Mem2ActBench: A Benchmark for Evaluating Long-Term Memory Utilization in Task-Oriented Autonomous Agents

Anonymous ACL submission

Abstract

Large Language Model (LLM)-based agents are increasingly deployed for complex, tool-based tasks where long-term memory is critical to driving actions. Existing benchmarks, however, primarily test a angent's ability to passively retrieve isolated facts in response to explicit questions. They fail to evaluate the more crucial capability of actively applying memory to execute tasks. To address this gap, we introduce MEM2ACTBENCH, a benchmark for evaluating whether agents can proactively leverage long-term memory to execute tool-based actions by selecting appropriate tools and grounding their parameters. The benchmark simulates persistent assistant usage, where users mention the same topic across long, interrupted interactions and expect previously established preferences and task states to be implicitly applied. We build the dataset with an automated pipeline that merges heterogeneous sources (ToolACE, BFCL, Oasst1), resolves conflicts via consistency modeling, and synthesizes 2,029 sessions with 12 user–assistant–tool turns on average. From these memory chains, a reverse-generation method produces 400 tool-use tasks, with human evaluation confirming 91.3% are strongly memory-dependent. Experiments on seven memory frameworks show that current systems remain inadequate at actively utilizing memory for parameter grounding, highlighting the need for more effective approaches to evaluate and improve memory application in task execution. Code and data are available at https://anonymous.4open.science/r/Mem2ActBench-29AC/.

1 Introduction

Large language model (LLM)-based agents are increasingly used as persistent assistants, interacting with users over extended periods. In these scenarios, users rarely restate all task constraints explicitly. Instead, preferences, requirements, and partial task states are gradually established across prior interactions, often interrupted by unrelated conversations, and are implicitly assumed to be remembered and applied in later requests. A realistic assistant is therefore expected not only to store long-term memory, but to actively retrieve and apply relevant past information to execute concrete actions, such as grounding missing arguments in tool invocations. Current memory benchmarks primarily test an agent's ability to retrieve isolated information from memory based on explicit questions, such as MSC (Xu et al., 2022) and Lo-

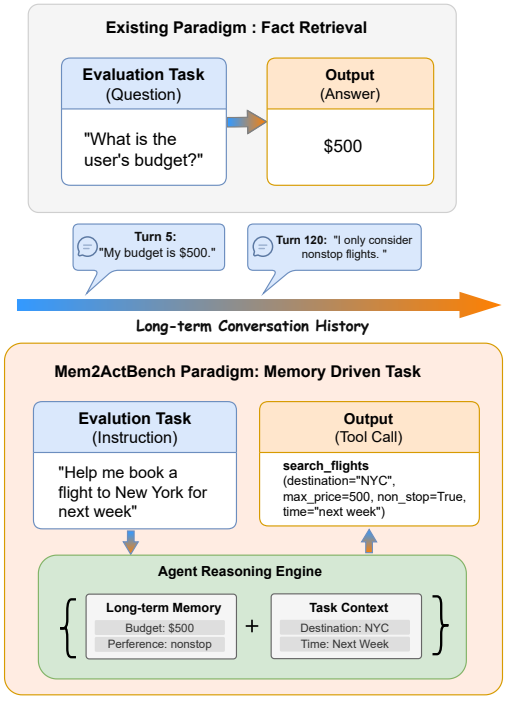

Figure 1: **Fact retrieval vs. memory-driven task execution.** Existing benchmarks focus on direct queries for a factual answer. In contrast, our benchmark requires the agent to combine past memories and generate a grounded tool call.

como (Maharana et al., 2024) (e.g., "What is the user's budget?"), but may under-test a more realistic challenge: given an underspecified instruction, the agent must infer what constraints to retrieve from long-term memory and ground them into an executable tool invocation, as shown in Figure 1.

To bridge this gap, we introduce **MEM2ACTBENCH**, a benchmark that evaluates whether agents can reconstruct executable tool arguments from dispersed long-term memory. Unlike prior benchmarks that test explicit memory retrieval, MEM2ACTBENCH models scenarios where task is clear but critical execution constraints are distributed across long, interruption-heavy histories.Each instance is constructed so that the correct tool invocation is uniquely grounded in memory but cannot be inferred from the final query alone. We construct an automated pipeline to interleave task-oriented tool-use data with natural dialogue to construct long-term interaction histories that reflect realistic assistant usage. To transform these histories into usable long-term memory, we resolve conflicting states and organize extracted facts into a coherent memory evolution chain that captures how topics are updated over time, before applying reverse query generation with strict leakage control to ensure genuine memory dependence.

