MEMTRACK: Evaluating Long-Term Memory and State Tracking in Multi-Platform Dynamic Agent Environments

Darshan Deshpande* **Varun Gangal*** **Hersh Mehta** **Anand Kannappan**
Rebecca Qian **Peng Wang**
Patronus AI
{darshan, varun.gangal, hersh, anand, rebecca, peng}@patronus.ai

Abstract

Recent works on context and memory benchmarking have primarily focused on conversational instances but the need for evaluating memory in dynamic enterprise environments is crucial for its effective application. We introduce MEMTRACK, a benchmark designed to evaluate long-term memory and state tracking in multi-platform agent environments. MEMTRACK models realistic organizational workflows by integrating asynchronous events across multiple communication and productivity platforms such as Slack, Linear and Git. Each benchmark instance provides a chronologically platform-interleaved timeline, with noisy, conflicting, cross-referring information as well as potential codebase/file-system comprehension and exploration. Consequently, our benchmark tests memory capabilities such as acquistion, selection and conflict resolution. We curate the MEMTRACK dataset through both manual expert driven design and scalable agent based synthesis, generating ecologically valid scenarios grounded in real world software development processes. We introduce pertinent metrics for Correctness, Efficiency, and Redundancy that capture the effectiveness of memory mechanisms beyond simple QA performance. Experiments across SoTA LLMs and memory backends reveal challenges in utilizing memory across long horizons, handling cross-platform dependencies, and resolving contradictions. Notably, the best performing GPT-5 model only achieves a 60% Correctness score on MEMTRACK. This work provides an extensible framework for advancing evaluation research for memory-augmented agents, beyond existing focus on conversational setups, and sets the stage for multi-agent, multi-platform memory benchmarking in complex organizational settings. MEMTRACK instances are available at [1]

1 Introduction

Recent advances in adaptivity of AI agents have surfaced the need for dynamic memory components that can acquire, understand and utilize relevant information from previous interactions [1]. More specifically, memory has been used to enhance personalization [2] and performance of LLM agents in robotics [3], financial trading [4], healthcare [5] and science research [6]. While these advances bring real-life impact and are scaling to multi-agent systems faster [7, 8], benchmarking memory in agentic systems has been largely limited to conversational setups. Benchmarks like `LoCoMo` [9] and `LongMemEval` [10] are focused on single thread conversational benchmarking but with the adoption

*These authors contributed equally to this work

[1] MEMTRACK Dataset: `https://drive.google.com/file/d/1ymMXmOIhCUcwC1WKOW8kioZgeYyrt-qe/view?usp=sharing`

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Scaling Environments for Agents (SEA).

of automatic context acquisition techniques [11], it becomes important to evaluate memory for real-world agentic tasks that require live context switches.

In ecologically valid [12] tasks, memory can be used for reasoning, managing and evolving enterprise components like codebases and documentation [13]. In this paper, we structure our evaluation setup as a unique sandboxed environment with realistic situations covering key concepts of contradictory memory management, and multi-hop and cross-knowledgebase content retrieval and reasoning. Unlike previous works that depend on specific implementation of memory and inclusion of external components like archival memory [14], MEMTRACK is agnostic to the backend mechanism for storing and maintaining memory. This makes MEMTRACK suitable for long context models as well as agentic systems with memory components attached. Furthermore, benchmarks like MemoryAgentBench [15] are made on top of existing datasets like ∞Bench [16], making them non-cohesive and susceptible to biases like the MCQ output format bias [17].

We build a robust set of asynchronous tools including Linear [2], Slack [3]and live notification servers, along with a dockerized file system and Git (using Gitea[4]) to faithfully replicate environments that modern agentic systems live in [5]. MEMTRACK contains 47 carefully curated and novel long context datapoints simulating realistic enterprise SWE workflows. This paper introduces three complementary data curation methodologies: 1) a top-down manual curation protocol, 2) a scalable, bottom-up, agent-based approach that generates novel memory data points by working backward from closed pull requests in widely used open-source repositories, and 3) a hybrid approach that combines human supervision with automated iterative refinement. To minimize non-deterministic behavior and overfitting on multiple choice QA for end-to-end evaluation, MEMTRACK supports brief phrase outputs that can be conveniently evaluated via direct and approximate matching using an LLM-as-judge.

The research questions we try to answer in this paper are as follows:

1. Can modern LLMs reason over large event histories and codebases to answer SWE questions?
2. Can agents use memory components such as Zep [18] and Mem0 [19] to more effectively reason over and about large event platform timelines?
3. Can LLM agents optimally access and remember information from across platforms?

Through our findings, we show that state-of-the-art LLMs fail to perform effective multi-platform context reasoning. Furthermore, we show that LLMs cannot use memory tools effectively and using such tools increases redundancy in planning and overall tool use. In our qualitative studies, we find that LLMs often struggle with context retention and need to repeatedly read tickets, conversations and code files throughout the conversation. Thus, MEMTRACK proposes a scalable design for automated data curation for memory benchmarking and surfaces key error patterns in modern LLMs with respect to memory usage.

2 Relevant Work

Memory for agents is an essential component, especially for applications such as LLM personalization [20, 21], social simulation [22, 23], gaming [24, 25, 26, 27]. Evaluating agentic systems has primarily focused on four key aspects: Tool usage, Planning, Reflection, and Memory [28]. These aspects make agents unique and distinguish them from standard conversational RAG systems [29].

Agentic Evaluation Evaluations for agentic tools revolve around task planning, tool selection, tool invocation, and response generation but existing benchmarks test for capabilities in a fragmented manner [30]. For example, state of the art benchmarks like WTU-Eval [31], ToolBench [32], AppWorld [33] and T-Eval [34] are restricted in focus to only a subset of these categories. In the sector of evaluating planning capabilities, datasets such as AgentBench [35], τBench [36] and WebArena [37] have gained popularity but lack ecological validity or are simplistic for real world

[2] `https://linear.app` and competitor Jira are well-known ticketing and project tracking platform offerings

[3] `https://slack.com`

[4] `https://about.gitea.com/`

[5] `https://cursor.com/dashboard?tab=integrations`, `https://docs.anthropic.com/en/docs/claude-code/mcp`

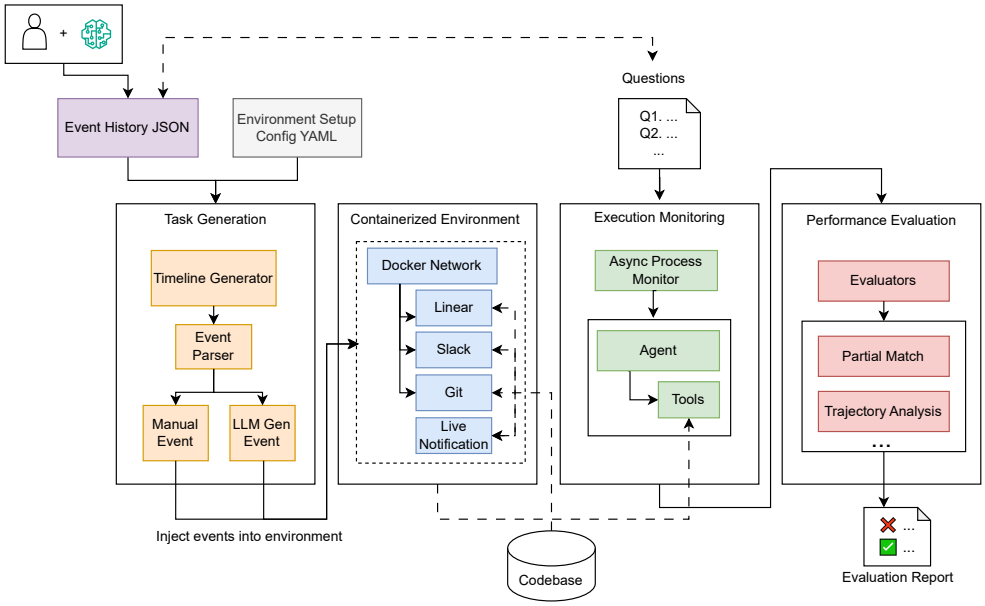

Figure 1: MEMTRACK's Data Generation and Evaluation procedure is divided into four parts: task generation, injection of events into containerized environment, agent execution monitoring and sequential question injection, and performance evaluation

use-case testing [38]. Planning trajectories for agents and their evaluation have been studied for specific tasks such as web search [39] and while efforts have been made to synthetically generate agentic trajectories [40], evaluation of techniques for generating long range planning and evaluating such trajectories still remains an open problem. Since memory is often implemented as a wrapper around tools, for example, in the form of a vector database [14, 19] or graphs [18], core concepts of evaluating tools such as tool planning, selection, invocation and response generation can be directly mapped to memory evaluations. One additional component to memory frameworks is the ability of these frameworks to forget or retain information based on requirements [41] which is understudied.

