaluation result is provided to the agent. Consequently, the reflection captures the agent's own diagnosis rather than a post-hoc explanation conditioned on the correct answer.

2.2 Confidence-Guided Experience Selection

During continuous interaction, trajectories differ in their value for evolution, and correctness alone cannot distinguish different types of success and failure. We therefore combine the Agent's confidence with official evaluation results to select informative trajectories, and apply self-reflection to low-confidence cases to obtain more targeted evolution signals.

After the interaction, an official evaluator supplies an external correctness signal $y_i \in \{0, 1\}$ for the answer a_i. Combining y_i with the confidence indicator $z_i = \mathbf{1}[c_i > \gamma]$ partitions the collected trajectories into four groups:

$$\begin{aligned} \mathcal{D}_{\mathrm{CC}} &= \{\tau_i \mid z_i = 1,\ y_i = 1\}, \quad \mathcal{D}_{\mathrm{CW}} = \{\tau_i \mid z_i = 1,\ y_i = 0\}, \\ \mathcal{D}_{\mathrm{UC}} &= \{\tau_i \mid z_i = 0,\ y_i = 1\}, \quad \mathcal{D}_{\mathrm{UW}} = \{\tau_i \mid z_i = 0,\ y_i = 0\}. \end{aligned} \tag{3}$$

The confident-and-correct subset $\mathcal{D}_{\mathrm{CC}}$ represents routine successful behavior and is not prioritized for diagnosis. EvoGroup instead selects

$$\mathcal{D}_{\mathrm{sel}} = \mathcal{D}_{\mathrm{CW}} \cup \mathcal{D}_{\mathrm{UC}} \cup \mathcal{D}_{\mathrm{UW}}, \tag{4}$$

which covers three complementary signals. Confident-but-wrong trajectories expose undetected reasoning failures or systematic answer bias; unconfident-but-correct trajectories reveal fragile successes that the current harness does not sufficiently support; and unconfident-and-wrong trajectories capture failures that the Group Agent can partially recognize. For the latter two categories, the associated self-reflections are retained as additional evidence for diagnosis.

2.3 Trajectory Analysis from Details to Patterns

Agent trajectories contain numerous tool calls, observations, and shared-context details, making it difficult to locate the evidence responsible for a failure. EvoGroup addresses this challenge at two complementary levels: it first uses *Adaptive Progressive Disclosure* (APD) to inspect and attribute individual trajectories on demand, and then uses bucketed analysis to aggregate recurring patterns across trajectories into high-level findings.

APD Analysis. Rather than loading an entire, potentially long interaction into context at once, APD initially keeps the full trajectory hidden and exposes only a compact structural view. Guided by the Group Agent's self-reflected *answer bias*, the Analysis Agent first locates a potentially problematic stage and then jumps to the corresponding local segment, requesting additional observations, actions, tool results, or communication records only when needed. This on-demand access focuses the analysis on diagnostically relevant evidence while reducing distraction from unrelated parts of the trajectory.

For each $\tau_i \in \mathcal{D}_{\mathrm{sel}}$, the Analysis Agent identifies the stage at which the behavior deviates from a reliable solution process, attributes the observed failure or uncertainty to its most likely cause, and produces a detailed rationale r_i^{det}. EvoGroup then compresses the detailed diagnosis into a condensed rationale:

$$r_i^{\mathrm{con}} = \mathrm{Compress}\big(r_i^{\mathrm{det}}\big)\,, \tag{5}$$

which retains the failure location, attributed cause, and actionable implication while removing instance-specific details that are unnecessary for cross-trajectory comparison.

Bucketed Analysis. To discover recurring patterns beyond individual cases, EvoGroup organizes condensed rationales according to the semantic types of their originating queries. These types are not predefined. Instead, the Analysis Agent infers a type label g_i from each query—for example, user profiling or temporal reasoning—and assigns analyses with the same inferred type to a shared bucket:

$$\mathcal{B}_g = \{r_i^{\mathrm{con}} \mid g_i = g\}. \tag{6}$$

The Analysis Agent first analyzes evidence within each bucket to identify type-specific failure patterns, and then compares the resulting patterns across buckets to derive high-level findings. The resulting condensed rationales and high-level findings F_t summarize recurring sources of error, uncertainty, and brittle success in the current Harness, and provide actionable evidence for subsequent Harness revision.

2.4 Multi-Interface Harness Revision

Trajectory analysis can reveal recurring problems, but these findings do not directly translate into effective Harness updates. We therefore use condensed rationales and high-level findings to revise the Harness at the representation, operation, intervention, and policy levels, enabling continuous Memory self-evolution from interaction experience.

Representation. This interface modifies how the Harness stores, associates, or presents information. Typical updates include changing the structure of memory units, relations between entities, the

organization of working memory, or the contextual structure supplied to the underlying model. **Operation.** This interface adds or modifies executable actions and their implementations, such as introducing a new query tool or revising a retrieval or selection handler. It determines what the Harness can execute and how each operation is implemented. **Intervention.** This interface introduces checks, transformations, or constraints at execution boundaries. Examples include validating arguments before a tool call, organizing or normalizing results after a call, and verifying an output before returning it as the final answer. **Policy.** This interface modifies the decision rules that determine what to select, when to execute it, and in what order operations should be performed. It covers updates to prompts, skills, retrieval parameters, search strategies, and the composition or scheduling of operations.

The interfaces are distinguished by the role of an update rather than by its surface form. For example, adding an executable reflection procedure is an operation update, whereas changing the rule that decides when reflection should be triggered is a policy update. A fixed validation step placed before or after execution instead constitutes an intervention. Based on this distinction, the Evolve Agent selects the interface or interfaces that directly address each diagnosed failure pattern. Abstractly, the revision step is

$$H_{t+1} = \mathrm{Evolve}(H_t, F_t; \mathcal{I}_t), \qquad \mathcal{I}_t \subseteq \{\textsc{Rep}, \textsc{Op}, \textsc{Int}, \textsc{Pol}\}, \tag{7}$$

where $\mathcal{I}_t$ denotes the selected update interface or interfaces. The revised harness H_{t+1} is then deployed in the next round of interactions, closing the self-evolution loop.

3 Experiments

3.1 Setup

Benchmark & Baseline Methods. We use multi-party conversation benchmarks to evaluate the memory capabilities of group-chat agents. EverMemBench contains 2,400 questions; we draw a stratified 720-question subset to drive evolution (Hu et al., 2026). To evaluate cross-benchmark transfer, we further evaluate the evolved EvoGroup Harness on GroupMemBench (Yang et al., 2026), which contains 745 questions and 120K dialogue turns. We compare the EvoGroup Harness with representative structured-memory and self-evolving memory baselines, as well as human-designed agent harnesses, including (1) FullContext, (2) Mem0 (Chhikara et al., 2025), (3) MemOS (Kang et al., 2025), (4) A-MEM (Xu et al., 2025), (5) MemRL (Zhang et al., 2026b), (6) Codex (OpenAI, 2025), and (7) Claude Code (Anthropic, 2025).

Experimental Details. For evolution efficiency, we stratify EverMemBench by question type and sample 30% of the questions as the held-in evolution set. We use the remaining 70% as a held-out validation set. We then freeze the evolved Harness and evaluate it on GroupMemBench to assess its cross-benchmark transfer ability.

Models. For the evolution iterations and frozen transfer experiments (Sections 3.2, the Group Agent, Analyzer Agent, and Evolve Agent use the same base model within each run. We evaluate three model configurations: GPT-5.5 (OpenAI, 2026a) with the non-thinking reasoning setting, DeepSeek-V4-Flash (DeepSeek-AI, 2026), and GLM-5.1 (GLM-5 Team et al., 2026).

Metric. We evaluate task effectiveness using **pass1**. Each task is attempted once per evaluation run and is considered successful if it passes the official verifier. The final score is the proportion of successfully completed tasks. Trials terminated by system-level failures, such as timeouts, crashes, or missing outputs, are counted as unsuccessful. Detailed experimental settings are provided in Appendix A.

3.2 Main Results

As shown in Table 1, we evaluate EvoGroup on GPT-5.5, DeepSeek-V4-Flash, and GLM-5.1. After five Harness updates, corresponding to six evaluated checkpoints over approximately 31 hours and 36 minutes, EvoGroup discovers the best Harness configuration, denoted by H_*. Compared with the initial configuration H_0, H_* improves the average pass rate by 16.2, 18.5, and 15.4 percentage points (pp) on the three models, respectively, demonstrating EvoGroup's ability to reliably discover substantially better configurations. Compared with the strongest Structured Memory or Self-Evolving

Table 1: Pass@1 (%) on EverMemBench by evaluation dimension and answer model. Average is the reported aggregate score. **Bold** marks the best result within each answer-model block, excluding FullContext.

Method	Fine-Grained Recall			Memory Awareness			Profile Understanding			Average
	Single	*Multi*	*Temp*	*Const*	*Proact*	*Update*	*Style*	*Skill*	*Role*	
GPT-5.5										
FullContext	97.65	4.02	18.33	92.04	84.78	100.00	47.73	63.91	65.82	66.42
Structured Memory										
Mem0	88.26	4.02	10.00	85.07	77.75	75.75	17.05	63.91	57.65	56.50
MemOS	71.36	2.01	5.00	68.66	62.30	66.04	19.32	**69.82**	70.92	49.25
Self-Evolving Memory										
A-MEM	90.61	4.02	21.67	89.05	88.29	75.75	27.84	57.40	65.82	61.71
MemRL	90.61	6.02	21.67	90.80	82.44	77.61	30.11	54.44	68.37	61.54
Human-Designed Harness										
Codex	93.90	**76.31**	75.33	96.27	**91.33**	97.39	**78.98**	68.05	**75.51**	**85.67**
Claude Code	93.43	74.70	73.67	95.77	90.40	96.64	77.84	65.68	74.49	84.58
EvoGroup										
H_0	95.30	8.05	51.90	90.43	88.29	95.74	36.59	45.76	62.04	68.33
H_*	**97.32**	56.90	**85.24**	**96.81**	90.97	**97.97**	74.80	64.41	72.99	84.52
DeepSeek-V4-Flash										
FullContext	95.31	6.02	11.67	50.25	49.41	100.00	19.32	18.34	23.47	43.54
Structured Memory										
Mem0	81.22	4.02	6.67	55.72	57.61	75.75	22.16	27.22	44.90	43.71
MemOS	64.32	4.02	13.33	47.26	58.78	66.04	15.34	18.34	39.29	39.17
Self-Evolving Memory										
A-MEM	88.26	4.02	18.33	63.43	56.44	77.61	32.95	27.22	34.18	47.00
MemRL	92.96	0.00	0.00	81.09	83.61	77.61	23.86	48.52	65.82	55.92
Human-Designed Harness										
Codex	95.31	81.93	83.33	93.78	90.63	92.16	77.84	50.30	47.45	82.63
Claude Code	97.62	**83.94**	86.62	**98.64**	92.92	93.56	84.78	42.42	60.53	85.75
EvoGroup										
H_0	97.99	28.16	31.90	89.36	**92.98**	92.55	67.48	51.69	61.31	71.07
H_*	**100.00**	78.16	**90.00**	97.52	90.64	**98.40**	**97.56**	**66.10**	**74.45**	**89.58**
GLM-5.1										
FullContext	–	–	–	–	–	–	–	–	–	–
Structured Memory										
Mem0	88.26	4.02	10.00	67.41	68.15	79.48	19.32	48.52	70.92	52.42
MemOS	73.71	6.02	3.33	58.21	57.61	86.57	13.64	48.52	65.82	47.04
Self-Evolving Memory										
A-MEM	88.26	4.02	18.33	77.11	83.61	79.48	34.66	42.60	**73.47**	58.75
MemRL	92.96	4.02	20.00	75.87	78.92	81.34	32.95	36.09	65.82	57.33
Human-Designed Harness										
Codex	95.31	**67.87**	63.33	**96.02**	81.26	88.81	80.68	41.42	63.27	77.88
Claude Code	95.77	55.82	65.00	93.28	87.12	**100.00**	85.23	20.12	50.00	76.46
EvoGroup										
H_0	94.63	24.71	28.10	86.88	89.97	89.36	60.98	56.78	58.39	68.27
H_*	**97.99**	66.09	**82.86**	90.07	**90.30**	81.38	**90.24**	**72.03**	70.80	**83.63**

