LongMemEval-V2: Evaluating Long-Term Agent Memory Toward Experienced Colleagues

Di Wu [1] **Zixiang Ji** [1] **Asmi Kawatkar** [1] **Bryan Kwan** [1] **Jia-Chen Gu** [1] **Nanyun Peng** [1] **Kai-Wei Chang** [1]

Abstract

Long-term memory is crucial for agents in specialized web environments, where success depends on recalling interface affordances, workflows, state dynamics, and recurring failure modes. We introduce LongMemEval-V2 (LME-V2), a benchmark for evaluating whether memory systems can accumulate environment-specific experience from multimodal web agent trajectories. LME-V2 contains 451 manually curated questions from customized shopping, forum, admin, and ServiceNow-style environments, with histories ranging from 25M to 115M tokens. Frontier LLMs reach at most 14.1% without trajectory evidence, confirming that LME-V2 requires learned experience beyond parametric knowledge. We evaluate memory under a context-gathering formulation and propose AgentRunbook: AgentRunbook-R is an efficient RAG pipeline over raw states, transitions, and notes, while AgentRunbook-C uses a scaffolded coding agent to gather evidence from trajectory files. AgentRunbook-C achieves the best overall accuracy, reaching 74.9% on LME-V2-Small and 70.1% on LME-V2-Medium, while improving the accuracy and latency trade-off over an off-the-shelf coding agent. We release the benchmark and memory implementations at https://github.com/xiaowu0162/LongMemEval-V2.

1. Introduction

Long-term memory helps large language models (LLMs) operate beyond their context and parameters by storing and recalling information over long horizons (Wu et al., 2022; Packer et al., 2023; Wang et al., 2024b). Memory is especially important for agent systems, where LLMs interact with specialized environments over many steps. Recent works show that memorizing task procedures, interface affordances, and hidden failure modes improve agent performance (Wang et al., 2024a; Zhao et al., 2024; Bouzenia et al., 2025; Wang et al., 2025b; Tang et al., 2025).

However, benchmarks for memory in the agentic context remain limited. Existing memory works mainly evaluate retrieval and reasoning over long documents or user chat histories (Hsieh et al., 2024; Bai et al., 2025; Wu et al., 2025a; Maharana et al., 2024; Tavakoli et al., 2025). Recent works consider evaluating memorization over agent trajectories, but often use simplified game environments (Fang et al., 2026; Li et al., 2026), emphasize limited dependencies within one or a few trajectories (He et al., 2026; Zhao et al., 2026b), or evaluate indirectly through downstream task success (He et al., 2026). As a result, they provide limited insight into whether memory systems can accumulate holistic, environment-specific knowledge from sustained interaction with a complex environment. To highlight this perspective, this paper uses the following framing:

A high-quality memory makes an agent an experienced colleague in a specialized environment.

Driven by this view, we introduce LongMemEval-V2 (LME-V2), a benchmark for evaluating whether memory systems can help web agents acquire the experience needed to become knowledgeable colleagues. LME-V2 leverages customized websites including Magento shopping, shopping admin, Postmill forum, and ServiceNow from WebArena (Zhou et al., 2024) and WorkArena (Drouin et al., 2024; Boisvert et al., 2024). From task-solving web agent trajectories, we manually curate 451 questions covering five core memory abilities: static state recall, dynamic state tracking, workflow knowledge, environment gotchas, and premise awareness. We provide examples in Figure 1 and ability definitions in §3.1. These questions are specific to the customized environments and thus remain generally unanswerable by recent frontier LLMs (§3.4). LME-V2 further pairs the questions with a sequence of web agent trajectories ("haystacks", following Kamradt (2023)), where only a small fraction bears the answers to each question ("needles"). LME-V2-Small provides a 100-trajectory haystack

[1]University of California, Los Angeles. Correspondence to: Di Wu, Kai-Wei Chang <{diwu,kwchang}@cs.ucla.edu>.

Proceedings of the 43rd International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026. Copyright 2026 by the author(s).

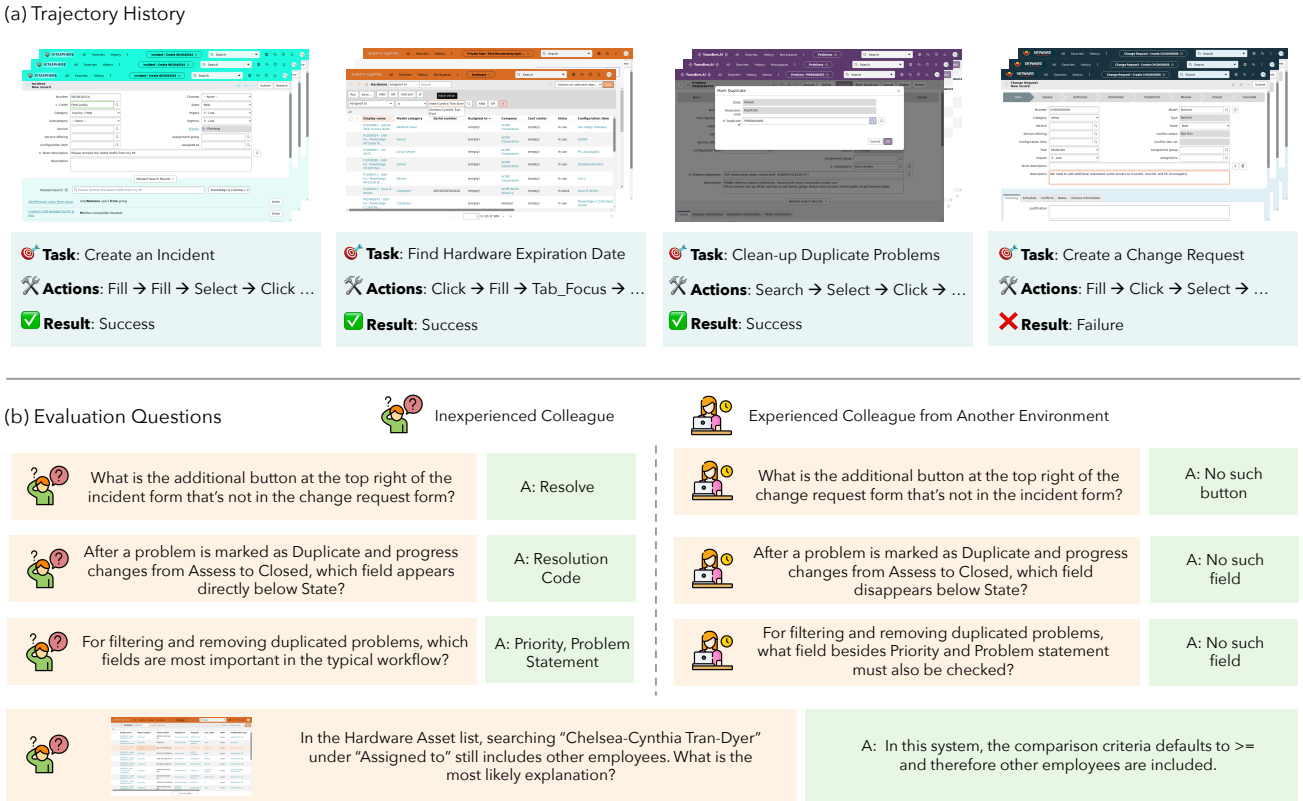


Figure 1. Examples of LongMemEval-V2 questions. We display examples from the WorkArena domain. LME-V2 questions exercise diverse memory abilities: static (row 1), dynamic (row 2), workflow (row 3), gotchas (row 4), and premise awareness (right column).

shared by all questions, and LME-V2-Medium has 500-trajectory question-specific haystacks. Compared to prior benchmarks, LME-V2 poses new challenges with its deep context (25M/115M tokens in the small/medium tiers) and comprehensive memory ability coverage (Table 1).

LME-V2 evaluates memory with a *context gathering* formulation (§3.3). A memory system implements two APIs: `Insert`, which consumes a trajectory, and `Query`, which returns a multimodal memory context for a question. For each question, we stream the associated trajectories sequentially into memory, invoke `Query`, truncate the returned context to a fixed token budget, and ask a fixed reader LLM to answer. This provides a direct evaluation of memory quality with a practical interface that a downstream agent would use. We report both answer accuracy and query latency.

To succeed in LME-V2, a memory system needs to intelligently store and filter information from the noisy agent trajectories, retaining both low-level observations as well as higher-level environment dynamics and procedural knowledge. As a result, naive application of popular agent memory methods could be ineffective as they are biased towards less noisy conversational contexts (Chhikara et al., 2025) or high-level strategic knowledge (Wang et al., 2025b; Ouyang et al., 2025). In this paper, we propose AgentRunbook, a simple yet effective baseline consisting of two variants, optimized separately for efficiency and accuracy. AgentRunbook-R is an efficient retrieval-augmented generation (RAG) pipeline inspired by agentic memory works such as Xu et al. (2025). It prompts an LLM controller to update and to actively query three knowledge pools: raw observations, state transition events, and high-level strategy notes (§4.1). AgentRunbook-R is efficient and covers major memory abilities, but its simple design is not optimized for detailed evidence selection. Inspired by Cao et al. (2026), we propose AgentRunbook-C, a coding agent-based memory method that casts memory management as a file management problem. AgentRunbook-C stores raw trajectories directly as files. At query time, it augments an off-the-shelf coding agent harness with workflow documents, memory manifests, and helper scripts, then invokes the agent to assemble a compact evidence set (§4.2).

We evaluate the memory designs on the small and medium tiers of LME-V2. To begin with, a simple RAG method that retrieves state slices can only achieve an overall accuracy of 40.1%, and AgentRunbook-R further improves to 57.8%. Accuracy-wise, we find the off-the-shelf Codex agent (OpenAI) has competitive performance, achieving a surprisingly high 69.3% accuracy. However, the agent achieves this at a cost of about 182 seconds per query, about

Table 1. Comparison with existing memory and long-context benchmarks. # Sess., # Tok., and # Q denote the history size in sessions or tokens, as well as the total number of questions. MM denotes whether the context or question is a multimodal mixture of text and images. For benchmarks with multiple preset length tiers, we report a range between the minimum length tier and the maximum length tier. For the other benchmarks, we report the averaged context size over all examples.

Benchmark	**Domain**	**Context Profile**			**Question Profile**		**Memory Ability**				
		# Sess.	# Tok.	MM	# Q	MM	Static	Dynamic	Workflows	Gotchas	Premise
General Long Context											
LongBench V2	Mixed	N/A	260k	✗	503	✗	✓	✗	✗	✗	✗
MemoryAgentBench	Mixed	N/A	285k	✗	2,071	✗	✓	✗	✗	✗	✗
CL-Bench	Mixed	N/A	10k	✗	1,899	✗	✓	✗	✓	✗	✗
Conversational Long Context											
LoCoMo	User-user chat	28	∼16k	✓	7,512	✗	✓	✓	✗	✗	✓
LongMemEval-V1	User-assistant chat	48–475	115k–1.5M	✗	500	✗	✓	✓	✗	✗	✓
PersonaMem	User-assistant chat	5–60	26k–951k	✗	5,990	✗	✓	✓	✗	✗	✗
PersonaMem-v2	User-assistant chat	10–20	33k–124k	✓	5,000	✗	✓	✓	✗	✗	✗
BEAM	User-assistant chat	4.5–100	124k–10M	✗	2,000	✗	✓	✓	✗	✗	✓
Agentic Long Context											
MemoryArena	Agent (mixed)	7	40k+	✗	766	✗	✓	✗	✓	✓	✗
AgentLongBench	Game agent	1	31k–4M	✗	6,400	✗	✓	✓	✗	✗	✗
EMemBench	Game agent	1	2k–∞	✓	1,280+	✗	✓	✗	✓	✓	✓
FileGramBench	File-system agent	12	11k	✓	4,333	✗	✓	✗	✓	✗	✗
AMA-Bench	Agent (mixed)	1	57k	✗	2,496	✗	✓	✓	✓	✓	✗
LongMemEval-V2	**Web agent**	**100–498**	**25M–115M**	✓	**451**	✓	✓	✓	✓	✓	✓

6.9 times slower than AgentRunbook-R. With our specialization designs, AgentRunbook-C performs best overall with 72.5% accuracy while being 32% faster than Codex at query time. Our further analyses reveal that AgentRunbook-C significantly advances the accuracy-latency frontier, but the room for future improvement remains large (§5.2). Overall, LME-V2 formulates a new standard for agent memory evaluation and provides a concrete testbed for memory modules that make long-running agents more reliable, adaptive, and useful in real-world environments.

2. Related Work

Long-Context and Memory Evaluation Long-term memory evaluation can be seen as part of the long-lasting effort to evaluate LLMs and retrieval systems on recalling information across extended context. Early line of benchmarks focuses on testing information retrieval, aggregation, and instruction following over long input documents (Hsieh et al., 2024; Karpinska et al., 2024; Modarressi et al., 2025; Bai et al., 2025; Dou et al., 2026). Subsequent work expanded the focus to personalized memory, covering explicit user facts and implicit preferences, with benchmarks such as LoCoMo (Maharana et al., 2024), DialSim (Kim et al., 2024), PerLTQA (Du et al., 2024), LongMemEval (Wu et al., 2025a), PersonaMem (Jiang et al., 2025a;b), and BEAM (Tavakoli et al., 2025). LMEB (Zhao et al., 2026a) isolates the retrieval component and evaluates dense retrievers on memory workloads. In contrast, LME-V2 targets experience memory with context constructed from web agent history trajectories. This shift introduces substantially more complex contexts, a new ability taxonomy, and memory designs centered on agent experience.

Memory Systems for Agents As LLM agents tackle long-horizon tasks in complex environments, memory becomes important both for recalling earlier detailed trajectory context (Hu et al., 2025) and for consolidating high-level knowledge across trajectories (Wang et al., 2025b; Ouyang et al., 2025; Wu et al., 2025b). Memory has also been linked to improving inference-time performance through extended exploration and sleep-time offline consolidation (Ouyang et al., 2025; Lin et al., 2025). Despite this progress, direct evaluation of memory quality in agent settings remains limited. MemoryArena (He et al., 2026) measures memory indirectly through the success rate of interdependent task sequences. AgentLongBench (Fang et al., 2026) and EMemBench (Li et al., 2026) use synthetic agent histories and test recall of details from those traces. FileGram (Liu et al., 2026a) studies reasoning over file system behavior traces. AMA-Bench (Zhao et al., 2026b) is closest to our setting, as it curates questions from agent trajectories in diverse domains such as embodied, web, and gaming agents. However, AMA-Bench focuses on understanding one trajectory, while LME-V2 focuses on environment knowledge induced across many past trajectories. To our knowledge,

LME-V2 is also the first benchmark in this setting to scale the history length to tens or even over 100 million tokens.