Agentic Memory Shan et al. [41], Du et al. [42] show that memory evaluation can be divided into three categories: acquisition, utilization and maintenance. To test accuracy of utilization, agentic benchmarks like Hu et al. [15], Wu et al. [10], Maharana et al. [9] use long-context, needle in the haystack-like tests for retrieval benchmarking. While acquisition of memory is relatively less standardized, works such as Reflexion [43] and MemInsight [44] rely on LLM self reflection capabilities to decide which memories are relevant and useful. Xu et al. [45] utilize concepts of link generation and automatic memory evolution to qualitatively show better t-SNE clusters of memory embeddings. Despite of these efforts, quantitatively evaluating which memories to retrieve has not been thoroughly studied. In parallel, existing works show different methods of memory maintenance. Hu et al. [15] specifically use existing coarse to fine classification data mappings to simulate test time learning whereas Maharana et al. [9], Li and Goyal [46] introduce conflicting information throughout the event history to introduce confusion and promote deep reasoning. Zhong et al. [47] cover concepts of long range memory management such as forgetting that are derived from human cognitive concepts.

Agentic Memory Implementations Packer et al. [14] were early adopters of using file systems for storage and retrieval of information across long contexts. In the following years, several other techniques were built that involved databases [19, 48] and graphs [49, 18] for storing long range information scalably. In parallel, Kang et al. [50] revealed that a hierarchical implementation of memory helps performance on the LoCoMo dataset. Zeng et al. [51] show that iterative retrieval consistently outperforms other methods across various scenarios and that mixed memory techniques are better than any single memory implementation type. Lee et al. [52] show that simply storing and retrieving gists as memory from long range text can improve performance. Other works [53, 54]

show how hierarchical text compression techniques can help RAG systems reason over long range texts. from Wang et al. [55], Xu et al. [45] build memory to improve long context performance for specific workflows which suggest that diverse evaluation workflows are needed for better testing of generalizable memory components. This work, MEMTRACK, is an effort to create complex and diverse agentic workflows by scaling agent search spaces with the help of a variety of tools and tasks.

Reinforcement Environments for LLM Agents The recent shift towards better alignment techniques [56] has resulted in better generalization for different tasks[57]. Following this progress, the need for unique simulation environments is growing since this becomes an automated source of useful data for improving agent performance [58]. Several studies recently have shown that well formed, high quality environments can scale performance beyond simple supervised fine tuning [59]. One such example of an evaluation environment is AndroidWorld [60] which benchmarks autonomous computer control agents. MEMTRACK is a step in this direction of creating a benchmark for memory augmented agent systems.

3 MEMTRACK Dataset

The MEMTRACK dataset is developed to answer the important research questions on long-term and cross-platform memory in agentic systems. MEMTRACK dataset includes 47 instances. Each instance includes a series of events, called **timeline** denoted as T, and question-expected answer pairs $< Q_i, A_i^* >$. The system is designed such that the evaluated LLM is unaware of the number of question-answer pairs within one instance and these questions are introduced strictly sequentially to remove the possibility of preemptive solution planning. An example of a MEMTRACK instance can be found in Table 2.

In MEMTRACK, a timeline represents a series of events in the environment that are accessible by tools or notifications. The event timeline is noted as $T = (E_1, ...E_n)$. Each event E_i includes a set of attributes, including meta-information such as timestamp, platform types ${e_i}^P$ where $P = \{Slack, Linear, Git\}$, and content such as Slack messages. During the creation of the benchmark environments, the timeline is loaded onto the corresponding Linear, Slack and Git servers, and is ready to be accessed by agent memory components. The timeline events will be only accessible through tools set by the servers and are not available to agents as a whole. This necessitates multi-platform context switching, information retention and effective reasoning. At instantiation (as shown in Figure 1), an event parser separates manually curated events from prompts that need to be passed to LLMs to generate an event. If LLM generation is required, we simulate the event data using CLAUDE-4-SONNET and then continue to inject both event types into the containerized environment. The dataset is compiled through a consolidation of three event history creation approaches.

Bottom-Up Approach
Ensuring the cohesive utilization of all communication channels and tools in the system, we design a bottom-up approach to creation of data points and unique realistic scenarios. Similar to SWE Bench [62], we first select a list of popular, open source repositories from GitHub. We then design an explorative agent (details available in subsubsection A.1.1) with access to web search tools and bash commands to explore closed issues on each one of these repositories. The agent is tasked to select a unique set of issues that have successfully merged PRs linked to them and preferably discussions leading to the resolution of the issue. The agent is further instructed to focus on smaller scale Pull Requests (PRs)[6] of up to two changed files that can be deterministically verified. Once the list is curated, two human annotators reviewed the list of extracted issues and resolutions, and agreed on the removal of the issues that do not have clear deterministic answers. After this filtering stage, we create another agent that takes the issues, resolutions and discussions as context and simulates natural slack conversations and Linear tickets pertaining to the issue. Once generated, the agent is instructed to augment the given data with contextualized counterfactual data and irrelevant distractors. We ensure that the distractors remain as grounded as possible to real-world workplace conversational exchanges. The generated timeline is then evaluated manually. Finally, the task is generated from the created issues and the actual solution implemented in the merged PR.

Top-Down Approach

[6]Small scale refers to PRs with less than 3 files changed

Table 1: Dataset Metrics: Definitions and Statistics

Metric	Description	Mean	Max
Per-Instance Event Count	Defined as $\|E_i\|$, the count of events in an instance's timeline T. Longer event timelines indicate more turns, longer context, and more challenging to acquire memory and memorize.	39.9	115
Event Tokens	Measures the total number of tokens in the event timeline E_i. Larger tokens are a factor indicative of harder tasks.	4.01K	11.1K
Per-Instance Platform Entropy	For each timeline E_i, we calculate its events' distribution on each platform $\hat{P}(e^{Platform} = p \| e \sim E_i)$, and then compute the entropy $H = -\sum_{p \in P} \hat{P}(p\|e) \log \hat{P}(p\|e)$. Higher entropy is desirable because the events are spread more evenly across multiple platforms.	0.668	0.989
Cross-Platform References	An LLM-judge (rubric in §A.3.1) assigns a soft multi-label 0-1 score to indicate the likelihood of an event e_i referring other platforms $p \in P, p \neq e_i^{Platform}$. An event is called cross-platform-referring, if the score exceeds a threshold (0.3). Higher number of cross-platform-referring events indicates more communication across platforms between the members within an organization, while a lower number indicates the echo-chamber problem [61], which would defeat the goal of evaluating the cross-platform dynamics.	2.1	19
Chronological Heterophily	Another form of trivialization in cross-platform dynamics can arise through events on each platform occurring in neatly separated sub-phases within a timeline without ever interleaving. To check if this anti-pattern is prevalent, we measure Chronological Heterophily, specifically the probability of an event being followed by an event of another platform. We get the maximum likelihood estimation (MLE) at the per-example level by computing the fraction of times this happens for non-terminal events. We see that we have a healthy number for this at 0.364.	0.364	0.714
Timeline Span in Hours	Besides the number of events, another distinct way of gauging the length of a timeline is the timespan it covers in its own frame of reference, which is the difference between the first and last event times $e_{i,\|E_i\|}.\tau - e_{i,1}.\tau$ measured in hours. From 1, we can see that our timelines span 878 hours on average, stretching across 3000 hours in some cases.	878	3049
Per-Instance Question Count	Defined as $\|Q_i\|$, the total number of questions asked in an instance, which includes an initial question and optional follow-up questions.	3.2	5

We ask a team of 4 in-house experts, who have worked in product and engineering organizations and have experience developing products that require collaborations among multiple teams, to provide examples in their own language, based on their own experience. Those examples represent the scenarios about how the real-world problems are defined, communicated, and eventually solved in an iterative way to address encountered challenges. Then we use their descriptions in the prompt to generate the timeline events using LLMs. Then the experts manually inspect the generated timeline and update it to make the generated timeline align with the task descriptions. The question and answer pairs are manually generated by reviewing generated timelines and ensure that the tasks are related to the task descriptions.