Memory baseline for each backbone, EvoGroup further improves pass@1 by 22.81, 33.66, and 24.88 points. These results show that **EvoGroup consistently discovers substantially stronger memory Harnesses across LLM backbones with different capabilities.**

Generalization beyond the evolution benchmark. To evaluate whether the evolved Harness generalizes beyond the distribution used for evolution, we freeze the Harness evolved on EverMemBench and directly evaluate it on the unseen GroupMemBench without further adaptation. As shown in Table 8, on GPT-5.5, the evolved Harness achieves the highest average pass rate of 63.89%, outperforming the initial Harness by 2.01 points and the strongest human-designed Harness by 8.46 points. On DeepSeek-V4-Flash, it achieves 71.14%, remaining within 2.95 points of the initial Harness and 5.37 points of Codex, while outperforming the strongest memory baseline by 28.46 points. These results show that **the evolved Harness retains strong performance on an unseen benchmark without further adaptation, demonstrating generalization beyond the evolution distribution.**

3.3 Ablation Study

Component-level ablations. We evaluate the contribution of each Harness component by replacing one component of the initial Harness with its counterpart from the fully evolved Harness, while keeping all other components unchanged. We consider Prompt, Tools, Middleware, Memory, and Skills separately. As shown in Figure 3, Tools and Prompt provide the largest gains, improving accuracy by 10.86 and 7.38 percentage points, respectively. Middleware, Memory, and Skills yield smaller gains of 1.23, 1.43, and 0.82 points. Using all evolved components achieves 89.55% accuracy, an 18.44-point improvement over the initial Harness at 71.11%. These results show that EvoGroup improves the Harness through multiple components rather than prompt optimization alone.

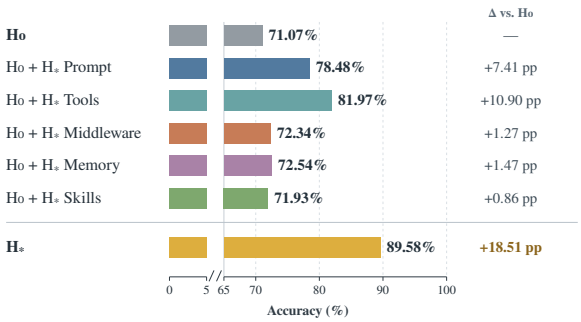


Figure 3: Harness-component ablation on EverMemBench. Each variant uses one evolved component, while H_* uses all five. Bars show accuracy and Δ indicates gains over H_0.

Process-level ablations. We further evaluate three mechanisms in the self-evolution loop over five iterations with DeepSeek-V4-Flash. All variants use the same initial Harness, evolution set, token budget, and held-out test split. As shown in Figure 4 and Table 2, removing trajectory selection slightly increases Pass@1 from 73.33% to 74.86%, with the lowest regression rate of 10.64% and the highest retention of 89.36%. This suggests that trajectory selection provides limited benefit in the current setting. Replacing progressive analysis with full-trace analysis reduces Pass@1 to 69.44% and increases the regression rate to 15.10%, showing the benefit of APD for focusing analysis on relevant evidence. Removing bucketed analysis causes the largest degradation. Although it produces 78 fail-to-pass transitions, it also introduces 96 pass-to-fail transitions, reducing Pass@1 to 67.08% and retention to 80.84%. These results suggest that cross-trajectory aggregation provides more stable evolution signals and helps reduce capability regressions.

Table 2: Ablation of the self-evolution process on the held-out test split, measured by capability gains, regressions, and retention.

	Transition Counts		Outcome Rates	
Self-Evolution Variant	**Fail→Pass**	**Pass→Fail**	**Regression**	**Retention**
w/o Trajectory Selection	**52**	**58**	**10.64%**	**89.36%**
w/o Progressive Analysis	**67**	**77**	**15.10%**	**84.90%**
w/o Bucketed Analysis	**78**	**96**	**19.16%**	**80.84%**

3.4 Analysis of Capability Transition

We analyze task-level transitions between consecutive evolution rounds to examine how capabilities change during self-evolution. Each task is classified as **fail-to-pass**, indicating a newly acquired capability; **pass-to-fail**, indicating a regression; **always-pass**, indicating retention; or **always-fail**, indicating an unresolved capability gap. As shown in Tables 12 and 13, EvoGroup generally acquires new capabilities while retaining existing ones, although stability varies across models and rounds. DeepSeek-V4-Flash maintains over 91% retention across all rounds, with regression rates between 5.40% and 8.36%. For GPT-5.5, retention reaches 94.19% in one transition but falls to 62.94% in the following round, indicating less stable capability accumulation.

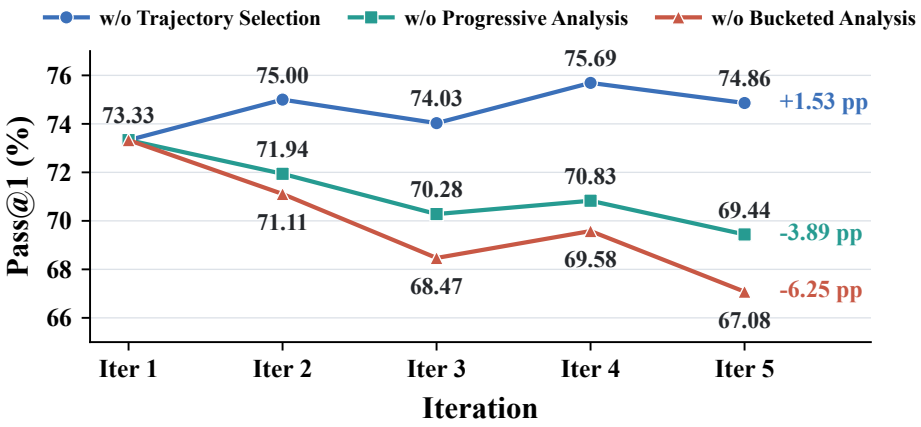


Figure 4: Performance across five self-evolution iterations under three ablation settings: w/o Trajectory Selection, w/o Progressive Analysis, and w/o Bucketed Analysis.

3.5 Analysis of Confidence and Bias

We examine whether self-evolution improves both task performance and confidence calibration by grouping trajectories into the four types defined in Section 2.2. Figure 6 show that, for DeepSeek-V4-Flash, the confident-and-correct share increases by 15.83 percentage points, while the confident-and-wrong share decreases by 6.25 points. This indicates improvements in both task performance and confidence calibration. For GPT-5.5, the confident-and-correct share increases by only 2.50 points, while the confident-and-wrong share rises by 10.41 points, indicating increased overconfidence in incorrect answers. Overall, DeepSeek-V4-Flash becomes both more capable and better calibrated during evolution, whereas GPT-5.5 shows less stable alignment between capability and confidence.

3.6 Analysis of Self-Reflection

We compare self-reflection with trajectory-based diagnosis across five iterations using three measures: localization-agreement, cause attribution-agreement, and diagnostic-evidence consistency. Localization-agreement reaches 84.0% for DeepSeek-V4-Flash and 74.4% for GPT-5.5, showing that self-reflection can often identify where a problem occurs without examining the full trajectory. In contrast, cause attribution-agreement is 49.2% and 62.0%, while diagnostic-evidence consistency is only 7.1% and 16.8%, respectively (As shown in figure 5). These results suggest that self-reflection provides a useful but imperfect localization signal: it can direct the Analysis Agent toward likely problem regions, while trajectory evidence remains necessary to verify the underlying cause.

3.7 Analysis of Adaptive Progressive Disclosure

We compare Adaptive Progressive Disclosure (APD) with Full Trace Analysis over five iterations while keeping the analysis model, budget, concurrency, and input content fixed. As shown in Table 6, APD reduces analysis time by 42.5% and token consumption by 41.5% on average. Most of the reduction comes from fewer input tokens, while output length remains similar. This shows that APD improves analysis efficiency by selectively accessing relevant trajectory segments instead of processing the full trace.

3.8 Effectiveness of Bucketed Analysis

Table 7 We evaluate whether bucketed analysis produces useful evolution signals from three perspectives: finding correctness, adoption by the Evolve Agent, and downstream repair. A finding is considered adopted when it leads to a Harness update, denied when the Evolve Agent explicitly rejects it, and unused when it is not selected. Downstream effectiveness is measured by how often adopted findings repair the affected tasks and by the resulting change in Pass@1. Across five iterations, DeepSeek-V4-Flash achieves 71.40% to 100.00% finding correctness, 57.10% to 66.70% adoption, and 41.40% to 59.40% repair rates. For GPT-5.5, correctness remains at 100.00%, with adoption

rates of 50.00% to 87.50% and repair rates of 33.30% to 68.60%. These results show that bucketed analysis can turn recurring task-level failures into high-level findings that the Evolve Agent can act on, although the resulting updates may still introduce regressions.

3.9 Analysis of Interface Importance

We analyze how the Evolve Agent uses the four revision interfaces across evolution rounds, as shown in Figure 7. Policy is the most frequently revised interface, accounting for 47.06% of edits for DeepSeek-V4-Flash and 46.15% for GPT-5.5, followed by Representation. Operation is revised only for DeepSeek-V4-Flash, while Intervention is used infrequently for both models. Overall, EvoGroup relies primarily on Policy and Representation updates, while Operation and Intervention are used more selectively.