Agents as Memory Controllers Recent work on agentic memory proposes memory systems in which memory write and read operations are controlled by an LLM rather than a fixed pipeline. MemGPT (Packer et al., 2023) and StateLM (Liu et al., 2026b) enable models to manage context programmatically. A-MEM (Xu et al., 2025) and Mem0 (Chhikara et al., 2025) introduce scaffolding that allows an LLM to evolve memory content and structure over time. Memory-R1 (Yan et al., 2025) and Mem-α (Wang et al., 2025a) learn memory update actions via reinforcement learning. MemSkill (Zhang et al., 2026) learns memory skills to guide memory update behavior at a finer granularity. In this work, we further expand the notion of agentic memory. Inspired by Cao et al. (2026) and Team et al. (2026), we view a general coding agent with tool use and file system manipulation abilities as a strong controller for file-based memory. Based on this perspective, we design AgentRunbook-C, which augments a coding agent harness with workflow documents, query-time rendered artifacts, and helper scripts, yielding a strong accuracy-latency trade-off on LME-V2.

3. LongMemEval-V2

3.1. Core Memory Ability Definition

What does an experienced colleague internalize after repeatedly working in an environment? We categorize the learned experience into five memory abilities:

- **Static State Recall**. An experienced colleague remembers important landmarks, page layouts, module affordances, and subtle differences across states.
- **Dynamic State Tracking**. An experienced colleague can act as a world model of the environment: given states and actions, they understand how the environment changes.
- **Workflow Knowledge**. An experienced colleague knows the steps needed to perform common tasks in the customized environment.
- **Environment Gotchas**. An experienced colleague is aware of common recurring issues in the current environment and can avoid environment-specific failures.
- **Premise Awareness**. An experienced colleague can recognize assumptions that are valid in another environment but wrong in the current one.

3.2. Annotation

To holistically evaluate these memory abilities, we curate LongMemEval-V2 from multimodal web agent trajectories. The annotation has four steps: trajectory collection, question annotation, answer trajectory labeling, and haystack creation. We present full details in Appendix Section A.

Trajectory Collection We collect trajectories from three web agent benchmarks: WebArena (Zhou et al., 2024), WorkArena (Drouin et al., 2024), and WorkArena++ (Boisvert et al., 2024), leveraging their **OneStopShop**, **CMS**, **Reddit**, **ServiceNow** environments. The trajectories are collected using the AgentLab[1] library, which provides unified state representations, action spaces, and a ReAct-style base agent implementation (Yao et al., 2023). Using the base agent and Codex (OpenAI), we perform rejection sampling with GPT-5.2 (OpenAI, 2025) and GPT-5-mini (OpenAI, 2026) as the LLMs. The final pool contains 599 trajectories from WebArena and 941 from WorkArena/WorkArena++. The overall success rate is 52.0%, and each trajectory contains 28.1 states on average.

Question Annotation All questions are constructed through **manual annotation**. Following the memory ability taxonomy, human experts first inspect the trajectories to identify various information an experienced colleague would naturally learn. We then curate and filter questions to ensure strong proprietary LLMs cannot answer from parametric knowledge alone[2]. Gotchas questions are framed as scenarios where an inexperienced worker sends a message with a screenshot, while the other questions are expressed as text-only true/false, multiple choice, or short answer questions. Finally, based on existing static, dynamic, and workflow questions, we curate abstention questions with wrong premises that the model must identify to succeed. Figure 1 shows example questions in each category. Figure 2 presents source domain, type, and format distribution of the final question pool. On average, questions require 1.4 trajectories to answer (min 1, max 5). However, many dynamic and workflow questions require evidence synthesized from many states within a supporting trajectory.

Answer Trajectory Labeling During annotation, annotators identify a seed set of answer-bearing trajectories for each question. To construct shared history haystacks where we can jointly minimize the number of answer-bearing trajectories for all questions, we perform additional annotation to label all trajectories that contain the answer for each question. We use the Codex coding agent to generate initial proposals. Human experts then verify that the question-trajectory correspondence for trajectories included in the final core haystack set. We provide details in Section A.

[1] https://github.com/ServiceNow/AgentLab.

[2] We manually tested Gemini-3-Pro (Google DeepMind, 2025), GPT-5.2 (OpenAI, 2025), Grok-4.1-thinking (xAI, 2025), and Claude-Opus-4.6 (Anthropic, 2026) and ensured that at least two out of four models answered the questions incorrectly.

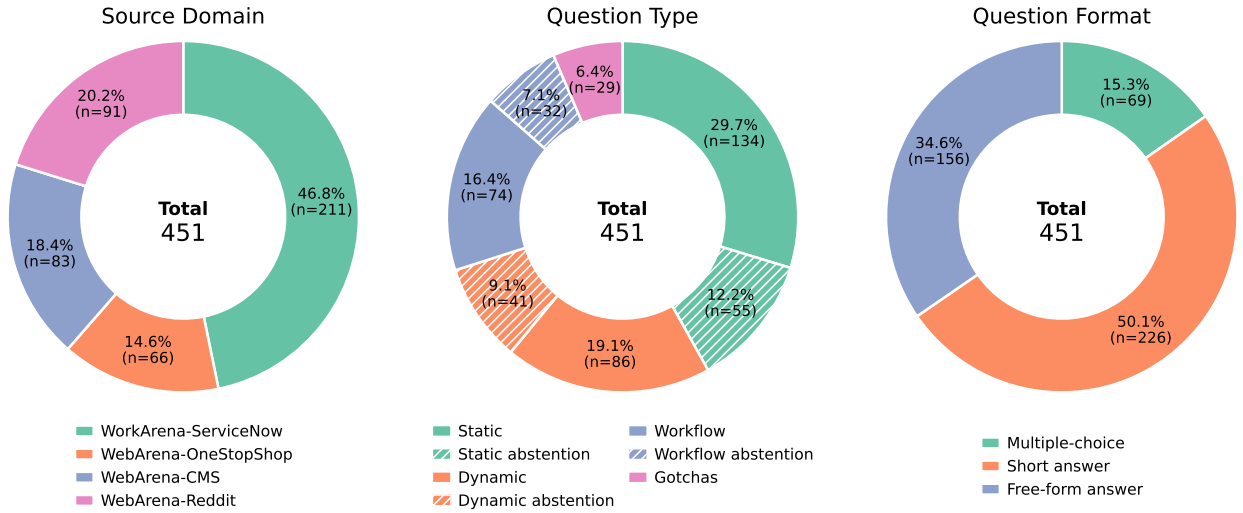


Figure 2. LME-V2 questions cover diverse source domains, question types, and formats.

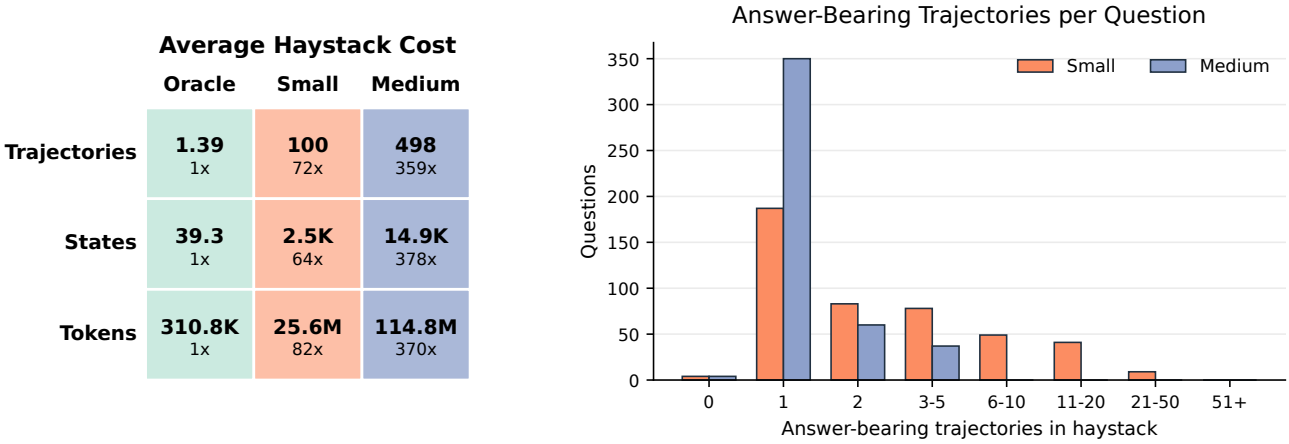


Figure 3. Basic statistics of history haystacks in LME-V2. The small and medium tiers cover a large number of states and tokens (left). Meanwhile, the trajectories containing answers to each question remain generally sparse in the haystack (right).

Haystack Creation Based on the answer trajectory labels, we programmatically assemble two tiers of history trajectory haystacks: a small variant that contains 100 trajectories shared by all questions, and a medium variant that contains roughly 500 trajectories per question. We refer to them as LME-V2-Small and LME-V2-Medium for the rest of the paper. For LME-V2-Small, we create one haystack for the ServiceNow questions and one haystack for the WebArena domains. All haystacks contain a balanced ratio of successful and failed trajectories, and many questions can only be answered from failed trajectories. Figure 3 presents further statistics of the haystacks. The final history lengths of LME-V2-Small and LME-V2-Medium are approximately 25M and 115M tokens, while each question's answer-bearing trajectory set remains sparse in the haystack.

Table 1 compares LME-V2 with previous long-term memory benchmarks. LME-V2 has substantially longer histories than prior long-term memory benchmarks, naturally includes multimodal evaluation, and provides a broad coverage of crucial agent memory capabilities.

3.3. Evaluation Formulation

We formulate LME-V2 as a **context gathering** task. For each question q_i with gold answer y_i, a memory system receives an ordered trajectory haystack $\mathcal{H}_i = \{h_{i,1}, \ldots, h_{i,m_i}\}$, where each $h_{i,j}$ is a trajectory. The system must support two APIs, $\texttt{Insert}(h)$ and $\texttt{Query}(q)$. We sequentially insert all trajectories in $\mathcal{H}_i$, query the final memory with q_i, and obtain a returned context c_i:

$$\mathcal{M}_{i,j} = \texttt{Insert}_{\mathcal{M}_{i,j-1}}(h_{i,j}),$$
$$c_i = \texttt{Query}_{\mathcal{M}_{i,m_i}}(q_i).$$

A fixed reader model R answers from the question and a bounded memory context: $\hat{y}_i = R(q_i, \text{Trunc}(c_i))$[3]. We

[3]We set the truncation budget to 200k tokens empirically.

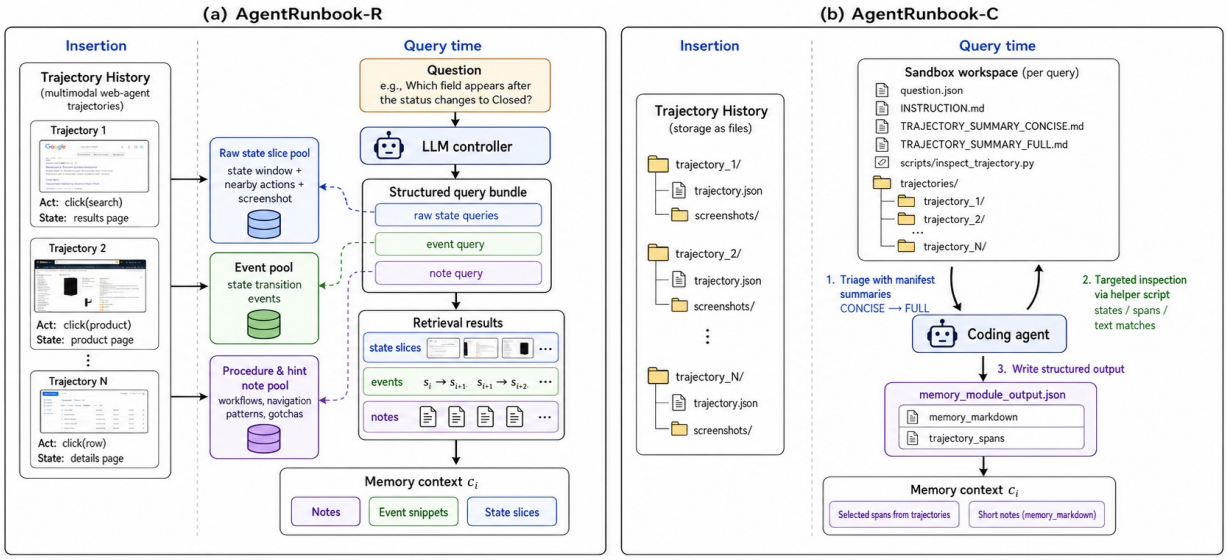


Figure 4. An illustration of the AgentRunbook memory modules. (a) AgentRunbook-R digests trajectories at insertion time and creates dedicated knowledge pools for raw states, events, and notes. At query time, an LLM controller generates multi-stream queries for each pool. (b) AgentRunbook-C stores trajectories as files at insertion time and constructs a sandbox workspace with specialized instructions and manifests for a coding agent to collect the required evidence efficiently.

report answer accuracy and query latency. Accuracy is computed by normalized string matching for structured answers and an LLM judge for free-form answers.

3.4. Pilot Studies

We perform two pilot studies. First, we evaluate whether LME-V2 questions require environment-specific trajectory evidence. Then, we sanity check whether answer-bearing trajectories are sufficient for reliable question answering. These studies use a direct question answering setup rather than the context gathering formulation used in the main experiments, and evaluate non-abstention questions only. Full per-category results, prompts, and sandbox instructions are provided in Appendix Section B. To begin with, can recent frontier LLMs answer LME-V2 questions without the trajectory history? We prompt strong LLMs with only the question. All LLMs perform poorly in this setting: the best model reaches only 14.1% overall accuracy, suggesting that LME-V2 questions generally cannot be answered from public or parametric knowledge alone. Second, we give models oracle access to the answer-bearing trajectories to isolate the difficulty of reading and grounding trajectory evidence. Long-context prompting shows much higher accuracy but remains limited due to the trajectory size exceeding the model's context window. These findings confirm that the answer trajectory labelings are accurate enough and motivate our memory method design.