The main contributions of this work are:

- We introduce a principled benchmark design for evaluating inference-driven long-term memory utilization in tool-augmented agents, targeting scenarios where task execution requires grounding underspecified requests using historical constraints.
- We construct and release MEM2ACTBENCH, a benchmark comprising 400 memory-dependent tool-use tasks derived from 2,029 long-context dialogue sessions. Human verification confirms that 91.3% of the tasks cannot be solved without access to long-term memory, ensuring the reliability of the evaluation.
- We conduct a comprehensive evaluation of seven representative memory frameworks and systematically analyze their failure modes, revealing persistent bottlenecks in memory retrieval and parameter grounding for tool-using tasks.

2 Related Work

2.1 Agent Memory Architectures

For autonomous agents in long-horizon, multi-stage tasks, memory has shifted from passive storage to an active module that supports planning and tool use. Existing approaches broadly fall into: (i) extending the context window to include long histories, which can suffer from the "lost-in-the-middle" effect and higher inference cost (Liu et al., 2024); (ii) external memory banks (e.g., vector stores) that retrieve stored interaction fragments or facts on demand, such as RET-LLM (Modarressi et al., 2024) and MemoryBank (Zhong et al., 2024); and (iii) explicit memory managers that organize and update memory structures, including Generative Agents (Park et al., 2023), MemGPT (Packer et al., 2024), and A-Mem (Xu et al., 2025). Despite these advances, most evaluations still emphasize memory *storage* and *retrievability*, rather than whether agents can decide *what* to retrieve and *how* to apply it under tool and task constraints.

2.2 Agent Memory Benchmarks

Most memory benchmarks follow an explicit query-based paradigm. Early work targets short-dialogue consistency (e.g., Persona-Chat (Yamashita et al., 2023)), while MSC (Xu et al., 2022) extends to cross-session long-term attribute retention. More recent benchmarks, such as LoCoMo (Maharana et al., 2024), LongMemEval (Wu et al., 2024), and MemoryAgentBench (Hu et al., 2025), increase time span and retrieval difficulty, but still largely instantiate "Question → Retrieval → Answer" with an explicitly provided query. This design under-tests realistic settings where retrieval intent must be inferred from underspecified task demands (e.g., missing tool arguments), rather than directly asked. Tool-oriented benchmarks similarly provide limited coverage of long-term memory usage: even approaches that incorporate memory into tool invocation, such as BFCL-v4 (Patil et al., 2025), typically operate over short interaction horizons.

As summarized in Table 1, prior benchmarks emphasize explicit query memory matching and static memory usage. MEM2ACTBENCH instead targets inference-driven long-term memory utilization, evaluating whether agents can infer task-critical constraints from evolving interaction histories and ground them into executable tool calls.

Dataset	*Session*	*Turns*	*Tokens*	*QA pairs*	*Reasoning*	*Memory Evolution*	*Tool use*
MSC(Xu et al., 2022)	4	53	564	/	Retrieval	✗	✗
MemoryBank(Zhong et al., 2024)	10	38	3094	7	Retrieval	✓	✗
Locomo(Maharana et al., 2024)	27.2	21.6	16618.1	199	Retrieval	✓	✗
DialSim(Kim et al., 2025)	1313	1310	352k	1056	Retrieval	✓	✗
LongmemEval(Wu et al., 2024)	48 / 500	10.34	115k / 1.5M	500	Retrieval	✓	✗
Mem2ActBench(Our work)	2029	13	3238	400	Inference	✓	✓

Table 1: **Comparison of representative agent-memory benchmarks.** *Session* is the number of discrete conversation segments per sample (temporal span); *Turns* is the total dialogue turns (interaction length); *Tokens* is the total token count aggregated over all sessions (text scale); *QA pairs* is the number of question–answer instances used for evaluation; *Reasoning* indicates whether tasks primarily test factual retrieval or inference; *Memory Evolution* indicates whether memory states are dynamically updated during interaction; *Tool use* indicates whether external tool invocation is supported in evaluation.

