# AMA-BENCH: EVALUATING LONG-HORIZON MEMORY FOR AGENTIC APPLICATIONS

**Yujie Zhao**[1], **Boqin Yuan**[1], **Junbo Huang**[1], **Haocheng Yuan**[1], **Zhongming Yu**[1], **Haozhou Xu**[1], **Lanxiang Hu**[1], **Abhilash Shankarampeta**[1], **Zimeng Huang**[1], **Wentao Ni**[1], **Yuandong Tian**[2], **Jishen Zhao**[1]
[1]University of California, San Diego, USA, [2]Independent Research

## ABSTRACT

Large Language Models (LLMs) are deployed as autonomous agents in increasingly complex applications, where enabling long-horizon memory is critical for achieving strong performance. However, a significant gap exists between practical applications and current evaluation standards for agent memory: existing benchmarks primarily focus on dialogue-centric, human-agent interactions. In reality, agent memory consists of a continuous stream of agent-environment interactions that are primarily composed of machine-generated representations. To bridge this gap, we introduce **AMA-Bench** (**A**gent **M**emory with **A**ny length), to evaluates long-horizon memory for LLMs in real agentic applications. It features two key components: (1) a set of real-world agentic trajectories across representative agentic applications, paired with expert-curated QA, and (2) a set of synthetic agentic trajectories that scale to arbitrary horizons paired with rule-based QA. Our comprehensive study shows that existing memory systems underperform on AMA-Bench primarily because they lack causality and objective information, and are constrained by the lossy nature of similarity-based retrieval employed by many memory systems. To address these limitations, we propose **AMA-Agent**, an effective memory system featuring a causality graph and tool-augmented retrieval.

**Code** **Dataset** **Leaderboard**

## 1 INTRODUCTION

Large Language Models (LLMs) have rapidly evolved from solving closed-form reasoning tasks (Fig. 2 (a)) and serving as chatbots (Fig. 2 (b)), to serving as autonomous agents (Fig. 2(c)). Autonomous agents require long-horizon reasoning and experience reuse to complete tasks like open-space navigation, code editing, and web search. To empower LLMs with these capabilities, agent memory has become an important component of agent design to manage LLM context Wang et al. (2025b); Agashe et al. (2024). Strong memory modules are expected to satisfy two core capabilities: (1) effective memory processing, where complete agentic trajectories are transformed into structured factual representations, such as summaries, fact tables, or embeddings Edge et al. (2025); Packer et al. (2023); Liu et al. (2026); and (2) effective memory retrieval, reliably selecting and leveraging the most relevant memory to guide decision-making. Existing benchmarks typically evaluate these capabilities in dialogue-centric or synthetic retrieval tasks Hsieh et al. (2024); Maharana et al. (2024), focusing on specific subcomponents, such as single- or multi-hop questions for memory retrieval or state updating and memory condensation questions for memory processing. There is a lack of benchmarks and evaluations of memory modules in long-horizon agentic tasks.

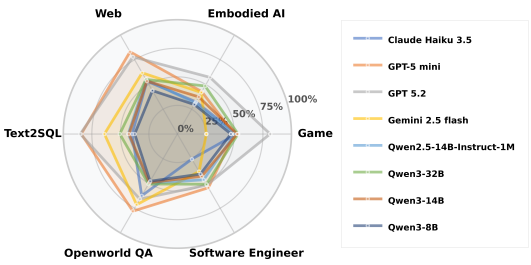


Figure 1: Model performance across agent task families in AMA-Bench.

Real-world agents mainly operate in machine generated environments such as databases, code executors, and web interfaces, where they must process large volumes of *machine generated representations*.

Yet, most existing memory benchmarks are still natural language centric and have three key limitations: (1) **A lack of representation types**: Agent trajectories encompass diverse machine generated symbolics (e.g., ASCII tables, JSON data, Unicode snippets, Python or HTML code blocks), whereas current benchmarks predominantly center on free-form natural languages; (2) **A lack of causality**: agent trajectories are causally grounded, where each action induces a latent environment state transition that constrains subsequent observations; but existing benchmarks follow unconstrained linguistic flow; (3) **Sparse objective information**: agent trajectories are machine-generated and objective, whereas dialog-centric benchmarks contain abundant redundant information such as phatic chit-chat.

To bridge this gap, we introduce AMA-Bench (Benchmarking Agent Memory with Any length), which comprises two complementary subsets: a real-world subset and a synthetic subset. The real-world component consists of expert-annotated and sanity-checked Question-Answer (QA) pairs sourced from six representative agent domains: Web, open-world QA, Text2SQL, Software Engineering, Gaming, and Embodied AI (see Fig. 3). Furthermore, we construct a synthetic subset in programmatic agent environments with automatically generated QA pairs. This design enables controlled synthesis of tasks at arbitrary horizons while keeping the agent-environment interaction pattern.

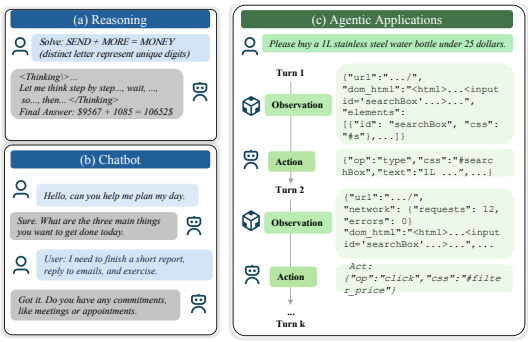


Figure 2: Memory comparison across reasoning, chatbots, and agents. Agent trajectories exhibit unique properties: causally grounded, diverse symbolic artifacts, and dense objective information.

As shown in Fig. 1, our systematic evaluation indicates that agent memory remains challenging even for frontier commercial models, with GPT 5.2 achieving only 72.26% accuracy. Evaluating memory systems on AMA-Bench yields **three key insights**: (1) While existing agent memory techniques often outperform long-context LLM baselines on dialogue-centric benchmarks, they fall short to the baselines in many long-horizon agentic tasks, highlighting the unique diagnostic value of our benchmark (Sec. 4); (2) Our analysis reveals that suboptimal memory system design, rather than base model capability, serves as the primary bottleneck for their poor performance (Sec. 4); (3) Existing lossy compression and similarity-based retrieval techniques are insufficient for the nuanced demands of agent memory, necessitating a paradigm shift toward agent-centric memory management strategies (Sec. 4).

Motivated by these insights, we present AMA-Agent, a framework designed to address the memory demands of agentic applications. Moving beyond lossy compression or similarity-based graphs, AMA-Agent implements a Causality Graph to preserve the objective information and explicit causal dependencies within interaction histories. To transcend the limitations of similarity-based retrieval, we introduce a Hybrid Tool-Augmented Retrieval mechanism. This approach enables more efficient information extraction and synthesis in machine-generated representations.

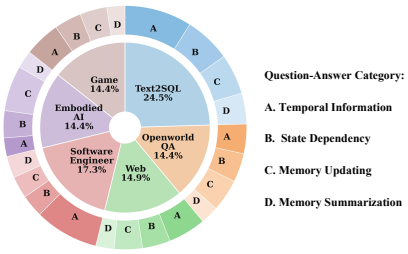


Figure 3: Domain and question type distribution in AMA-Bench.

This paper makes the following main contributions:

**AMA-Bench.** We introduce the first benchmark suite built for evaluating memory in agent applications, AMA-Bench, with two complementary subsets: *Real world* preserves authentic machine-generated interaction patterns, and a *Synthetic* suite that enables controlled scaling of any horizon length and complexity.

**Comprehensive Evaluations.** Through comprehensive evaluation using AMA-Bench, we show that many existing agent memory designs underperform the long-context baseline, as errors introduced by lossy memory compression and similarity-based retrieval accumulate and compound over long-horizon agentic tasks; this highlights the critical need for agent-centric memory designs.

**AMA-Agent.** We propose AMA-gent that addresses the identified bottlenecks with two mechanisms: (i) Causality Graph that preserves the integrity of objective information and causal dependencies, and

Table 1: Comparison of memory benchmarks. NL: Natural Language.

| Category | Benchmark | Interaction Paradigm | Content Source | Average Length (tokens) | Representation Types | Memory Organization |
|---|---|---|---|---|---|---|
| Dialogue-Centric | LoCoMo (Maharana et al., 2024) | Dialogue | Real + Synthetic | 9K | NL + Vision | Episodic |
| | LongMemEval (Wu et al., 2025) | Dialogue | Synthetic | 115K | NL | Multi-session |
| | MemoryAgentBench (Hu et al., 2025b) | Dialogue | Real + Synthetic | 100K–500K | NL | Multi-domain |
| | MemoryBench (Ai et al., 2025) | Dialogue | Real + Synthetic | ∼30–380K | NL | Multi-session + Feedback |
| | RealTalk (Lee et al., 2025) | Dialogue | Real | 17K | NL | Multi-day |
| Long-Context | QuALITY (Pang et al., 2022) | Long-Context | Real | 5K | NL | Multi-hop |
| | RULER (Hsieh et al., 2024) | Long-Context | Synthetic | 4K–128K | NL | Single-turn |
| | LongBench v2 (Bai et al., 2025) | Long-Context | Real | 8K–2M | NL | Document-level |
| Agent-Centric | **AMA-Bench** | **Agent-Env** | **Real + Synthetic** | **57K** | **NL + Machine** | **Trajectory-based** |

(ii) Tool-Augmented Retrieval, which utilizes both graph node search and keyword-based search. Experimental results show that AMA-Agent outperforms the strongest existing memory baselines by 11.16% on average.

## 2 Related Work

### 2.1 Agent Memory Evaluation

We categorize existing memory benchmarks into two primary classes (as shown in Tab. 1): Dialogue-Centric and Long-Context. Dialogue-centric benchmarks evaluate memory retention over multi-turn human-agent interactions. LoCoMo (Maharana et al., 2024) and LongMemEval (Wu et al., 2025) evaluate long-term interactive memory in assistant-style chats; MemoryAgentBench (Hu et al., 2025b) tests multiple long-term memory competencies across diverse memory capabilities; MemoryBench (Ai et al., 2025) unifies diverse memory tasks into a continual learning suite; and RealTalk (Lee et al., 2025) grounds long-term memory evaluation in multi-day human dialogues. Long-context benchmarks such as QuALITY (Pang et al., 2022), RULER (Hsieh et al., 2024), and LongBench v2 (Bai et al., 2025) evaluate static document-level reasoning, focusing on multi-hop comprehension over long inputs rather than interactive or incremental memory. In contrast, AMA-Bench evaluates memory for agent applications, where the interaction trajectory is characterized by machine-generated representations, causal dependencies, and dense, objective information.

### 2.2 Agent Memory Mechanisms

Three main approaches have been explored to equip agents with long-horizon memory.