4. AgentRunbook

LME-V2 is challenging because its history is too large for reliable direct processing. In addition, the evidence needed for a question can mix low-level UI observations, state transitions, and reusable task procedures. Memory modules therefore need to organize trajectories into compact representations and index them for targeted recall. We propose two memory designs: AgentRunbook-R, a structured RAG pipeline with separate knowledge pools, and AgentRunbook-C, a coding agent based method that casts trajectory memory as a file management problem. The two methods represent different accuracy-latency operating points.

4.1. AgentRunbook-R

AgentRunbook-R, where R denotes RAG, extracts structured memory items at insertion time and retrieves them at query time. To recall information at different granularities, AgentRunbook-R uses separate knowledge pools and a retrieval mechanism over these pools. Given a trajectory h, AgentRunbook-R builds three memory pools. The **raw state slice** pool stores windows centered at trajectory states, including local UI observations and nearby actions. This pool preserves fine-grained visual and textual evidence. The **state transition event** pool stores events extracted from consecutive states. These events describe how actions change the environment, accumulating evidence for an environment world model. The **procedure and hint note** pool stores trajectory-level notes that capture reusable workflows, navigation patterns, and environment-specific gotchas. This

Table 2. Main results with baselines and ablations. The downstream reader is always Qwen3.5-9B. We boldface the best results in each method family. † means statistically significantly outperforming the non-ablation baselines via paired bootstrap test ($p < 0.05$). AgentRunbook strongly outperforms the baseline in RAG family and achieves a superior latency in coding agent family.

Method	LME-V2-Small						LME-V2-Medium					
	Overall	Static	Dynamic	Workflow	Gotchas	Latency	Overall	Static	Dynamic	Workflow	Gotchas	Latency
No retrieval	0.013	0.000	0.008	0.094	0.138	0s	0.013	0.000	0.008	0.094	0.138	0s
RAG Methods (Controller = Qwen3.5 9B, thinking enabled)												
RAG: query → slice	0.428	0.471	0.425	0.415	0.207	0.1s	0.381	0.434	0.405	0.293	0.242	0.1s
RAG: query → slice + notes	0.510	0.524	0.496	0.528	0.414	0.2s	0.459	0.487	0.472	0.434	0.310	0.3s
AgentRunbook-R	**0.586**†	**0.661**†	0.583†	0.528	0.310	26.9s	**0.570**†	**0.630**†	**0.614**†	**0.472**†	**0.345**	25.8s
– raw slice pool	0.423	0.286	0.551	**0.538**	**0.345**	16.7s	0.335	0.233	0.433	0.377	0.413	17.1s
– event pool	0.556	0.614	0.559	0.528	0.276	19.1s	0.484	0.534	0.496	0.434	0.276	18.5s
– note pool	0.579	0.651	**0.614**	0.481	0.310	22.8s	0.499	0.561	0.543	0.396	0.276	20.5s
Coding Agent Methods (Controller = GPT-5.4-mini, xhigh reasoning)												
Codex	0.699	0.804	0.670	0.575	**0.586**	177.2s	0.687	0.783	0.646	0.613	**0.517**	185.8s
AgentRunbook-C	**0.749**†	0.820†	**0.724**†	**0.726**†	0.483	108.3s	0.701	0.788	**0.701**†	0.613	0.449	139.9s
– workflow	0.701	0.772	0.677	0.632	**0.586**	167.9s	0.641	0.709	0.646	0.575	0.414	231.9s
– manifest artifacts	0.747	**0.847**	0.709	0.698	0.448	155.0s	0.681	0.767	0.685	0.576	0.483	211.6s
– helper functions	0.714	0.783	**0.724**	0.660	0.414	145.9s	**0.718**	**0.804**	0.693	**0.689**	0.380	182.5s

pool is inspired by prior works that consolidate trajectory experience into compact reusable knowledge (Wang et al., 2025b; Ouyang et al., 2025).

At query time, AgentRunbook-R uses an LLM controller to reason about the query and the current memory snapshot, then generate retrieval queries for the knowledge pools: multiple raw state queries for exact UI evidence, one event query for important state changes, and one note query for procedural knowledge. The controller may skip irrelevant streams. Each query retrieves from its corresponding pool using dense retrieval, and the results are rendered as a multi-modal memory context c_i. This design keeps query latency low while supporting evidence at different granularities.

4.2. AgentRunbook-C

AgentRunbook-C, where C denotes coding agent, is motivated by the observation that general coding agents are effective file system manipulators and tool users (Cao et al., 2026). Rather than compressing retrieval behavior into a fixed vector search pipeline, AgentRunbook-C stores trajectories directly as files and uses a coding agent to search, inspect, and select evidence at query time.

An off-the-shelf coding agent, however, is not optimized as a memory module. It may over-explore, under-explore, or inspect the data inefficiently. AgentRunbook-C adds three lightweight scaffolding components to the coding agent. First, a **workflow document** instructs the agent to act as a memory module and outlines the steps for collecting evidence. Second, query-time **manifest artifacts** summarize the current memory layout, helping the agent shortlist relevant trajectories before detailed inspection. Third, a **helper script** exposes common trajectory inspection operations, such as viewing a state span or searching within a trajectory. At insertion time, AgentRunbook-C stores each trajectory on disk. At query time, it creates a sandbox with the question, workflow document, helper script, and rendered manifest artifacts. The coding agent then writes a structured retrieval output containing a short memory note and selected trajectory state spans, which are rendered into c_i.

5. Experiments

We evaluate all methods on LME-V2 under the context gathering formulation. The returned memory context is truncated to 200K tokens and answered by a fixed Qwen3.5-9B reader (Qwen Team, 2026). For RAG methods, we use Qwen3.5-9B as the memory controller and Qwen3-Embedding-8B for retrieval. For coding agent methods, we use Codex and GPT-5.4-mini with different reasoning efforts. The full implementation details of AgentRunbook and baselines are provided in Section C.

5.1. Main Results

As shown in Table 2, the no-retrieval baseline is near zero, confirming that the reader cannot answer without memory context. A simple query-to-slice RAG baseline reaches 42.8% on LME-V2-Small and 38.1% on LME-V2-Medium, while adding trajectory notes improves performance to 51.0% and 45.9%. AgentRunbook-R further improves over the strongest RAG baseline, reaching 58.6% on LME-V2-Small and 57.0% on LME-V2-Medium. The ablations show that the raw slice pool is important for static questions, while removing the event pool harms static, dynamic, and gotchas questions. Workflow questions benefit from consolidating trajectory experience into reusable events and notes rather than only retrieving local observations.

AgentRunbook-C achieves the best overall accuracy, reaching 74.9% on LME-V2-Small and 70.1% on LME-V2-Medium. It also improves over vanilla Codex, which reaches

Figure 5. AgentRunbook improves the query accuracy-latency frontier of memory modules.

69.9% and 68.7%, respectively. The ablations show that workflow instructions are consistently important, while manifest artifacts mainly improve efficiency. Helper functions affect the performance in a mixed way: they improve the small-tier result and reduce latency compared with the most expensive ablations, but their effect on medium-tier accuracy is not uniformly positive. Overall, the results show that coding agents can serve as strong memory controllers with a proper file-based environment. In Appendix Section D, we further analyze the error patterns and the tool calling behavior of AgentRunbook-C.

5.2. Accuracy and Latency Trade-off

An ideal memory method should support both accurate and efficient querying. Across multiple factors, we find the reasoning effort of the memory controller has a large and direct effect on the overall query latency. We thus use it to analyze the methods across different operating points. As shown in Figure 5, AgentRunbook-R provides a moderate-accuracy, low-latency baseline: it substantially improves over the slice-plus-note baseline while keeping latency around 26 seconds, or much lower without thinking. This makes it a strong choice when query efficiency is prioritized. AgentRunbook-C moves the accuracy and latency frontier upward. Across reasoning effort settings, the scaffolded coding agent memory consistently offers a better trade-off than directly using the off-the-shelf coding agent. This suggests that coding agents are more effective as memory controllers when paired with workflow, manifests, and tools.

6. Conclusion

We introduce LongMemEval-V2, a long-term memory benchmark that formulates a new standard for agent memory evaluation: memory systems should help agents become experienced operators of specialized environments. LME-V2 holistically covers five memory abilities with beyond 100M-token context from large multimodal web-agent histories. We further propose AgentRunbook-R, which improves standard RAG-based methods with dedicated memory pools, and AgentRunbook-C, which leverages the file manipulation abilities of coding agents and further improves the accuracy and latency through lightweight workflow guidance, manifests, and inspection tools. We hope LME-V2 provides a concrete testbed for memory modules that support more reliable real-world agents.

References

Anthropic. Claude Opus 4.6 System Card. https://www-cdn.anthropic.com/14e4fb01875d2a69f646fa5e574dea2b1c0ff7b5.pdf, 2026. Accessed: 2026-04-23.

Bai, Y., Tu, S., Zhang, J., Peng, H., Wang, X., Lv, X., Cao, S., Xu, J., Hou, L., Dong, Y., Tang, J., and Li, J. Longbench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. In Che, W., Nabende, J., Shutova, E., and Pilehvar, M. T. (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pp. 3639–3664. Association for Computational Linguistics, 2025. URL https://aclanthology.org/2025.acl-long.183/.

Boisvert, L., Thakkar, M., Gasse, M., Caccia, M., de Chezelles, T. L. S., Cappart, Q., Chapados, N., Lacoste, A., and Drouin, A. Workarena++: Towards compositional planning and reasoning-based common knowledge work tasks. In Globersons, A., Mackey, L., Belgrave, D., Fan, A., Paquet, U., Tomczak, J. M., and Zhang, C. (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, Decem-*

ber 10 - 15, 2024, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/0b82662b6c32e887bb252a74d8cb2d5e-Abstract-Datasets_and_Benchmarks_Track.html.

Bouzenia, I., Devanbu, P. T., and Pradel, M. Repairagent: An autonomous, llm-based agent for program repair. In *47th IEEE/ACM International Conference on Software Engineering, ICSE 2025, Ottawa, ON, Canada, April 26 - May 6, 2025*, pp. 2188–2200. IEEE, 2025. doi: 10.1109/ICSE55347.2025.00157. URL https://doi.org/10.1109/ICSE55347.2025.00157.

Cao, W., Yin, X., Dhingra, B., and Zhou, S. Coding agents are effective long-context processors. *CoRR*, abs/2603.20432, 2026. doi: 10.48550/ARXIV.2603.20432. URL https://doi.org/10.48550/arXiv.2603.20432.

Chhikara, P., Khant, D., Aryan, S., Singh, T., and Yadav, D. Mem0: Building production-ready AI agents with scalable long-term memory. In Lynce, I., Murano, N., Vallati, M., Villata, S., Chesani, F., Milano, M., Omicini, A., and Dastani, M. (eds.), *ECAI 2025 - 28th European Conference on Artificial Intelligence, 25-30 October 2025, Bologna, Italy - Including 14th Conference on Prestigious Applications of Intelligent Systems (PAIS 2025)*, Frontiers in Artificial Intelligence and Applications, pp. 2993–3000. IOS Press, 2025. doi: 10.3233/FAIA251160. URL https://doi.org/10.3233/FAIA251160.

Dou, S., Zhang, M., Yin, Z., Huang, C., Shen, Y., Wang, J., Chen, J., Ni, Y., Ye, J., Zhang, C., Xie, H., Hu, J., Wang, S., Wang, W., Xiao, Y., Liu, Y., Xu, Z., Guo, Z., Zhou, P., Gui, T., Wu, Z., Qiu, X., Zhang, Q., Huang, X., Jiang, Y., Wang, D., and Yao, S. Cl-bench: A benchmark for context learning. *CoRR*, abs/2602.03587, 2026. doi: 10.48550/ARXIV.2602.03587. URL https://doi.org/10.48550/arXiv.2602.03587.

Drouin, A., Gasse, M., Caccia, M., Laradji, I. H., Verme, M. D., Marty, T., Vázquez, D., Chapados, N., and Lacoste, A. Workarena: How capable are web agents at solving common knowledge work tasks? In Salakhutdinov, R., Kolter, Z., Heller, K. A., Weller, A., Oliver, N., Scarlett, J., and Berkenkamp, F. (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, Proceedings of Machine Learning Research, pp. 11642–11662. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/drouin24a.html.

Du, Y., Wang, H., Zhao, Z., Liang, B., Wang, B., Zhong, W., Wang, Z., and Wong, K. Perltqa: A personal long-term memory dataset for memory classification, retrieval, and synthesis in question answering. *CoRR*, abs/2402.16288, 2024. doi: 10.48550/ARXIV.2402.16288. URL https://doi.org/10.48550/arXiv.2402.16288.

Fang, S., Wang, Y., Liu, X., Lu, J., Tan, C., Chen, X., Zheng, Y., Huang, X., and Qiu, X. Agentlongbench: A controllable long benchmark for long-contexts agents via environment rollouts. *CoRR*, abs/2601.20730, 2026. doi: 10.48550/ARXIV.2601.20730. URL https://doi.org/10.48550/arXiv.2601.20730.

Google DeepMind. Gemini 3 Pro Model Card. https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-3-Pro-Model-Card.pdf, 2025. Accessed: 2026-04-23.

He, Z., Wang, Y., Zhi, C., Hu, Y., Chen, T., Yin, L., Chen, Z., Wu, T. A., Ouyang, S., Wang, Z., Pei, J., McAuley, J. J., Choi, Y., and Pentland, A. Memoryarena: Benchmarking agent memory in interdependent multi-session agentic tasks. *CoRR*, abs/2602.16313, 2026. doi: 10.48550/ARXIV.2602.16313. URL https://doi.org/10.48550/arXiv.2602.16313.