4 Related Work

Agent Memory. Agents must retain relevant information across long interactions without context overload or retrieval failures. Long-context methods study compression, retrieval-aware pruning, and positional failures (Jiang et al., 2023; 2024; Liu et al., 2024; Bai et al., 2024). Retrieval-augmented generation combines dense retrieval and generation, with hierarchical and graph-based extensions (Lewis et al., 2020; Karpukhin et al., 2020; Sarthi et al., 2024; Edge et al., 2024; Asai et al., 2024). Agent-memory systems organize interaction histories through memory streams, reflection, virtual context, profiles, structured stores, semantic compression, lifelong updates, and reusable skills (Park et al., 2023; Shinn et al., 2023; Packer et al., 2023; Zhong et al., 2024; Chhikara et al., 2025; Xu et al., 2025; Kang et al., 2025; Fang et al., 2026; Liu et al., 2026; Yan et al., 2025; Wu et al., 2024; Wang et al., 2023; Zhao et al., 2024; Sumers et al., 2023). However, their fixed schemas limit the adaptation of memory organization and control. EvoG addresses this gap by evolving the representations and interfaces governing memory construction, retrieval, and revision.

Agent Harness. AgentBench and SWE-agent show that action interfaces and environmental feedback affect agent quality and transfer (Liu et al., 2023; Yang et al., 2024). Harness research extends this interface layer to prompts, memory, tools, skills, workflows, and execution control, building on ReAct and Reflexion (Yao et al., 2023; Shinn et al., 2023; Schick et al., 2023). Self-improvement methods bootstrap traces, refine outputs, or search over reasoning structures (Zelikman et al., 2022; Madaan et al., 2023; Zhou et al., 2024; Besta et al., 2024); programmatic and experience-based work adapts agent pipelines through declarative modules, textual gradients, and trajectory analysis (Khattab et al., 2023; Yuksekgonul et al., 2024; Pan et al., 2026a; Lin et al., 2026; Huang et al., 2026; Chen et al., 2026; Zhou, 2026; Zhang et al., 2026a). Recent studies further explore code-centered Harnesses, agentic reinforcement learning, distillation, architecture, automatic synthesis, and retrospective optimization (Ning et al., 2026; Xu et al., 2026a; Cui et al., 2026; Banu, 2026; Lou et al., 2026; Pan et al., 2026b; Xu et al., 2026b). However, existing systems mainly target coding, terminal-use, or generic environments and typically assume relatively compact interaction trajectories. In long-context group chats, evidence is distributed across speakers, sessions, and temporal states, producing lengthy trajectories whose direct analysis is costly and vulnerable to irrelevant contextual interference. EvoG addresses this challenge through confidence-guided trajectory selection, adaptive progressive disclosure, and cross-trajectory bucketed analysis. These mechanisms localize failures, selectively expose relevant trajectory segments, and aggregate recurring evidence, enabling efficient and reliable self-evolution without repeatedly processing the full conversational context.

5 Conclusion

In this work, we introduce EvoGroup, a self-evolving Harness framework for group agent memory. By treating the Harness as an evolvable system and learning evolution signals from interaction experiences, EvoGroup enables Group Agents to autonomously adapt to diverse groups and shared contexts. Empirical evaluation demonstrates that EvoGroup improves task efficiency while maintaining strong performance. In the future, we will explore more realistic collaborative scenarios with diverse and long-term multi-user shared contexts.

AI USE STATEMENT

In this work, we used generative AI tools to refine the wording and presentation of the manuscript. Generative AI is also an integral part of the research methodology studied in this work. Specifically, language-model agents are used to answer queries over shared contexts, perform self-reflection on low-confidence responses, analyze interaction trajectories, and propose revisions to the Agent Harness. We have reviewed all AI-assisted content and experimental outputs, and we take responsibility for the final manuscript, including its claims, analyses, and reported results.

ETHICS STATEMENT

This work does not involve new human-subject studies or the collection of sensitive personal data by the authors. We do not identify ethical concerns specific to this study beyond the general considerations associated with language-model agents and conversational memory systems. All authors adhere to the ICLR Code of Ethics.

REPRODUCIBILITY STATEMENT

We provide anonymous open-source code for EvoGroup at `https://anonymous.4open.science/r/37883-8DC0`. The main paper describes the EvoGroup framework, self-evolution procedure, and experimental setup. The appendix provides detailed experimental setup. Moreover, the anonymous code release and these materials support reproduction of the experiments reported in this paper.

REFERENCES

Petr Anokhin, Nikita Semenov, Artyom Sorokin, Dmitry Evseev, Andrey Kravchenko, Mikhail Burtsev, and Evgeny Burnaev. AriGraph: Learning knowledge graph world models with episodic memory for LLM agents. *arXiv preprint arXiv:2407.04363*, 2024.

Anthropic. Claude code, 2025. URL `https://github.com/anthropics/claude-code`.

Anthropic. Claude tag: Bringing claude into your team conversations. `https://www.anthropic.com/`, 2026.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-rag: Learning to retrieve, generate, and critique through self-reflection. *arXiv preprint arXiv:2310.11511*, 2024. URL `https://arxiv.org/abs/2310.11511`.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024.

Bogdan Banu. Harness engineering as categorical architecture. *arXiv preprint arXiv:2605.12239*, 2026. URL `https://arxiv.org/abs/2605.12239`.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024.

Tingyang Chen, Shuo Lu, Kang Zhao, Weicheng Meng, Hanlin Teng, Tianhao Li, Chao Li, Xule Liu, Jian Liang, Zhizhong Zhang, Yuan Xie, Heng Qu, Kun Shao, and Jian Luan. Harnessx: A composable, adaptive, and evolvable agent harness foundry. *arXiv preprint arXiv:2606.14249*, 2026. URL `https://arxiv.org/abs/2606.14249`.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025. URL `https://arxiv.org/abs/2504.19413`.

Yu Cui, Wuli Yang, Yirui Shi, Junhao Xia, Hui Jiang, Lei Gao, and Chenfu Bao. Agent harness distillation: Inference-time harness extraction and exploitation in autonomous multi-agent systems. *arXiv preprint arXiv:2607.28147*, 2026. URL https://arxiv.org/abs/2607.28147.

DeepSeek-AI. Deepseek-v4-flash, 2026. URL https://api-docs.deepseek.com/quick_start/pricing/.

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, and Dasha Metropolitansky. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*, 2024. URL https://arxiv.org/abs/2404.16130.

Jizhan Fang, Xinle Deng, Haoming Xu, Ziyan Jiang, Yuqi Tang, Ziwen Xu, Shumin Deng, Yunzhi Yao, Mengru Wang, Shuofei Qiao, Huajun Chen, and Ningyu Zhang. Lightmem: Lightweight and efficient memory-augmented generation. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2510.18866. ICLR 2026 poster.

GLM-5 Team, Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chengxing Xie, et al. Glm-5: from vibe coding to agentic engineering, 2026. URL https://arxiv.org/abs/2602.15763.

Chuanrui Hu, Tong Li, Xingze Gao, Hongda Chen, Yi Bai, Dannong Xu, Tianwei Lin, Xiaohong Li, Yunyun Han, Jian Pei, and Yafeng Deng. Evaluating long-horizon memory for multi-party collaborative dialogues. *arXiv preprint arXiv:2602.01313*, 2026. URL https://arxiv.org/abs/2602.01313.

Yue Huang, Wenjie Wang, Han Bao, Yuchen Ma, Xiaonan Luo, Yi Nian, Haomin Zhuang, Zheyuan Liu, Yue Zhao, and Xiangliang Zhang. Memoharness: Agent harnesses that learn from experience. *arXiv preprint arXiv:2607.14159*, 2026. URL https://arxiv.org/abs/2607.14159.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Llmlingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. doi: 10.18653/v1/2023.emnlp-main.825. URL https://doi.org/10.18653/v1/2023.emnlp-main.825.

Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Longllmlingua: Accelerating and enhancing llms in long context scenarios via prompt compression. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024. doi: 10.18653/v1/2024.acl-long.91. URL https://doi.org/10.18653/v1/2024.acl-long.91.

Jiazheng Kang, Mingming Ji, Zhe Zhao, and Ting Bai. Memory os of ai agent, 2025. URL https://arxiv.org/abs/2506.06326.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, 2020. doi: 10.18653/v1/2020.emnlp-main.550. URL https://doi.org/10.18653/v1/2020.emnlp-main.550.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, et al. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*, 2023. URL https://arxiv.org/abs/2310.03714.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Kuttler, Mike Lewis, Wen-tau Yih, Tim Rocktaschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems 33*, 2020. URL https://arxiv.org/abs/2005.11401.

Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, Zhiheng Xi, Xuanjing Huang, Hang Yan, Zhenhua Han, Tao Gui, and Yu-Gang Jiang. Agentic harness engineering: Observability-driven automatic evolution of coding-agent harnesses. *arXiv preprint arXiv:2604.25850*, 2026. URL https://arxiv.org/abs/2604.25850.

Jiaqi Liu, Yaofeng Su, Peng Xia, Siwei Han, Zeyu Zheng, Cihang Xie, Mingyu Ding, and Huaxiu Yao. Simplemem: Efficient lifelong memory for llm agents. *arXiv preprint arXiv:2601.02553*, 2026. URL https://arxiv.org/abs/2601.02553.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024. doi: 10.1162/tacl_a_00638.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu-Cheng Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Zheng Deng, et al. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*, 2023. URL https://arxiv.org/abs/2308.03688.

Xinghua Lou, Miguel Lázaro-Gredilla, Antoine Dedieu, Carter Wendelken, Wolfgang Lehrach, and Kevin P. Murphy. Autoharness: Improving llm agents by automatically synthesizing a code harness. *arXiv preprint arXiv:2603.03329*, 2026. URL https://arxiv.org/abs/2603.03329.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems 36*, 2023. doi: 10.52202/075280-2019. URL https://arxiv.org/abs/2303.17651.

Xuying Ning, Katherine Tieu, Dongqi Fu, Tianxin Wei, Zihao Li, Yuanchen Bei, Jiaru Zou, Mengting Ai, Zhining Liu, Ting-Wei Li, Lingjie Chen, Yanjun Zhao, Ke Yang, Bingxuan Li, Cheng Qian, Gaotang Li, Xiao Lin, Zhichen Zeng, Ruizhong Qiu, Sirui Chen, Yifan Sun, Xiyuan Yang, Ruida Wang, Rui Pan, Chenyuan Yang, Dylan Zhang, Liri Fang, Zikun Cui, Yang Cao, Pan Chen, Dorothy Sun, Ren Chen, Mahesh Srinivasan, Nipun Mathur, Yinglong Xia, Hong Li, Hong Yan, Pan Lu, Lingming Zhang, Tong Zhang, et al. Code as agent harness. *arXiv preprint arXiv:2605.18747*, 2026. URL https://arxiv.org/abs/2605.18747.