**Long-Context Models.** The first direction adapts LLMs to process memory directly as the context. For instance, GPT 5.2 (OpenAI, 2025) exposes an effective context window of approximately 400,000 tokens. On the open-source side, the Qwen2.5 1M (Yang et al., 2025b) series extends models to 100,000 tokens. Although simple and often strong in practice, this strategy remains bounded by physical context limits.

**Retrieval-Augmented Generation (RAG).** Another prominent research direction is RAG, which externalizes information into external storage during the memory construction stage and fetches relevant items based on similarity to augment the model's context during retrieval. Traditional methods, such as BM25 (Robertson & Zaragoza, 2009) and the Qwen3 Embedding series (Zhang et al., 2025b), store memory by partitioning data into discrete chunks. Structured RAG approaches have emerged to capture more complex relationships. For instance, GraphRAG (Edge et al., 2025) utilizes graph-based retrieval by constructing and aggregating entity-document graphs to capture these structural dependencies. Furthermore, HippoRAG2 (Gutiérrez et al., 2025) formalizes retrieval as a nonparametric form of Despite these advances, existing methods primarily rely on similarity-based or entity-centric retrieval, often neglecting the underlying causality within stored information.

**Memory Agent Systems.** Recent research has shifted from rule-based RAG pipelines to agent-centric memory management, where LLM agents autonomously decide how to perform memory construction and retrieval. MemoryBank (Zhong et al., 2023) enables models to autonomously summon and update the stored memories. MemGPT (Packer et al., 2023) formulates memory access as a decision problem, where the LLM learns when to retrieve and how to manage the long-term context. MemoRAG (Qian et al., 2025) proposes a dual system RAG architecture that maintains a global memory store and retrieves semantically similar clues to assemble a high-level draft for answering. MEM1 (Zhou et al.,

2025), Mem-$\alpha$ (Wang et al., 2025a), Mem0 (Chhikara et al., 2025), MemAgent (Yu et al., 2025), and A MEM (Xu et al., 2025) construct memory by iterative compression or edit-style operations such as insertion, deletion, and modification, and then directly condition generation on the compressed memory. SimpleMem (Liu et al., 2026) introduced a structured compression pipeline that filters redundancy, organizes memories into hierarchies, and adaptively retrieves relevant contexts. However, memory compression and similarity-based retrieval perform poorly on agent memory tasks for two main reasons. First, most compression methods are designed for natural language, where redundancy and subjective fillers are common. However, agent trajectories contain dense, causally structured state transitions. Second, agent memories are often machine-generated representations, and similarity retrieval frequently fails to extract the required evidence.

# 3 AMA-BENCH

## 3.1 PROBLEM FORMULATION

**Agent Environment Interactions.** We consider LLM agents operating within the reason and act paradigm Yao et al. (2022); Shinn et al. (2023); Wang et al. (2023), where sequential decision-making is formulated as a Partially Observable Markov Decision Process (POMDP) defined by the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{O}, P, r)$ (See Fig. 4 (a)). At each time step $t$, the environment resides in a latent state $s_t \in \mathcal{S}$. Upon executing an action $a_t \in \mathcal{A}$, the agent receives an observation $o_t \in \mathcal{O}$ sampled from the observation function $O(s_t)$, and the environment transitions to $s_{t+1}$ according to the dynamics $P(s_t \mid s_t, a_{t+1})$. Given a task instruction $x$, the interaction generates a trajectory history $h_t = (x, a_1, o_1, ..., o_t)$. The partial observability motivates an explicit memory mechanism to persist the agent memory.

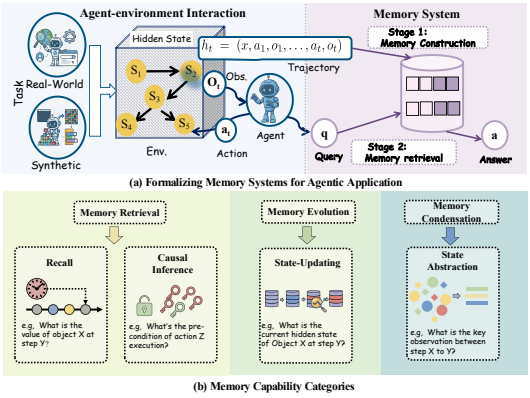


Figure 4: Formalizing memory system and capability for agentic applications.

**The Memory System.** We formalized a memory system through two stages (see Fig. 4 (a)), memory construction (Build) and memory retrieval,(Retrieve). The construction stage, $\mathsf{Build}: \mathcal{H} \to \mathcal{M}_{\text{mem}}$, maps the interaction history $h_t$ to an external memory state $m_t \in \mathcal{M}_{\text{mem}}$. The memory space $\mathcal{M}_{\text{mem}}$ accommodates diverse structured representations, such as recursive summaries, knowledge graphs, and vector embeddings Edge et al. (2025); Packer et al. (2023); Liu et al. (2026). Upon receiving a query $q_t$, the retrieval module extracts a query-relevant context $c_t = \mathsf{Retrieve}(m_t, q_t)$. The agent policy $\pi$ then determines the response based on the retrieved context and query: $a_t \sim \pi(\cdot \mid q_t, c_t)$.

## 3.2 MEMORY CAPABILITY CATEGORIES

The proposed formulation, supported by recent literature Du et al. (2025); Zhang et al. (2024), underscores that an effective memory system must facilitate three core capabilities: 1. Memory Retrieval: targeted access to the correct evidence. 2. Memory Evolution: continually updates memory as new observations arrive. 3. Memory Condensation: precisely extracting and condensing memory without information loss. Aligning these essential mechanisms with the specific requirements of agent-based tasks, we categorize memory capabilities into three functional modules. The formal definitions are detailed in Tab. 2, while illustrative examples are provided in Fig. 4 (b). These modules encompass: Recall and Causal Inference (Retrieval), State Updating (Evolution), and State Abstraction (Condensation).

Table 2: Memory capability described in Sec 3.2. We group evaluation dimensions into three mechanisms and four capabilities.

| Mechanism | Capability | Description |
|---|---|---|
| **Memory Retrieval** | A. Recall | Identification of Temporal and sequential information. |
| | B. Causal Inference | Verification of action preconditions and dependency relations between states. |
| **Memory Evolution** | C. State Updating | Tracking updates to states, including explicit observations and hidden states. |
| **Memory Condensation** | D. State Abstraction | Filtering redundant content while extracting precise and condensed key information. |

### 3.3 BENCHMARK CONSTRUCTION

With the above formulation and capability taxonomy, we now describe how we build AMA-Bench to jointly capture real-world complexity and provide controllable scaling complexity. AMA-Bench comprises two complementary components: (i) a real-world subset and (ii) a synthetic subset.

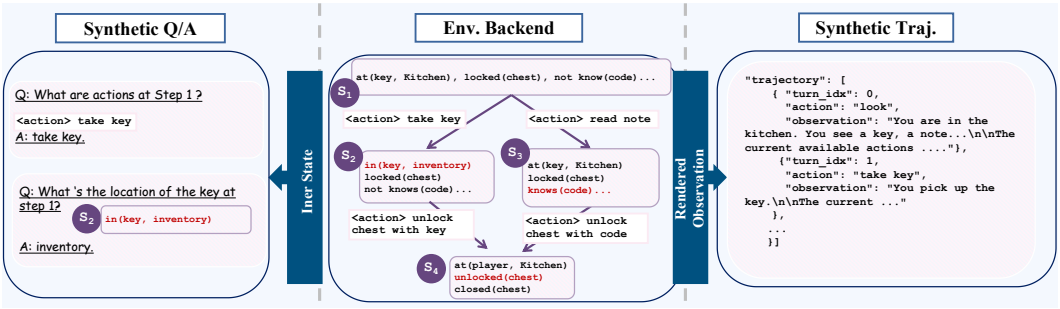


Figure 5: Synthetic subset construction pipeline. We synthesize an executable environment backend with explicit latent states and transitions, render machine-generated observations to form trajectories, and programmatically generate trajectory-grounded QA pairs.

#### 3.3.1 REAL-WORLD SUBSET

We curate high-quality, long-horizon trajectories from six representative real-world agent tasks, including web navigation, software engineering, text-to-SQL, embodied AI, gaming, and open-world tool with 2496 QA pairs (see Fig. 3 and Tab. 8 in Appendix A for the detailed category). For each task family, we gathered action-observation interaction traces from representative benchmarks using either state-of-the-art agent frameworks or expert-level trajectories provided directly by the environment. From this pool, we curated a subset for annotation, prioritizing longer trajectories while maintaining the original task distribution within each family. Specific details regarding the benchmarks and frameworks used are provided in Appendix A.

The real-world environments are treated as black boxes: we only observe agent-environment interaction logs (action and observation trajectories) and do not have access to the environment backend state. Building on the capability taxonomy in Sec. 3.2, we manually annotate each selected trajectory with 12 memory-intensive QA pairs that collectively cover all categories in Tab. 2. Each question is formulated such that its answer is supported by explicit and unambiguous evidence within the trajectory, ensuring that the correctness of the question can be verified from the log itself.

QA pairs are then authored by graduate-level annotators with research experience in LLM agents, following shared guidelines that standardize evidence grounding and category coverage across six domains. To improve annotation reliability, each annotated trajectory undergoes a cross-review sanity check by a second annotator. This protocol yields expert-level QA annotations that are trajectory-grounded, category-aligned, and consistent across the task families. Examples of trajectories and QA are listed in Appendix E.1.

#### 3.3.2 SYNTHETIC SUBSET

To systematically evaluate agent memory scaling, we construct a synthetic subset via programmatic environment synthesis. Each instance comprises an executable backend with controllable state transitions and a tunable perception interface, enabling the generation of verifiable trajectories with arbitrary horizons. Fig. 5 shows the pipeline of the synthetic subset construction. We synthesize tasks from two distinct environments characterized by long-range dependencies and partial observability: TextWorld Côté et al. (2018), BabyAI Chevalier-Boisvert et al. (2019). The description of the two tasks is listed in Appendix A.2.

**Environment Synthesis.** We parameterize each instance using a difficulty vector $\phi$ and a random seed to synthesize the environment backend. The latent state $s_t$ and transition kernel $s_{t+1} = P_\phi(s_t, a_{t+1})$ are programmatically defined and machine-verifiable. This allows us to systematically scale the interaction context length $L$ by increasing the environmental difficulty as dictated by $\phi$. For instance,

in BabyAI, $\phi$ encompasses parameters such as the grid dimensions, the number of rooms, and the length of the instruction chain. By increasing the map size or adding nested dependencies, we can provably extend the trajectory length $L$.