Hsieh, C., Sun, S., Kriman, S., Acharya, S., Rekesh, D., Jia, F., Zhang, Y., and Ginsburg, B. RULER: what's the real context size of your long-context language models? *CoRR*, abs/2404.06654, 2024. doi: 10.48550/ARXIV.2404.06654. URL https://doi.org/10.48550/arXiv.2404.06654.

Hu, M., Chen, T., Chen, Q., Mu, Y., Shao, W., and Luo, P. Hiagent: Hierarchical working memory management for solving long-horizon agent tasks with large language model. In Che, W., Nabende, J., Shutova, E., and Pilehvar, M. T. (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pp. 32779–32798. Association for Computational Linguistics, 2025. URL https://aclanthology.org/2025.acl-long.1575/.

Jiang, B., Hao, Z., Cho, Y., Li, B., Yuan, Y., Chen, S., Ungar, L. H., Taylor, C. J., and Roth, D. Know me, respond to me: Benchmarking llms for dynamic user profiling and personalized responses at scale. *CoRR*, abs/2504.14225, 2025a. doi: 10.48550/ARXIV.2504.14225. URL https://doi.org/10.48550/arXiv.2504.14225.

Jiang, B., Yuan, Y., Shen, M., Hao, Z., Xu, Z., Chen, Z., Liu, Z., Vijjini, A. R., He, J., Yu, H., Poovendran, R., Wornell, G. W., Ungar, L. H., Roth, D., Chen, S., and Taylor, C. J. Personamem-v2: Towards personalized intelligence via learning implicit user personas and agentic memory. *CoRR*, abs/2512.06688, 2025b. doi: 10.48550/ARXIV.2512.06688. URL https://doi.org/10.48550/arXiv.2512.06688.

Kamradt, G. Needle in a haystack: Pressure testing llms. https://github.com/gkamradt/LLMTest_NeedleInAHaystack, 2023. GitHub repository.

Karpinska, M., Thai, K., Lo, K., Goyal, T., and Iyyer, M. One thousand and one pairs: A "novel" challenge for long-context language models. In Al-Onaizan, Y., Bansal, M., and Chen, Y. (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, pp. 17048–17085. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.EMNLP-MAIN.948. URL https://doi.org/10.18653/v1/2024.emnlp-main.948.

Kim, J., Chay, W., Hwang, H., Kyung, D., Chung, H., Cho, E., Jo, Y., and Choi, E. Dialsim: A real-time simulator for evaluating long-term dialogue understanding of conversational agents. *CoRR*, abs/2406.13144, 2024. doi: 10.48550/ARXIV.2406.13144. URL https://doi.org/10.48550/arXiv.2406.13144.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

Li, X., Zhu, Z., Liu, S., Ma, Y., Zang, Y., Cao, Y., and Sun, A. Emembench: Interactive benchmarking of episodic memory for VLM agents. *CoRR*, abs/2601.16690, 2026. doi: 10.48550/ARXIV.2601.16690. URL https://doi.org/10.48550/arXiv.2601.16690.

Lin, K., Snell, C., Wang, Y., Packer, C., Wooders, S., Stoica, I., and Gonzalez, J. E. Sleep-time compute: Beyond inference scaling at test-time. *CoRR*, abs/2504.13171, 2025. doi: 10.48550/ARXIV.2504.13171. URL https://doi.org/10.48550/arXiv.2504.13171.

Liu, S., Tian, S., Hu, K., Dong, Y., Yang, Z., Li, B., Yang, J., Loy, C. C., and Liu, Z. Filegram: Grounding agent personalization in file-system behavioral traces. *arXiv preprint arXiv:2604.04901*, 2026a.

Liu, X., Liang, T., Ma, D., Zhou, D., Mi, H., He, P., and Wang, Y. The pensieve paradigm: Stateful language models mastering their own context. *CoRR*, abs/2602.12108, 2026b. doi: 10.48550/ARXIV.2602.12108. URL https://doi.org/10.48550/arXiv.2602.12108.

Maharana, A., Lee, D., Tulyakov, S., Bansal, M., Barbieri, F., and Fang, Y. Evaluating very long-term conversational memory of LLM agents. In Ku, L., Martins, A., and Srikumar, V. (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pp. 13851–13870. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.ACL-LONG.747. URL https://doi.org/10.18653/v1/2024.acl-long.747.

Modarressi, A., Deilamsalehy, H., Dernoncourt, F., Bui, T., Rossi, R. A., Yoon, S., and Schütze, H. Nolima: Long-context evaluation beyond literal matching. In Singh, A., Fazel, M., Hsu, D., Lacoste-Julien, S., Berkenkamp, F., Maharaj, T., Wagstaff, K., and Zhu, J. (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, Proceedings of Machine Learning Research. PMLR / OpenReview.net, 2025. URL https://proceedings.mlr.press/v267/modarressi25a.html.

OpenAI. Codex CLI. https://github.com/openai/codex. Open-source coding agent software. Accessed: 2026-04-24.

OpenAI. Update to GPT-5 System Card: GPT-5.2. https://openai.com/index/gpt-5-system-card-update-gpt-5-2/, 2025. Accessed: 2026-04-23.

OpenAI. GPT-5 System Card, 2026. URL https://arxiv.org/abs/2601.03267.

OpenRouter. OpenRouter. https://openrouter.ai/, 2026. AI model routing and inference API. Accessed: 2026-04-24.

Ouyang, S., Yan, J., Hsu, I., Chen, Y., Jiang, K., Wang, Z., Han, R., Le, L. T., Daruki, S., Tang, X., Tirumalashetty, V., Lee, G., Rofouei, M., Lin, H., Han, J., Lee, C., and Pfister, T. Reasoningbank: Scaling agent self-evolving with reasoning memory. *CoRR*, abs/2509.25140, 2025. doi: 10.48550/ARXIV.2509.25140. URL https://doi.org/10.48550/arXiv.2509.25140.

Packer, C., Fang, V., Patil, S. G., Lin, K., Wooders, S., and Gonzalez, J. E. Memgpt: Towards llms as operating systems. *CoRR*, abs/2310.08560, 2023. doi: 10.48550/ARXIV.2310.08560. URL https://doi.org/10.48550/arXiv.2310.08560.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Tang, X., Hu, T., Ye, M., Shao, Y., Yin, X., Ouyang, S., Zhou, W., Lu, P., Zhang, Z., Zhao, Y., Cohan, A., and Gerstein, M. Chemagent: Self-updating library in large language models improves chemical reasoning. *CoRR*, abs/2501.06590, 2025. doi: 10.48550/ARXIV.

2501.06590. URL https://doi.org/10.48550/arXiv.2501.06590.

Tavakoli, M., Salemi, A., Ye, C., Abdalla, M., Zamani, H., and Mitchell, J. R. Beyond a million tokens: Benchmarking and enhancing long-term memory in llms. *CoRR*, abs/2510.27246, 2025. doi: 10.48550/ARXIV.2510.27246. URL https://doi.org/10.48550/arXiv.2510.27246.

Team, C., Hao, S., Zhang, Z., Liang, Z., Liu, T., Zha, Y., Gao, Q., Chen, J., Wang, Z., Cheng, Z., Zhang, H., Wang, J., Jin, H., Zheng, B., Zhou, K., Wang, Y., Yao, F., Liu, L., Li, Y., Li, Z., Han, Z., Promthaw, P., Cerruti, T., Fu, X., Ma, Z., Shang, J., Qin, L., McAuley, J., Xing, E. P., Liu, Z., Srivastava, R. K., and Hu, Z. Cocoabench: Evaluating unified digital agents in the wild. 2026. URL https://arxiv.org/abs/2604.11201.

Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., and Anandkumar, A. Voyager: An open-ended embodied agent with large language models. *Trans. Mach. Learn. Res.*, 2024, 2024a. URL https://openreview.net/forum?id=ehfRiF0R3a.

Wang, Y., Gao, Y., Chen, X., Jiang, H., Li, S., Yang, J., Yin, Q., Li, Z., Li, X., Yin, B., Shang, J., and McAuley, J. J. MEMORYLLM: towards self-updatable large language models. In Salakhutdinov, R., Kolter, Z., Heller, K. A., Weller, A., Oliver, N., Scarlett, J., and Berkenkamp, F. (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, Proceedings of Machine Learning Research, pp. 50453–50466. PMLR / OpenReview.net, 2024b. URL https://proceedings.mlr.press/v235/wang24s.html.

Wang, Y., Takanobu, R., Liang, Z., Mao, Y., Hu, Y., McAuley, J. J., and Wu, X. Mem-α: Learning memory construction via reinforcement learning. *CoRR*, abs/2509.25911, 2025a. doi: 10.48550/ARXIV.2509.25911. URL https://doi.org/10.48550/arXiv.2509.25911.

Wang, Z. Z., Mao, J., Fried, D., and Neubig, G. Agent workflow memory. In Singh, A., Fazel, M., Hsu, D., Lacoste-Julien, S., Berkenkamp, F., Maharaj, T., Wagstaff, K., and Zhu, J. (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, Proceedings of Machine Learning Research. PMLR / OpenReview.net, 2025b. URL https://proceedings.mlr.press/v267/wang25bx.html.

Wu, D., Wang, H., Yu, W., Zhang, Y., Chang, K., and Yu, D. Longmemeval: Benchmarking chat assistants on long-term interactive memory. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025a. URL https://openreview.net/forum?id=pZiyCaVuti.

Wu, W., Zhou, K., Yuan, R., Yu, V., Wang, S., Hu, Z., and Huang, B. Auto-scaling continuous memory for GUI agent. *CoRR*, abs/2510.09038, 2025b. doi: 10.48550/ARXIV.2510.09038. URL https://doi.org/10.48550/arXiv.2510.09038.

Wu, Y., Rabe, M. N., Hutchins, D., and Szegedy, C. Memorizing transformers. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=TrjbxzRcnf-.

xAI. Grok 4.1 Model Card. https://data.x.ai/2025-11-17-grok-4-1-model-card.pdf, 2025. Accessed: 2026-04-23.

Xu, W., Liang, Z., Mei, K., Gao, H., Tan, J., and Zhang, Y. A-MEM: agentic memory for LLM agents. *CoRR*, abs/2502.12110, 2025. doi: 10.48550/ARXIV.2502.12110. URL https://doi.org/10.48550/arXiv.2502.12110.

Yan, S., Yang, X., Huang, Z., Nie, E., Ding, Z., Li, Z., Ma, X., Schütze, H., Tresp, V., and Ma, Y. Memory-r1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. *CoRR*, abs/2508.19828, 2025. doi: 10.48550/ARXIV.2508.19828. URL https://doi.org/10.48550/arXiv.2508.19828.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K. R., and Cao, Y. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Zhang, H., Long, Q., Bao, J., Feng, T., Zhang, W., Yue, H., and Wang, W. Memskill: Learning and evolving memory skills for self-evolving agents. *CoRR*, abs/2602.02474, 2026. doi: 10.48550/ARXIV.2602.02474. URL https://doi.org/10.48550/arXiv.2602.02474.

Zhao, A., Huang, D., Xu, Q., Lin, M., Liu, Y., and Huang, G. Expel: LLM agents are experiential learners. In Wooldridge, M. J., Dy, J. G., and Natarajan, S. (eds.), *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth*

Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada, pp. 19632–19642. AAAI Press, 2024. doi: 10.1609/AAAI.V38I17.29936. URL https://doi.org/10.1609/aaai.v38i17.29936.

Zhao, X., Hu, X., Xu, J., Tang, D., Zhang, X., Zhou, M., Zhong, Y., Zhou, Y., Shan, Z., Zhang, M., Hu, B., and Zhang, M. LMEB: long-horizon memory embedding benchmark. *CoRR*, abs/2603.12572, 2026a. doi: 10.48550/ARXIV.2603.12572. URL https://doi.org/10.48550/arXiv.2603.12572.

Zhao, Y., Yuan, B., Huang, J., Yuan, H., Yu, Z., Xu, H., Hu, L., Shankarampeta, A., Huang, Z., Ni, W., Tian, Y., and Zhao, J. Ama-bench: Evaluating long-horizon memory for agentic applications. *CoRR*, abs/2602.22769, 2026b. doi: 10.48550/ARXIV.2602.22769. URL https://doi.org/10.48550/arXiv.2602.22769.

Zhou, S., Xu, F. F., Zhu, H., Zhou, X., Lo, R., Sridhar, A., Cheng, X., Ou, T., Bisk, Y., Fried, D., Alon, U., and Neubig, G. Webarena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=oKn9c6ytLx.

A. LongMemEval-V2: Further Benchmark Construction Details

A.1. Trajectory Collection

Domain and task selection. We collect trajectories from WebArena, WorkArena, and WorkArena++. For WebArena, we use the OneStopShop, CMS, and Reddit tasks because our pilot inspection found that these websites contain more environment-specific customizations than Wikipedia, Map, and GitLab. WorkArena and WorkArena++ are both based on ServiceNow, so we include all tasks from both benchmarks.

AgentLab harness and action space. We use AgentLab to collect trajectories under a unified web agent interface. AgentLab provides benchmark wrappers, observation preprocessing, high-level browser actions, and trajectory logging. The agent observes accessibility tree representations and screenshots, and interacts with the browser through a high-level BrowserGym action API. Elements are referenced by `bid` identifiers from the accessibility tree, and the agent emits exactly one Python-like action call at each step.

For WorkArena and WorkArena++, we use the following action space:

```
noop(wait_ms=1000), scroll(delta_x, delta_y),
fill(bid, value, enable_autocomplete_menu=False),
select_option(bid, options), click(bid, button='left', modifiers=[]),
dblclick(bid, button='left', modifiers=[]),
hover(bid), press(bid, key_comb), focus(bid), clear(bid),
drag_and_drop(from_bid, to_bid), tab_focus(index), new_tab(), tab_close(),
go_back(), go_forward(), goto(url),
send_msg_to_user(text), report_infeasible(reason).
```

For WebArena, we use:

```
noop(wait_ms=1000), scroll(delta_x, delta_y), keyboard_press(key),
click(bid, button='left', modifiers=[]),
fill(bid, value, enable_autocomplete_menu=False), hover(bid),
tab_focus(index), new_tab(), go_back(), go_forward(), goto(url),
tab_close(), select_option(bid, options),
send_msg_to_user(text), report_infeasible(reason).
```

The main difference is that WebArena includes `keyboard_press(key)`, while WorkArena and WorkArena++ include additional element-specific actions such as `dblclick`, `press`, `focus`, `clear`, and `drag_and_drop`.