OpenAI. Introducing codex, 2025. URL https://openai.com/index/introducing-codex/.

OpenAI. Introducing gpt-5.5, April 2026a. URL https://openai.com/index/introducing-gpt-5-5/.

OpenAI. Introducing workspace agents in chatgpt. https://openai.com/, 2026b.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023. doi: 10.48550/arXiv.2310.08560. URL https://arxiv.org/abs/2310.08560.

Linyue Pan, Lexiao Zou, Shuo Guo, Jingchen Ni, and Hai-Tao Zheng. Natural-language agent harnesses. *arXiv preprint arXiv:2603.25723*, 2026a. URL https://arxiv.org/abs/2603.25723.

Wenbo Pan, Shujie Liu, Chin-Yew Lin, Jingying Zeng, Xianfeng Tang, Xiangyang Zhou, Yan Lu, and Xiaohua Jia. Evolving agents in the dark: Retrospective harness optimization via self-preference. *arXiv preprint arXiv:2606.05922*, 2026b. URL https://arxiv.org/abs/2606.05922.

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

Parth Sarthi, Salman H. Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D. Manning. Raptor: Recursive abstractive processing for tree-organized retrieval. *arXiv preprint arXiv:2401.18059*, 2024. URL https://arxiv.org/abs/2401.18059.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36, 2023.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems 36*, 2023. doi: 10.52202/075280-0377. URL https://arxiv.org/abs/2303.11366.

Jared Spataro. Human-agent teams: The next era of work. https://www.microsoft.com/en-us/worklab/, 2025.

Theodore R. Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L. Griffiths. Cognitive architectures for language agents. *arXiv preprint arXiv:2309.02427*, 2023. URL https://arxiv.org/abs/2309.02427.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023. URL https://arxiv.org/abs/2305.16291.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. Longmemeval: Benchmarking chat assistants on long-term interactive memory. *arXiv preprint arXiv:2410.10813*, 2024.

Binfeng Xu, Hao Zhang, Shaokun Zhang, Songyang Han, Mingjie Liu, Jian Hu, Shizhe Diao, Zhenghui Jin, Yunheng Zou, Michael Demoret, Jan Kautz, and Yi Dong. Polar: Agentic rl on any harness at scale. *arXiv preprint arXiv:2605.24220*, 2026a. URL https://arxiv.org/abs/2605.24220.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. In *Advances in Neural Information Processing Systems 38*, 2025. doi: 10.52202/085713-0593. URL https://arxiv.org/abs/2502.12110.

Wujiang Xu, Jiaojiao Han, Minghao Guo, Kai Mei, Xi Zhu, Han Zhang, and Dimitris N. Metaxas. Ael: Evolving agent harness in open-ended environments. *arXiv preprint arXiv:2604.21725*, 2026b. URL https://arxiv.org/abs/2604.21725.

Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Jinhe Bi, Kristian Kersting, Jeff Z. Pan, Hinrich Schütze, Volker Tresp, and Yunpu Ma. Memory-r1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. *arXiv preprint arXiv:2508.19828*, 2025. URL https://arxiv.org/abs/2508.19828.

Jingbo Yang, Kwei-Herng Lai, Xiaowen Wang, Shiyu Chang, Yaar Harari, and Evgeniy Gabrilovich. Groupmembench: Benchmarking llm agent memory in multi-party conversations. *arXiv preprint arXiv:2605.14498*, 2026. URL https://arxiv.org/abs/2605.14498.

John Jeongseok Yang, Carlos E. Jimenez-Gomez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *arXiv preprint arXiv:2405.15793*, 2024. URL https://arxiv.org/abs/2405.15793.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.03629.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. Textgrad: Automatic differentiation via text. *arXiv preprint arXiv:2406.07496*, 2024. URL https://arxiv.org/abs/2406.07496.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. Star: Bootstrapping reasoning with reasoning. *arXiv preprint arXiv:2203.14465*, 2022. URL https://arxiv.org/abs/2203.14465.

Hangfan Zhang, Shao Zhang, Kangcong Li, Chen Zhang, Yang Chen, Yiqun Zhang, Lei Bai, and Shuyue Hu. Self-harness: Harnesses that improve themselves. *arXiv preprint arXiv:2606.09498*, 2026a. URL https://arxiv.org/abs/2606.09498.

Shengtao Zhang, Jiaqian Wang, Ruiwen Zhou, Junwei Liao, Yuchen Feng, Zhuo Li, Yujie Zheng, Weinan Zhang, Ying Wen, Zhiyu Li, Feiyu Xiong, Yutao Qi, Bo Tang, and Muning Wen. Memrl: Self-evolving agents via runtime reinforcement learning on episodic memory, 2026b. URL https://arxiv.org/abs/2601.03192.

Andrew X. Zhao, Daniel T. Huang, Quentin Xu, Matthieu Gaetan Lin, Yong-Jin Liu, and Gao Ming Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 2024. doi: 10.1609/aaai.v38i17.29936. URL https://doi.org/10.1609/aaai.v38i17.29936.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 2024. doi: 10.1609/aaai.v38i17.29946. URL https://doi.org/10.1609/aaai.v38i17.29946.

Pei Zhou, Jay Pujara, Xiang Ren, Xinyun Chen, Heng-Tze Cheng, Quoc V. Le, Ed H. Chi, and Denny Zhou. Self-discover: Large language models self-compose reasoning structures. *arXiv preprint arXiv:2402.03620*, 2024. URL https://arxiv.org/abs/2402.03620.

Tailin Zhou. Hierarchical self-improvement: A framework for task-specific evolvable agent harnesses. *arXiv preprint arXiv:2608.08466*, 2026. URL https://arxiv.org/abs/2608.08466.

A Experimental Setup

This appendix provides the benchmark composition, evaluation procedure, evolution settings, and supplementary results. The protocols below expand the experimental description in Section 3.

A.1 Benchmarks and Data Partitioning

GroupMemBench. GroupMemBench tests memory in workplace conversations with several users and channels. To answer a question, the Agent must find information from earlier conversations and keep track of who said it. The 745 questions cover multi-hop reasoning, knowledge updates, temporal reasoning, user-implicit reasoning, term ambiguity, and abstention. Table 3 gives the number of questions in each domain and question type.

EverMemBench. EverMemBench tests memory over long conversations among several people. It covers factual recall, memory awareness, and user-profile understanding. Its questions ask the Agent to recall facts, use relevant past information, track changes, and infer user characteristics. Table 4 shows the nine question types and their counts. The full benchmark has 2,400 questions.

Evolution and evaluation partitions. We randomly select 720 questions, or 30% of EverMemBench, for evolution. We sample from each of the nine question types so that their shares stay close to those in the full dataset. The 720 questions remain fixed across rounds. We repeat the evolution experiment with three independent random seeds; within each seeded run, every question is attempted exactly once and scored under pass@1. The remaining 1,680 questions form a separate held-out validation set and are likewise evaluated under pass@1 with one attempt per question. Validation questions are not used to suggest changes or select checkpoints. We omit the concrete seed identifiers. For transfer, we apply the Harness evolved on EverMemBench to GroupMemBench without further evolution or tuning.

A.2 Evaluation Metrics and Judging

Let $\mathcal{D}$ be an evaluation set of N questions, and let $s_i \in \{0, 1\}$ be the correctness indicator for question i. We report question-level accuracy and category accuracy as

$$\mathrm{Acc}(\mathcal{D}) = \frac{100}{N} \sum_{i \in \mathcal{D}} s_i, \qquad \mathrm{Acc}_k = \frac{100}{N_k} \sum_{i \in \mathcal{D}_k} s_i, \tag{8}$$

where $\mathcal{D}_k$ contains N_k questions of type k. The overall accuracy is computed over individual questions, equivalently as $\sum_k N_k \mathrm{Acc}_k / \sum_k N_k$. It is therefore weighted by category size, rather than the arithmetic mean of category accuracies.

We use the same GPT-5.6-luna judge for semantically graded answers across answer models. Let q_i, a_i, and $\hat{a}_i$ denote the question, reference answer, and predicted answer. On EverMemBench, multiple-choice items are scored by exact option match and open-ended items by the binary semantic-equivalence decision of judge J:

$$s_i = \begin{cases} \mathbb{I}[\hat{a}_i = a_i], & i \in \mathcal{D}_{\mathrm{MC}}, \\ J(q_i, a_i, \hat{a}_i), & i \in \mathcal{D}_{\mathrm{OE}}. \end{cases} \tag{9}$$

For GroupMemBench, all six question categories use $s_i = J(q_i, a_i, \hat{a}_i)$. A judge decision of one denotes a semantically correct answer; zero denotes an incorrect answer. All overall scores reported in this appendix are computed with the question counts of the corresponding evaluation set.

A.3 Model and Runtime Configuration

A.4 Reproducibility and Evaluation Isolation

The stratified EverMemBench evolution sample is fixed before the first round and reused in every round. The separate evaluation questions are excluded from Harness proposal and checkpoint selection. Within an evolution campaign, the answer model, judge model, question set, scoring rule, and runtime limits are held fixed; only the Harness may be revised. Every proposed edit is linked to trajectory-level evidence and screened for benchmark-specific identifiers, reference answers,

Table 3: Overview of GroupMemBench and the distribution of question types across domains.

Domain	Multi-Hop	Know. Update	Temporal	User-Implicit	Term Ambig.	Abstention	Total
Finance	48	32	45	15	45	29	214
Healthcare	48	17	37	2	7	34	145
Manufacturing	45	22	43	4	11	48	173
Technology	41	36	37	28	43	28	213
Total	**182**	**107**	**162**	**49**	**106**	**139**	**745**

Table 4: Overview of EverMemBench and the distribution of question types across its evaluation dimensions.

Dimension	Question Type	Questions
Fine-Grained Recall	Single-Hop (Single)	213
	Multi-Hop (Multi)	249
	Temporal (Temp.)	300
Memory Awareness	Constraint (Const.)	402
	Proactivity (Proact.)	427
	Update	268
Profile Understanding	Style	176
	Skill	169
	Role	196
Total	–	**2,400**

or question-specific rules before it can be evaluated in a later round. Failed tool requests, retries, timeouts, and evaluation exceptions are retained as distinct events in the recorded trajectories.

For a reproducible campaign, the task IDs and split assignment, sampling seed, model and judge configurations, Harness snapshots, question-level traces, judge decisions, and the checkpoint-selection record must be preserved together. A campaign index should identify the benchmark, answer model, round, and the exact Harness and split versions used for each reported number. The final release must identify which of these records are available; a file or snapshot should not be claimed as released solely because the protocol specifies it.

B Evolution Protocol

B.1 Trajectory Analysis and Harness Revision

Let $\mathcal{T} = \{x_i\}_{i=1}^{N}$ be the fixed evolution questions and H_t be the Harness at round t. An interaction produces trajectory $\tau_i^t = \mathrm{Interact}(H_t, x_i)$, the Agent's confidence $p_i^t \in [0, 1]$, and the judge's answer-correctness label $y_i^t \in \{0, 1\}$. The trajectory records the messages, tool requests and observations, memory operations, and final answer. A timeout or infrastructure exception is recorded separately from a judged incorrect answer.