Hybrid Approach

The third approach is a mixture of the top-down and bottom-up approaches. The dataset annotators adapted an interactive approach to leverage high-level ideation from experts and the generation power of LLMs. In this process, the annotators start the LLM dialogue with task context, motivation, end

Table 2: An example of instance, including timeline events and question-answer pairs. Note that the Git events are set up in the filesystem in the Git server and are not shown here.

Platform	Events
Linear	"timestamp": "20250410T0900", "title": "MLE-Bench Integration for Competition Analysis Pipeline", "description": "Integrate MLE-Bench framework into our ML competition analysis pipeline to automate competitor analysis and leaderboard tracking. This will enable automated data preparation and performance benchmarking for internal ML competitions.", "team": "ml", "priority": "high", "lead": "sarah_chen", "attached_resources": ["/design/mle-bench-integration-spec.pdf", "/research/competition-analysis-requirements.md"]
...	...
Slack	"timestamp": "20250410T1430", "channel": "#ml", "sender": "sarah_chen", "message": "The error is happening during dataset download. Looks like it's in one of the core files - either mlebench/data.py, mlebench/utils.py, or maybe mlebench/registry.py. Import failure in some exception handling code around lines 180-200."
Linear	"timestamp": "20250410T1630", "title": "Code Analysis: MLE-Bench Import Error Source Investigation", "description": "Source code analysis of import failures in MLE-Bench data preparation module. Multiple import statements identified in core data processing files. Issues located in exception handling code sections. Detailed line-by-line analysis required for complete understanding of scope.", "team": "ml", "priority": "medium", "lead": "sarah_chen", "status": "done"
...	...
Slack	"timestamp": "20250411T1440", "channel": "#ml", "sender": "david_wong", "message": "Check the compatibility testing ticket - I documented the specific working version there. The one that passed all our preparation tests."
Slack	"timestamp": "20250411T1600", "channel": "#engineering", "sender": "sarah_chen", "message": "@alex_kim Could be related to the new Istio service mesh rollout. I'll check the telemetry data."
...	...
Questions	**Question 1**: Based on sarah_chen's vague mention of 'lines 180-200' and 'exception handling code', clone the repository and examine the source: how many total Python files in the entire mlebench/directory (including subdirectories) contain any import statement that references the kaggle package? **Question 2**: Following conversations about 'competition-specific modules' and 'experiments directory', investigate the actual repository structure: what is the exact filename (without path) of the Python file in the mlebench/competitions/subdirectory that contains the longest individual function by line count? **Question 3**: After correlating references to 'data preparation pipeline' and 'dataset download functionality', analyze the repository code: what is the total number of function definitions (def statements) in the mlebench/data.py file? ...
Answers	**Answer 1**: 9 **Answer 2**: prepare.py **Answer 3**: 4 ...

goals and the codebase. In an iterative manner, the LLM is then prompted to elaborate and adapt its plan as the annotators increase complexity of the event history. Following this, the annotators have finer control over the areas in which complexity needs to be added which allows them to tune questions and situations accordingly, thereby making the process more flexible. Once the complete event history is generated, the annotator then manually attempts to solve the task to ensure completeness of the procedure.

Dataset Statistics We provide a set of seven metrics to validate the quality of generated dataset, to ensure that the dataset is indeed diverse and includes long-term memory across multiple platforms, as designed. Detailed metric definitions and statistics are provided in Table 1.

4 Experimental Setup

4.1 Methods

LLM+NoMem: This method has the LLM driving the agent without any within-agent memory components included.

LLM+Mem0: In this method, the agent consists of a LLM with an additional within-agent Mem0-based [19] memory component that exposes an additional set of its own tools, for example, *search_memory()*, *store_memory()*, *get_memories()*, to the agent. We use GPT-4o-mini [63] LLM-based embeddings and ChromaDB for the vector store in Mem0.

LLM+Zep: This has a similar agent architecture to the LLM+Mem0 approach except that the memory component used is Zep [18]. We choose Zep as the second memory component since, having additional knowledge-graph based aspects to its internal memory component's architecture.

For our memory experiments involving Mem0 and Zep, we use the default settings from their respective APIs.

4.2 Evaluation Measures

Correctness: For each of the questions $Q_i = (q_{i,1}, q_{i,2} \dots q_{i,n})$, our benchmark provides a set of respective expected answers $A_i^* = (a_{i,1}^*, a_{i,2}^* \dots a_{i,n}^*)$. When the agent is given the tool-accessible timeline, it returns respective agent answers as it is prompted with each q_i in order, yielding through its responses the agent answers $A_i = (a_{i,1}, a_{i,2} \dots a_{i,n})$. Using a LLM-judge $Judge(q, a, a*)$ (GPT-4o), we output a Correctness score for each respective triple of question, expected answer and agent answer. The average of this score across all indices gives us the Correctness for an instance.

Efficiency: The number of tool calls made to access elements of the timeline, noted TC, is undesirable if it is excessive. To check for efficiency of tool calling, we measure it as Efficiency $= e^{-\frac{|TC|-TC_{min}}{TC_{min}}}$ for $TC >= TC_{min}$ and 1 for $TC < TC_{min}$, where $TC_{min} = 10$.

Redundancy: Consider $TC_i = \{tcname_{i,k}(tcargs_{i,k})\}$ be the set of tool calls where $tcname_{i,k}$ and $tcargs_{i,k}$ are tool name and arguments respectively. We want to capture to what extent the agent is making tool calls (in the complete sense, including args) that are equivalent to or subsumed by tool calls made already i.e. $TC_{i,j} \subseteq TC_{i,k}$ s.t $j > k$ and are hence redundant. Measuring this is particularly important given we ask several questions. In this setting, maximally reusing the information obtained already while answering earlier questions is a desirable behavior.

Since this needs both soft matching as well as noticing subsumption (e.g. *get_ticket(id="abc")* vs *list_all_tickets()*), we prompt a LLM judge with our definition and the completely described TC, asking it to enumerate if each tool call is redudant given the previous and return the aggregate. Redundancy is defined as the fraction of TC that is redundant.

Note that Efficiency and Redundancy are intrinsic metrics that measure the efficiency of the memory acquisition mechanism the model exhibits while doing the task, and higher Efficiency or lower Redundancy at the cost of a non-trivial drop in Correctness is in general not desirable.

5 Results and Discussions

A compilation of our results from our approaches with the frontier LLMs at the time of writing, namely GPT-5 [64] and Gemini-2.5-pro [65] can be found in Table 3.

We see that the GPT-5 family based methods considerably outperforms the Gemini family ones irrespective of the accompanying memory method used, reaching roughly four times the performance in terms of correctness.

5.1 Quantitative Analysis

Do Agent Responses Invoke Tool Calls Successfully? All approaches show high mean as well as median fraction of successful tool calls (tool calls which execute and return back) $\approx\in (0.90, 0.96)$, with a minimum between 0.7 and 0.8 (exact numbers in Appendix Table 5). On an

Table 3: Results on MEMTRACK across methods recording the Correctness (averaged per-example over questions), Efficiency, Redundancy as well as extent and cross-platform entropy of tool calling. All results are averaged across 5 runs and statistical significance is reported in Table 6.