Trajectory collection agents. Most trajectories are collected with AgentLab's generic ReAct-style agent using GPT-5-mini and GPT-5.2 as the underlying LLMs. In addition, we use a manual action agent controlled by Codex with GPT-5.2 at xhigh reasoning effort. The manual action agent does not change the environment interface or action space. It displays the current observation, accepts a proposed high-level action string from the operator, validates it against the same BrowserGym action API, and logs the resulting transition in the standard AgentLab trajectory format. We use this Codex-controlled manual-action setup mainly for Level-3 WorkArena++ tasks, where additional targeted collection is needed to obtain successful trajectories across task categories.

Rejection sampling and trajectory filtering. Our primary goal during rejection sampling is to obtain successful trajectories for each task, while also retaining useful failure trajectories. We initially aim to collect both one successful and one failed trajectory per task instance when possible. In practice, many tasks end up with only one successful or one failed trajectory because additional sampling becomes costly. The final trajectory pool contains 599 WebArena trajectories and 941 WorkArena/WorkArena++ trajectories. On average the trajectories contain 28.1 states. Overall, 73.1% of trajectories come from GPT-5-mini, 22.9% from GPT-5.2, and 4.0% from Codex with GPT-5.2 (xhigh reasoning).

We also considered trajectories released through the official WebArena leaderboard, but did not use them as the main data source. Some public trajectories lack screenshots, which are essential for our multimodal haystack design. In addition, many

Table 3. Examples of goal sanitization for WorkArena and WorkArena++ Level-2/3 trajectories. The rewritten goals preserve task intent and task-specific values while removing explicit navigation routes and step-by-step module hints.

Task family	**Original goal excerpt**	**Sanitized goal excerpt**
Duplicate problem cleanup	Clean-up your duplicate problems. Concretely, navigate to the "Assigned to me" module of the "Problem" application. Create a filter where "Problem statement" contains `#SERIES-5ea261ef-8`. Mark problems with duplicated problem statements as such.	Clean-up your duplicate problems. Review your own assigned problems where "Problem statement" contains `#SERIES-5ea261ef-8`. Mark problems with duplicated problem statements as such.
Dashboard-driven catalog restocking	Retrieve information from the chart with title `#CAT012007808`. Navigate to `Reports > View/Run`, search for the report, then navigate to `Self-Service > Service Catalog` and place an order for the least available item.	Retrieve information from the chart with title `#CAT012007808`. Find the greatest stock value and the least available item in the chart. For the least available item, place an order for extra items such that its quantity matches the value you found.
Requested-item reorder	Order same item as Kathryn-Lisa Ibarra-Stewart. Navigate to the "Requested Items" module of "Self-Service", filter by "Requested for", then navigate to the "Service Catalog" module and order the item with the specified quantity and configuration.	Order same item as Kathryn-Lisa Ibarra-Stewart. Find the item previously requested for Kathryn-Lisa Ibarra-Stewart. Order the item with the specified quantity and configuration.
Hardware asset lookup	Find the warranty expiration date for Julia-Dylan Ray-Mclean's laptop. Navigate to `Portfolios > Hardware Assets`, filter where "Assigned to" is Julia-Dylan Ray-Mclean, and extract the "Warranty expiration" field.	Find the warranty expiration date for Julia-Dylan Ray-Mclean's laptop and report it.

leaderboard trajectories are produced by agents specialized for WebArena and are often very short, making them less natural for studying environment experience accumulated from general agents. By contrast, trajectories generated by GPT-5-mini and GPT-5.2 use strong general models and provide successful examples across all selected task categories. We use Codex sparingly because it is more expensive, and because the coding agent setup may inadvertently bring in additional external knowledge from e.g., web searches.

Final trajectory artifact and goal sanitization. For each retained trajectory, the final artifact consists of the task goal and an ordered sequence of state-action pairs. Each state contains the screenshot and the accessibility tree, and each action is the high-level BrowserGym action emitted at that step. For WorkArena and WorkArena++ Level-2 and Level-3 tasks, the original goal descriptions sometimes contain detailed navigational hints, which can make procedure questions answerable by reading the route directly from the goal rather than learning it from the trajectory. We therefore sanitize the goal descriptions while preserving the task intent and task-specific payload. We also sanitize repeated appearances of the original goal in trajectory states, such as the initial task description and later copied task text. Table 3 shows representative examples.

A.2. Question Annotation

The annotation team consists of one graduate student and three undergraduate students. Before writing questions, annotators familiarize themselves with the target environments by interacting with the sandbox websites and inspecting collected trajectories. The project lead explains the memory ability definitions as we defined in the main text and annotators discuss ambiguous cases with the project lead throughout the process.

Annotators first inspect trajectories to identify environment-specific facts, state changes, workflows, and gotchas that an experienced colleague should know. They then write questions whose answers require this experience rather than general knowledge of the public base platforms. Each question is validated by at least one additional annotator, who checks that the question is answerable from the intended trajectory evidence, that the answer is correct, and that the question type matches the intended memory ability. When uncertainty remains, the annotators resolve it through discussion with the project lead.

We use the LMArena webpage[4] to test whether strong LLMs can answer candidate questions from parametric knowledge

[4] https://arena.ai/

alone. Empirically, we find that models often answer many candidate questions correctly because they know the public versions of Magento, Postmill, and ServiceNow. We therefore filter many questions and rewrite or perturb others until the models fail. In some cases, we first write a free-form question, observe the models' incorrect answers, and then use these wrong answers as distractors to construct a more challenging multiple-choice question. In the final context gathering evaluation, the reader is also prompted to answer based on the returned memory context and to avoid guessing, which further reduces the chance that accuracy comes from parametric knowledge rather than retrieved experience.

For all questions, annotators verify that the relevant answer evidence is visible in screenshots. Although accessibility trees sometimes expose information more clearly than screenshots, especially for hidden labels or structured fields, we avoid questions whose answer can only be inferred from the accessibility tree and not from the visual trajectory evidence.

A.3. Answer Trajectory Labeling

During question annotation, annotators record a seed set of trajectories that contain the answer for each question. However, the final haystacks are shared across questions, so we perform an additional answer-trajectory labeling pass to identify all trajectories that contain sufficient evidence for each question. The output of this step is a coverage map from each question ID to the set of answer-bearing trajectory IDs. For questions that require multiple independent pieces of evidence, such as cross-form comparisons or gotchas involving both symptom and resolution, the coverage map also records required evidence hops. The haystack builder later enforces that at least one trajectory is selected for each required hop.

We use Codex to assist this labeling process. Each worker is assigned a small batch of questions together with the relevant trajectory pool, metadata, and workflow rules. We found batching important for accuracy: it lets the agent focus on a coherent question family, such as static form comparison, dynamic transition, workflow, or gotcha questions, while keeping the search space small enough for careful inspection. The workflow emphasizes both recall and precision. The agent first narrows candidate trajectories using metadata, task family, URL patterns, and trajectory summaries, and then inspects candidate trajectories directly. Inclusion requires screenshot-visible evidence; accessibility trees and string matching can be used for triage, but they are not sufficient for inclusion. Failure trajectories are treated equally with successful trajectories, since many questions are answerable only from failed or partially completed runs. When the evidence is ambiguous, the agent records the trajectory as uncertain with a rationale rather than forcing an inclusion decision.

The workflow also enforces question-type-specific rules. Static questions require direct visual evidence of the relevant page, field, value, or form. Dynamic questions require evidence of the relevant before/after state change rather than only a related task family. Procedure questions require the trajectory to show the key procedural steps, not merely a similar entity or goal. Gotcha questions are marked as high-risk by default because the required evidence is often subtle and context-dependent. For multi-hop questions, each hop must have its own candidate trajectory set, and a single trajectory may satisfy multiple hops only when the cited screenshots directly support each hop.

After the Codex-assisted pass, humans manually resolve the ambiguous cased marked by the agent and manually validate the question-trajectory correspondence for the trajectories included in the final core haystack set. Validators check that each selected trajectory contains the intended evidence, that the evidence is visible in screenshots, and that multi-hop constraints are satisfied. This final human validation is used to ensure that the minimal answer core used in haystack construction preserves answer coverage while avoiding spurious answer-bearing trajectories.

A.4. Haystack Creation

We construct haystacks in three stages: answer-core selection, small-haystack expansion, and medium-haystack expansion. The input to the builder is the final coverage map from the previous step, which lists all answer-bearing trajectories for each question. Some questions require multiple evidence hops, so the coverage map can specify multiple required hops, each with its own candidate trajectory set. Abstention questions with no direct answer-bearing trajectory are handled separately through a manually verified anchor mapping to the corresponding source question.

Minimal answer core. We first build a minimal answer core for each domain. The goal is to select a small shared set of trajectories such that every nonzero-support question has at least one selected answer-bearing trajectory for each required hop. We formulate this as an assignment problem: for each question-hop requirement, the program chooses one trajectory from its candidate answer-bearing set. The objective favors trajectories that cover more questions and more requirements, while also encouraging sparse coverage, avoiding unnecessary overcoverage, controlling the success/failure ratio, and minimizing the number of unique selected trajectories. We solve the assignment with deterministic greedy initialization

and local search over multiple restarts. The resulting core contains 44 trajectories for WebArena and 49 trajectories for ServiceNow. We then manually verify the minimal cores to ensure that the selected trajectories contain the intended evidence for the questions and that multi-hop requirements are represented.

Small haystack. Starting from the verified minimal core, we expand each domain to a shared 100-trajectory haystack. The WebArena questions share one 100-trajectory haystack, and the ServiceNow questions share another. Fillers are selected from trajectories outside the minimal core. The filler ranking encourages diversity over task families, prefers trajectories with low global answer-support count, applies a weak preference for harder trajectories, and controls the success/failure balance toward a 1:1 ratio. After construction, the trajectory order is deterministically shuffled.

Medium haystack. The medium tier is built per question rather than as a single shared haystack. Importantly, it reuses the same per-question answer seed selected by the small-haystack builder, so the answer-bearing core does not change between tiers. For each question, the builder starts from the selected answer trajectories for its required hops and then adds fillers until the target size is reached, using 500 trajectories for the reported medium tier. Fillers are sampled after excluding the full answer-bearing set for that question, ensuring that the expansion does not introduce extra answer trajectories beyond the chosen seed. The filler ranking promotes task-family diversity, low overlap between filler goals and answer-core goals, low global answer-support count, and outcome balance. For zero-support abstention questions, the builder reuses the haystack of the manually verified anchor question exactly.

A.5. Evaluation

We evaluate each memory system with the context gathering protocol. For each question, the evaluation harness first inserts all trajectories in the question's haystack into the memory module, queries the final memory with the question, validates that the returned context consists only of text and image items, and truncates the context to 200K Qwen3.5-9B tokens. The fixed reader then receives the domain-specific system prompt, the returned memory context, the question text, and the question image if present. The reader output is parsed by extracting the last `\boxed{}` expression; if no boxed answer is found, the full response is used as the prediction. A prediction of `UNKNOWN` is always counted as incorrect for accuracy.

Structured answers are scored with deterministic evaluators specified by each question, including normalized phrase matching, ordered phrase matching, single-choice matching, and multi-select choice matching. Gotchas and abstention questions are evaluated with an LLM judge because they require semantic comparison to the reference insight or flawed-premise explanation. The judge outputs a binary JSON label, which is used as the final score. We aggregate accuracy over the full set and also report category-level breakdowns.

Experimental Details For the fixed reader model, we use the Qwen3.5-9B model hosted with vllm (Kwon et al., 2023) on a local machine with Nvidia A100 GPUs. We use temperature 0.6, top_p 0.95, and top_k 20 for a sampling evaluation. We perform manual prompt tuning to make sure that the prompt template is reliable such that (1) with no memory context the model's performance is near zero and (2) with the oracle image+text state slices, the model's performance is as high as possible. For evaluating the correctness for abstention and gotchas questions we use the GPT-5.2 model with medium reasoning effort as the LLM judge.

B. LongMemEval-V2: Pilot Studies

We conduct two pilot studies to understand whether the benchmark questions can be answered without trajectory evidence and whether oracle access to answer-bearing trajectories is sufficient for reliable question answering. These studies use a direct question answering setup rather than the context gathering formulation used in the main experiments. We report results on non-abstention questions only, since abstention questions are intentionally constructed with misleading premises and do not primarily test ordinary evidence use.

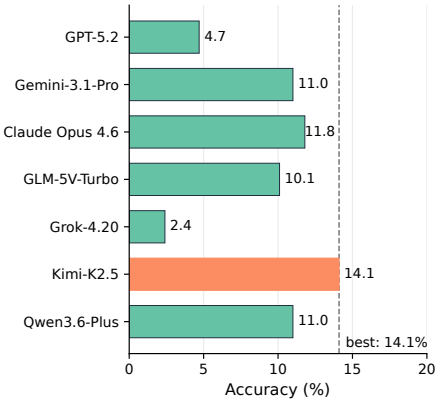
B.1. Can Frontier Models Answer Without Trajectory History?

We first evaluate whether recent frontier models can answer LME-V2 questions from parametric knowledge alone. Each model receives only the question and, when applicable, the question image. The model is instructed to answer directly and to output `\boxed{UNKNOWN}` rather than guessing. We use OpenRouter (OpenRouter, 2026) to perform the evaluation and medium reasoning effort is used across all models. As shown in Table 6, all models perform poorly without trajectory

Table 4. Reader prompt used in the main context gathering evaluation. The same template is used for all memory systems.

Component	Template
System prompt, WebArena	You are an experienced colleague in a web browsing environment that has a customized Magento-based shopping website, a customized Magento-based shopping admin CMS website, as well as a customized forum website based on Reddit/Postmill. Answer based on your memory of the environment. If you do not know the answer, output exactly `\boxed{UNKNOWN}`. Do not guess. Never attempt to guess an answer if you are not sure. If you believe the question's construction/premise is wrong, provide an explanation in `\boxed{}` explaining why the question is flawed.
System prompt, WorkArena	You are an experienced colleague working in a customized ServiceNow environment. Answer based on your memory of the environment. If you do not know the answer, output exactly `\boxed{UNKNOWN}`. Do not guess. Never attempt to guess an answer if you are not sure. If you believe the question's construction/premise is wrong, provide an explanation in `\boxed{}` explaining why the question is flawed.