**Trajectory Synthesis.** The synthetic nature of our environments grants full access to the environment MDP, enabling the derivation of an optimal policy $\pi^*$ despite the agent's partial observability. We generate stepwise sequences $\{(a_t,o_t)\}_{t=1}^{T}$ grounded in these gold-standard transitions to resolve the issue of low-structural density. To further address representational diversity and robustness, we introduce two auxiliary perturbations beyond $\phi$: (1) Action Stochasticity ($\epsilon$): we inject random noise into $\pi^*$ to simulate sub-optimal action ratios, testing memory robustness under varying agent policies; and (2) Observation Verbosity ($\gamma$): we employ various symbolic representations with controllable descriptive granularity $\gamma$ for $o_t$.

**QA Synthesis.** Since we have access to the full MDP, we can programmatically generate golden QA pairs anchored to backend state variables such as state $s_t$ or transition kernel $s_{t+1} = P_\phi(s_t,a_t)$.

**Needle Synthesis.** Following the widely used needle-in-a-haystack (NIAH) paradigm Nelson et al. (2024); Hsieh et al. (2024); Kamradt for evaluating memory capabilities, we also instantiate a needle protocol in AMA-Bench. The *needle* here is the minimal set of trajectory turn IDs that contains all the evidence necessary to answer a query. Crucially, because AMA-Bench is backed by a programmatic environment, we can automatically synthesize and verify the needles. More details about the needle generation pipeline are listed in Appendix F.

## 4 Empirical Motivation

We benchmarked a broad set of representative memory systems on the AMA-Bench (see Fig. 6). The results reveal three key empirical insights that highlight the current limitations and directly motivate the design of our proposed method in Sec. 2.2.

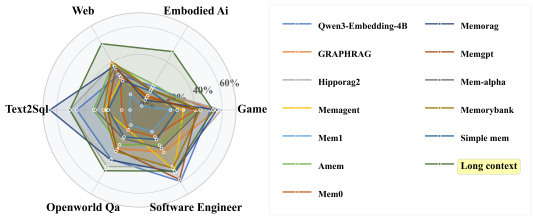


Figure 6: LLM-as-a-judge accuracy across different memory systems using Qwen3-32B as the base model.

### Motivation1: Memory systems fall short of the long-context baseline.

Fig. 6 compares representative memory systems against a long context baseline across six agent task families. A clear pattern emerges: the long context baseline is consistently strong and often achieves the best performance, whereas existing memory systems exhibit large variance across families and frequently underperform, even when they introduce structured memory construction or retrieval augmentation.

### Motivation2: Memory Design bottlenecks the model performance.

A central question in building memory-augmented agents is whether performance bottlenecks reside in the backbone capacity or memory system design. Fig. 7 illustrates that scaling from 8B to 32B provides only marginal improvements (avg. improvement is 0.038), whereas varying the memory architecture induces significantly higher variance, with score ranges reaching 0.45.

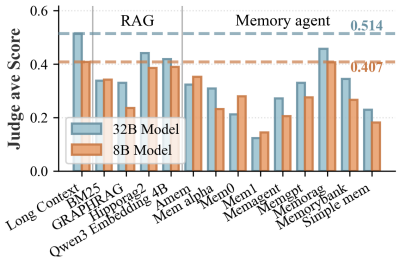


Figure 7: Impact of Model Scale vs. Memory Architecture.

### Motivation3: Limitations of Existing Memory System Designs.

To further pinpoint bottlenecks, we performed needle protocol ablation in BabyAI with three settings. *Full Observation (Needle)* provides the raw needle turns and serves as an upper bound. *Constructed Memory (Needle)* replaces them with method specific constructed memory to isolate construction loss. *End to End System* evaluates the full pipeline with retrieval.

Tab. 3 demonstrates two limitations. First, many methods degrade sharply after construction, e.g., MemoryBank drops by 41.3%, suggesting that compression tuned for redundant natural language fails to preserve dense state and causal information in agent memory. Second, similarity-based retrieval is unreliable: HippoRAG2 remains strong under constructed memory with needle turn but drops by 43.2% performance end-to-end.

| Method | Full Observation w/ Needle ACC | Consturcted Memory w/ Needle ACC | End to End System ACC |
|---|---|---|---|
| HippoRAG2 | 0.46 | 0.37 (−19.6%) | 0.21 (−43.2%) |
| Mem1 | | 0.29 (−37.0%) | 0.20 (−31.0%) |
| AMem | | 0.29 (−37.0%) | 0.24 (−17.2%) |
| MemoryBank | | 0.27 (−41.3%) | 0.26 (−3.7%) |

Table 3: Ablation study on performance bottlenecks under the needle protocol in BabyAI, evaluated by **accuracy (ACC)**. Dark red values denote relative decrease vs. the previous column.

# 5 The AMA-Agent

Motivated by the observations in Sec. 4, we develop the AMA-Agent with two core mechanisms: (A) a Causality Graph for memory construction to minimize information loss; and (B) a tool augmented retrieval module that complements standard retrieval with graph node traversal and keyword search to improve retrieval effectiveness.

## 5.1 Memory Construction: Causality Graph

AMA-Agent constructs a **Causality Graph** from the agent's trajectory. The construction proceeds in three stages: For each timestep $t$, the agent parses the adjacent turn pairs $(o_{t-1}, a_t, o_t)$ to extract environment and object states, identifying latent inter-state causal dependencies and state-object associations (Fig. 8 (A) 1). These signals are instantiated as directed causality edges and undirected association edges connecting the respective state nodes (Fig. 8 (A) 2). Finally, these local interactions are integrated into a global Causality Graph, where nodes are mapped into a latent embedding space to facilitate similarity-based retrieval and relational reasoning (Fig. 8 (A) 3 ).

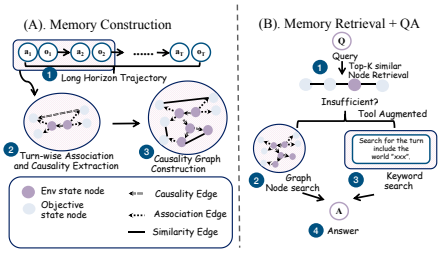


Figure 8: Overview of the AMA-agent. (A) Illustrates the transition from trajectories to a structured causality graph. (B) depicts the retrieval mechanism, utilizing tool-augment search.

## 5.2 Memory Retrieval: Tool-Augmented Search

Beyond similarity-based retrieval, AMA-Agent adopts a tool-augmented search mechanism. It first retrieves the top $K$ nodes based on embedding similarity (Fig. 8 (B) 1) and performs self-evaluation to assess whether the retrieved evidence is sufficient to answer the query. If the evidence is insufficient, the agent categorizes the missing context and invokes either the *graph node search tool* or the *keyword search tool*.

Under the *graph node search tool* route, the agent performs depth-controlled neighborhood traversal to aggregate multi-hop context and causal relations (Fig. 8 (B) 2). Under the *keyword search tool* route, the AMA-Agent uses a tool interface that allows it to write and execute scripts for programmatic analysis, enabling precise keyword matching and statistical aggregation (Fig. 8 (B) 3). Finally, the AMA-Agent synthesizes the retrieved evidence to produce a response (Fig. 8 (B) 4).

# 6 Evaluation

## 6.1 Experimental Setup

**Benchmarks.** We evaluate our baselines on two complementary subsets: 1. Real-world Subset: This subset comprising a total of 2,496 QA pairs. 2. Synthetic Subset: we utilize two tasks with a total of 1,200 QA pairs. These tasks are stratified across five trajectory lengths (8K,16K,32K,64K, and 128K tokens), with 240 samples per interval.

**Baselines.** We consider three categories of baselines: long context models, RAG, and memory agents.

Table 4: Performance of different models on real-world subset.

| Method | Recall | Causal Inference | State Updating | State Abstraction | Average |
|---|---|---|---|---|---|
| Claude Haiku 3.5 (Anthropic, 2025) | 0.4943 (0.3510) | 0.4507 (0.2792) | 0.4287 (0.3015) | 0.3090 (0.2648) | 0.4361 (0.3067) |
| GPT-5-mini (OpenAI, 2025) | 0.6951 (0.4010) | 0.7157 (0.3027) | **0.6575** (0.3288) | **0.6235** (0.3262) | 0.6784 (0.3464) |
| GPT 5.2 (Wailgum, 2025) | **0.7741** (0.4758) | **0.8047** (0.3512) | 0.6563 (0.3686) | 0.6037 (0.3582) | **0.7226** (0.3988) |
| Gemini 2.5 flash (Gemini Team, 2025) | 0.5834 (0.3682) | 0.5087 (0.2628) | 0.5000 (0.2395) | 0.4196 (0.2361) | 0.5168 (0.2878) |
| Qwen2.5-14B-1M (Yang et al., 2025b) | 0.5570 (0.4157) | 0.4111 (0.3209) | 0.4728 (0.3348) | 0.3368 (0.3560) | 0.4638 (0.3622) |
| Qwen3-32B (Yang et al., 2025a) | 0.6149 (0.4074) | 0.5178 (0.3289) | 0.4903 (0.3334) | 0.3657 (0.3172) | 0.5181 (0.3545) |
| Qwen3-14B (Yang et al., 2025a) | 0.5675 (0.3636) | 0.4430 (0.2931) | 0.4502 (0.3204) | 0.3176 (0.2716) | 0.4659 (0.3203) |
| Qwen3-8B (Yang et al., 2025a) | 0.5024 (0.3801) | 0.3776 (0.2830) | 0.3987 (0.3177) | 0.2923 (0.2792) | 0.4109 (0.3240) |

*Note: Results are reported as Accuracy (F1). The **best** and second best Accuracy values are highlighted.*

Table 5: Performance comparison of Agent Memory and RAG methods using base model Qwen-32B on real-world subset.