Method	Correct. (↑)	Eff. (↑)	Red. (↓)	$TC_i(mean/max)$ (↑)	$(H(TC_i))$ (↑)
GPT-5+NOMEM	0.601	**0.667**	**0.206**	13.22/45.4	0.978
GPT-5+MEM0	**0.610**	0.656	0.214	13.29/45.0	**1.023**
GPT-5+ZEP	0.601	0.660	0.214	13.23/**51.4**	0.949
GEMINI-2.5-PRO+NOMEM	0.144	0.638	0.237	**13.72**/48.4	0.840
GEMINI-2.5-PRO+MEM0	0.118	0.658	0.240	12.53/46.8	0.838
GEMINI-2.5-PRO+ZEP	0.140	0.662	0.238	12.51/47.8	0.747

average, GEMINI-2.5-PRO . Though there is minor room for improvement here, both GPT-5 and GEMINI-2.5-PRO seem fairly fluent at this aspect.

Do The Memory Components Improve Performance? We see that both MEM0 and ZEP components do not cause any significant improvement in the performance of GPT-5 as well as GEMINI-2.5-PRO. Furthermore, in the case of GEMINI-2.5-PRO+MEM0, we also notice a slight degradation in performance due to the introduction of MEM0.

Does Performance Decay with Follow-Ups?: We observe a small but notable drop in mean correctness when only follow-up questions are considered. This pattern holds consistently across all methods.

Table 4: Drop in performance on follow-up questions

	NOMEM		MEM0		ZEP	
	GPT-5	GEMINI	GPT-5	GEMINI	GPT-5	GEMINI
Overall	0.601	0.144	0.588	0.118	0.604	0.140
Follow-Up	0.571	0.121	0.553	0.094	0.585	0.113

5.2 Qualitative analysis

5.2.1 Illustrative GPT-5 Run

In this example run, (of GPT-5 solving the *nemo_run_cli_crisis* example), shown in full in Appendix §B.1 , we observe how solving an instance in our environment makes a GPT-5 driven agent richly navigate through making a total of 42 platform-interleaving tool calls spread over filesystem (magenta) , Linear (olive) and Slack (blue) before finally leading it to answer the final two follow-up questions [which are relatively phrased] correctly. At the outset, it also has to clone the repository of interest in its own filesystem, to facilitate the filesystem exploration and navigation. During this run, it has to switch between tool-calling against different platforms 7 times. (we show an excerpt here)

list_tickets() → get_tickets(ticket_id = e8da2eed-bb7c-4d9f-93b2-89977fe48f2d) . . . (Getting More Tickets) → search_file_content(pattern="", case_sensitive = False, max_matches=200, context_lines=1) → get_unread_messages() → list_channels() → get_channel_messages(channel="engineering", limit=100) → get_ticket(ticket_id="171754fc-bca8-4057-8c5f-410ed9721590") → Answer("2025-05-15") ✓ → Question 3 Asked → list_directory(path="NeMo-Run/nemo_run", detailed=true) → search_file_content(pattern="32", search_path= "NeMo-Run/nemo_run") → Answer(142) ✓

5.2.2 Redundancy Patterns

Given 20% redundancy, we examine the sequence of tool calls made by models across examples as well as the reasoning outputs of our Redundancy LLM judge to ferret out some notable patterns.
General-To-Specific Info Access Redundancy: Agents often repeat up a general platform enlisting call (e.g. *list_tickets()*) by a more specific repeated call (e.g. *list_tickets(limit=100)*).

Repeated Info Access After Interlude: Given a sufficient gap of 3 or more turns, we notice a tendency to repeat information accessing tool calls E.g. a GPT-5 based agent on the *config_plexe_autocroll_crisis* repeats *get_unread_messages()* to list all slack messages as its 11th and

23rd tool calls and, in another run, *get_ticket(ticket_id=x)* to re-access the same Linear ticket with id x on its 8th and 25th tool call. This is not limited to Slack and Linear, in *config_vg_12*, the file *evals/elsuite/hr_ml_agent_bench/benchmarks/cifar10* is read two times at the 9th and 17th tool calls made by the GPT5+ZEP agent. Some instances repeat file reads with overlapping line ranges.

Progressively Widening Exploration: In some cases, the agent chooses to explore a platform very gradually with increasing limits, if the tool parameters allow this e.g. *get_channel_messages(channel='backend',limit=50)* followed by limit 100. Agents being conservative in this regard is justified, so this is not an anti-pattern unless its overdone to point of small increments.

5.3 Answering Research Questions

RQ1: Can modern LLMs reason over large event histories and codebases to answer SWE questions? As seen in Table 3, frontier LLM families such as GPT-5 exhibit suboptimal performance on the MEMTRACK($\approx 60\%$) irrespective of memory components. This suggests that there is a significant room for improvement, particularly on improving large context reasoning and understanding of follow-up questions as seen in Table Table 4. SoTA models are unable to properly utilize memory implementations and addition of memory tools leads to a clear increase in redundancy of tool calls. Finally, the increase in tool call entropy as displayed in Table 3 suggests that diverse information gathering using tools helps answer questions more reliably.

RQ2: Can agents use memory databases and backends such as ZEP and MEM0 effectively to reason over large codebases? According to Table 3, we can observe that memory equipped LLMs fail to call memory tools effectively. LLMs with memory tools consistently display increased redundancy as well as drop in performance efficiency. As observed in the qualitative analysis above, models generally prefer repeatedly accessing information over using the memory component.

RQ3: Can LLM agents optimally access and remember information from across platforms? We observe that retention of cross-platform information is suboptimal and models need to access information repeatedly. Given a $\geq 20\%$ redundancy in tool calls provides support to this suboptimality. Additionally, as observed in Table 4, LLMs display poor follow-up performances suggesting their inability to remember and utilize cross platform information effectively. Finally, the tool call entropy displayed in Table 3 suggests that diverse tool use is still a limiting factor improving on which can drive up performance for current approaches. Thus, the lack of saturation on this metric shows the gap between memory assimilation across multiple threads.

6 Conclusion & Future Work

By combining manual, bottom-up, and hybrid data curation approaches, we developed MEMTRACK, a benchmark to test agent memory in a multi-platform environments that faithfully reflect real organizational workflows. Our experiments show that while frontier LLMs like GPT-5 exhibit some capacity to reason across large event histories, performance declines on follow-up queries. Memory components such as MEM0 and ZEP provide limited gains, and tool-use to access memory remains inefficient. Our framework provides an umbrella to build future environments where the agent is not only given access to a rich set of events, but furthermore contextually act, such as creating Linear tickets, sending Slack messages, to get involved in the future unfolding of the organization timeline and aim to drive the overall organization to complete the tasks. We avoided exploring this harder class of settings in this initiating work to focus on crafting an ecologically valid base. In the future, MEMTRACKcan be extended to other domains such as marketing or sales that commonly involve a large overlap of internal and external communication contexts.

References

[1] Zeyu Zhang, Quanyu Dai, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model based agents. *ACM Transactions on Information Systems*, 2024.

[2] Hao Li, Chenghao Yang, An Zhang, Yang Deng, Xiang Wang, and Tat-Seng Chua. Hello again! llm-powered personalized agent for long-term dialogue. *arXiv preprint arXiv:2406.05925*, 2024.

[3] Marc Glocker, Peter Hönig, Matthias Hirschmanner, and Markus Vincze. Llm-empowered embodied agent for memory-augmented task planning in household robotics. *arXiv preprint arXiv:2504.21716*, 2025.

[4] Yang Li, Yangyang Yu, Haohang Li, Zhi Chen, and Khaldoun Khashanah. Tradinggpt: Multi-agent system with layered memory and distinct characters for enhanced financial trading performance. *arXiv preprint arXiv:2309.03736*, 2023.

[5] Mahyar Abbasian, Iman Azimi, Amir M Rahmani, and Ramesh Jain. Conversational health agents: A personalized llm-powered agent framework. *arXiv preprint arXiv:2310.02374*, 2023.