User prompt	`### Memory context:`
	`{memory context returned by the memory module, after truncation}`
	`### Question to answer:`
	`{question text}`
	If the question contains an image, the image is appended after the question text.



Context	Overall	Static	Dynamic	Workflow	Gotchas
Qwen3.5-9B (thinking enabled)					
No context	0.016	0.000	0.015	0.155	0.136
Oracle trajectories	0.596	0.566	0.668	0.718	0.310
Oracle slices + notes	0.825	0.908	0.879	0.750	0.484
GPT-5.4-mini (medium reasoning)					
No context	0.045	0.025	0.010	0.075	0.171
Oracle trajectories	0.653	0.660	0.696	0.697	0.484
Oracle slices + notes	0.863	0.950	0.843	**0.905**	0.467
Codex + GPT-5.4-mini (xhigh reasoning)					
Oracle trajectory files	**0.897**	**0.986**	**0.947**	0.815	**0.517**

Figure 6. Pilot studies on LME-V2 non-abstention questions. (a) Frontier LLMs perform poorly without trajectory history, suggesting that parametric knowledge alone is insufficient for LME-V2. (b) LME-V2 is challenging to answer even with oracle answer-bearing trajectories and optimizations such as evidence slicing with notes or using a coding agent harness help improve performance.

evidence, with the best overall accuracy reaching only 14.1%. This confirms that the questions generally depend on environment-specific experience rather than public or parametric knowledge. In other words, although these models have substantial knowledge of the public versions of the environments (Magento, Postmill, ServiceNow, etc.), they still lack the environment-specific knowledge to enable them as experienced colleagues in the WebArena and WorkArena websites.

B.2. Can Models Reliably Answer with Oracle Trajectory Access?

We next evaluate whether models can answer when given oracle access to the trajectories that contain the answer. This setting removes the retrieval problem and isolates the difficulty of reading, grounding, and reasoning over trajectory evidence. We compare three variants. First, *oracle trajectories* provides the full answer-bearing trajectories. Second, *oracle slices + notes* provides procedure and hint notes together with radius-1 evidence windows around the annotated answer states. The notes are generated using a reflection prompt conditioned on a compressed view of the trajectory's content. Third, we evaluate a coding agent direct QA setup, where Codex explores a local sandbox containing the oracle trajectories and writes the answer to a JSON file. The Codex runs use Codex binary v0.117.0 with `gpt-5.4-mini (xhigh)`.

Table 7 shows the prompt template used for direct QA with rendered oracle trajectory context. The data-dependent trajectory content is abbreviated. For the coding agent experiments, we package each question as an isolated local sandbox. The model is not asked to return a memory context. Instead, it directly answers the question by inspecting local files and writing the

Table 5. LLM-judge prompts for semantic evaluation of abstention and gotchas questions. Both judges return JSON with a binary label.

Judge	Prompt
Abstention judge	**System:** You are a strict grader for flawed-premise (abstention) questions. Judge whether a model answer correctly identifies that the question premise is wrong, consistent with the reference answer. If the model follows the flawed premise and gives a concrete answer under that premise, it must be graded 0. If the model's final answer is just UNKNOWN / cannot determine without identifying the flaw, grade 0. If the model is contradictory (both rejects premise and also gives a concrete premise-following answer), grade 0. Paraphrases are allowed when they preserve the same core flaw described by the reference answer. **User:** Evaluate whether the model correctly handled a flawed-premise question. **Question:** {question_text} **Reference answer:** {reference_answer} **Model full response:** {model_full_response} **Model extracted final answer:** {model_final_answer} **Scoring rubric:** Label 1 only if the model clearly identifies the flawed premise and reaches a conclusion equivalent to the reference answer. Label 1 also if the model explicitly says it lacks access to the user's specific live environment/instance/configuration and therefore cannot verify, provided it does not give a concrete premise-following answer. Label 0 if the model follows the flawed premise and gives a concrete answer under that premise. Label 0 for generic UNKNOWN/insufficient-info replies that do not identify a flaw and do not make the explicit environment-access limitation clear. Label 0 if contradictory. **Output JSON only:** {"label": 0 or 1, "reason": "short rationale"}
Gotchas judge	**System:** You are a strict grader for gotchas-style insight questions. The reference answer describes the key insight(s). Grade 1 if the model response includes at least one correct insight point from the reference answer (paraphrase allowed), and does not contradict any reference point. If the model's direction is wrong, or it contains contradictions against any reference point, grade 0. If the model gives multiple points, partial coverage is enough for 1 as long as no contradictions appear. **User:** Evaluate whether the model answer captures the gotcha insight. **Question:** {question_text} **Reference answer:** {reference_answer} **Model full response:** {model_full_response} **Model extracted final answer:** {model_final_answer} **Scoring rubric:** Label 1 if the model includes at least one correct insight point from the reference answer (paraphrase acceptable), and does not contradict any reference point. Label 1 even if only part of a multi-point reference answer is covered, as long as there is no contradiction. Label 0 if direction is wrong (suggests opposite action/cause), even if some wording overlaps. Label 0 if any point in the model response contradicts any reference point. Label 0 if the response is irrelevant or generic without insight. **Output JSON only:** {"label": 0 or 1, "reason": "short rationale"}

result to `answer.json`. Table 8 summarizes the layout and instruction.

The Figure 6 right table shows two main findings. First, full oracle trajectories are not sufficient for reliable direct QA: Qwen3.5-9B reaches 59.6% and GPT-5.4-mini (medium) reaches 65.3%. Second, evidence slicing and notes substantially improve direct QA, reaching 82.5% and 86.3%, respectively. Finally, Codex direct QA further improves to 89.7%, suggesting that file-system exploration and iterative evidence inspection are effective ways to process agent trajectories. With inherent file system manipulation and tool use capabilities, general coding agents promise as effective memory controllers, motivating our AgentRunbook-C memory module design.

C. Implementation Details

C.1. Baselines

Since most existing memory systems mainly consider the conversation as the main input modality and facts as the corresponding memory value representations, LME-V2 poses a significant challenge and a direct naive adaptation would lead to suboptimal performance. As a result, we mainly consider two RAG and coding agent methods that we found strongest during our pilot study during our development of AgentRunbook, as well as studying ablation versions of our proposed system.

Slice+Note RAG Based on our pilot studies in Section B, a RAG-based memory system have strong performance if its state recall is accuracy and combines high-level insights with the low-level details. We thus inherit from this design

Table 6. No-context direct QA results on non-abstention questions. Frontier LLMs perform poorly across problem types.

Model	Overall	Static	Dynamic	Workflow	Gotchas
GPT-5.2	0.047	0.000	0.000	0.032	0.210
Gemini-3.1-Pro-Preview	0.110	0.104	0.096	0.147	0.241
Claude Opus 4.6	0.118	0.096	0.121	0.134	0.379
GLM-5V-Turbo	0.101	0.126	0.107	0.091	0.205
Grok-4.20	0.024	0.000	0.029	0.151	0.102
Kimi-K2.5	0.141	0.183	0.197	0.115	0.171
Qwen3.6-Plus	0.110	0.118	0.078	0.091	0.310

Table 7. Prompt template for oracle slices and notes direct QA. The same template is used for full oracle trajectories.

Component	Template
System prompt, web	You are an experienced colleague in a web browsing environment that has a customized Magento-based shopping website, a customized Magento-based shopping admin CMS website, as well as a customized forum website based on Reddit/Postmill. Answer based on your memory of the environment. If you do not know the answer, output exactly `\boxed{UNKNOWN}`. Do not guess. Never attempt to guess an answer if you are not sure. If you believe the question's construction/premise is wrong, provide an explanation in `\boxed{}` explaining why the question is flawed.
System prompt, ServiceNow	You are an experienced colleague working in a customized ServiceNow environment. Answer based on your memory of the environment. If you do not know the answer, output exactly `\boxed{UNKNOWN}`. Do not guess. Never attempt to guess an answer if you are not sure. If you believe the question's construction/premise is wrong, provide an explanation in `\boxed{}` explaining why the question is flawed.

User prompt	`# Memory context:`
	`## Procedure and Hint Notes Learned from Previous Tasks in the Environment`
	For each oracle trajectory: procedure note title, description, and bullet content; hint note title, description, and bullet content.
	`## Oracle Trajectories and Relevant State Slices from Previous Tasks in the Environment`
	For each selected trajectory: goal, outcome, start URL, action list, and evidence windows centered at annotated answer states. Each evidence window includes states from radius 1 around the annotated state, with URL, action, accessibility-tree text, and screenshots according to the rendering configuration.
	`# Question to answer:`
	`{question}`
	If the question contains an image, the image is appended after the question text.

and build two separate knowledge pools: the slice pool and the high-level note pool. We use the same hyperparameters as AgentRunbook-R to construct both pools. For the hyperparameters, we conducted a grid search for slice radius size (value 1, 2, 3), slice modality (image only, axtree text only, image+axtree), and top-k (3, 6, 9). We find that radius 1 with image+axtree slices and k=6 works the best. At query time, the question itself is directly used as the query for both pools. We retain top-6 states and top-3 notes. Qwen3-8B-Embedding is used as the embedding for all the experiments.

Codex Agent As our pilot study found promising performance of off-the-shelf coding agents, we include Codex as the baseline following the same sandbox setting we used for the pilot study. We mainly used Codex due to its performance, popularity, and its availability as an open-source project. Codex v0.117.0 binary is used for all the experiments. We download the software from github release[5] and run from a local linux server. The server has the common software required by Codex, especially `ripgrep` and `find`. In the early stages of the project, we manually tuned the instruction for this baseline to make sure that it is clear and informative so that the evaluation results do not underrepresent the agent's ability. We use GPT-5.4-mini due to its competitive latency and cost.

[5]We use the binary https://github.com/openai/codex/releases/download/rust-v0.117.0/codex-x86_64-unknown-linux-musl.tar.gz.

Table 8. Sandbox layout and instruction for the Codex oracle direct-QA pilot study.

Component	**Content**
Sandbox layout	`question.json`: question text and optional copied question image. `INSTRUCTION.md`: task instruction. `answer.json`: initialized as {`"answer": ""`}. `trajectories/{trajectory_id}/trajectory.json`: oracle trajectory with `id`, optional `original_goal`, optional `outcome`, and state content. `trajectories/{trajectory_id}/screenshots/`: copied trajectory screenshots.
Package instruction	You are an experienced colleague working in a customized web environment. Read `question.json` and inspect every trajectory under `trajectories/`. This package comes from a public-environment-based setup that has been customized. Do not rely on prior knowledge of the public environment. Work only from the provided trajectories and copied question assets. Use only the question and the provided trajectories. Do not browse anywhere else, do not inspect other question folders, and do not use outside resources. If the question specifies an answer format, follow it exactly. For multiple-choice questions, write only the boxed letter corresponding to your answer, e.g., `\boxed{A}`, into the answer field. Write your final answer to `answer.json` using this exact schema: {`"answer": "<your final answer>"`}.
Codex invocation prompt	You are an experienced colleague working in a customized web environment. Read the local files in this directory, especially `INSTRUCTION.md` and `question.json`. If `question.json` refers to a screenshot, view it carefully. Use only local files in this directory. Solve the task and write your final answer to `answer.json` as valid JSON with a non-empty string field named `answer`. If `answer.json` already exists, update only the `answer` value. Follow the formatting instructions in `question.json`. For multiple-choice questions, write only the boxed letter corresponding to your answer, e.g., `\boxed{A}`, into the answer field.

C.2. AgentRunbook-R

AgentRunbook-R maintains three retrieval pools. For each inserted trajectory, we first materialize a simplified trajectory containing state indices, URLs, actions, accessibility-tree text, screenshots, and task metadata. The raw-state pool contains one entry per state. Each entry stores a radius-1 local state window, the full trajectory action sequence, the local action sequence, the trajectory goal, and the center-state screenshot. The event pool contains one entry per adjacent state transition. Each event is generated by the memory controller LLM from the pre-state, post-state, and annotated action trace, and contains a concise transition title and description. The note pool contains two trajectory-level entries: one procedure note and one hint note. The procedure note summarizes reusable task steps, while the hint note records durable environment-specific observations and pitfalls.

All pool entries are embedded when the trajectory is inserted. At query time, the memory controller LLM reasons about the current memory state and the query and emits a structured JSON query bundle:

$$\{\texttt{"raw_state_queries"}: [\ldots], \texttt{"event_query"}: \ldots, \texttt{"note_query"}: \ldots\}.$$

The raw-state field may contain up to five focused queries, each targeting a distinct UI surface or entity. The event query targets interaction effects, navigation changes, popups, blockers, and other state transitions. The note query targets reusable procedures, module paths, disambiguation rules, and gotchas. AgentRunbook-R retrieves the top entries from each pool with dense similarity search, merges duplicate raw-state results, and renders the retrieved notes, events, and raw state slices in this order. The rendered context includes text and screenshots when available.

Prompt templates. Table 9 summarizes the fixed prompts used by AgentRunbook-R.

Experimental Details For the main experiments in the paper, we limit the number of raw state queries to 5. We maintain top-6 events and top-3 notes. As the number of queries for the raw states is flexible, we retain top-`m` results for each query, where `m=min(2,6//n_queries)`. We use Qwen3-Embedding-8B model for all the retrieval purposes. The input is truncated to 4,096 tokens. We use Qwen3.5-9B as the memory controller. We temperature 0.6, top_p 0.95 across settings. For the index-time knowledge pool generation, we always enable thinking. For query-time query generation, we experimented with both thinking enabled and disabled and report both in Figure 5. For the main results in Table 2 the results are measured with thinking enabled. All the embedding and generation models are served with vllm on a local machine with Nvidia A100 GPUs.

C.3. AgentRunbook-C

AgentRunbook-C uses file storage at insertion time. Each inserted trajectory is stored under the memory workspace as a trajectory directory containing the trajectory JSON and copied screenshots. Query-time retrieval is executed inside an isolated sandbox. The coding agent is instructed to act as a memory retrieval module rather than a final answerer. It inspects local files, identifies compact supporting evidence, and writes a structured memory output.