| Method | Recall | Causal Inference | State Updating | State Abstraction | Average |
|---|---|---|---|---|---|
| **RAG** | | | | | |
| BM25 | 0.3301 (0.1465) | 0.4264 (0.1549) | 0.3450 (0.1325) | 0.2498 (0.1623) | 0.3436 (0.1475) |
| Qwen3-Emb-4B Zhang et al. (2025b) | 0.4843 (0.1590) | 0.4974 (0.1549) | 0.3520 (0.1353) | 0.3011 (0.1610) | 0.4227 (0.1522) |
| GraphRAG Edge et al. (2025) | 0.3077 (0.2769) | 0.3905 (0.2634) | 0.3140 (0.2551) | 0.2879 (0.2588) | 0.3258 (0.2650) |
| HippoRAG2 Gutiérrez et al. (2025) | 0.4579 (0.2356) | 0.5080 (0.1966) | 0.4403 (0.1892) | 0.3538 (0.1785) | 0.4480 (0.2048) |
| **Agent Memory Methods** | | | | | |
| MemAgent Yu et al. (2025) | 0.2550 (0.1489) | 0.3380 (0.1606) | 0.2849 (0.1432) | 0.2202 (0.1655) | 0.2768 (0.1530) |
| Mem1 Zhou et al. (2025) | 0.1180 (0.1857) | 0.1427 (0.1732) | 0.1205 (0.1659) | 0.1080 (0.2042) | 0.1229 (0.1807) |
| Amem Xu et al. (2025) | 0.3084 (0.2707) | 0.3653 (0.2731) | 0.3088 (0.2480) | 0.2873 (0.2953) | 0.3186 (0.2695) |
| Mem0 Chhikara et al. (2025) | 0.2011 (0.2413) | 0.2645 (0.2443) | 0.2101 (0.2225) | 0.1516 (0.2241) | 0.2104 (0.2343) |
| MemoRAG Qian et al. (2025) | 0.4708 (0.1789) | 0.5497 (0.1811) | 0.4257 (0.1713) | 0.3659 (0.2073) | 0.4606 (0.1822) |
| MemGPT Packer et al. (2023) | 0.3289 (0.1318) | 0.4404 (0.1475) | 0.2809 (0.1259) | 0.2526 (0.1431) | 0.3304 (0.1359) |
| Mem-alpha Wang et al. (2025a) | 0.2876 (0.2325) | 0.4172 (0.1993) | 0.3064 (0.2000) | 0.2171 (0.2135) | 0.3117 (0.2130) |
| MemoryBank Zhong et al. (2023) | 0.3231 (0.3128) | 0.4100 (0.2861) | 0.3006 (0.2678) | 0.3332 (0.3011) | 0.3397 (0.2928) |
| Simple mem Liu et al. (2026) | 0.2012 (0.2039) | 0.1884 (0.1612) | 0.1764 (0.1594) | 0.1373 (0.1689) | 0.1811 (0.1764) |
| **AMA-Agent** (AMA) | **0.6238** (0.3280) | **0.6145** (0.3103) | **0.5305** (0.2625) | **0.4719** (0.2825) | **0.5722** (0.2992) |

*Note: Results are reported as Accuracy (F1). The **best** and second best Accuracy values are highlighted.*

**Implementation Details.** To ensure a fair comparison, we evaluate all RAG baselines, memory-based agents, and our proposed AMA-Agent using the same backbone architectures: **Qwen3-32B** and **Qwen3-8B**. For each baseline, we adhere to the original authors' default embedding models and indexing configurations (refer to Appendix B for reproduction details). For AMA-Agent, we employ Qwen3-4B-embedding to map the causality graph into the latent space and set $K = 5$ for similarity-based node retrieval.

**Metrics.** We report both Accuracy and F1-score. Accuracy measures the instances judged as correct by an LLM-as-judge based on Qwen3-32B. Additional details and validation of the LLM-as-judge protocol are provided in Appendix C.

### 6.2 Key Results

**Real-world Subset.** We report the main results on the real-world subset in Tab. 4 (evaluating models with long contexts) and Tab. 5 (comparing different memory systems). While GPT 5.2 achieves the highest average accuracy (0.73) as Tab. 4 shows , its performance suggests that even strong commercial models have not fully mastered trajectory-based agent memory capabilities. Crucially, when controlled under the Qwen3 32B backbone (Tab. 5), **AMA-Agent** establishes a new state-of-the-art across all dimensions—Recall (0.6238), Causal Inference (0.6145), State Updating (0.5305), and State Abstraction (0.4719)—reaching an average of 0.5722. This significantly outperforms the strongest RAG baseline HippoRAG2 (0.4480) and the leading memory method MemoRAG (0.4606). These results demonstrate that explicit modeling of long-horizon state dynamics and causal memory organization provides a more robust framework for agent reasoning than standalone retrieval-based approaches.

**Synthetic Subset**. We also evaluate all memory methods on the Synthetic subset and compare their scores against the real-world subset. Fig. 9 A illustrates the strong ranking correlation between real-world scenarios and our synthetic subset. The close alignment of most methods with the diagonal line demonstrates that the synthetic environment serves as a high-fidelity proxy for real-world performance, which is crucial given the high costs of real-world data acquisition and manual annotation.

Fig. 9 evaluates the performance stability of our method compared to other baselines as the sequence length increases from 8K to 128K. While the Long Context approach maintains competitive accuracy at shorter scales, its performance degrades significantly beyond 32K, revealing the inherent limitations of fixed context window. In contrast, AMA-agent exhibits superior scalability, maintaining robust and consistently high accuracy even at 128K. .

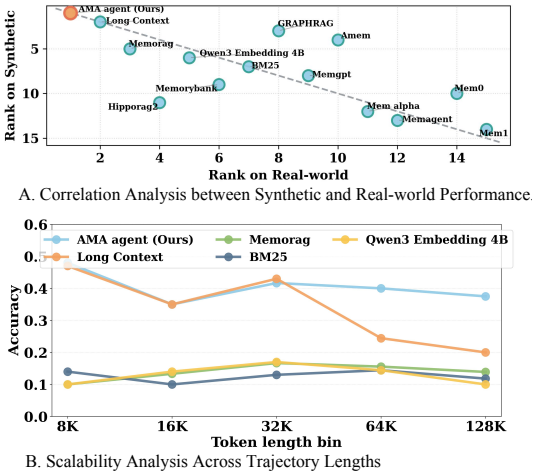


Figure 9: Performance Benchmarking. We evaluate 15 memory methods across Qwen 8B and 32B backbones.

### 6.3 Ablation Study

To validate the contributions of our key components, we perform ablation studies on the Causality Graph and tool augmented retrieval. The variant *w/o Causality Graph* replaces our structured Graph-based memory with a vanilla Qwen3 Embedding 4B indexs the context directly, while *w/o Tool Augmented Retrieval* disables tool calls and relies solely on embedding similarity retrieval. The results in Tab. 6 show that both modules are necessary for strong performance. Removing the Causality Graph causes a substantial degradation, with the average score dropping from 0.57 to 0.43, indicating that causality-aware representations are critical for agent memory. Likewise, removing tool augmented retrieval reduces performance to 0.44, suggesting that similarity search alone is insufficient and that tools provide complementary evidence access for robust reasoning.

Table 6: Ablation results for AMA-Agent

| Method | Recall | Causal Inference | State Updating | State Abstraction | Avg. |
|---|---|---|---|---|---|
| AMA-agent | 0.62 | 0.61 | 0.53 | 0.47 | 0.57 |
| w/o Causality Graph | 0.48 (−22.6%) | 0.48 (−21.3%) | 0.36 (−32.1%) | 0.35 (−25.5%) | 0.43 (−24.6%) |
| w/o Tool-Augmented Retrieval | 0.47 (−24.2%) | 0.51 (−16.4%) | 0.42 (−20.8%) | 0.31 (−34.0%) | 0.44 (−22.8%) |

*Note:* Values in dark red indicate the relative performance decrease compared to the full AMA agent.

## 7 Conclusion

In this paper, we introduced AMA-Bench to bridge the disparity between natural language-centric evaluations and the machine-generated, causally grounded nature of real-world agent trajectories. Our systematic analysis revealed that memory architecture is the primary determinant of performance, highlighting the limitation of lossy compression and similarity-based retrieval for dense, objective information. To address these challenges, we proposed AMA-Agent, which leverages a Causality Graph and Hybrid Tool-Augmented Retrieval to significantly outperform state-of-the-art baselines. A limitation of this study is its focus on in-episode memory; future work should extend these rigorous standards to cross-task scenarios involving lifelong learning.

## IMPACT STATEMENT

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted. Regarding the introduced benchmark, we confirmed that it was constructed entirely from open-source data sources.

## REFERENCES

Saaket Agashe, Jiuzhou Han, Shuyu Gan, Jiachen Yang, Ang Li, and Xin Eric Wang. Agent S: An open agentic framework that uses computers like a human. *arXiv preprint arXiv:2410.08164*, 2024.

Qingyao Ai, Yichen Tang, Changyue Wang, Jianming Long, Weihang Su, and Yiqun Liu. Memorybench: A benchmark for memory and continual learning in llm systems, 2025.

Anthropic. Claude models overview. Claude API Documentation, 2025. URL https://platform.claude.com/docs/en/about-claude/models/overview. Model overview page listing Claude Sonnet 4 and 4.5 with up to 200K / 1M token context windows.

Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. Longbench v2: Towards deeper understanding and reasoning on realistic long-context multitasks, 2025. URL https://arxiv.org/abs/2412.15204.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. Babyai: A platform to study the sample efficiency of grounded language learning. In *International Conference on Learning Representations (ICLR)*, 2019. URL https://openreview.net/forum?id=rJeXCo0cYX.

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

Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, et al. Textworld: A learning environment for text-based games. In *Workshop on Computer Games at IJCAI*, 2018. URL https://arxiv.org/abs/1806.11532.

Yiming Du, Wenyu Huang, Danna Zheng, Zhaowei Wang, Sebastien Montella, Mirella Lapata, Kam-Fai Wong, and Jeff Z. Pan. Rethinking memory in ai: Taxonomy, operations, topics, and future directions, 2025. URL https://arxiv.org/abs/2505.00675.

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. From local to global: A graph rag approach to query-focused summarization, 2025.

Gemini Team. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. Technical report, Google DeepMind, 2025. URL https://storage.googleapis.com/deepmind-media/gemini/gemini_v2_5_report.pdf. Technical report describing the Gemini 2.X family and their long-context capabilities.

Bernal Jiménez Gutiérrez, Yiheng Shu, Weijian Qi, Sizhe Zhou, and Yu Su. From rag to memory: Non-parametric continual learning for large language models, 2025.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models?, 2024.

Lanxiang Hu, Mingjia Huo, Yuxuan Zhang, Haoyang Yu, Eric P. Xing, Ion Stoica, Tajana Rosing, Haojian Jin, and Hao Zhang. lmgame-bench: How good are LLMs at playing games? *arXiv preprint arXiv:2505.15146*, 2025a. URL https://arxiv.org/abs/2505.15146.

Yuanzhe Hu, Yu Wang, and Julian McAuley. Evaluating memory in llm agents via incremental multi-turn interactions, 2025b.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. SWE-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023. URL https://openreview.net/forum?id=VTF8yNQM66.

Greg Kamradt. Llmtest_needleinahaystack. https://github.com/gkamradt/LLMTest_NeedleInAHaystack. GitHub repository, accessed 2026-01-28.

Dong-Ho Lee, Adyasha Maharana, Jay Pujara, Xiang Ren, and Francesco Barbieri. Realtalk: A 21-day real-world dataset for long-term conversation, 2025. URL https://arxiv.org/abs/2502.13270.

Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. Spider 2.0: Evaluating language models on real-world enterprise text-to-sql workflows. *arXiv preprint arXiv:2411.07763*, 2024. URL https://arxiv.org/abs/2411.07763.

Zhuofeng Li, Haoxiang Zhang, Seungju Han, Sheng Liu, Jianwen Xie, Yu Zhang, Yejin Choi, James Zou, and Pan Lu. In-the-flow agentic system optimization for effective planning and tool use. *arXiv preprint arXiv:2510.05592*, 2025. URL https://arxiv.org/abs/2510.05592.