3 Methodology

3.1 Overview

To evaluate an agent's ability to proactively apply long-term memory for task execution, we introduce MEM2ACTBENCH, constructed via a three-stage automated pipeline. First, we simulate realistic, interruption-heavy interactions by interleaving task-oriented data with conversational noise, creating fragmented contexts that necessitate long-term memory. Next, we synthesize these interactions into a logically coherent Fact Evolution Chain to serve as a ground-truth memory. Finally, we employ a reverse-generation paradigm, creating underspecified queries derived from ground-truth tool calls. This design ensures that successful task completion strictly requires reasoning over the historical memory chain, thereby directly evaluating the agent's ability to apply memory for inference-driven tasks rather than just retrieving facts.

3.2 Heterogeneous Data Integration

Task-oriented Dialogue. We construct the dataset by synthesizing multi-step tool-use trajectories from ToolACE and BFCL via LLM-based generation. For ToolACE(Liu et al., 2025), we process 8,000 samples, parsing raw interaction traces and employing LLM to reconstruct them into coherent, natural multi-turn dialogues. For BFCL_v3(Patil et al., 2025), we aggregate diverse subsets (including live, parallel, and multi-turn scenarios). Using the same synthesis approach, we transform static task queries and ground-truth tool calls into dynamic multi-round conversations, ensuring the dataset faithfully reflects realistic user-assistant interactions and precise tool execution flows.

Conversational Noise. We inject conversational noise from OASST1 (Köpf et al., 2023), a tree-structured corpus with ranked assistant candidates. We keep only `rank=0` responses and reconstruct full threads by tracing selected leaves to the root.

We collect these dialogues and normalize them into a unified multi-turn format. After alignment, all processed interactions serve as the historical dialogue repository for subsequent memory construction and task generation. Details are provided in Appendix A.

3.3 Constructing the Fact Evolution Chain

3.3.1 Fact Extraction and Grouping

We prompt an LLM to extract structured facts from each dialogue. Each fact is represented as a triple (**attribute**, **fact**, **source ID**), where **fact** is the atomic user statement and **source ID** uniquely identifies the originating dialogue. To prevent unrelated events from being merged under overly generic labels, we instruct the LLM to produce entity-bound attributes whenever applicable (e.g., account modification (YouTube)), which reduces spurious cross-entity comparisons.

We then cluster the extracted **attributes** using **BERTopic** (Grootendorst, 2022) with HDBSCAN as backend. For each cluster, we select one attribute as the canonical representative and map all attributes in the cluster to it, flagged as outliers. The resulting attribute clusters are used as **fact groups** for subsequent conflict detection and evolution analysis.

3.3.2 Memory Evolution Chain Construction

Local Conflict Resolution. For each fact group (sharing the same attribute), we use an LLM to produce a locally consistent evolution chain. Concretely, the LLM (i) orders facts by their temporal

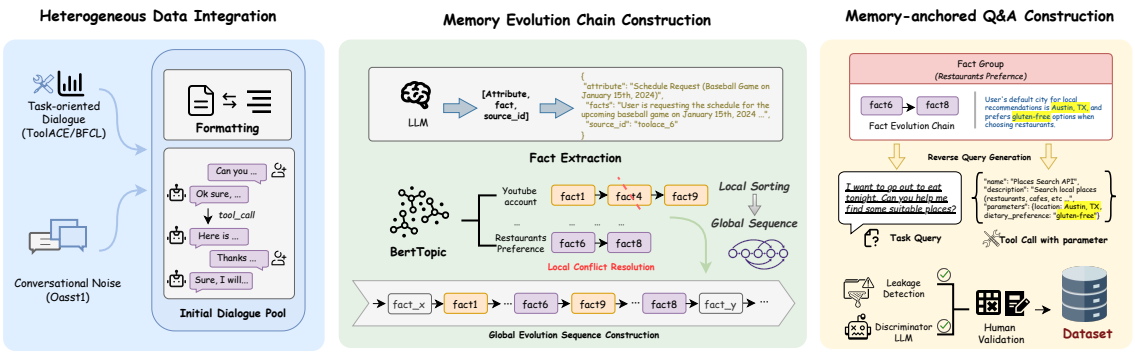

Figure 2: This diagram illustrates the MEM2ACTBENCH framework, a benchmark used to evaluate the long-term memory capabilities of an agent. The framework first constructs a globally consistent and conflict-free "memory evolution chain" by integrating multi-source dialogue data. Then, based on this memory chain, it reverse-engineers question-answering tasks that require long-term memory to correctly select and use tools. Through this automated process, MEM2ACTBENCH can effectively measure an agent's ability to proactively use its memory to complete tasks in complex, long dialogues.

cues, (ii) preserves logically valid updates, such as refinement from coarse to specific (e.g., sports" $\rightarrow$ basketball") and valid multi-valued trajectories (e.g., residences over time), (iii) removes statements that are off-context or in strong logical conflict with other facts under predefined rules, and (iv) drops near-duplicates that provide no information gain. The remaining facts are then compressed into a clean local sequence, which serves as ordering constraints for global integration.