Before invoking the coding agent, AgentRunbook-C renders two manifest artifacts for the current haystack: a concise trajectory summary and a fuller trajectory summary. These artifacts provide trajectory-level metadata and lightweight previews that help the agent shortlist likely trajectories before detailed inspection. The sandbox also includes a trajectory inspection helper script, which supports targeted operations such as inspecting one trajectory, one state, one state span, or text matches within a trajectory.

The coding agent writes its result to `memory_module_output.json` with the following schema:

```
"memory_markdown": string,
"trajectory_spans": [ {"trajectory_id": string, "start_state_index": int, "end_state_index": int} ].
```

The `memory_markdown` field contains brief support analysis and any relevant procedure or hint notes. The trajectory spans are zero-based inclusive state ranges, with a total span budget of 20 states. After the coding agent finishes, AgentRunbook-C validates the JSON output, filters invalid spans, renders the selected states and screenshots into the returned memory context, and passes this context to the fixed reader.

Sandbox Table 10 shows the query-time sandbox provided to the coding agent. Table 11 summarizes the fixed prompt and workflow instruction used by AgentRunbook-C. Long examples and repeated rules are omitted for compactness.

Experimental Details Codex v0.117.0 binary is used for all the experiments. The query process is implemented by a python harness that prepares the sandbox and invokes `codex exec` directly. To accelerate evaluation while ensuring a fair latency measurement, we restrict the parallel query invocation to 3 query processes to avoid overloading the disk or network.

D. Further Analyses

This appendix section provides three analyses of the main methods in Table 2. Unless stated otherwise, web and enterprise questions are pooled within each LME-V2 tier.

D.1. Error Analyses

We categorize each incorrect final answer into one mutually exclusive error type. Retrieval errors are defined against the final human labeled answer-trajectory coverage maps. For each question, we take the answer-bearing trajectory set and derive each trajectory's task family, which contains the tasks with the same goal but slightly different data and starting points. A major retrieval miss means that the returned raw slices or state spans do not hit any answer-bearing task family. A minor retrieval miss means that the returned evidence hits the answer task family, but not an exact answer-bearing trajectory. If the returned context hits an exact answer-bearing trajectory, answer URL, or gold-answer text, but the reader still answers incorrectly, we label the failure as a reading error. We remark that under this categorization, the memory module often still has the responsibility to the reading errors as it has the freedom to use better state slices and better evidence presentation methods. Gotcha and premise/abstention failures are kept separate for better visualization.

As shown in Figure 7, AgentRunbook-R significantly reduces retrieval errors and downstream reading errors compared to the RAG+notes baseline. It does not improve abstention as it directly presents the relevant evidence to the downstream model without an analysis and thus the model can be misled into using the evidence to the question instead of rejecting it. Both Codex and AgentRunbook-C make fewer retrieval errors and downstream reader errors compared to the RAG methods. AgentRunbook-C further improves the abstention performance as the memory module is also instructed to explicitly identify the inconsistencies and wrong question premises and present them to the downstream model.

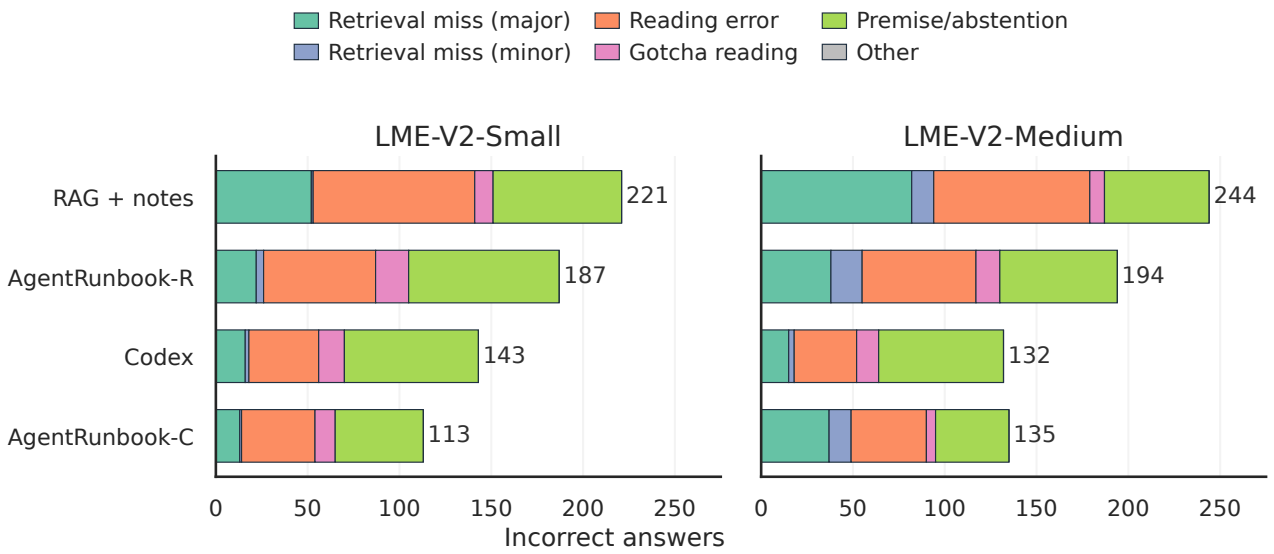


Figure 7. Error composition for the main methods on LME-V2-Small and LME-V2-Medium. Each horizontal bar decomposes the incorrect answers for one method into retrieval misses, reading failures, gotcha failures, and premise-awareness failures.

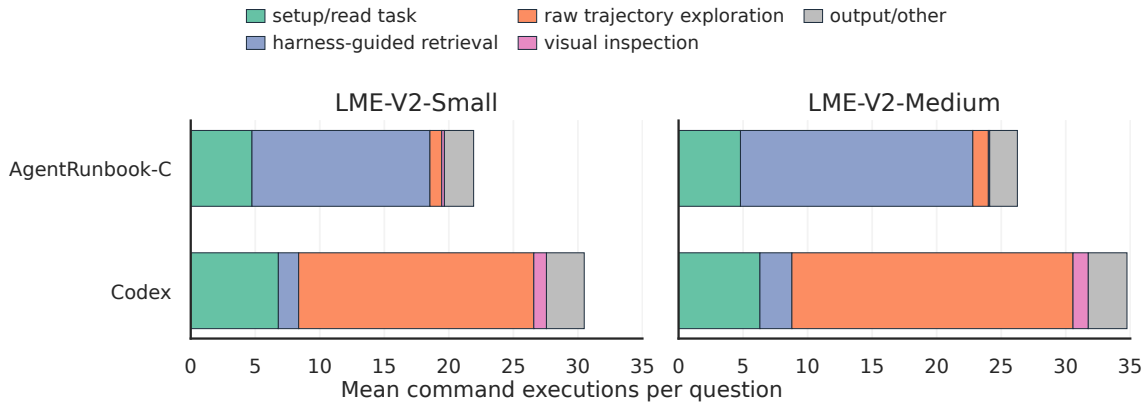


Figure 8. Mean number of command executions per query divided by command types.

D.2. Tool Calling Behavior

We compare AgentRunbook-C with the Codex baseline by parsing the coding agent event streams and grouping shell commands into five behavior classes. The *setup/read task* group includes orientation and prompt/question reads. *Harness-guided retrieval* includes manifest/summary reads and AgentRunbook helper calls. *Raw trajectory exploration* includes broad filesystem searches, direct trajectory reads, and ad-hoc Python scans. The remaining groups capture visual inspection and output validation or other commands.

As shown in Figure 8, AgentRunbook-C reduces the total number of commands and shifts work from raw trajectory exploration toward harness-guided retrieval. On LME-V2-Medium, Codex uses 21.8 raw trajectory-exploration commands per question on average, while AgentRunbook-C uses 18.0 harness-guided retrieval commands and only 1.2 raw trajectory exploration commands. In Figure 9, we further visualize the command distribution in the first rounds. Both methods begin with setup and task-reading commands. AgentRunbook-C then moves quickly into harness-guided retrieval, while Codex increasingly falls back to raw trajectory exploration in later early rounds.

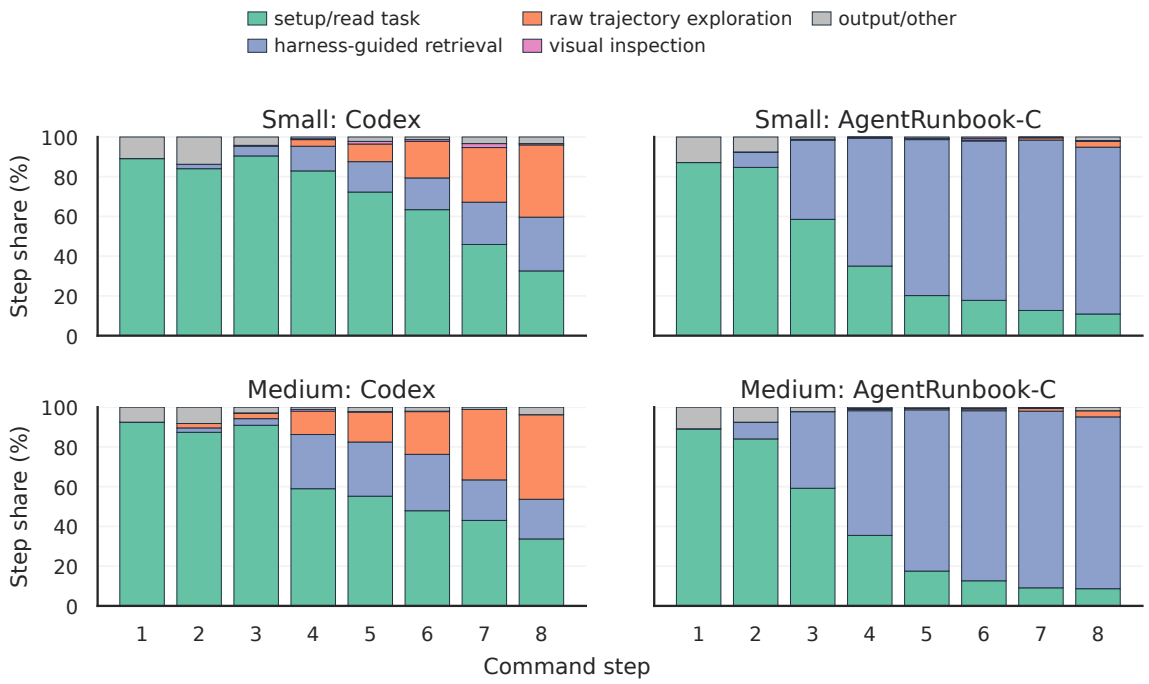


Figure 9. Distribution of grouped command classes during the first eight command rounds.

D.3. Qualitative Examples

Finally, we present qualitative examples for the two AgentRunbook variants from some successful queries in Table 12 and Table 13. For AgentRunbook-R, we select one example where each memory pool provides the answer-bearing evidence: procedure/hint notes, state-transition events, and raw state slices. For AgentRunbook-C, we show two selected evidence spans from different question types to illustrate the form of evidence passed to the reader.

E. Limitations and Ethics Statements

E.1. Limitations

Benchmark scope and formulation. LME-V2 focuses on web agents operating in customized browser environments. Browser use is a broad and practically important domain, but it does not cover the full space of digital agents, such as coding agents, computer-use agents, or domain-specific enterprise agents, which may involve different memory requirements and risk profiles. In addition, for reproducibility and controlled comparison, LME-V2 evaluates memory over pre-collected trajectory histories rather than online learning during live task execution. This design makes the benchmark easier to distribute and reproduce, but it may not fully capture distribution shifts caused by an agent's own evolving behavior. Finally, our context gathering formulation evaluates whether a memory module can return useful evidence for a fixed reader model, rather than directly measuring end-to-end task success. This is an intentional design choice to isolate memory quality, but downstream agent correctness may also depend on planning, tool use, and action execution.

Methodological scope. AgentRunbook explores practical memory designs built around retrieval, file organization, and agent-native evidence inspection, rather than proposing new model architectures or training procedures. While this keeps the methods simple and easy to analyze, their performance depends on the quality of trajectory representations, retrieval models, prompts, and coding agent behavior. AgentRunbook-C significantly improves over an off-the-shelf Codex harness in our experiments, but we do not build a fully customized coding agent harness from scratch. Future work could study tighter integration between memory, planning, and execution, as well as learned memory controllers that adapt their storage and retrieval strategies across environments.

E.2. Ethics Statements

Human annotators. The benchmark was constructed by four student authors, consisting of one graduate student and three undergraduate students. The graduate student was compensated through wage support, and the undergraduate students received research credits; all annotators are also credited through authorship. The annotation process included regular weekly annotation sessions, question-review discussions, and trajectory-label verification meetings with the project lead. Annotators were briefed on the research goals and downstream use of their annotations, and they provided consent for their annotation work to be used in the benchmark. Under the institution's research policy, this work was determined to be exempt from IRB approval.

Dataset privacy, bias, and safeguards for reuse. LME-V2 is built from sandboxed web-agent environments derived from WebArena, WorkArena, and WorkArena++, rather than from real user browsing histories. The environments use synthetic tasks, names, records, and personal information, so the risk of exposing real private user data is minimal. We manually inspected the curated LME-V2 questions and did not identify additional information leakage beyond the intended synthetic environment content. We also rely on the upstream benchmark creators' sanitization of their released environments and assets. Since the benchmark primarily covers synthetic professional workflows and controlled web environments, it is not intended to represent real demographic populations and is unlikely to directly propagate or amplify demographic bias. Nevertheless, released artifacts will be documented with intended-use guidance, and users should not treat LME-V2 as evidence that memory systems are safe for deployment on real user histories or sensitive enterprise data without additional privacy review, access control, and data minimization safeguards.