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

lmgame-org. GamingAgent: Llm/vlm gaming agents and lmgame-bench. https://github.com/lmgame-org/GamingAgent, 2025.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of llm agents. *arXiv preprint arXiv:2402.17753*, 2024.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: A benchmark for general ai assistants. *arXiv preprint arXiv:2311.12983*, 2023.

Elliot Nelson, Georgios Kollias, Payel Das, Subhajit Chaudhury, and Soham Dan. Needle in the haystack for memory based large language models, 2024. URL https://arxiv.org/abs/2407.01437.

OpenAI. Introducing SWE-bench verified. OpenAI Research Blog, 2024. URL https://openai.com/research/introducing-swe-bench-verified. Updated February 24, 2025.

OpenAI. Gpt-5 models. OpenAI Model Overview, 2025. URL https://openai.com/gpt-5/. Official model index for the GPT-5 series, listing 400K token context windows for GPT-5-family models.

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

Davide Paglieri, Bartlomiej Cupial, Samuel Coward, Ulyana Piterbarg, Maciej Wolczyk, Akbir Khan, Eduardo Pignatelli, Lukasz Kucinski, Lerrel Pinto, Rob Fergus, Jakob Nicolaus Foerster, Jack Parker-Holder, and Tim Rocktäschel. BALROG: Benchmarking agentic LLM and VLM reasoning on games. *arXiv preprint arXiv:2411.13543*, 2024. URL https://arxiv.org/abs/2411.13543.

Richard Yuanzhe Pang, Alicia Parrish, Nitish Joshi, Nikita Nangia, Jason Phang, Angelica Chen, Vishakh Padmakumar, Johnny Ma, Jana Thompson, He He, and Samuel R. Bowman. Quality: Question answering with long input texts, yes!, 2022. URL https://arxiv.org/abs/2112.08608.

Hongjin Qian, Zheng Liu, Peitian Zhang, Kelong Mao, Defu Lian, Zhicheng Dou, and Tiejun Huang. Memorag: Boosting long context processing with global memory-enhanced retrieval augmentation, 2025.

Stephen E. Robertson and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. In *Foundations and Trends in Information Retrieval*, volume 3, pp. 333–389. Now Publishers Inc., 2009. doi: 10.1561/1500000019.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 2023.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. 2020a. URL https://arxiv.org/abs/1912.01734.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2020b. URL https://arxiv.org/abs/2010.03768.

Thomas Wailgum. Openai launches gpt-5.2 'garlic' with 400k context window. *eWeek*, 2025. URL https://www.eweek.com/news/openai-launches-gpt-5-2/. News article summarizing GPT-5.2, its 400K token context window, and pricing.

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.

Xingyao Wang, Boxin Li, Yaobo Song, Frank F. Xu, Xiaoyu Tang, Ming Zhuge, Jialu Chen, Haoming Yuan, Xinyu Li, Hantian Wang, et al. OpenHands: An open platform for AI software developers as generalist agents. *arXiv preprint arXiv:2407.16741*, 2024. URL https://arxiv.org/abs/2407.16741.

Yu Wang, Ryuichi Takanobu, Zhiqi Liang, Yuzhen Mao, Yuanzhe Hu, Julian McAuley, and Xiaojian Wu. Mem-$\alpha$: Learning memory construction via reinforcement learning, 2025a.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *International Conference on Machine Learning (ICML)*, 2025b. URL https://openreview.net/forum?id=NTAhi2JEEE. Also available as arXiv:2409.07429.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. Longmemeval: Benchmarking chat assistants on long-term interactive memory, 2025. URL https://arxiv.org/abs/2410.10813.

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

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

An Yang, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoyan Huang, Jiandong Jiang, Jianhong Tu, Jianwei Zhang, Jingren Zhang, Junyang Lin, Kai Dang, Kexin Yang, Le Yu, Mei Li, Minmin Sun, Qin Zhu, Rui Men, Tao He, Weijia Xu, Wenbiao Yin, Wenyuan Yu, et al. Qwen2.5-1m technical report. *arXiv preprint arXiv:2501.15383*, 2025b. URL https://arxiv.org/abs/2501.15383.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Hongli Yu, Tinghong Chen, Jiangtao Feng, Jiangjie Chen, Weinan Dai, Qiying Yu, Ya-Qin Zhang, Wei-Ying Ma, Jingjing Liu, Mingxuan Wang, and Hao Zhou. Memagent: Reshaping long-context LLM with multi-conv RL-based memory agent. *arXiv preprint arXiv:2507.02259*, 2025.

Hongwei Zhang, Ji Lu, Shiqing Jiang, Chenxiang Zhu, Li Xie, Chen Zhong, Haoran Chen, Yurui Zhu, Yongsheng Du, Yanqin Gao, Lingjun Huang, Baoli Wang, Fang Tan, and Peng Zou. Co-Sight: Enhancing LLM-based agents via conflict-aware meta-verification and trustworthy reasoning with structured facts. *arXiv preprint arXiv:2510.21557*, 2025a. URL https://arxiv.org/abs/2510.21557.

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

Zeyu Zhang, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Quanyu Dai, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model based agents, 2024. URL https://arxiv.org/abs/2404.13501.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory, 2023. URL https://arxiv.org/abs/2305.10250.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.

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

# APPENDIX

## A DETAILS OF DATASET

Here, we provide a detailed introduction to the datasets used for evaluating the four core competencies, including the dataset curation, corresponding metrics, average context length, and a brief description.

### A.1 REAL-WORLD SUBSET

This section details the composition of the real-world subset, which comprises multi-turn trajectories curated from **six diverse domains** of agent-environment interactions (Table 7).

Table 7: Implementation details of collected agent trajectories across task families.

| Field | Benchmark | Trace Source | Total | Selected |
|---|---|---|---|---|
| **Embodied AI** | ALFWorld-verified (seen) | ALFRED (Li et al., 2025) | 140 | 33 |
| **Gaming** | BALROG / lmgame bench | BALROG / GamingAgent | 367 | 30 |
| **Web Task Execution** | WebArena (Zhou et al., 2023) | WebArena (Zhou et al., 2023) | 162 | 31 |
| **Software Engineering** | SWE-bench (Jimenez et al., 2023) | OpenHands (Wang et al., 2024) | 162 | 34 |
| **Open World Tool QA** | GAIA (Mialon et al., 2023) | CoSight (Zhang et al., 2025a) | 100 | 30 |
| **Text 2 SQL** | Spider 2.0 (Lei et al., 2024) | Spider2 agent (Lei et al., 2024) | 120 | 51 |

**Embodied AI.** We collect trajectories from both the **seen** and **unseen test splits** of ALFWorld (Shridhar et al., 2020b), a text-based embodied environment aligned with the ALFRED benchmark. These trajectories are generated using the expert-level demonstrations from ALFRED (Shridhar et al., 2020a) to ensure high-quality task completion in both familiar and novel environments.

**Gaming.** We curate gaming trajectories from two sources: BALROG (Paglieri et al., 2024), which includes Crafter (resource management), Baba is AI (long-horizon puzzle solving), and MiniHack (navigation); and LMGame-Bench (Hu et al., 2025a), which includes 2048 and Candy Crush. Trajectories are collected using GPT-5.1 with the BALROG agent framework and GamingAgent (lmgame-org, 2025) with memory and perception modules. For 2048, we use rule-based methods due to the extensive action sequences required. We select 30 trajectories totaling 360 QA pairs, with an average of 150 turns per episode.

**Web Task Execution.** We use WebArena (Zhou et al., 2023), a realistic web environment featuring fully functional websites across e-commerce, social forums, software development, and content management domains. Trajectories are collected using GPT-4.1 with the WebArena agent framework. We select 31 trajectories comprising 372 QA pairs, with an average of 25 turns and 34K tokens per trajectory, reaching up to 166K tokens for complex tasks.

**Software Engineering.** We collect trajectories from SWEBench Verified (Jimenez et al., 2023; OpenAI, 2024), which consists of real GitHub issues and pull requests from popular Python repositories. Trajectories are generated using Claude Sonnet 4 with the OpenHands framework (Wang et al., 2024), an open platform for AI software developers. We select 36 trajectories totaling 432 QA pairs, with an average of 103 turns and 19K tokens per trajectory.

**Open World Tool QA.** We use the GAIA benchmark (Mialon et al., 2023), which tests general AI assistants on real-world questions requiring reasoning, multi-modality handling, web browsing, and tool-use proficiency. Trajectories are collected using GPT-5 with the Co-Sight framework (Zhang et al., 2025a), which achieves state-of-the-art performance on open-sourced agent benchmarks. We select 30 trajectories across all three difficulty levels from the validation set, comprising 360 QA pairs with an average of 41 turns and 289K tokens – the longest among all domains, reaching up to 997K tokens.

**Text-to-SQL.** We collect trajectories from the Spider 2.0 benchmark (Lei et al., 2024), specifically sampling from the Spider2-Snow subset which focuses on enterprise-level text-to-SQL tasks with Snowflake databases. Spider 2.0 comprises three subsets (Snow, DBT, and Lite), with the Snow subset containing 547 examples. Among these, gold answers are provided for 120 examples to enable verification of generated SQL queries. We sample 51 trajectories from these verified examples to ensure answer correctness can be validated. Trajectories are generated using Claude Sonnet 4.5 with the Spider2-Agent framework, totaling 612 QA pairs with an average of 22 turns and 6K tokens per trajectory.

A comprehensive breakdown of the Real-world Subset is provided in Table 8. To provide a clearer illustration of our defined problem types, we present a representative example from the Web Task Execution domain (Figure E.1). This example shows how an agent must use different memory operations, such as tracking incremental UI changes, recognizing high level strategic failures, and handling long horizon interactions. Aligned with the three memory capabilities (Table 2), we define four QA categories that probe whether an agent has acquired the corresponding competencies required to answer them reliably, and we instantiate all four categories in this example. For additional qualitative visualizations of data samples, please refer to Appendix E.