[6] Cilin Yan, Jingyun Wang, Lin Zhang, Ruihui Zhao, Xiaopu Wu, Kai Xiong, Qingsong Liu, Guoliang Kang, and Yangyang Kang. Efficient and accurate prompt optimization: the benefit of memory in exemplar-guided reflection. *arXiv preprint arXiv:2411.07446*, 2024.

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

[8] Jiayan Nan, Wenquan Ma, Wenlong Wu, and Yize Chen. Nemori: Self-organizing agent memory inspired by cognitive science. *arXiv preprint arXiv:2508.03341*, 2025.

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

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

[11] Lingrui Mei, Jiayu Yao, Yuyao Ge, Yiwei Wang, Baolong Bi, Yujun Cai, Jiazhi Liu, Mingyu Li, Zhong-Zhi Li, Duzhen Zhang, et al. A survey of context engineering for large language models. *arXiv preprint arXiv:2507.13334*, 2025.

[12] Samuel R. Bowman and George Dahl. What will it take to fix benchmarking in natural language understanding? In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou, editors, *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4843–4855, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.385. URL https://aclanthology.org/2021.naacl-main.385/.

[13] Thomas Joshi, Shayan Chowdhury, and Fatih Uysal. Swe-bench-cl: Continual learning for coding agents. *arXiv preprint arXiv:2507.00014*, 2025.

[14] Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as Operating Systems. *arXiv preprint arXiv:2310.08560*, 2024. URL https://arxiv.org/abs/2310.08560.

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

[16] Xinrong Zhang, Yingfa Chen, Shengding Hu, Zihang Xu, Junhao Chen, Moo Khai Hao, Xu Han, Zhen Leng Thai, Shuo Wang, Zhiyuan Liu, et al. ∞ bench: Extending long context evaluation beyond 100k tokens. *arXiv preprint arXiv:2402.13718*, 2024.

[17] Chujie Zheng, Hao Zhou, Fandong Meng, Jie Zhou, and Minlie Huang. Large language models are not robust multiple choice selectors. *arXiv preprint arXiv:2309.03882*, 2023.

[18] Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. Zep: A temporal knowledge graph architecture for agent memory, 2025. URL https://arxiv.org/abs/2501.13956.

[19] Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025.

[20] Yunfan Shao, Linyang Li, Junqi Dai, and Xipeng Qiu. Character-LLM: A trainable agent for role-playing. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13153–13187, Singapore, December 2023. Association for Computational Linguistics. URL https://aclanthology.org/2023.emnlp-main.814/.

[21] Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19724–19731, 2024.

[22] Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior, 2023. URL https://arxiv.org/abs/2304.03442.

[23] Maximilian C. Fink, Seth A. Robinson, and Bernhard Ertl. Ai-based avatars are changing the way we learn and teach: benefits and challenges. *Frontiers in Education*, 9, 2024. ISSN 2504-284X. doi: 10.3389/feduc.2024.1416307. URL https://www.frontiersin.org/journals/education/articles/10.3389/feduc.2024.1416307.

[24] Zihao Wang, Shaofei Cai, Anji Liu, Yonggang Jin, Jinbing Hou, Bowei Zhang, Haowei Lin, Zhaofeng He, Zilong Zheng, Yaodong Yang, Xiaojian Ma, and Yitao Liang. Jarvis-1: Open-world multi-task agents with memory-augmented multimodal language models, 2023. URL https://arxiv.org/abs/2311.05997.

[25] Xizhou Zhu, Yuntao Chen, Hao Tian, Chenxin Tao, Weijie Su, Chenyu Yang, Gao Huang, Bin Li, Lewei Lu, Xiaogang Wang, Yu Qiao, Zhaoxiang Zhang, and Jifeng Dai. Ghost in the minecraft: Generally capable agents for open-world environments via large language models with text-based knowledge and memory, 2023. URL https://arxiv.org/abs/2305.17144.

[26] Zaijing Li, Yuquan Xie, Rui Shao, Gongwei Chen, Dongmei Jiang, and Liqiang Nie. Optimus-1: Hybrid multimodal memory empowered agents excel in long-horizon tasks. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 49881–49913. Curran Associates, Inc., 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/5949a8750a110ce1f0631b1776c500a2-Paper-Conference.pdf.

[27] Sihao Hu, Tiansheng Huang, Gaowen Liu, Ramana Rao Kompella, Fatih Ilhan, Selim Furkan Tekin, Yichang Xu, Zachary Yahn, and Ling Liu. A survey on large language model-based game agents, 2025. URL https://arxiv.org/abs/2404.02039.

[28] Asaf Yehudai, Lilach Eden, Alan Li, Guy Uziel, Yilun Zhao, Roy Bar-Haim, Arman Cohan, and Michal Shmueli-Scheuer. Survey on evaluation of llm-based agents. *arXiv preprint arXiv:2503.16416*, 2025.

[29] Wenqi Fan, Yujuan Ding, Liangbo Ning, Shijie Wang, Hengyun Li, Dawei Yin, Tat-Seng Chua, and Qing Li. A survey on rag meeting llms: Towards retrieval-augmented large language models. In *Proceedings of the 30th ACM SIGKDD conference on knowledge discovery and data mining*, pages 6491–6501, 2024.

[30] Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. Tool learning with large language models: A survey. *Frontiers of Computer Science*, 19(8):198343, 2025.

[31] Kangyun Ning, Yisong Su, Xueqiang Lv, Yuanzhe Zhang, Jian Liu, Kang Liu, and Jinan Xu. Wtu-eval: A whether-or-not tool usage evaluation benchmark for large language models. *arXiv preprint arXiv:2407.12823*, 2024.

[32] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*, 2023.

[33] Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. *arXiv preprint arXiv:2407.18901*, 2024.

[34] Zehui Chen, Weihua Du, Wenwei Zhang, Kuikun Liu, Jiangning Liu, Miao Zheng, Jingming Zhuo, Songyang Zhang, Dahua Lin, Kai Chen, et al. T-eval: Evaluating the tool utilization capability of large language models step by step. *arXiv preprint arXiv:2312.14033*, 2023.

[35] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*, 2023.

[36] Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains, 2024. URL https://arxiv.org/abs/2406.12045.

[37] Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023. URL https://webarena.dev.

[38] Xu Huang, Weiwen Liu, Xiaolong Chen, Xingmei Wang, Hao Wang, Defu Lian, Yasheng Wang, Ruiming Tang, and Enhong Chen. Understanding the planning of llm agents: A survey, 2024. URL https://arxiv.org/abs/2402.02716.

[39] Xing Han Lù, Amirhossein Kazemnejad, Nicholas Meade, Arkil Patel, Dongchan Shin, Alejandra Zambrano, Karolina Stańczak, Peter Shaw, Christopher J. Pal, and Siva Reddy. Agentrewardbench: Evaluating automatic evaluations of web agent trajectories, 2025. URL https://arxiv.org/abs/2504.08942.

[40] Yiheng Xu, Dunjie Lu, Zhennan Shen, Junli Wang, Zekun Wang, Yuchen Mao, Caiming Xiong, and Tao Yu. Agenttrek: Agent trajectory synthesis via guiding replay with web tutorials. 2024. URL https://arxiv.org/abs/2412.09605.

[41] Lianlei Shan, Shixian Luo, Zezhou Zhu, Yu Yuan, and Yong Wu. Cognitive memory in large language models, 2025. URL https://arxiv.org/abs/2504.02441.

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

[43] Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023. URL https://arxiv.org/abs/2303.11366.

[44] Rana Salama, Jason Cai, Michelle Yuan, Anna Currey, Monica Sunkara, Yi Zhang, and Yassine Benajiba. Meminsight: Autonomous memory augmentation for llm agents, 2025. URL https://arxiv.org/abs/2503.21760.

[45] Wujiang Xu, Kai Mei, Hang Gao, Juntao Tan, Zujie Liang, and Yongfeng Zhang. A-mem: Agentic memory for llm agents, 2025. URL https://arxiv.org/abs/2502.12110.

[46] Aochong Oliver Li and Tanya Goyal. Memorization vs. reasoning: Updating llms with new knowledge, 2025. URL https://arxiv.org/abs/2504.12523.