Dataset and artifact licensing. We respect the intended use and license terms of the upstream artifacts used in this work. WebArena, AgentLab, and the Codex GitHub repository are released under the Apache-2.0 license, and our use of these assets follows their respective licensing requirements. For Codex-based experiments, we use OpenAI API access rather than commercial ChatGPT subscriptions to avoid violating consumer product usage terms. We plan to release our code and derived benchmark artifacts under the Apache-2.0 license as well. We do not redistribute the original benchmark datasets or the agent harnesses used for trajectory collection. Instead, we release the derived trajectory traces, including accessibility trees and screenshots, as necessary benchmark artifacts for reproducing and evaluating LME-V2.

Broader societal impacts. This work aims to improve the efficiency and reliability of long-running agents by evaluating and designing memory systems that reuse prior experience instead of repeatedly rediscovering the same environment knowledge. More effective memory may reduce redundant agent computation, which can lower cost and environmental impact, and may also support self-improving systems that develop useful expertise in specialized domains. At the same time, persistent memory introduces risks. Agents may drift in behavior if they rely on stale, incorrect, or context-dependent observations, and coding agent-based memory controllers require careful sandboxing because their file system and tool use abilities can introduce additional security risks. More capable self-learning agents may also accelerate automation in economically valuable workflows, which could contribute to labor displacement. These risks reinforce the need to evaluate memory systems under controlled settings, document intended uses, and apply deployment safeguards such as sandboxing, access control, memory expiration, and human oversight.

E.3. LLM and Agent Use

No LLMs or agents were used for research ideation. The benchmark questions in LME-V2 were written by human annotators. As described in the main paper and appendix, we used Codex to generate initial proposals for the large-scale answer trajectory labeling which were then verified by humans. We also used Codex to assist with experiment implementation and all generated code was reviewed and validated by the authors. Codex was used to generate the figures and tables in the paper from human-provided data, except for Figure 1 and Figure 4. ChatGPT is used to generate the illustration in Figure 4, which we rigorously validate to ensure its consistency with the actual method implementation. ChatGPT, specifically GPT-5.5, is used to polish the paper from a purely human-written draft. The authors take full responsibility for the content, claims, experiments, code, figures, and tables in this submission.

Table 9. Prompt templates used by AgentRunbook-R. Trajectory-specific content is abbreviated.

Prompt	Template
Procedure and hint note generation	**System:** You convert one UI task trajectory into two reusable memory notes for a future agent. Assume these notes will later be retrieved for unknown future questions. Preserve the workflow and the highest-value reusable facts from the touched pages. Write `procedure_note` and `hint_note`. Each note must contain `title`, `description`, and `content`. Use only evidence grounded in the provided goal, outcome, thoughts, annotated actions, and screenshots. Do not invent unseen fields, filters, modules, or outcomes. If the run failed, describe only the intended or attempted workflow where the evidence supports it. Keep the procedure note focused on the reliable core workflow and use the hint note for durable facts, pitfalls, option sets, confirmation signals, absent functionality, and distinctions between easily confused controls. Return only valid JSON: `{"procedure_note":{"title":"...","description":"...","content":"-...."},"hint_note":{"title":"...","description":"...","content":"-...."}}`. **User:** Extract two reusable notes from this UI task run. Goal: {`goal`}. Outcome: {`outcome`}. Start URL: {`start_url`}. Each state block is followed by the screenshot for that state. The action line is the action taken from that state, annotated with recoverable object or module details. *Then include the ordered state blocks, thoughts, annotated actions, and screenshots.*
State-transition event generation	**System:** You convert one UI transition from a longer task trajectory into retrieval-ready event text. You will be given the task goal and outcome, the full annotated action trace, and one target transition defined as pre-state, annotated action, and post-state. Return exactly one JSON object: `{"overview":"...","state_transition":"..."}`. The overview briefly recaps the task goal and workflow stage. The state-transition field explicitly compares the post-state to the pre-state and describes what changed after the action, such as a new page, revealed panel, form fields, changed values, confirmation signal, blocker, popup, navigation, or lack of visible change. Ground the output only in the provided evidence and preserve exact UI labels when available. **User:** Generate an event for transition {`state_i` -> `state_j`}. Goal: {`goal`}. Outcome: {`outcome`}. Full action trace: {`actions`}. Pre-state: {`url, thoughts, action, AXTree, screenshot`}. Post-state: {`url, thoughts, action, AXTree, screenshot`}.
Query generation	**System:** You generate structured retrieval queries for an active memory system with three pools: raw state slices, state-transition events, and procedure/hint notes. Return exactly one JSON object: `{"raw_state_queries":["..."],"event_query":"...","note_query":"..."}`. Maximize retrieval of memory entries that would help answer the question later. Do not answer the question yourself. Use raw-state queries for exact UI surface evidence, such as pages, forms, records, tabs, fields, buttons, dropdowns, options, labels, values, counts, and missing controls. Use the event query only when navigation, before/after change, revealed content, confirmation, blocker, popup, or workflow stage matters. Use the note query for reusable procedures, module paths, disambiguation, absent functionality, pitfalls, and durable hints. Remove formatting instructions and final-answer wrappers. Preserve exact entity names and literal UI labels. Deduplicate raw-state queries and cap them at five. Return JSON only. **User:** Memory pool summary: {`runtime_summary`}. Output schema example: {`schema_example`}. Prompt examples: {`few_shot_examples`}. Question ID: {`question_id`}. Question type: {`question_type`}. Question text: {`question`}. Question image path: {`image_path or <none>`}. Original goals attached to this benchmark question: {`original_goals`}. Return only the JSON object.

Table 10. AgentRunbook-C query-time sandbox layout.

Path	Content
question.json	Question text, question id, metadata, and optional copied question image path.
INSTRUCTION.md	Workflow instruction for using the sandbox as a memory retrieval module.
trajectories/	Symlink to the inserted trajectory haystack.
trajectories/<trajectory_id>/trajectory.json	One full trajectory, including goal, start URL, outcome, actions, and ordered states.
trajectories/<trajectory_id>/screenshots/	Screenshots referenced by trajectory states.
trajectories/TRAJECTORY_SUMMARY_CONCISE.md	Compact trajectory-level manifest for quick triage.
trajectories/TRAJECTORY_SUMMARY_FULL.md	Fuller manifest with detailed thought and action traces for shortlist selection.
scripts/inspect_trajectory.py	Helper script for inspecting one trajectory, state, span, or text match.
memory_module_output.json	Structured output written by the coding agent.

Table 11. Prompt and workflow instruction for AgentRunbook-C. Codex invocation prompt is the prompt for invoking the codex binary software. The other rows in the table are the content in the INSTRUCTION.md.

Component	Template
Codex invocation prompt	You are acting as the query-time agent for Coding Agent Memory. Read the local files in this directory, especially INSTRUCTION.md and question.json. The local trajectories/ directory contains the current haystack for this evaluation item, and you must explore trajectories/ before returning your final result. If question.json refers to an image, view it carefully. Write your final result to memory_module_output.json as valid JSON. Use the local inspection helper under scripts/ when you need to inspect one trajectory, one state, one span, or match text within one trajectory quickly.
Task overview in INSTRUCTION.md	You are acting as a quick memory retrieval module to provide contexts from agent trajectories collected from a customized web environment for a downstream reader to answer questions specific to that environment. The question is in question.json. Aggregate information from the local trajectories/ directory. Follow the workflow and do not attempt to re-verify or rebuild maps unnecessarily, since the task has latency constraints. Be quick and do not over-explore unless necessary. Work inside the current directory and never explore outside it.
Output requirement	Write the final result to memory_module_output.json as valid JSON: {"memory_markdown":"## Support Analysis\n...\n\n## Relevant Procedure and Hint Notes\n...","trajectory_spans":[{"trajectory_id":"...", "start_state_index":0,"end_state_index":0}]}. The support analysis should briefly describe where the evidence can be found. If the evidence contradicts the premise of the question, clearly say that the premise is wrong and include the contradicting evidence. The trajectory spans must use zero-based inclusive indices and preserve order by importance.
Workflow instruction	First classify the question before opening trajectories in detail. If the question contains an image, inspect it and align it with the matching surface or state. For direct lookup questions, find an exact state showing the requested field, value, button, or page. For comparison questions, find one supporting state per side when needed. For procedure questions, stay within the same workflow family unless the question explicitly asks for a shared pattern across workflows. Start from TRAJECTORY_SUMMARY_FULL.md and shortlist only a few likely trajectories using the goal, start URL, action sequence, and final reward. Prefer the helper script for exact verification: python scripts/inspect_trajectory.py <trajectory_id>, --state <i>, --span <i:j>, or --match "<pattern>". Use the helper on shortlisted trajectories rather than performing broad raw-file search. Keep the final evidence package small, usually no more than three states per span, and use at most 20 states in total.
Final rules	Move fast and prefer targeted exploration. Put the most important evidence first. Avoid redundant trajectories when multiple trajectories support the same fact. Reject nearby but non-exact matches. Do not copy screenshots or large AXTree blocks into the JSON output. You may write scratch files in the current directory if needed.

Table 12. Successful AgentRunbook-C examples with the selected question markdown, evidence span content, and corresponding screenshot.

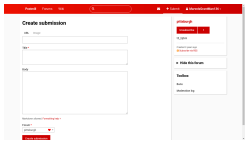
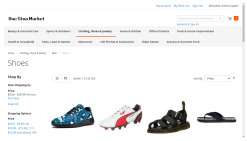
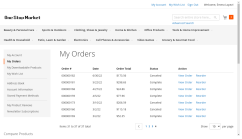
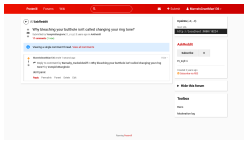
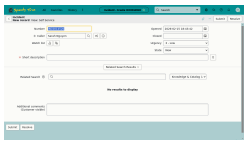
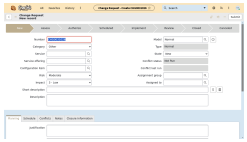
Question type	Question markdown	Selected span content	Screenshot
Static environment	**QID: `98b62f3d`. Question.** I am using our reddit-based custom forum website. For the create submission form, what are the names of the mandatory fields? Mark your final answer as a comma-separated list of short phrases in `\boxed{}`. **Gold / reader answer.** *Title, Forum*.	**Support analysis.** The clearest evidence is trajectory `4ba5e9cb`, state 2, on the *Create submission* form for `/submit/pittsburgh`. That state marks *Title* and *Forum* as required, while *Body* is optional and the URL/Image controls are submission-type selectors. **Trajectory state span.** `4ba5e9cb`: states 2–2. **State 2 AXTree excerpt.** `[136]` LabelText: *Title* ⋆; `[138]` textbox *Title This field is required.*, required; `[141]` LabelText: *Body*; `[143]` textbox *Body*; `[148]` checkbox *Formatting help* +; `[277]` LabelText: *Forum* ⋆; `[384]` combobox value *pittsburgh*.	 Trajectory `4ba5e9cb`, state 2.
Dynamic environment	**QID: `609acb91`. Question.** I am using our magento-based custom shopping website. I am now browsing the item list that appears after I click a specific item category from the home page. If I then narrow the display scope by selecting a specific price range in the left column, two new links will appear after the selected range. What are the names of the links? Your final answer should be a comma-separated list of two phrases wrapped in `\boxed{}`. **Gold / reader answer.** *Remove This Item, Clear All*.	**Support analysis.** The supporting evidence is trajectory `dddd8aa2`, state 5. That state shows the Men > Shoes category page after the *Price: $0.00–$29.99* filter is applied, and immediately after the selected range the sidebar lists the two links *Remove This Item* and *Clear All*. **Trajectory state span.** `dddd8aa2`: states 5–5. The action sequence opens Men > Shoes, selects *Price*, and loads the filtered page with `price=0-30`. **State 5 AXTree excerpt.** `[1922]` strong: *Shop By*; `[1925]` heading: *Now Shopping by*; `[1927]` listitem: *Price: $0.00–$29.99*; `[1930]` link: *Remove This Item*, clickable, visible; `[1933]` link: *Clear All*, clickable, visible.	 Trajectory `dddd8aa2`, state 5.

Table 13. Successful AgentRunbook-R examples where different memory pools provide the answer-bearing evidence. The controller queries and retrieved items are copied from the corresponding runs.

Pool	Question	Controller query	Retrieved evidence from that pool	Figure
Procedure and hint notes	Magento storefront order-history question: the user is already on *My Orders*; the table says "Items 1 to 10 of 37 total"; which pagination label should be clicked first to reach the oldest orders most directly? Choices: A. 2, B. 3, C. 4, D. Next, E. Last. **Correct answer: C.**	*Magento My Orders page pagination and date sorting behavior for finding oldest purchase*	**Procedure note, rank 1** (score 0.7465, trajectory `19022110`) is titled *Find Earliest Purchase Date in My Orders*. It states that on *My Orders*, the reader should note the pagination text, compute the final page from the total count, click the final page number, and verify the final range, e.g., "Items 31 to 37 of 37 total." The paired hint note also records that oldest orders are on the last page. With 37 orders and 10 per page, the first direct click is page 4.	 Trajectory `19022110`: final order-history page.
State-transition events	Postmill forum dynamic question: after replying to a nested comment, a blue banner appears above the reply; what does the link in that banner say? **Correct answer: View all comments.**	*submit comment reply to nested thread and observe blue banner overlay appearance*	**Event result 2** (similarity 0.5923, trajectory `2e8f6477`) retrieves the transition from state 9 to state 10 after the nested reply was posted. The event is attached to the single-comment-thread view for the AskReddit post and includes the post-state screenshot where the banner text is visible. The banner reads "Viewing a single comment thread." and its link reads "View all comments."	 Trajectory `2e8f6477`: reply result banner.
Raw state slices	ServiceNow form-comparison question: between *Create Change Request* and *Incident*, what additional top-right button appears on the incident form but not the change-request form? **Correct answer: Resolve.**	*ServiceNow Incident form top right button area visible controls*; *ServiceNow Change Request form top right button area visible controls*	**Raw state result 1** (similarity 0.7168, trajectory `454485ca`, center state 7) retrieves the incident creation form; the top-right controls include *Submit* and *Resolve*. **Raw state result 4** (similarity 0.7087, trajectory `afa62eac`, center state 6) retrieves the change-request creation form; its top-right controls show *Submit* without *Resolve*. The contrast isolates the extra incident-only button.	 Incident: *Submit*, *Resolve*.  Change Request: *Submit* only.