Table 8: Statistics of QA pairs, evaluation type distribution, and interaction complexity.

| Field | #Samples | #QA | Evaluation Type | | | | Avg. Turns | Avg. Tokens | Max Tokens |
|---|---|---|---|---|---|---|---|---|---|
| | | | Type A | Type B | Type C | Type D | | | |
| Text 2 SQL | 51 | 612 | 223 | 153 | 134 | 102 | 21.80 | 6,049 | 10,718 |
| Open World Tool QA | 30 | 360 | 98 | 95 | 107 | 60 | 41.40 | 288,651 | 996,826 |
| Web Task Execution | 31 | 372 | 125 | 93 | 93 | 61 | 24.77 | 34,265 | 166,260 |
| Gaming | 30 | 360 | 120 | 90 | 90 | 60 | 149.87 | 14,909 | 33,360 |
| Embodied AI | 30 | 360 | 61 | 90 | 150 | 59 | 130.33 | 26,306 | 60,717 |
| Software Engineering | 36 | 432 | 212 | 75 | 73 | 72 | 103.22 | 19,296 | 28,615 |
| **TOTAL** | **208** | **2,496** | **839** | **596** | **647** | **414** | **73.29** | **57,506** | **996,826** |

### A.2 SYNTHETIC SUBSET

**BabyAI** are generated from the BabyAI environment Chevalier-Boisvert et al. (2019), which supports six difficulty levels: `easy`, `medium`, `medium_hard`, `hard`, `very_hard`, and `hard_large`. Each trajectory is paired with 12 questions by default, and we collect a total of 50 trajectories, with an average length of 563 turns and 30,042 tokens per trajectory.

**TextWorld** are generated from the TextWorld Côté et al. (2018) environment. We consider three game types: `coin_collector`, `cooking`, and `treasure_hunter`. The environment supports eight difficulty levels: `easy`, `medium`, `medium_hard`, `hard`, `very_hard`, `extreme`, `ultra`, and `mega`. On average, each trajectory contains 57 turns and 31,662 tokens.

To provide a concrete realization of the Synthetic subset described in Section 3.3, we present qualitative case studies from BabyAI and TextWorld. These examples are designed to illustrate how our programmatic framework evaluates specific agent capabilities by modulating synthesis parameters. For additional qualitative visualizations of data samples, please refer to Appendix E.

- **Probing Memory Robustness under Action Stochasticity ($\epsilon$):** Figure E.2 presents a diagnostic episode from the **BabyAI** environment under high action stochasticity ($\epsilon$). This setup evaluates the agent's ability to maintain the task goal within its interaction context when $\pi^*$ is perturbed by suboptimal exploratory noise. The observed failure—a task truncation—indicates that the agent's memory mechanism fails to distinguish gold-standard goal alignment from the increased "interaction noise," even when target objects are clearly rendered via the perception interface $O_\phi(s_t)$.
- **Evaluating State Tracking across Subgoal Chains ($\phi$):** Figure E.2 captures a failure in **TextWorld** that probes the agent's internal state-tracking of the backend transition kernel $P_\phi$. By scaling the difficulty vector $\phi$ to increase subgoal chain length, we test whether the agent can correctly update its "memory slots" based on transition events $\Delta s$. The repetitive invalid actions (e.g., attempting a `put` before a verified `take` event) reveal a breakdown in causal reasoning,

Table 9: Maximum context lengths used for long context baselines.

| Model | Max context tokens | Notes |
|---|---|---|
| Claude 3.5 Haiku | 200,000 | API context window |
| OpenAI GPT 5 mini | 400,000 | API context window |
| OpenAI GPT 5.2 | 400,000 | API context window |
| Gemini 2.5 Flash | 1,048,576 | Max input tokens |
| Qwen2.5 14B Instruct 1M | 1,010,000 | Long context checkpoint |
| Qwen3 32B | 32,768 | Native |
| Qwen3 14B | 32,768 | Native |

where the agent loses track of the latent state $s_t$ despite having navigated the correct spatial transitions.

# B Baseline Implementation Details

**Long-Context Model Baseline.** For long-context baselines, we directly *pack* the trajectory into the model input without retrieval or compression until reaching the maximum context length permitted by each API or checkpoint in Tab. 9. We reserve a fixed 4K-token budget for the model to generate the final answer, and use the remaining tokens as the effective input budget. When a trajectory exceeds this budget, we apply a simple truncation strategy that preserves both early and late interactions: we keep the first 50% and the last 50% budget length of the trajectory (by token count) and discard the middle portion to fit the context window.

## B.1 RAG Baseline

**GraphRAG.** constructs memory by using an LLM (Qwen3-8B/32B) to extract entities (object, location, action) and their relationships from trajectory text, storing them as a knowledge graph in parquet format. The trajectory is first chunked into semantic units of 15 turns per chunk with a maximum of 24,000 tokens. This construction process is inherently lossy, as it discards substantial raw trajectory details, particularly the detailed observation states and fine-grained action sequences present in the original data. During retrieval, GraphRAG selects the top-$k$ most relevant entities and relationships from the knowledge graph based on their descriptions, concatenating these structured elements into the prompt as context rather than including the full trajectory. We follow the default GraphRAG configuration with $k=50$ entities and relationships, `max_gleanings=0`, and a description summarization disabled to preserve extraction fidelity.

**HippoRAG.** constructs memory by applying OpenIE-style extraction to trajectory text, yielding entities and relation triples that form a heterogeneous graph of passage (trajectory chunk), entity, and fact nodes; synonymy edges are added via nearest-neighbor search over entity embeddings. This construction is lossy because raw trajectories are distilled into triples and edges, potentially omitting fine-grained state transitions or action details. At retrieval time, HippoRAG computes query–fact similarity with dense embeddings, reranks top facts, maps them to linked entities, and runs personalized PageRank over the graph; the graph scores are combined with dense passage retrieval to select top-$k$ passages for QA. We use the default HippoRAG configuration with top-$k$ fact/entity linking $=5$, passage retrieval top-$k=200$, and QA context limited to the top 5 passages. All passage, entity, and fact embeddings use the same model BAAI/bge-m3.

## B.2 Memory Agents Baseline

**MemoryBank.** constructs a hierarchical memory by first chunking the trajectory into segments of 5,000 tokens with 500-token overlap, then using the LLM to summarize each chunk into a compact memory piece that preserves key subgoals, actions, observations, and failures. Each memory piece is embedded using a local sentence-transformer model (all-MiniLM-L6-v2), and a global summary is generated from all memory pieces to capture the overall strategy and critical facts. This summarization process is lossy, as fine-grained trajectory details are compressed into concise text.

During retrieval, MemoryBank computes cosine similarity between the question embedding and memory piece embeddings, combined with an Ebbinghaus-inspired retention score that accounts for memory strength and recency of recall. The top-$k$ memory pieces are retrieved and concatenated with the global summary as context for answering. We use the default configuration with $k = 6$, forget decay $\tau = 5.0$, and strength increment of 1 upon each recall.

**MemAgent.** processes the trajectory as a stream of fixed-length sections, iteratively updating a recurrent memory state. For each chunk of 5,000 tokens, the LLM reads the current trajectory section along with the previous memory, then generates an updated memory that summarizes the agent's progress while retaining relevant details from earlier sections. This recurrent summarization is inherently lossy, as information from earlier chunks may be progressively compressed or forgotten as new sections are processed. During retrieval, MemAgent directly reads from its final accumulated memory without additional retrieval mechanisms, the memory itself serves as the complete context for answering questions. We follow the default configuration with a 4,096-token context window partitioned into: the current trajectory chunk (5,000 tokens, truncated if needed), the accumulated memory (dynamically sized), and generation budget (1,024 tokens), with memory truncated from the beginning when context limits are exceeded to preserve more recent information.

**Mem-alpha.** employs a three-tier hierarchical memory architecture with an agentic approach to memory management. The system maintains: (1) Core Memory for high-level task understanding and rules, (2) Semantic Memory for storing factual knowledge as embedded vectors, and (3) Episodic Memory for recording specific events with temporal context. Trajectories are chunked using sentence-aware tokenization into segments of 4,096 tokens, preserving sentence boundaries. For each chunk, an agent equipped with memory tools (insert, update, delete, retrieve) autonomously decides which information to store and in which memory tier. Both semantic and episodic memories are embedded using text-embedding-3-small (1,536 dimensions) and retrieved via Top-K similarity search. During question answering, MemAlpha retrieves relevant memories using BM25 sparse retrieval, fetching the top-20 most relevant items per memory type. We follow the default configuration with a thinking budget of 1,024 tokens, maximum generation of 2,048 tokens, and memory consolidation occurring every 5 items.

**Mem1.** processes the trajectory through recurrent memory consolidation, iteratively updating a compact memory state with each new chunk of observations. For each chunk of 5,000 tokens, the LLM reads the current trajectory section along with the previous accumulated memory, then generates an updated memory that integrates new information while maintaining context and discarding redundant details. After processing all chunks, a final comprehensive summary is generated that consolidates key actions, important observations, overall progress, and patterns encountered. MEM1 directly uses this global consolidated memory as the complete context for answering all questions. We follow the default configuration with a 120,000-token maximum context window, trajectory chunks of 5,000 tokens, memory update budget of 1,024 tokens per chunk.

**Mem0.** constructs its memory layer through an LLM-driven fact extraction process, distilling raw trajectory data into a series of "atomic facts." These facts are subsequently embedded and stored in a vector database. To ensure memory consistency, Mem0 incorporates a conflict resolution mechanism that updates or replaces outdated information (e.g., evolving user locations). However, this extraction-based approach is inherently lossy for structured trajectory data, as it often omits critical low-level details. Empirical observations during our experiments indicate that bypassing the extraction layer and utilizing raw data directly can significantly enhance performance on trajectory-based benchmarks. During retrieval, Mem0 employs vector-based cosine similarity to identify the top-$k$ most relevant facts, which are then injected into the LLM prompt as context

**A-Mem.** implements a recurrent memory processing strategy to handle long-term trajectories. The input is segmented into chunks, and new memory states are built recursively by integrating the current chunk with preceding memory. Each memory entry consists of a concatenated representation of content, context, keywords, and tags, which is then embedded and stored in a vector database. This recursive construction, while thorough, introduces significant computational latency for long-context sequences. Furthermore, A-Mem supports memory evolution: it retrieves top-$k$ neighboring entries

via vector search, and an LLM determines whether to establish new relational connections or update existing metadata. We set the `RECURRENT_CHUNK_SIZE` to $8{,}000$ tokens.

**MemGPT.** implements a hierarchical memory architecture that separates memory into core memory (in-context) and archival memory (external storage with retrieval). The trajectory is inserted directly into archival memory as a complete text block, which is then indexed for retrieval. MemGPT uses an agentic approach where the LLM autonomously manages memory through function calls. It can search, insert, and retrieve from archival memory as needed during question answering. The archival memory is embedded using a local embedding model (BAAI/bge-small-en-v1.5) for vector-based retrieval. Unlike other memory agents that pre-process trajectories into summaries, MemGPT stores the raw trajectory text and relies on the agent's retrieval capabilities to fetch relevant portions at query time. During question answering, the agent receives the question and uses its memory tools to search archival storage, with retrieved content brought into the limited core memory context window. We use the default MemGPT configuration with auto-save disabled, maximum chaining steps set to 5 to prevent infinite tool-calling loops, and observations truncated to 8,000 characters when exceeding length limits.