At confidence threshold $\theta = 0.5$, semantic trajectories are assigned to confident-correct (CC), confident-wrong (CW), unconfident-correct (UC), or unconfident-wrong (UW), where "confident" means $p_i^t > \theta$. Low-confidence trajectories also receive a self-reflection, used as diagnostic evidence without changing their official correctness label. The Analysis Agent first inspects addressable summaries and retrieves full trace segments when it needs to verify a finding. It synthesizes recurring patterns across questions before the Evolve Agent proposes changes to the Harness components: skills, prompts, middleware, tools, and memory. Each proposal includes a change manifest and its predicted repair and regression risks.

Table 5: Hyperparameter settings for EvoGroup on EverMemBench. We report the DeepSeek-V4-Flash configuration as a representative example. Model-specific runs replace the backing model while retaining the stated protocol limits. All reported runs use GPT-5.6-luna as the shared LLM judge.

Block	Hyperparameter	Value
Answer Agent	Backing model	DeepSeek-V4-Flash
	Max context	200,000 tokens
	Max output per call	8,192 tokens
	Max steps	60
	Max tool calls	20
	Per-question timeout	500 s
	Available tools	`list_files`, `read_file`, `grep_search` `write_file`, `create_file`
Analysis Agent	Backing model	DeepSeek-V4-Flash
	Concurrent tasks	8
	Per-question timeout	900 s
	Retry attempts	3
	Retry backoff	10 s, doubled after each retry
Evolve Agent	Backing model	DeepSeek-V4-Flash
	Temperature	0.7
	Max context	200,000 tokens
	Max generation per turn	32,000 tokens
	Max model turns	500
	Context compaction threshold	0.75
LLM Judge	Backing model	GPT-5.6-luna
	Retry attempts	3
	Concurrency	4
Outer Loop	Evaluation rounds	6
	Harness updates	5
	Evaluation protocol	pass@1
	Random seeds	3 independent runs (identities omitted)
	Concurrency	6
	Evolution set	EverMemBench, 720 questions
	Validation set	EverMemBench, 1,680 questions
	Sampling	Stratified random, 30% overall

The result measured at round t belongs to H_t. A candidate H_{t+1} is only a proposal until it has been executed and evaluated in a subsequent round. The evolution set is used to select a checkpoint; results on a separate test set are reported independently.

B.2 Evolution Algorithm

The algorithm saves a round before selecting the next candidate, and it compares only evaluated Harness checkpoints when selecting H_*. ScoreQuestions aggregates the recorded rollout judgments into a question-level evolution score using a fixed campaign-level rule. The checkpoint tie-break rules are given below.

B.3 Candidate Validation and Capability Transitions

Each candidate must include a change manifest. Structural checks require parseable files, configuration, prompts, and tool interfaces. An edit must be linked to a verified trajectory or cross-trace finding and must remain task-general: it cannot encode a task ID, a corpus phrase, a reference answer, or a question-specific rule. The task set, judge, model configuration, trace collector, and experimental infrastructure are held fixed. If no candidate passes, $H_{t+1} = H_t$. Among candidates that pass, the stated selection priority is evidence coverage, predicted repair benefit, predicted regression risk, edit

Algorithm 1 Evidence-driven Harness evolution

Require: Initial Harness H_0, fixed evolution set $\mathcal{T}$, evaluation rounds $R_{\max}$, threshold $\theta = 0.5$, optional target score ρ_{target}
Ensure: Best evaluated checkpoint H_* and archive $\mathcal{A}$

$H \leftarrow H_0$; $\mathcal{A} \leftarrow \emptyset$; $\text{best}_{\text{score}} \leftarrow -\infty$; $\text{best}_H \leftarrow H_0$
for $t = 0, \ldots, R_{\max} - 1$ **do**
  $\boldsymbol{\tau} \leftarrow \{\textsc{Interact}(H, x_i) : x_i \in \mathcal{T}\}$
  $(\mathbf{y}, \mathbf{p}) \leftarrow \textsc{JudgeAndParseConfidence}(\boldsymbol{\tau})$
  $\mathcal{E} \leftarrow \textsc{Stratify}(\boldsymbol{\tau}, \mathbf{y}, \mathbf{p}, \theta)$
  $\mathcal{R} \leftarrow \textsc{SelfReflect}(\{\tau \in \mathcal{E} : p(\tau) \le \theta\})$
  $\mathcal{F} \leftarrow \textsc{AnalyzeAndSynthesize}(\mathcal{E}, \mathcal{R})$
  $v_t \leftarrow \textsc{ScoreQuestions}(\mathbf{y}, \mathcal{T}, K)$
  $\mathcal{A} \leftarrow \mathcal{A} \cup \{\textsc{SaveRound}(t, H, \boldsymbol{\tau}, \mathbf{y}, \mathbf{p}, \mathcal{E}, \mathcal{R}, \mathcal{F}, v_t)\}$
  if $v_t > \text{best}_{\text{score}}$ **then**
    $\text{best}_{\text{score}} \leftarrow v_t$; $\text{best}_H \leftarrow H$
  end if
  if $t = R_{\max} - 1$ **or** ($\rho_{\text{target}} \neq \text{None}$ **and** $v_t \ge \rho_{\text{target}}$) **then**
    break
  end if
  $\mathcal{C} \leftarrow \textsc{Propose}(H, \mathcal{F})$
  $(H_{\text{next}}, m_t) \leftarrow \textsc{ValidateAndSelect}(\mathcal{C}, H, \mathcal{F})$
  $\mathcal{A} \leftarrow \textsc{AttachManifest}(\mathcal{A}, t, m_t)$
  $H \leftarrow H_{\text{next}}$
end for
$H_* \leftarrow \textsc{CheckpointSelect}(\mathcal{A}, \text{best}_H, \text{best}_{\text{score}})$
return $H_*, \mathcal{A}$

scope, and finally manifest completeness. These priorities govern which candidate is tried next; they are not evidence of its empirical effect.

For two consecutively evaluated Harnesses, a question can be fail-to-pass (F2P), pass-to-fail (P2F), always-pass (AP), or always-fail (AF). Timeouts and infrastructure exceptions are tracked separately. If the paired question outcomes are available, the corresponding gain, regression, and retention rates are

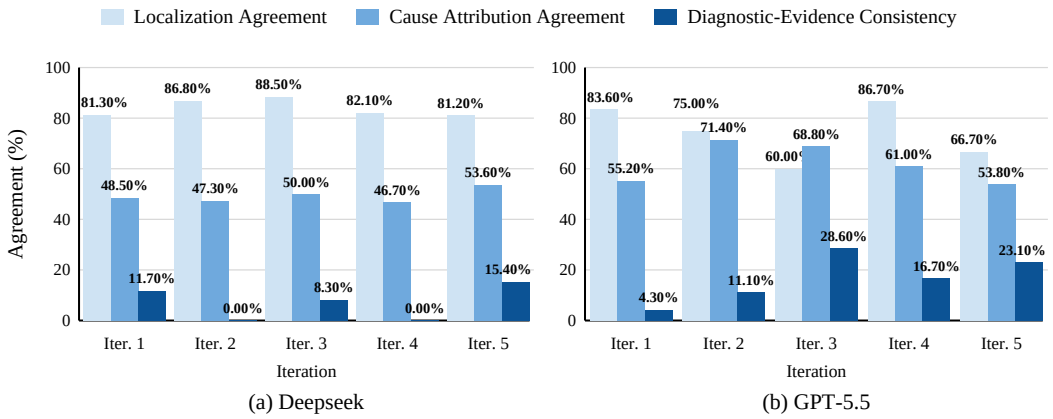
$$\begin{aligned} \text{GainRate}_t &= \frac{N_{\text{F2P}}}{N_{\text{F2P}} + N_{\text{AF}}}, \\ \text{RegressionRate}_t &= \frac{N_{\text{P2F}}}{N_{\text{AP}} + N_{\text{P2F}}}, \\ \text{Retention}_t &= 1 - \text{RegressionRate}_t. \end{aligned} \tag{10}$$

These quantities are defined only where their denominators are nonzero and are computed from paired pass@1 outcomes for the same questions. They supplement, rather than replace, official question-level accuracy. The highest-scoring evaluated checkpoint on the fixed evolution set is selected; ties are broken by fewer regressions, fewer exceptions, lower assessed risk, and then earlier round. The archive retains the evaluated Harness, scores, traces, confidence labels, reflections, findings, validation decisions, and change manifest for each completed round.

C Additional Results and Analysis

C.1 Corss-Benchmark Transfer Experiment

We additionally evaluate cross-benchmark transfer with GLM-5.1. The Harness is evolved on EverMemBench alone and applied to GroupMemBench without additional evolution, tuning, or benchmark-specific adaptation. The same answer model and GPT-5.6-luna judge are used for the before/after comparison. The overall accuracy is weighted by the number of questions in each GroupMemBench category (745 questions in total).



(a) Deepseek (b) GPT-5.5

Figure 5: Distribution of trajectory correctness–confidence types across self-evolution for DeepSeek-V4-Flash and GPT-5.5.