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

[48] Chenxu Hu, Jie Fu, Chenzhuang Du, Simian Luo, Junbo Zhao, and Hang Zhao. Chatdb: Augmenting llms with databases as their symbolic memory. *arXiv preprint arXiv:2306.03901*, 2023.

[49] Shilong Li, Yancheng He, Hangyu Guo, Xingyuan Bu, Ge Bai, Jie Liu, Jiaheng Liu, Xingwei Qu, Yangguang Li, Wanli Ouyang, et al. Graphreader: Building graph-based agent to enhance long-context abilities of large language models. *arXiv preprint arXiv:2406.14550*, 2024.

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

[51] Ruihong Zeng, Jinyuan Fang, Siwei Liu, and Zaiqiao Meng. On the structural memory of llm agents. *arXiv preprint arXiv:2412.15266*, 2024.

[52] Kuang-Huei Lee, Xinyun Chen, Hiroki Furuta, John Canny, and Ian Fischer. A human-inspired reading agent with gist memory of very long contexts. *arXiv preprint arXiv:2402.09727*, 2024.

[53] Chanwoong Yoon, Taewhoo Lee, Hyeon Hwang, Minbyul Jeong, and Jaewoo Kang. Compact: Compressing retrieved documents actively for question answering. *arXiv preprint arXiv:2407.09014*, 2024.

[54] Mengkang Hu, Tianxing Chen, Qiguang Chen, Yao Mu, Wenqi Shao, and Ping Luo. Hiagent: Hierarchical working memory management for solving long-horizon agent tasks with large language model, 2024. URL https://arxiv.org/abs/2408.09559.

[55] Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory, 2024. URL https://arxiv.org/abs/2409.07429.

[56] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

[57] Zhiqing Sun, Longhui Yu, Yikang Shen, Weiyang Liu, Yiming Yang, Sean Welleck, and Chuang Gan. Easy-to-hard generalization: Scalable alignment beyond human supervision, 2024. URL https://arxiv.org/abs/2403.09472.

[58] Alireza Rashidi Laleh and Majid Nili Ahmadabadi. A survey on enhancing reinforcement learning in complex environments: Insights from human and llm feedback, 2024. URL https://arxiv.org/abs/2411.13410.

[59] Qi Zhao, Haotian Fu, Chen Sun, and George Konidaris. Epo: Hierarchical llm agents with environment preference optimization, 2024. URL https://arxiv.org/abs/2408.16090.

[60] Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Toyama, Robert Berry, Divya Tyamagundlu, Timothy Lillicrap, and Oriana Riva. Androidworld: A dynamic benchmarking environment for autonomous agents, 2025. URL https://arxiv.org/abs/2405.14573.

[61] Srijan Kumar, William L Hamilton, Jure Leskovec, and Dan Jurafsky. Community interaction and conflict on the web. In *Proceedings of the 2018 world wide web conference*, pages 933–943, 2018.

[62] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

[63] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

[64] Openai gpt-5 system card. https://openai.com/index/gpt-5-system-card/.

[65] Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

A Prompts

A.1 Dataset Generation

A.1.1 Bottom-Up Approach

```
Refer to the task details attached here and create a situationally aware dataset that
    tests LLM's memory skills. Be comprehensive, careful (verify your steps) and
    optimal while creating the dataset. Analyze the given examples, especially their
    questions and expected answers to understand the level of complexity and cross
    platform communication that is expected. You must run the script and reiterate the
     process in case the models pass all tests (i.e. the dataset is too easy for the
    model). You must always verify that the model fails on your dataset and you must
    carefully iterate while making sure that you don't lose track of the task and
    trajectory that you want to follow to get to the answer. Always be ecologically
    valid when creating questions and think from the perspective of actual user
    requirements and complexity.

{task_details}
```

The task description is as follows:

```
Configuration Setup
Before beginning, configure the following variables:

OWNER: The GitHub repository owner (e.g., "openai", "microsoft", "google")
REPO: The repository name (e.g., "mle-bench", "vscode", "tensorflow")
ORGANIZATION: The organization name for internal communications (can be same as OWNER or
     different)

Creating the Setup
Step 1: Repository Analysis
Analyze the target repository: https://github.com/{OWNER}/{REPO}.git
Pull all solved issues using the following script:
bashCopycurl -H "Accept: application/vnd.github.v3+json" \
    -H "User-Agent: curl" \
    "https://api.github.com/repos/{OWNER}/{REPO}/issues?state=closed" | jq -r '.[] | .
        number, .title, ""'
Step 2: Issue Investigation
Identify interesting issues that involve technical discussions, feature implementations,
     or bug resolutions. For promising issues, pull detailed information:
bashCopycurl -H "Accept: application/vnd.github.v3+json" \
    -H "User-Agent: curl" \
    "https://api.github.com/repos/{OWNER}/{REPO}/issues/{ISSUE_NUMBER}" | jq -r '.title,
         "", .body'
Pull the complete comment history:
bashCopycurl -H "Accept: application/vnd.github.v3+json" \
    -H "User-Agent: curl" \
    "https://api.github.com/repos/{OWNER}/{REPO}/issues/{ISSUE_NUMBER}/comments" | \
jq -r '.[] | "Author: " + .user.login + "\nDate: " + .created_at + "\n" + .body + "\n" +
     ("-" * 50) + "\n"'
Creating the Multi-Platform Scenario
Core Requirements for Task Difficulty
The scenario must be designed such that:

No single platform contains the complete answer - Information must be distributed across
     Slack, Linear, and potentially email/documents
Minimum 5 reasoning hops required - Each hop should require connecting information from
    different sources
Minimum 3 platform switches - Must require switching between Slack channels, Linear
    tickets, and other platforms
Temporal dependencies - Some information should only be meaningful when combined with
    timestamps from different platforms

Multi-Platform Information Distribution Strategy
Structure your scenario with information scattered as follows:
Platform 1 (Slack #engineering): Contains initial problem identification and high-level
    technical approach decisions
Platform 2 (Linear tickets): Contains detailed implementation specifications, status
    updates, and technical requirements
Platform 3 (Slack #random or #general): Contains casual mentions of blockers,
    dependencies, or context that affects the technical decision
Platform 4 (Slack DMs or #leadership): Contains resource allocation decisions, priority
    changes, or approval processes
Platform 5 (Additional tickets/email): Contains external dependencies, vendor
    communications, or compliance requirements
Scenario Creation Guidelines

Create realistic organizational personas with distinct roles:

Engineering leads, senior developers, product managers
DevOps, security, QA team members
External stakeholders or vendor contacts


Implement strategic information fragmentation:
```

Key decision rationale split across multiple Slack threads
Implementation details spread between Linear descriptions and comments
Critical context buried in seemingly unrelated conversations
Timeline information requiring cross-referencing multiple platforms

Add high-quality distractors:

Similar but unrelated technical discussions
Multiple tickets with similar names or purposes
Conversations about abandoned approaches or rejected solutions
Time-overlapping discussions about different features

Ensure temporal complexity:

Decisions that get revised across platforms
Information that becomes relevant only after later context
Status changes that affect interpretation of earlier discussions

Question Design Requirements
Ecological Validity Principles
Questions should reflect real workplace information needs:
Instead of: "What was the exact configuration parameter set for the Redis cache timeout ?"
Design: "I'm trying to understand why our caching layer behaves the way it does. Can you help me figure out what drove the performance decisions we made last quarter?"
Instead of: "Which team member proposed the authentication fix?"
Design: "The client is asking about our security implementation timeline. Who should I credit for the breakthrough that got us past that login issue?"
Instead of: "What was the final status of ticket LIN-403?"
Design: "I need to prep for the retrospective meeting. Can you remind me how that problematic feature request got resolved?"
Strategic Vagueness Implementation
Questions should be contextually ambiguous to require genuine reasoning:
Time-based Vagueness

"That issue from a few weeks back" (multiple issues in timeframe)
"The recent performance discussion" (several performance topics)
"After the client escalation" (need to identify which escalation)

Domain-based Vagueness

"The API changes" (multiple API modifications)
"The security fix" (several security-related tickets)
"The database optimization" (multiple DB-related work streams)