**MemoRAG.** constructs memory by first building a global memory representation of the entire trajectory using a dedicated memory model, then enabling retrieval-augmented generation for question answering. The trajectory is converted to text format and processed by a memory encoder (Qwen2-7B-Instruct with beacon compression at ratio 4) that compresses the long context into a compact memory representation. This memory is then used to guide retrieval from the original text chunks. During retrieval, MemoRAG uses a dual-model architecture: the memory model generates retrieval cues based on the query and global memory, while a separate retriever (BAAI/bge-m3) fetches the top-$k$ most relevant chunks from the original trajectory. The retrieved chunks are then passed to a generation model to produce the final answer. This approach is lossy during memory encoding, as the beacon compression mechanism reduces the original context to a fraction of its size. We use the default configuration with retrieval chunk size of 512 tokens, top-$k = 3$ retrieved hits, beacon ratio of 4, and maximum generation length of 256 tokens, retrieval via bge-m3.

**SimpleMem.** constructs memory through an LLM-driven extraction process that converts raw trajectory data into atomic memory entries. The trajectory is processed in sliding windows, where each window is passed to an LLM that extracts structured fields including lossless restatements with forced coreference resolution (eliminating pronouns and converting relative time to absolute timestamps), keywords, locations, persons, entities, and topics. These atomic entries are embedded and stored in a vector database. This extraction process is lossy, as trajectory details are abstracted into discrete semantic units. During retrieval, SimpleMem employs a hybrid strategy combining semantic vector similarity, lexical keyword matching, and symbolic metadata filtering. The system supports multi-query planning that decomposes complex questions into targeted sub-queries, and reflection-based refinement that iteratively checks information completeness and generates additional queries to fill gaps. We use the default configuration with parallel memory building (2 workers), parallel retrieval (3 workers), reflection enabled with maximum 2 rounds, and planning enabled for query decomposition.

## C LLM-AS-JUDGE CALIBRATION PROTOCOL

We include the following materials to ensure reproducibility and transparency of our LLM-as-judge evaluation.

### C.1 JUDGE PROMPT AND OUTPUT FORMAT

We use QWEN3 32B as the primary evaluator. The judge receives the input triplet (question, reference answer, predicted answer) and returns a binary decision. The judge is required to output only one token in $\{\texttt{yes}, \texttt{no}\}$.

Binary Correctness Judgement Prompt

**System or Instruction Prompt**
You are an expert evaluator. You will be given a question, a reference answer, and a predicted answer. Your task is to determine if the predicted answer is correct based on:
1. Factual correctness compared to the reference
2. Completeness of the answer
3. Relevance to the question

{context_str}

**Question:** {question}

**Reference Answer:** {golden_answer}

**Predicted Answer:** {predicted_answer}

Is the predicted answer correct? Respond with ONLY `yes` or `no`. Do not include any thinking process, explanation, or additional text.

**Answer**

### C.2 HUMAN-JUDGE AGREEMENT

To validate the reliability of our LLM-as-judge evaluation, we conducted human annotation on a sample of 300 instances (50 per subset) from the GPT-5.2 results. We obtain gold labels via independent human annotation. Each instance is labeled by at least two annotators with access to the question, reference answer, and predicted answer. Labels are binary: `yes` if the predicted answer is correct, otherwise `no`. Disagreements are resolved by majority vote, with a third annotator used for adjudication when needed.

Table 10 presents the confusion matrix and performance metrics aggregated across all subsets. The judge achieves 92.67% accuracy, indicating reliable alignment with human judgment.

Table 10: Confusion matrix (left) and performance metrics (right) for LLM-as-judge vs. human annotations.

| | **Human Label** | | |
|---|---|---|---|
| **Judge Label** | Correct | Incorrect | Total |
| Correct | 190 (TP) | 7 (FP) | 197 |
| Incorrect | 15 (FN) | 88 (TN) | 103 |
| Total | 205 | 95 | 300 |

| **Metric** | **Value** |
|---|---|
| Accuracy | 92.67% |
| Precision | 96.45% |
| Recall | 92.68% |
| F1 Score | 94.53% |

## D PROMPT TEMPLATES FOR AMA-AGENTS

This appendix provides the prompt templates used by AMA-agent in both phases. The first phase constructs the Causality Graph by extracting objective inventories, detecting environment and objective state changes, and emitting structured, machine parsable records in Markdown. The second phase performs retrieval time routing, including chunk sufficiency judgement and trajectory based code generation.

### D.1 MEMORY CONSTRUCTION PROMPT

Memory Construction Prompt Template

```
MEMORY_CONSTRUCTION_PROMPT_TEMPLATE = """
You are given an agent trajectory consisting of action and observation pairs.
Your task is to analyze the trajectory turn by turn and produce a Markdown report
that is strictly machine parsable.

Input will provide:
(1) current turn: {turn_t}
(2) previous turn: {turn_t_minus_1}
(3) optional task description: {task}

You must do the following in order.

Section 1: OBJECTIVE INVENTORY
List all objectives that the agent is tracking or manipulating.

Section 2: ENV STATE CHANGE DETECTION
Decide whether the environment state changed at this turn.

Section 3: OBJECT STATE CHANGE DETECTION
Decide whether the object state changed at this turn.

OUTPUT FORMAT
Return a Markdown block with the following exact structure.

# OBJECTIVES
1. <objective_name>: <description>
2. ...

# STATE_CHANGES
env_changed: <true or false>
objective_changed: <true or false>
evidence:
  - "<quote 1>"
  - "<quote 2>"

# STATES
env_state:
  - <key>: <value>
  - ...
object_states:
  - name: <objective_name>
    state:
      - <key>: <value>
      - ...
  - ...
Constraints:
(1) Do not invent facts not supported by the provided turns.
(2) Keep rationales short and grounded.
(3) Use consistent objective names across turns.
"""
```

### D.2 CHUNK SUFFICIENCY JUDGEMENT PROMPT

Chunk Sufficiency Judgement Prompt Template

```
CHUNK_SUFFICIENCY_JUDGMENT_PROMPT_TEMPLATE = """
You have retrieved the top ranked most relevant turns from an agent trajectory.
Each turn has a UNIQUE TURN INDEX that you can reference.

Query: {query}

Retrieved Turns:
{retrieved_chunks}

Your Task
Carefully analyze the retrieved turns and determine ONE of the following.

1. SUFFICIENT
The retrieved turns contain enough information to answer the query completely.
If you choose this, you MUST provide the answer immediately in the same response.
Format:
SUFFICIENT
ANSWER: <your complete and accurate answer>

2. NEED_GRAPH
The query can likely be answered by looking at adjacent turns or specific ranges.
Use this when you found relevant information but need surrounding context.

You can specify retrieval in multiple ways.

A. Request adjacent turns
NEED_GRAPH: turn_5 before=2 after=1
NEED_GRAPH: turn_8 before=3 after=0, turn_15 before=0 after=2

B. Request turn ranges
NEED_GRAPH: turns 5 to 10
NEED_GRAPH: turns 3 to 8, turns 15 to 20

C. Request individual turns
NEED_GRAPH: turns 3, 7, 12, 18
NEED_GRAPH: turns 5, 8, 15

3. NEED_CODE
The query requires computational analysis, pattern finding, counting, or aggregation
across the full trajectory and cannot be answered from the retrieved turns alone.
Format:
NEED_CODE: <explain what computation or analysis is needed>

Guidelines
Choose SUFFICIENT only if you can answer completely right now.
Choose NEED_GRAPH when you need immediate context around retrieved turns.
Choose NEED_CODE for trajectory wide computation or when turns do not contain the answer.

Response:
"""
```

### D.3 TRAJECTORY BASED CODE GENERATION PROMPT

Trajectory Based Code Generation Prompt Template

```
CODE_GENERATION_PROMPT_TEMPLATE = """
Write Python code to extract information from a trajectory to answer the question.
Keep your thinking brief.

Question: {query}

Task: {task}

Trajectory Sample:
{trajectory_sample}

Trajectory JSON Structure
Variable trajectory_json contains:
{
  "trajectory": [
    {
      "turn_idx": 0,
      "action": "...",
      "observation": "..."
    },
    ...
  ],
  "task": "...",
  "episode_id": "..."
}

Requirements
(1) Only use Python standard library unless explicitly allowed otherwise.
(2) The code must be directly executable.
(3) Print intermediate results that justify the final answer.
(4) Reference turn indices in outputs whenever possible.
"""
```

# E DATASET EXAMPLES

We provide representative examples from subset below.

## E.1 REAL-WORLD SUBSET EXAMPLE

WebArena Case Study (ID: `task_webarena_17`)

**Task:** Star the eight most-starred repositories on GitLab.
**Total Tokens:** 38,289 **Turns:** 31

| Turn | Action | Environment Observation (Structural State Snippet) |
|---|---|---|
| 00 | `click [426]` | Tab 0: `Projects · Dashboard · GitLab` (link `Explore`) |
| 01 | `click [2040]` | Tab 0: `Projects · Explore · GitLab` (link `Most stars`) |
| 04 | `click [6170]` | Tab 0: `Projects · Explore · GitLab` (link `267`) |
| 05 | `click [7798]` | Tab 0: `Issues · Umano: News Read To You / AndroidSlidingUpPanel` |
| ... | ... | ... |
| 15 | `type [13642]` | Tab 0: `most starred repositoriesstars · Search · GitLab` |
| ... | ... | ... |
| 23 | `type [19543]` | Tab 0: `The A11Y Project / allyproject.com` (`textbox Search`) |
| 26 | `click [21429]` | Tab 0: `Projects · Dashboard` (link `Byte Blaze / ally...`) |
| ... | ... | ... |
| 30 | `action: stop` | *[Early stop: Reach max steps 30]* |

**QA Examples:**

- **Type A (Temporal Inference):** *The search query string begins concatenating across Steps 14–16. At which step does the title first show 'most starred repositoriesstars', and what action caused that?*
  **Ans:** Step 15 shows the title, caused by the Step 14 `type` action appending text to the query without clearing it.
- **Type B (State Dependency):** *Why did keyword searching fail to replicate the 'Most stars' ordering, and what prerequisite state was required instead?*
  **Ans:** 'Most stars' is a view-specific ordering on Explore; Search matches text and returned zero results. The prerequisite was staying on Explore with 'Most stars' selected.
- **Type C (State Update):** *Compare the 'Users' result count between Step 14 and Step 21. What is the change?*
  **Ans:** It increases from 'Users 0' to 'Users 1', a +1 change.
- **Type D (State Abstraction):** *Identify the redundant loop near the end and explain why it is unproductive.*
  **Ans:** Steps 26 and 29 both click link [21429] from the Dashboard list without a state change in between. This loop fails to advance toward the goal.