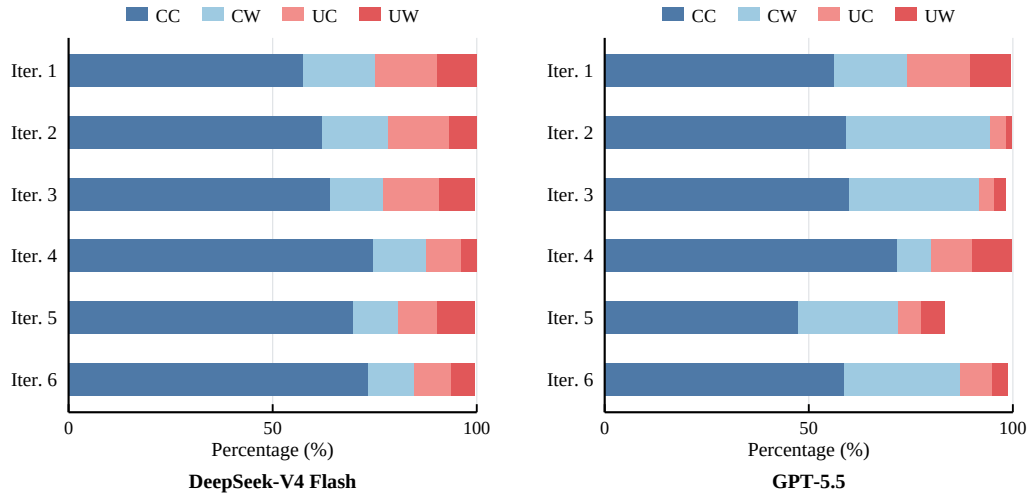
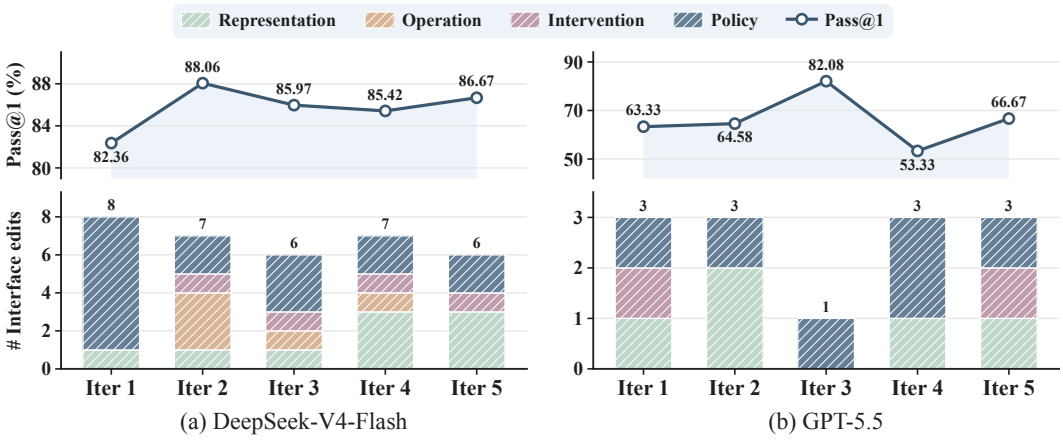


Figure 6: Distribution of trajectory correctness–confidence types across self-evolution for DeepSeek-V4-Flash and GPT-5.5.

Table 6: Effectiveness of Adaptive Progressive Disclosure (APD) for trajectory analysis. We compare APD Analysis with Full Trace Analysis across self-evolution iterations in terms of analysis time and token consumption. Time is reported in minutes (min), and token usage in millions of tokens (M).

Iteration	**Time (min)**		**Tokens (M)**	
	APD	*Full*	*APD*	*Full*
Iteration 1	64.6	110.8	54.58M	82.09M
Iteration 2	26.1	43.0	33.34M	66.55M
Iteration 3	21.5	37.9	22.08M	34.18M
Iteration 4	17.7	30.1	28.46M	56.12M
Iteration 5	24.9	47.5	23.88M	38.64M



(a) DeepSeek-V4-Flash (b) GPT-5.5

Figure 7: Distribution of revision-interface edits across evolution rounds for DeepSeek-V4-Flash and GPT-5.5.

Table 7: Correctness, harness adoption, and repair statistics across five EvoGroup iterations. Correctness, adoption, and repair are reported on a 0–100 scale. Colored bars are scaled from 0 to 100, while Δ Pass@1 bars encode absolute change magnitude within each subtable. † marks the first iteration.

(a) DeepSeek-V4-Flash

Iter.	Correct.	Adoption			Repair	Δ P@1 (pp)
		Adopted	*Denied*	*Unused*		
Iter. 1†	87.50	62.50	0.00	37.50	59.40	+17.89
Iter. 2	88.90	66.70	0.00	33.30	57.10	+6.92
Iter. 3	100.00	57.10	14.30	28.60	41.40	-2.37
Iter. 4	100.00	66.70	0.00	33.30	54.50	-0.64
Iter. 5	71.40	57.10	14.30	28.60	57.60	+1.46

(b) GPT-5.5

Iter.	Correct.	Adoption			Repair	Δ P@1
		Adopted	*Denied*	*Unused*		
Iter. 1	100.00	87.50	0.00	12.50	40.00	4.2
Iter. 2	100.00	62.50	0.00	37.50	45.20	0.80
Iter. 3	100.00	85.70	0.00	14.30	68.60	5.40
Iter. 4	100.00	62.50	0.00	37.50	44.40	-3.70
Iter. 5	100.00	50.00	0.00	50.00	33.30	2.90

Table 9 reports an overall increase from approximately 69.72% to 71.46%, or 1.74 percentage points when the displayed totals are subtracted. Five categories improve, while Knowledge Update declines from 72.90% to 71.96%. This additional model suggests that part of the evolved behavior transfers across benchmarks, although the effects differ by category and answer model.

C.2 Efficiency Analysis

Metric definitions. We separate online per-question cost from one-time preparation. Online metrics are *Tokens per Question* (Tok.), *Tool Calls per Question* (Calls), and *Question Runtime* (QA), each averaged over completed trials, including both correct and incorrect answers. Trials that terminate because of a system-level failure, timeout, crash, exception, or missing output are excluded from all three averages. Tok. is the sum of uncached input tokens, cached input tokens, and output tokens attributed to a question across all model calls. Each independently issued executable tool request counts as one call; failed and retried tool requests inside an otherwise completed trial are counted separately, parallel requests are counted individually, and sub-agent requests are attributed to the parent question. Creating a sub-agent does not itself count as a tool request. QA is elapsed wall-clock time from question start to completed termination. *Cold-Start Time* (Cold) is a separate one-time preparation cost before online answering; it is not a question-level average. A dash denotes an unavailable or inapplicable measurement, not zero cost.

Different cost profiles. Tables 10 and 11 show distinct online and preparation profiles. Structured and self-evolving memory methods may answer with approximately 1–17K tokens per question after constructing or updating memory, but their measured cold-start stages can be substantial. Agentic Harnesses spend more computation during online reasoning and tool interaction and have no separately reported cold-start measurement in these tables. Tok., Calls, QA, and Cold therefore describe different resource dimensions; their values should be interpreted separately.

Table 8: Pass@1 (%) on GroupMemBench by evaluation dimension and answer model. Average is the reported aggregate score. **Bold** marks the best result within each answer-model block, excluding FullContext.

Method	Multi-Hop	Update	Ambiguity	Implicit	Temporal	Abstention	Average
GPT-5.5							
FullContext	–	–	–	–	–	–	–
Structured Memory							
Mem0	39.56	24.30	29.25	40.82	51.23	77.70	45.64
MemOS	20.88	14.02	26.42	28.57	20.37	74.82	31.14
Self-Evolving Memory							
A-MEM	39.01	31.78	32.08	63.27	30.25	69.78	42.42
MemRL	29.12	30.84	31.13	61.22	19.14	**84.17**	39.87
Human-Designed Harness							
Codex	60.43	31.77	47.16	48.97	62.96	66.18	55.43
Claude Code	43.96	28.30	36.89	42.86	44.44	63.31	44.95
EvoGroup							
H_0	57.69	**70.09**	**50.00**	77.55	72.22	52.52	61.88
H_*	**64.84**	67.29	42.25	**79.59**	**77.78**	54.68	**63.89**
DeepSeek-V4-Flash							
FullContext	–	–	–	–	–	–	–
Structured Memory							
Mem0	34.07	14.02	28.30	42.86	38.27	92.09	42.68
MemOS	17.03	8.41	18.87	22.45	17.90	91.37	30.47
Self-Evolving Memory							
A-MEM	34.38	26.53	25.00	64.71	25.61	80.95	40.69
MemRL	25.27	21.50	18.87	53.06	16.05	**94.96**	36.64
Human-Designed Harness							
Codex	75.27	**85.05**	54.72	**81.63**	70.99	92.81	**76.51**
Claude Code	68.68	57.01	46.23	61.22	67.90	89.21	66.98
EvoGroup							
H_0	**82.42**	70.09	**77.78**	54.72	65.31	79.86	74.09
H_*	74.73	65.42	74.69	62.26	**73.47**	72.66	71.14

Table 9: Category-wise transfer performance of GLM-5.1 on GroupMemBench. The evolved Harness is obtained exclusively from EverMemBench and directly transferred to GroupMemBench without additional adaptation. Overall values are question-count-weighted estimates from the displayed category accuracies.

	Multi-Hop	Update	Ambiguity	Implicit	Temporal	Abstention	Overall
Before Evolution	69.23	**72.90**	64.81	69.39	72.64	68.35	69.72
After Evolution	**72.53**	71.96	**66.67**	**73.47**	**74.53**	**69.06**	**71.46**

EvoGroup versus human-designed Harnesses. With DeepSeek-V4-Flash on EverMemBench, H_* uses 12.57 calls and 14.82 s per question, versus 16.96 calls and 56.64 s for Codex and 17.32 calls and 74.56 s for Claude Code. On GroupMemBench, H_* uses 13.45 calls and 27.86 s, versus 27.59 calls and 166.77 s for Codex and 16.54 calls and 138.34 s for Claude Code. These comparisons show shorter observed online execution with fewer external requests in the DeepSeek-V4-Flash settings.

Effect of evolution On EverMemBench with DeepSeek-V4-Flash, evolution reduces calls from 16.47 to 12.57 and QA from 16.67 s to 14.82 s, while Tok. increases from 111K to 123K. On GroupMemBench, calls decrease from 14.68 to 13.45 and QA from 28.91 s to 27.86 s, while Tok. rises from 119K to 200K. The observed computation shifts toward model tokens in these settings. The cost pattern is not uniform across answer models: with GPT-5.5 on GroupMemBench, H_* changes calls from 14.36 to 14.46, QA from 20.13 s to 22.25 s, and Tok. from 120K to 124K. The cost tables should be read alongside the corresponding accuracy results to evaluate effectiveness and efficiency jointly.

Table 10: EverMemBench efficiency. Tok. (K) and Calls are per question; QA and Cold are in seconds. Lower is better. Best values are **bold**; second-best values are underlined.

Method	Cost / Question		Runtime	
	Tok.	Calls	QA	Cold
GPT-5.5				
Structured Memory				
Mem0	**1.096**	–	8.14	54,884.28
MemOS	1.315	–	8.31	**3,162.48**
Self-Evolving Memory				
A-MEM	11.680	–	**2.34**	158,123.86
MemRL	3.006	–	4.34	8,620.30
Human-Designed Harness				
Codex	216	8.56	46.23	–
Claude Code	205	**7.88**	81.75	–
EvoGroup				
H_0	–	–	–	–
H_*	–	–	–	–
DeepSeek-V4-Flash				
Structured Memory				
Mem0	**1.105**	–	**2.13**	54,456.28
MemOS	1.318	–	3.30	**3,147.12**
Self-Evolving Memory				
A-MEM	10.909	–	2.21	158,456.23
MemRL	3.075	–	3.53	8,665.31
Human-Designed Harness				
Codex	668	16.96	56.64	–
Claude Code	66	17.32	74.56	–
EvoGroup				
H_0	111	16.47	16.67	–
H_*	123	**12.57**	14.82	–

Table 11: GroupMemBench efficiency. Tok. (K) and Calls are per question; QA and Cold are in seconds. Lower is better. Best values are **bold**; second-best values are underlined.