Role-based Vagueness

"What the team decided" (need to identify which team/decision)
"The solution Alex proposed" (Alex proposed multiple solutions)
"The approach we ultimately went with" (multiple approaches discussed)

Outcome-based Vagueness

"The workaround that actually worked" (multiple workarounds attempted)
"The change that fixed the production issue" (multiple changes made)
"The configuration that resolved the bottleneck" (several configs modified)

Multi-Hop Reasoning Structure
Your question must require connecting information such as:
Hop 1: Vague question reference -> Identify specific context period/domain

```
Hop 2: Context clues -> Locate initial problem discussion (Slack #engineering)
Hop 3: Problem details -> Find corresponding implementation ticket (Linear)
Hop 4: Implementation ticket -> Discover blocking dependency (Slack #random)
Hop 5: Dependency context -> Locate resolution approach (Different Linear ticket)
Hop 6: Resolution approach -> Confirm final outcome (Slack #leadership)
Example Ecologically Valid + Vague Questions
Scenario: Performance optimization issue resolution
Poor Question: "What caching strategy did ticket LIN-427 implement on March 15th?"
Good Question: "Hey, I'm onboarding the new performance engineer and they're asking
    about that caching situation we dealt with after the big slowdown. Can you help me
    explain what we ended up implementing and why it worked?"
Required reasoning path:

Identify "big slowdown" -> find incident discussion in #engineering
Locate related performance tickets in Linear
Find caching strategy discussions across multiple threads
Connect implementation approach to success metrics
Verify final solution through status updates and retrospective comments

Scenario: Security vulnerability fix
Poor Question: "Who fixed the authentication bug in issue #234?"
Good Question: "The security team is doing their quarterly review and asked about that
    authentication problem that had everyone stressed. Can you remind me what the root
    cause turned out to be and how we prevented it from happening again?"
Required reasoning path:

Identify "authentication problem" from stress context clues
Find initial security discussions in relevant channels
Locate corresponding security tickets and investigations
Connect root cause analysis across multiple conversations
Identify prevention measures from implementation updates

Scenario: Feature rollback decision
Poor Question: "Why was the notification feature disabled?"
Good Question: "I'm updating the roadmap document and need to explain the notification
    feature situation to stakeholders. What ended up being the main concern that led to
     our decision there?"
Required reasoning path:

Identify notification feature context from roadmap implications
Find feature development discussions
Locate rollback decision conversations
Connect stakeholder concerns across multiple platforms
Verify decision rationale through leadership communications

Progressive Question Design
Primary Question (Multi-hop reasoning)
Start with an ecologically valid, appropriately vague question requiring the full
    reasoning chain.
Follow-up Questions (Memory testing)
Design 2-3 progressively specific follow-ups that test retention:
Follow-up 1: "And what was the timeline concern that influenced that decision?"
Follow-up 2: "Who were the key people who made that call?"
Follow-up 3: "How did this connect to the client requirements we discussed?"
Output Requirements
JSON Structure
Create a comprehensive event history JSON containing:

Chronologically ordered events across all platforms
Realistic message threading and references
Proper user attribution and timestamps
Status transitions for Linear tickets
Cross-platform references and mentions

Question-Answer Pairs
```

```
Design questions that:

Sound like actual workplace inquiries - colleagues asking for context, preparing for
    meetings, onboarding explanations
Require domain knowledge to interpret - understanding what "the performance issue" or "
    the client escalation" refers to
Have deterministic answers despite vague framing
Test both reasoning and memory through progressive follow-ups

Verification Criteria

Questions reflect realistic workplace information needs
Vagueness requires genuine reasoning rather than keyword matching
Answer requires information synthesis from minimum 3 platforms
No single message thread or ticket contains sufficient information
Human evaluator with JSON access can definitively verify correctness

File Deliverables

event_history_{REPO}.json - Complete multi-platform scenario
config_{REPO}.yaml - Question-answer configuration with ecological validity ratings
verification_guide.md - Step-by-step reasoning path for each question

Remember: The goal is to create questions that sound like natural workplace
    communication while requiring sophisticated multi-platform reasoning to answer
    accurately.
```

The outputs are then iteratively made more difficult if the evaluated agent passes the questions around the event history. Difficulty can be increased along the dimension of either question complexity or by making the event history more complicated. We explore both cases on a per-sample basis to ensure validity of the dataset.

A.1.2 Top-Down Approach

The system prompt is as below:

```
""" you are a data generator to create a timeline in the format as this example:

Example timeline:{timeline_examples}.

Your task is as below:

{task_description}

Only output the timeline in JSON format.
"""
```

This is an example of task description in natural language:

```
Create a timeline, starting from the PM Alice who wants to build a product feature that
    allows searching product images based on descriptions. The engineer bob created a
    linear ticket to list the step of implementation. Alice and bob started a slack
    message and confirmed that the search is based on the keyword of the product
    description. This is further confirmed with Diana in another thread between Diana
    and bob. The initial implementation was done with a few git commits and the
    original ticket was closed. But when Alice tested the product feature, she reported
     that a lot of times, the product cannot be found based on here description. Bob,
    Diana and Alice started another Slack thread, and after 20 messages, they found
    that what Alice meant by description is from users, not directly from product
    description in their cataglog. They decided to implement the whole system
    differently, and create a new linear ticket for it.
```

A.2 Prompts and Interesting for Evaluation Measures

A.2.1 Correctness

```
"""You are evaluating whether an AI agent correctly answered a memory-related question
    about a
workplace timeline.

QUESTION: {question}

EXPECTED ANSWER: {expected_answer}

AGENT'S EXTRACTED ANSWER: {extracted_answer}

Your task is to determine if the agent's answer is correct. Consider:
1. Exact matches are obviously correct
2. Semantic equivalence (same meaning, different wording)
3. Partial correctness (got part of a multi-part answer right)
4. Reasonable interpretations of ambiguous questions

Respond with ONLY a JSON object in this exact format:
{{
    "score": <float between 0.0 and 1.0>,
    "is_correct": <true or false>,
    "confidence": <float between 0.0 and 1.0>,
    "reasoning": "<brief explanation of your evaluation>"
}}

Examples:
- If expected "done" and agent said "completed": score 1.0, correct true
- If expected "alice, bob" and agent said "alice": score 0.5, correct false
- If expected "in_progress, charlie" and agent said "in_progress, urgent, charlie":
    score 1.0, correct true
  (extra info OK)
- If completely wrong: score 0.0, correct false"""
```

A.2.2 Redundancy

```
"""Analyze these tool calls for memory retrieval redundancy:

{chr(10).join(formatted_calls)}

Task: Identify redundant tool calls where the agent re-retrieves similar information.

Consider these as redundant:
- Same tool with very similar parameters (e.g., "get_ticket(id1)" then "get_ticket(id1)
    ")
- Semantically equivalent queries (e.g., "list_tickets(status='done')" then "
    list_tickets(status='completed')")
- Broad queries that contain info from specific queries (e.g., "get_ticket(id1)" then "
    list_tickets()")

Consider these as NOT redundant:
- Different tools entirely
- Same tool with meaningfully different parameters
- Natural progression from general to specific

Respond with ONLY this JSON format:
{{
    "redundant_calls": <number of redundant calls>,
    "total_calls": <total number of calls>,
    "redundancy_fraction": <redundant_calls / total_calls>,
    "efficiency_score": <1.0 - redundancy_fraction>,
    "reasoning": "<brief explanation of which calls were redundant>"
```

```
}}
"""
```

A.3 Prompts for Statistics

A.3.1 Cross Reference

```
"""
Analyze this {current_platform} content for references to other platforms.
Rate each reference type from 0.0 to 1.0:

Content: "{content}"

Rate how much this content references or alludes to:
- Linear tickets/issues (mentions like "ticket", "Linear", "BACK-456", "milestone", "bug
      report")
- Slack messages/channels (mentions like "channel", "message", "thread", "@someone said
     ", "in #engineering")
- Git commits/PRs/code (mentions like "commit", "PR", "pull request", "branch", "merge",
      "code change")

Respond only with a JSON object:
{{"linear": 0.0, "slack": 0.0, "git": 0.0}}

Examples:
- "Created BACK-456 for the TypedDict issue" -> {{"linear": 0.9, "slack": 0.0, "git":
    0.0}}
- "PR #12124 is already getting review" -> {{"linear": 0.0, "slack": 0.0, "git": 0.8}}
- "@alex_chen mentioned in #engineering" -> {{"linear": 0.0, "slack": 0.8, "git": 0.0}}
"""
```