### E.2 SYNTHETIC SUBSET EXAMPLE

BabyAI Example (ID: `task_babyai_22`)

**Task:** put the purple key next to the green key
**Total Tokens:** 19,310 **Turns:** 392 **Result:** *fail (truncated)*

| Turn | Action | Environment Observation (Structural State Snippet) |
|---|---|---|
| 00 | done | Mission: put the purple key next to the green key. You are facing south. |
| 03 | toggle | In your view: . Walls border the area. You are facing south. |
| 10 | forward | In your view: a purple key, a red ball, a grey box. You are facing east. |
| 12 | toggle | In your view: a green key, a grey ball, a red ball, a grey box, a purple box, a purple key. You are facing north. |
| ... | ... | ... |
| 24 | pickup | In your view: a red ball, a grey box, a purple box, a purple key, a green box. You are facing north. |
| 43 | drop | In your view: . Walls border the area. You are facing south. |
| ... | ... | ... |
| 391 | toggle | Mission: put the purple key next to the green key. *[Reach max steps 392]* |

**QA Examples:**

- **Type A (Temporal Inference):** *Which object is no longer visible after the agent moves at step 10?*
  **Ans:** After the agent executes the 'forward' action at step 10, '. Walls border the area' is no longer visible. It was visible at step 9, but the agent's movement changed its field of view, causing '. Walls border the area' to move out of sight.
- **Type B (State Dependency):** *Background: At step 6, the agent's inventory is empty. Question: At step 7, can the agent execute the 'drop' action? Why or why not?*
  **Ans:** At step 7, the agent cannot execute the 'drop' action because the inventory is empty at step 6. The agent needs to have an item in inventory to drop it.
- **Type C (State Update):** *What are the inventory changes from step 1 to step 4? At which steps did the inventory change respectively?*
  **Ans:** The inventory changes from step 1 to step 4 are: step 4: agent picks up item (inventory changes).
- **Type D (State Abstraction):** *Summarize the agent's trajectory from step 0 to step 3 for the task 'put the purple key next to the green key'. What are the key actions and milestones?*
  **Ans:** From step 0 to step 3, the agent's trajectory includes: Key actions: step 0: done; step 1: done; step 2: done; step 3: toggle.

TextWorld Example (ID: `cooking_medium_103008`)

**Task:** Get ready to pick stuff up and put it in places... First thing I need you to do is to doublecheck that the walnut gate in the serious workshop is wide open. And then, go to the east... retrieve the old lampshade from the rusty rack... sit the old lampshade on the shoddy table inside the cramped vault.
**Total Tokens:** 8,384 **Turns:** 80 **Result:** *fail (reached_max_steps)*

| Turn | Action | Environment Observation (Structural State Snippet) |
|---|---|---|
| 00 | `put old lampshade on...` | You can't see any such thing. (Inventory: tiny loaf of bread, teal keycard, mouse, synthesizer) |
| 14 | `look` | **-= Serious Workshop =-** Desk contains outmoded mouse and tiny loaf of bread. Closed walnut gate to the east. |
| 29 | `open walnut gate` | You open walnut gate. (Available actions: go east, close walnut gate, examine walnut gate, etc.) |
| 59 | `go east` | **-= Cramped Vault =-** You see a shoddy table (empty) and a rusty rack with an **old lampshade** on it. |
| 60 | `put old lampshade on...` | You need to take the old lampshade first. |
| 64 | `put old lampshade on...` | You need to take the old lampshade first. |
| 74 | `go north` | **-= Silent Cubicle =-** The agent moves away from the goal room. |
| 78 | `go south` | **-= Serious Workshop =-** Agent returns to the starting room. |
| 79 | `look` | *[Early stop: Reach max steps 80]* |

**QA Examples (Direct Extraction):**

- **Type A (Temporal Inference):** *What actions were performed between step 37 and step 41?*
  **Ans:** Between step 37 and step 41, the agent performed the following actions: at step 37, eat tiny loaf of bread; at step 38, put old lampshade on shoddy table; at step 39, put old lampshade on shoddy table; at step 40, put old lampshade on shoddy table; at step 41, look.
- **Type B (State Dependency):** *At step 27 with state: Inventory: empty... Object states: shoddy table [cramped vault], old lampshade [rusty rack], teal keycard [I]... can the agent perform "drop teal keycard"?*
  **Ans:** No, the agent cannot perform this action because the inventory is empty, so there is no object to put.
- **Type C (State Update):** *How did the state of I change throughout the trajectory, including what objects were placed in or removed from it?*
  **Ans:** I evolution: step 1: tiny loaf of bread was removed; step 2: outmoded mouse was removed; step 15: tiny loaf of bread was added; step 23: tiny loaf of bread was removed; step 31: small synthesizer was removed; step 32: tiny loaf of bread was added; step 37: tiny loaf of bread was removed. At step 41, it contains: teal keycard.
- **Type D (State Abstraction):** *Until step 79, what actions has the agent performed and how frequently?*
  **Ans:** Actions performed: 'put old lampshade on shoddy table' (18 times), 'inventory' (13 times), 'look' (8 times), 'take old lampshade from rusty rack' (6 times), 'open walnut gate' (5 times), 'put tiny loaf of bread on solid desk' (4 times), 'take tiny loaf of bread from solid desk' (3 times), 'go east' (3 times), 'examine small synthesizer' (3 times), 'put outmoded mouse on solid desk' (2 times), 'examine teal keycard' (2 times), 'examine solid desk' (2 times), 'close walnut gate' (2 times), 'examine walnut gate' (1 times), 'drop small synthesizer' (1 times).

## F NEEDLE-IN-A-HAYSTACK QA GENERATION PIPELINE

To evaluate the long-context retrieval capabilities of memory-augmented agents, we developed a structured pipeline to generate QA pairs where the answer is anchored to specific "needle" turns within a trajectory "haystack." The generation logic is formalized in Algorithm 1.

**Algorithm 1** QA Needle Generation for Trajectory Evaluation

**Require:** Source trajectory data $\mathcal{T}$, Bin sizes $\mathcal{B} \in \{8K, 16K, ..., 128K\}$, QA Types $\mathcal{Q} \in \{A, B, C, D\}$
**Ensure:** Final balanced dataset $\mathcal{D}_{final}$

$\mathcal{C} \leftarrow \emptyset$ {Initialize candidate pool}
$H \leftarrow$ split_by_turns($\mathcal{T}$) {Chunk haystack with unique turn identifiers}
**for all** $bin\_size \in \mathcal{B}$ **do**
  **for all** $qa\_type \in \mathcal{Q}$ **do**
    $\tau_{needle} \leftarrow$ sample($H$,strategy="diversity_first") {Ensure depth diversity}
    $qa_{needle} \leftarrow$ generate_qa($H, qa\_type, \tau_{needle}, bin\_size$)
    $qa_{needle}$.source_ids $\leftarrow \{t.id \mid t \in \tau_{needle}\}$ {Map for traceability}
    **if** verify_qa_quality($qa_{needle}$) **then**
      $\mathcal{C} \leftarrow \mathcal{C} \cup \{qa_{needle}\}$
    **end if**
  **end for**
**end for**
$\mathcal{D}_{final} \leftarrow$ select_balanced($\mathcal{C}$,quota$= \{A:4, B:3, C:3, D:2\}$)

**Key Strategies in Pipeline:**

- **Diversity-First Sampling:** Instead of random sampling, we pick "needle" turns from various depths of the trajectory (early, middle, and late stages) to prevent the LLM from exploiting positional biases.
- **Ground Truth Traceability:** By binding each QA pair to specific `turn_ids`, we can verify whether the agent's retrieval mechanism successfully identified the correct "needle" from the "haystack" during inference.
- **Balanced Distribution:** The final selection ensures that different reasoning types (e.g., spatial reasoning vs. object state tracking) are represented proportionally to avoid data skew.

## G EXAMPLE OF NEEDLE TURN FOR ABLATION STUDY

To further illustrate the distinction between the three evaluation branches discussed in Section 5.3, we provide a concrete example from a Type A3 task (Object Visibility Change).

Sample Query (Type A3)

**Question:** Which object is no longer visible after the agent moves at step 1?
**Ground Truth:** After the agent executes the 'forward' action at step 1, 'a purple box' is no longer visible. It was visible at step 0, but the agent's movement changed its field of view.

- **Raw Observation Branch:**

```
[Turn 0] Action: forward; Obs: "In your view: a yellow box, a
purple box, a green ball..."
[Turn 1] Action: forward; Obs: "In your view: a yellow box, a
green ball..."
```

*Note: The LLM must compare the two raw observation strings to infer the disappearance of the purple box.*

- **Oracle Memory Branch (and System Branch upon success):**

```
<memory>
- **Initial Position (Turn 0)**: Visible objects: Yellow box, purple box, green ball.
- **Progress (Turn 1)**: Action: Moved forward. Updated view: Yellow box and
  green ball visible; purple box no longer in sight.
- **Inference**: The disappearance of the purple box suggests movement progress.
</memory>
```

*Note: The state change is explicitly summarized. In the Oracle branch, this shard is force-fed to the LLM; in the System branch, the agent must retrieve this specific shard from the database.*

## H CASE-STUDY

To further illustrate the distinction between our raw data storage approach and the default Mem0 extraction process discussed in Section 3, we provide a comparative analysis using a $Qwen3-32B$ model. This example demonstrates why LLM-driven "fact distillation" is inherently lossy for structured agent experiences.

**Case Study 1: Narrative Data (Successful Extraction)**

**Input:** My name is John. I like pizza and I work as a software engineer. I have a dog named Max.
**Status:** Success (4 Facts Extracted)

- `Fact 1`: Name is John
- `Fact 2`: Likes pizza
- `Fact 3`: Works as a software engineer
- `Fact 4`: Has a dog named Max

**Case Study 2: Trajectory Data (Extraction Failure)**

**Input (Abridged Trajectory):**

```
[Turn 0] Action: look; Obs: "You are in the middle of
a room. Looking quickly around you, you see nothing.
Available actions: go to cabinet 1, go to cabinet 2..."
[Turn 1] Action: go to toilet 1; Obs: "You arrive at
toilet 1. On the toilet 1, you see a soapbottle 2..."
```

**Status:** Failure (0 Facts Extracted)

- **Narrative Extraction Logic:** Mem0's extraction prompt is optimized for high-level entities and static properties. In Case 1, the LLM successfully maps the input into an atomic "subject-predicate-object" structure, which is ideal for standard user profiling.
- **The Bottleneck in Trajectory Data:**

```
<internal_log>
- Input length: 5022 characters (15 turns)
- Model: Qwen3-32B
- Observation: The extractor fails to identify "facts" within the
  dynamic action-state-observation loops. Critical spatial
  identifiers (e.g., "toilet 1") are ignored as low-level noise.
</internal_log>
```

*Note: As evidenced by Case 2, the extraction-based approach is inherently lossy for agent-based tasks. By bypassing the `infer=True` extraction layer and utilizing raw trajectory segments directly in our **System Branch**, we preserve the environmental context that is otherwise discarded by the LLM-driven fact extractor.*