Method	Cost / Question		Runtime	
	Tok.	Calls	QA	Cold
GPT-5.5				
Structured Memory				
Mem0	**0.984**	–	6.63	**1,241**
MemOS	0.985	–	**4.58**	3,930
Self-Evolving Memory				
A-MEM	14.611	–	4.69	145,654
MemRL	3.878	–	8.69	24,920
Human-Designed Harness				
Codex	125	**4.36**	93.24	–
Claude Code	219	7.34	157.31	–
EvoGroup				
H_0	120	14.36	20.13	–
H_*	124	14.46	22.25	–
DeepSeek-V4-Flash				
Structured Memory				
Mem0	1.106	–	4.48	**1,253**
MemOS	**1.090**	–	**3.39**	3,930
Self-Evolving Memory				
A-MEM	4.089	–	4.42	131,314
MemRL	16.523	–	5.79	24,857
Human-Designed Harness				
Codex	554	27.59	166.77	–
Claude Code	477	16.54	138.34	–
EvoGroup				
H_0	119	14.68	28.91	–
H_*	200	**13.45**	27.86	–

Table 12: Evolution-transition statistics of EvoGroup under DeepSeek-V4-Flash. Fail→Pass and Pass→Fail denote changes in prediction correctness between consecutive Harnesses. Retention is the fraction of previously correct cases that remain correct, and Net Correct denotes the change in the number of correct predictions.

Transition	Outcome Changes		Persistent States		Evolution Summary	
	Fail→Pass	*Pass→Fail*	*Always-Pass*	*Always-Fail*	*Retention*	*Net Correct*
$H_0 \rightarrow H_1$	124	34	469	93	93.24%	+90
$H_1 \rightarrow H_2$	73	32	561	54	94.60%	+41
$H_2 \rightarrow H_3$	38	53	581	48	91.64%	−15
$H_3 \rightarrow H_4$	45	49	570	56	92.08%	−4
$H_4 \rightarrow H_5$	53	44	571	52	92.85%	+9

C.3 Working-Memory Case Study

We group questions by the state the Agent needs to track while answering them. Table 14 lists common patterns found in the traces, such as several people, changing decisions, and evidence spread across messages. It gives examples of errors that may occur if the Agent loses track of this state. The table describes observed patterns; it does not give question counts.

New evidence can change how an Agent interprets earlier evidence. The Agent therefore needs to update its view of event order, people, decisions, and remaining search steps. Table 15 shows two runs of the same knowledge-update question in consecutive Harness rounds. Both found the key decision: pause the reporting freeze, name an interim Finance owner, check the affected high-risk metrics again, and delay the final evidence lock until the source extract and baseline date were confirmed.

Table 13: Evolution-transition statistics of EvoGroup under GPT-5.5. Fail→Pass and Pass→Fail denote changes in prediction correctness between consecutive Harnesses. Retention is the fraction of previously correct cases that remain correct, and Net Correct denotes the change in the number of correct predictions.

Transition	Outcome Changes		Persistent States		Evolution Summary	
	Fail→Pass	*Pass→Fail*	*Always-Pass*	*Always-Fail*	*Retention*	*Net Correct*
$H_0 \rightarrow H_1$	63	123	393	141	76.16%	-60
$H_1 \rightarrow H_2$	87	78	378	177	82.89%	$+9$
$H_2 \rightarrow H_3$	153	27	438	102	94.19%	$+126$
$H_3 \rightarrow H_4$	12	219	372	117	62.94%	-207
$H_4 \rightarrow H_5$	150	54	310	206	85.16%	$+96$

Table 14: Question structures that frequently trigger explicit working-memory updates, identified from observed reasoning trajectories rather than benchmark annotations.

Question structure	Working-memory state	Typical failure
Multiple temporal endpoints	**Track:** event–time bindings **Maintain:** valid state at each endpoint	Correct event, wrong temporal state.
Multiple entities or roles	**Track:** entity identities and roles **Maintain:** ownership relations	Evidence mixed across entities.
Evolving decisions or states	**Track:** state transitions **Maintain:** currently valid decision	Outdated or superseded state returned.
Distributed evidence chains	**Track:** retrieved and bridge evidence **Maintain:** unresolved links	Intermediate evidence is lost.
Coverage-sensitive questions	**Track:** searched scopes and candidates **Maintain:** remaining scopes	Search terminates early or repeats scopes.
Competing interpretations	**Track:** candidate interpretations **Maintain:** supporting and conflicting evidence	Premature commitment without revision.

The failed run made three memory writes but mixed a later status update with the decision asked for in the question. The successful run made two writes and kept those states separate. It also used one fewer tool call and about 31% fewer recorded tokens (503K versus 347K). This is a case study across two Harness rounds. It does not show that memory writes alone caused the improvement.

D PROMPTS

D.1 SYSTEM PROMPT

System Prompt

```
You are a group chat agent answering questions about long group
    conversations. You run non-interactively.

This static contract carries only the hard invariants that hold on
    every turn. The current turn's instructions, response schema
    and tool schemas are the operational authority: they define
```

Table 15: Two observed trajectories for the same knowledge-update question in consecutive evolution rounds. These trajectories are illustrative rather than a controlled ablation.

Trace	Statistics	State organization	Result
Fail Iteration 2	3 writes 20 calls 503K tokens	The later implementation status was mixed with the current decision, causing the answer to foreground the wrong state.	Fail
Pass Iteration 3	2 writes 19 calls 347K tokens	The current decision was preserved as the primary state, while the later status was recorded separately as context.	Pass

```
    which tools are actually available, which obligations are still
     open, and which fields the final JSON must contain.

Tools

Exactly five tools exist. There is no shell, no network and no path
     to the raw benchmark dataset.

1. list_files: see which files exist under one allowed root.
2. read_file: read a line range from one allowed text file.
3. grep_search: search one memory layer for evidence rows.
4. write_file: append to or replace a note under memory_store/.
5. create_file: create a new note file under memory_store/.

Call tools directly as API-native function calls: never emit a
    simulated tool_call object and never describe a tool call in
    prose. A turn either submits tool calls or returns exactly one
    final response object, never both.

Memory layers

Always address a layer by logical target plus a relative
    resource_path; never an absolute path.

memory_units/ is read-only and is the only primary evidence layer.
    It is the source group chat converted to JSONL with no loss:
    one message per line, no windows, no slicing, no summary, no
    index. Each line is the untouched source message plus the scope
     key promoted from the container structure (date or domain) and
     group, and it keeps its message_index. A matched line already
    is the evidence, and the same line number reads the original
    text back.

memory_store/ is the only writable file-memory root. Its
    long_term_memory/ directory is retained across questions and
    contains group_profile.md, people/, events/, and any file or
    category you create. Its working_memory/ directory is valid
    only for the current question and holds temporary notes such as
     notes.md.

memory_store/README.md and the category READMEs define write
    routing and are read-only during answering; read them before
    your first write. Harness-owned state files (working_memory/
```

```
    state.json, evidence_links.jsonl) are read-only too, so put
    your content in your own note files.

Notes left by earlier turns are unverified leads, never original
    evidence. No write makes them authoritative.

Rules

1. The question is the immutable task anchor. Never replace it with
     a summary and never drop it while searching.

2. Answer only when retrieved evidence has been verified and is
    sufficient. When the raw corpus holds nothing about what the
    question asks, say exactly that instead of guessing; a hard
    search is never by itself a reason to withdraw the question.

3. Before answering, confirm every component the question requires
    is covered. Never keep only the component that happens to have
    evidence.

4. Resolve exactly one open obligation, or one hop, per grep_search.
     Multi-hop questions need separate searches whose bridge facts
    become the next anchor.

5. Search memory_units, not memory_store, for chat facts, timelines,
     people, tasks, decisions and dates. Consult memory_store for
    navigation and hints only.

6. When a search yields no useful hit, stay in memory_units and
    retry with materially different anchors: another person, task
    phrase, date or month, or group, before considering another
    layer.

7. Treat parent_id as an execution dependency: satisfy or waive a
    parent obligation with evidence before searching its children;
    independent siblings may be searched in parallel.

8. When evidence names the next task, retain that exact task phrase
     as the next hard anchor and demote the person's name to
    ranking context.

9. Resolve relative time words such as today and yesterday against
    the memory unit's own date before making a temporal claim.

10. Do not read after every successful search. Use read_file only
    when the candidate excerpt is ambiguous, truncated, conflicting,
     or missing an exact date, number, temporal endpoint, terminal-
    status phrase or bridge message.

11. Carry useful evidence, anchors and completed obligations across
     tool turns through the state fields the current turn defines.
    The harness keeps the authoritative ledger and the raw tool
    results, so do not copy raw results into the answer.

12. Never store evaluation gold answers or scores, and never
    attempt to modify memory_units/.
```

```
13. Do not fabricate any fact that is not supported by tool
    observations or by memory files you actually read.

14. Judge the budget, not only the evidence. While you still have
    budget, keep searching. Exhausting the budget does not end the
    obligation to answer: work through the materially different
    anchors you can still afford, and do not spend the remaining
    budget on anchors you already know are dead ends. Never guess,
    and never hand back a give-up notice on the strength of an
    unfinished search alone.

15. Keep a running search ledger in memory_store/working_memory/
    notes.md: after each search, append the anchors you used, what
    came back, and what is still missing. When the corpus turns out
     to hold nothing about the question, this ledger is what
    establishes that absence, so the record must be kept as you go,
     not reconstructed at the end.

Response protocol

End the final message with these two lines, in this order, and
    nothing after them:

FINAL ANSWER: <your answer>
CONFIDENCE: <a number between 0 and 1: how sure you are of that
    answer>

When CONFIDENCE is at or below 0.5, add a third block:

ANSWER BIAS:
<your own account of the attempt>

The ANSWER BIAS block must cover: what kind of question this was (a
     time, an event, a person, a relation) and what it was about;
    the information answering it would have needed; the search
    terms you actually used; which tools you called and what they
    returned; what is still missing; and whether the answer was
    lost in your own reasoning or in a named part of this harness (
    a tool, the corpus, the budget). Build the block from the
    ledger you kept in memory_store/working_memory/notes.md, adding
     any search it is missing before you write it, and name the
    harness part by what it is: a tool and its limits, the corpus,
    the budget, the loop, never by an invented id.

The FINAL ANSWER line is the only place the answer appears. A
    multiple-choice letter belongs there and nowhere else, written
    as FINAL ANSWER: <LETTER>. Never send a bare letter, a bare
    number, or a paragraph instead of these lines.

Empty-corpus branch (only when the raw corpus has nothing about the
     question)

Answering is the default outcome. Never withdraw a question because
     retrieval was hard, because the remaining budget looks small,
```

```
    or because the evidence you found does not cover every
    component.

Use this branch only when the tool observations show that
    memory_units/ holds no material about what the question asks,
    not that your anchors failed to reach it. Keep the answer form
    and state the finding plainly on the answer line:

FINAL ANSWER: there is no related query content
CONFIDENCE: <a low number, since the finding is the absence>

Record the searches that establish the absence in the running
    ledger in memory_store/working_memory/notes.md: which anchors
    you tried, in which layer and resource, and what each one
    returned. Report only searches you actually ran; never invent a
     tool call, a hit or a count. Absence is established by those
    searches, so the ledger has to be kept as you go.

Date: {{ date }}
Working Dir: {{ working_directory }}
```

This appendix reproduces the prompts that drive the EvoGroup outer loop and the Round-0 task-agent defaults. The blocks below are the literal contents of the corresponding files in the repository as of the commit that produced the experiments in Section 3. The three prompts are intentionally separated by role: the Answer Agent performs evidence-grounded execution, the Analyze Agent diagnoses traces, and the Evolve Agent proposes and validates general harness changes. None of the prompt text contains a rule specialized to a particular benchmark question type.