B Additional Results & Analysis

B.1 Illustrative GPT-5 Run - Complete Example

Q1: *Mike mentioned something about a default value he set in model configuration. Sarah referenced 'the specific case from the ticket' but never stated what it was. David mentioned finding 'the issue' but was vague about location details. By correlating across all these conversations and investigating the codebase, determine: what is the sum of all numeric values that appear as default parameters in factory function definitions within the test files?* A^*: 547
Q2: *The team communicated about a bug resolution timeline with vague date references like 'today', 'last month', and 'this afternoon'. By parsing all temporal references in context and correlating with the event timeline, determine: on what date did Sarah first report the factory function conversion issue?* A^*: 20250515
Q3: *Multiple conversations referenced configuration values, default parameters, and batch sizes across different contexts. By correlating* ***Mike's early comment about batch size*** *with the actual repository structure at the* ***specified commit****, determine: what is the exact line number where the first occurrence of the number 32 appears in any Python file within the nemo_run directory?* A^*: 142
Tool Calls Made:

list_remote_git_repositories() → run_shell_command(git clone http://git.local:3000/pgym/NeMo-Run.git) → list_directory(path = NeMo-Run, detailed = true) → list_directory(path = NeMo-Run/test, detailed = true) → read_file(path = NeMo-Run/test/dummy_factory.py) → search_file_content(pattern = "@run.cli.factory", search_path = "NeMo-Run/test", case_sensitive = false, max_matches = 100, context_lines = 2) → read_file(file_path = NeMo-Run/test/cli/test_api.py, start_line=1, max_lines = 1000) → ... ... (Further Navigation and Reading of Files) → Answer(4858.9001) ✘ → Question 2 Asked → list_tickets() → get_tickets(ticket_id = e8da2eed-bb7c-4d9f-93b2-89977fe48f2d) → get_tickets(ticket_id="39b5f839-d79c-438c-8eb3-fee9bb5af641") → get_tickets(ticket_id="970a53b2-ed24-4719-8ee5-153d03c3d983") → search_file_content(pattern="", case_sensitive = False, max_matches=200, context_lines=1) → get_unread_messages() → list_channels() → get_channel_messages(channel="engineering", limit=100) → get_ticket(ticket_id="171754fc-bca8-4057-8c5f-410ed9721590") → Answer("2025-05-15") ✓ → Question 3 Asked → list_directory(path="NeMo-Run/nemo_run", detailed=true) → search_file_content(pattern="32", search_path= "NeMo-Run/nemo_run") → Answer(142) ✓

Table 5: Tool Call Success Rates (SRs) Across Different Memory Benchmark Methods

Method	Mean SR	Median SR	Min SR
zep_gpt5_results_20250829_233315	0.9166	0.9698	0.7229
mem0_gemini_results_20250830_123405	0.9341	0.9535	0.7570
nomemory_gpt5_results_20250829_222210	0.9179	0.9731	0.6967
mem0_gpt5_results_20250829_233011	0.9307	1.0000	0.7017
mem0_claude_results_20250829_233021	0.9524	0.9766	0.7717
zep_claude_results_20250829_233311	0.9106	0.9082	0.7027
nomemory_claude_results_20250829_232558	0.9532	0.9854	0.7761
nomemory_gemini_results_20250830_123311	0.9305	0.9491	0.7100
zep_gemini_results_20250830_123111	0.9291	0.9534	0.6455

B.2 Hyperparameters

We use a temperature of 1 for all our experiments. We choose a higher temperature to allow reasoning and also to prevent an implicit length horizon from getting in the way of the relatively longer output token horizons and turn counts needed to solve the task.

B.3 σ of Correctness

Table 6: σ on Correctness numbers, averaged across 5 runs.

Method	σ(Correctness)
GPT-5+NoMem	0.0594
GPT-5+Mem0	0.0610
GPT-5+Zep	0.0452
Gemini-2.5-pro+NoMem	0.0480
Gemini-2.5-pro+Mem0	0.0342
Gemini-2.5-pro+Zep	0.0430

B.4 Tool/Component Specifications

B.4.1 Slack

1. $get\_unread\_messages(limit : int = 50) \rightarrow str$ - Retrieve unread messages grouped by channel/DM, automatically marked as read after retrieval
2. $send\_channel\_message(channel : str, message : str) \rightarrow str$ - Send message to a channel with @username mention support
3. $send\_direct\_message(to\_user : str, message : str) \rightarrow str$ - Send direct message to another user
4. $get\_channel\_messages(channel : str, limit : int = 50, after\_id : str = None) \rightarrow str$ - Retrieve channel messages with pagination support

5. $get\_direct\_messages(with\_user : str, limit : int = 50, after\_id : str = None) \rightarrow str$ - Retrieve DM conversation history with pagination
6. $list\_channels() \rightarrow str$ - List all available channels in workspace
7. $list\_users() \rightarrow str$ - List all users in the Slack workspace
8. $send\_offline\_message(to\_user : str, message : str) \rightarrow str$ - Send offline message when away from desk

B.4.2 Linear

1. $create\_ticket(title : str, description : str, team : str, ...) \rightarrow str$ - Create new ticket with comprehensive metadata including priority, dates, labels, and milestones
2. $update\_ticket(ticket\_id : str, ...) \rightarrow str$ - Update existing ticket fields with authorization checks
3. $get\_ticket(ticket\_id : str) \rightarrow str$ - Retrieve comprehensive ticket details by ID
4. $list\_tickets(team : str = None, status : str = None, ...) \rightarrow str$ - Query tickets with filtering by team, status, lead, milestone, priority, and date ranges
5. $delete\_ticket(ticket\_id : str) \rightarrow str$ - Permanently remove ticket (admin-only operation)
6. $assign\_ticket(ticket\_id : str, lead : str) \rightarrow str$ - Assign or reassign ticket ownership with notifications
7. $add\_team\_member(team : str, member : str) \rightarrow str$ - Add user to ML or Engineering team (admin-only)
8. $remove\_team\_member(team : str, member : str) \rightarrow str$ - Remove user from team (admin-only)
9. $list\_team\_members(team : str = None) \rightarrow str$ - List team membership for specified or all teams
10. $create\_milestone(name : str, description : str, start\_date : str, end\_date : str) \rightarrow str$ - Create project milestone with unique naming (admin-only)
11. $update\_milestone\_progress(milestone : str, completion\_rate : float) \rightarrow str$ - Update milestone completion and sync to associated tickets (admin-only)
12. $list\_milestones() \rightarrow str$ - Retrieve all milestones with current progress status

B.4.3 Git

1. $list\_remote\_git\_repositories() \rightarrow str$ - List available remote repositories with clone instructions and commit information

B.4.4 Shell Commands

The agent also has access to its own filesystems into which it can execute shell commands such as git clone as well as *list_directory()*, *read_file()*, *search_file()* etc

B.5 Memory Based Components

1. $store\_memory(content : str, metadata : dict = None) \rightarrow str$ - Store important information in persistent memory for future recall and context building
2. $search\_memory(query : str, limit : int = 5) \rightarrow str$ - Search for contextually relevant memories using hybrid semantic and keyword search
3. $get\_memories(limit : int = 10) \rightarrow str$ - Retrieve recent stored memories for context awareness and debugging

B.6 Compute Resources

The following configuration was used for running evaluation experiments on MEMTRACK.

Table 7: System Requirements to run environment

Resource	**Requirement**
Memory	2GB per concurrent benchmark (Docker containers + memory providers)
Ports	Ensure 3000-3999 range is available
Docker Networks	System should support 10+ custom networks