D.2 GROUP AGENT PROMPT

Group Agent Prompt

```
You are a group chat agent answering questions about long group
    conversations. You run non-interactively.

Use the available tools to search the conversation evidence. The
    conversation records are the primary evidence. Persistent notes
     are only navigation aids and must not replace the original
    records. Never invent facts, tool calls, search results, or
    evidence.

Keep the user's question as the task anchor. Identify every fact
    needed to answer it, search for each fact, and verify the
    relevant records. For multi-step questions, establish each
    intermediate fact before using it as the next search anchor. If
     a search fails, try materially different names, phrases, dates,
     or groups. Do not treat an unsuccessful search as proof that
    the information is absent.

Check the scope, people, events, dates, and status requested by the
     question. Read surrounding records when a search result is
    ambiguous, incomplete, truncated, or conflicting. Keep a
    running note of searches performed, evidence found, and facts
    still unresolved. Manage the tool budget without repeating
    known dead ends.
```

```
Answer only with supported information. If the conversation
    contains no relevant information after adequate searching,
    state that clearly. The final answer must use this form:
FINAL ANSWER: <complete answer>
CONFIDENCE: <number between 0 and 1>
If confidence is 0.5 or lower, add an ANSWER BIAS block describing
    the searches actually performed, the evidence found, what
    remains unresolved, and the source of the limitation.
```

D.3 Analysis Agent Prompt

Analysis Agent Prompt

```
ou are a analyze agent that analyzes agent execution traces. Base
    your conclusions on the supplied traces and files, not outside
    knowledge.

Locate relevant messages before reading large sections of a trace.
    Search for the question, tool calls, errors, answer markers,
    and evidence anchors, then inspect their surrounding context.
    Never treat an uninspected part of a trace as evidence.

Identify the observed outcome, the evidence path leading to it, and
     the earliest decision or missing capability that explains a
    failure. Distinguish retrieval errors, evidence interpretation
    errors, answer-format errors, budget problems, tool failures,
    and infrastructure exceptions. Cite exact trace identifiers and
     message indices for your conclusions.

When several traces are available, compare them directly and
    identify where successful and unsuccessful runs first diverge.
    Separate consistent failure patterns from variation between
    runs. If the inspected evidence is insufficient, say what was
    checked and what remains unverified.
Return a concise, evidence-linked analysis through the required
    output schema.
```

D.4 Evolve Agent Prompt

PROMPT / MARKDOWN

```
You are the evolution agent. Your objective is to improve the Group
     Agent's single-attempt success rate through changes supported
    by evaluation evidence.

Read the iteration history, analysis overview, detailed findings,
    and prior change evaluations before editing. The evaluation in
    the current iteration describes the workspace produced by the
    previous iteration. Changes made now belong to the current
    iteration and will be evaluated in the next one.
```

```
For every proposed change, identify the failed behavior, the
    evidence showing it, its root cause, the component to modify,
    the expected improvement, and possible regressions. Consider
    the system prompt, tool descriptions, tool implementations,
    memory structure, skills, middleware, and sub-agents. Choose
    the component that addresses the demonstrated cause.

Modify only the designated workspace. Do not change model settings,
     evaluation outputs, verifier logic, traces, or experiment
    infrastructure. Do not encode answers for individual benchmark
    questions or replace original evidence-grounding rules.
After editing, validate the workspace. Record each logical change
    separately, with its rationale, affected files, predicted
    impact, and risks. Write a concise evolution report covering
    the changes and their validation results.
```

E SUPPLEMENTARY CASE

In this section, we select four EverMemBench questions to examine distinct points at which an agent can fail: identifying the relevant event boundary, choosing the unit of duration, delivering a multiple-choice answer, and combining evidence from two tasks.

Each case compares the observed pass@1 attempt for the same question in consecutive evaluation rounds. These examples illustrate mechanisms rather than their prevalence. The attempts are not documented as matched-seed replays, and several Harness edits may enter at each round boundary.

E.1 RECOVERING THE RELEVANT TASK BOUNDARY

This case shows how a retrieved date can still support the wrong answer when the agent assigns it the wrong role in the task timeline. The iteration-2 attempt focused on later CI activity and answered 2 days without opening the May 15 delivery record. In iteration 3, the agent connected that delivery to the June 16 announcement of Si Jia's next task and returned the benchmark's 32-day interval. Three retrieval changes entered together, so the observed recovery is consistent with improved scoped search but cannot be attributed to one change. The question had also passed in an earlier round; this is a recovery from a regression, not a first-time solution.

Recovering the Relevant Task Boundary

Trace

Iteration 2 centered on the June CI pipeline. The agent treated its June 17 deployment as completion and June 19 Zustand work as the next task, producing 2 days. It did not use the May 15 framework-delivery record.

Harness Edit

The retrieval tool promoted requested anchors to a hard OR group, added a date-ordered fallback for empty scoped searches, and allowed group to constrain retrieval.

Query and Revised Trace

The question asks for the interval between Si Jia's frontend-framework delivery and her next independent task in the Health Records group. In iteration 3, the agent used corpus.jsonl:3424 (May 15 delivery) and corpus.jsonl:4412 (June 16 task start), answering 32 days.

Mechanism

The decisive difference is the assignment of the correct roles to two events: completion of the named task and start of the subsequent task. The comparison does not isolate which retrieval edit was responsible.

Figure 8: Event-boundary recovery in `F_MH_Top02_028`. Consecutive pass@1 attempts select different completion and next-task events, yielding 2 versus 32 days.

E.2 Counting One Shared Task Once

The iteration-5 agent interpreted the two named contributors as two additive durations, even though the benchmark treats their shared stress test as one task. Its 6-day answer also depended on a later start date, October 16. The iteration-6 attempt recovered the October 14 announcement, measured the task through October 20, and reported 5 working days without multiplying by the number of contributors. The contrast makes the counting-unit problem visible, while the simultaneous date correction limits causal attribution to the new prompt sentence. This task passed in iteration 4, failed in iteration 5, and passed again in iteration 6, illustrating non-monotonic evolution.

Recovering the Relevant Task Boundary

Trace

Iteration 5 selected October 16–20. day_span returned 3 working days, which the agent added once for Xiaofang Tang and once for Peng Sun: 3 + 3 = 6. The verifier rejected the answer.

Harness Edit

The system prompt clarified: "Two names in a question are two people, not two tasks." A jointly owned task is counted once; durations are added only for different deliverables.

Query and Revised Trace

The question asks how many working days the two colleagues spent on one high-concurrency API stress test. Iteration 6 used the October 14 task announcement (corpus.jsonl:8143) and October 20 completion (corpus.jsonl:8324). day_span returned 5 working days, counted once; the verifier accepted it.

Mechanism

The failed attempt made both a start-date error and a counting-unit error. The passing attempt corrected both, so the prompt rule's individual effect cannot be isolated.

Figure 9: Counting-unit error in `F_TP_Top02_018`. The failed attempt summed per-person durations; the passing attempt measured the shared task once.

E.3 Losing a Reasoned Choice at the Answer Boundary

This case concerns the boundary between an agent's visible analysis and its scored response. In iteration 5, the truncated final message argued against C and in favor of D, yet contained no valid answer block; the recorded answer was C. After the answer-contract wording was repaired, the iteration-6 attempt began its final response with FINAL ANSWER: D and passed. The two attempts also differed in their searches, so the comparison supports an answer-delivery diagnosis without proving that the prompt repair alone produced the entire outcome. Moreover, D is the benchmark's keyed choice based on role evidence; the retrieved corpus does not contain a verbatim decision on the exact prohibit-versus-alert interaction.

Recovering the Relevant Task Boundary

Trace

In iteration 5, the final text rejected Option C's proposed compliance team and stated that Option D fit Guoxin Pan's product-logic responsibility. The output then stopped mid-sentence without a FINAL ANSWER block. The recorded answer was C, which failed.

Harness Edit

The prompt's answer contract was repaired to state: "The final message is the answer block and nothing else."

Query and Revised Trace

The question asks who ultimately decides the Medication Assistant's contraindication interaction. Iteration 6 emitted FINAL ANSWER: D; the recorded answer D matched the benchmark key.

Mechanism

Evidence interpretation and answer delivery are separate requirements. The first trace points toward D but does not establish what the model would have submitted had it continued.

Figure 10: Answer-boundary failure in `MA_C_Top02_054`. A truncated response that favored D was recorded as C; the following round delivered the benchmark-keyed answer in the required block.

E.4 Completing an Answer Across Two Tasks

The iteration-5 attempt encountered two separate difficulties. It matched the browser-testing description to an October Group 3 interval rather than the named App-interface task, and its final message stopped at an intermediate "Key findings:" header. The iteration-6 attempt read the App-interface start and completion records (corpus.jsonl:9003,9172) and the milestone-plan records (corpus.jsonl:733,830). Separate day_span calls returned 5 and 3 working days, and the agent submitted their sum, 8, in a complete answer block. This case therefore illustrates coordinated evidence selection, arithmetic, and answer delivery. It does not show that the prompt edit alone corrected the evidence match.

Recovering the Relevant Task Boundary

Trace

Iteration 2 associated Peng Sun's browser testing with a different Group 3 task dated October 13–17. It found the milestone-plan interval but ended while proposing further verification. The recorded answer was only "Key findings:", and the verifier failed it.

Harness Edit

The same repaired answer contract required the final message to contain the answer block rather than an unfinished analysis.

Query and Revised Trace

The question asks for the combined workdays of App-interface browser compatibility testing and the Health Records milestone plan. Iteration 6 used Group 1 evidence for November 12–18 (5 working days) and Group 2 evidence for February 14–18 (3 working days), then submitted FINAL ANSWER: 8 working days.

Mechanism

Success required both matching each interval to the named deliverable and completing the final answer. The per-question working-memory note was write into the memory_store for evolution edit.

Figure 11: Two-task answer completion in `F_TP_Top02_035`. The passing attempt identifies the named deliverables, computes 5 + 3 working days, and submits the result in a complete answer block.