Global Evolution Sequence Construction. We merge the local sequences into one global evolution chain. This is achieved by first constructing a dependency graph where facts are nodes and temporal orderings are directed edges. Next, we apply a modified topological sorting method based on Kahns algorithm. To handle contradictions that manifest as cycles in the graph, we introduce a deterministic heuristic for conflict resolution. When a cycle is found, we identify the deadlocked nodes and remove the one with the highest out-degree. This removes the fact that forces the most downstream ordering constraints, which helps restore a valid order. The final outputs are the globally sorted sequence of facts and a list of any conflicting facts that were discarded.

3.4 Memory-anchored Q&A Construction

3.4.1 Target Tool Selection and Parameter Anchoring

Given a memory evolution chain $\mathcal{S}$, we first construct a fully specified gold-standard tool invocation $C = (t^*, P)$ that is strictly grounded in memory. The target tool t^* is selected via hybrid retrieval (BM25 + BGE-M3) followed by LLM-based decision-making. For parameter construction, all values in P must be either explicitly extracted from or logically inferred based on $\mathcal{S}$. To prevent spurious or hallucinated parameters, we enforce a memory-anchoring constraint: each parameter is validated through a combination of fuzzy matching and an LLM verifier, ensuring that its value can be traced back to the memory chain.

3.4.2 Reverse Implicit Query Generation and Filtering

Starting from the grounded tool invocation C and its supporting memory subset, we reverse-generate an underspecified user query Q. The generation process enforces three critical constraints: (i) Parameter Omission: Key values present in C must be omitted from Q to prevent information leakage; (ii) Reference Dependency: The query must rely on anaphoric expressions (e.g., "book *that* flight", "use *my previous* preference"); (iii) Intent Preservation: The query must remain semantically consistent with the execution of C.

To ensure that each generated query is genuinely memory-dependent, we first filter out samples that reveal parameter values through explicit mentions or implicit hints. We then introduce a discriminator LLM, which attempts to reconstruct the correct tool invocation C using only the query Q and the tools API documentation, without access to historical memory. A sample is retained only if the discriminator fails, guaranteeing that correct tool invocation is impossible without re-

trieving relevant memory. Details are provided in Appendix B.

We employ **Qwen3-Next-80B-A3B-Instruct** [1] as the backbone LLM for target tool selection and parameter grounding. For the reverse implicit query generation stage, we employ **Kimi-K2-Thinking**[2]. Finally, we generate a total of **400** memory-dependent tool-use queries grounded in **2,029** long conversational sessions, with an average of **13** turns per session. These samples constitute the final MEM2ACTBENCH dataset used in all subsequent experiments.

3.5 Task Formalization

We define the evaluation task as a conditional generation problem. Given a memory sequence $\mathcal{M}$ and a user query q, the agent generates the optimal tool invocation $\hat{c}$ by maximizing:

$$\hat{c} = \arg\max_{c \in \mathcal{C}} P(c \mid q, \mathcal{M}) \tag{1}$$

where c consists of a selected tool T and parameter values v_p. Each v_p is derived by reasoning over the context:

$$v_p = f_\theta(p, q, \mathcal{M}) \tag{2}$$

subject to the constraint that v_p is strictly grounded in $\mathcal{M}$.

3.6 Human Verification

We conduct expert verification to assess the reliability of MEM2ACTBENCH at three critical stages: fact extraction, conflict resolution, and memory-dependent task formulation. A total of five expert annotators, each holding advanced degrees in fields such as Computational Linguistics, Computer Science, or Artificial Intelligence, and with prior experience in evaluating AI models, were recruited. Each item was independently reviewed by at least two annotators, ensuring thorough evaluation. Disagreements between annotators were resolved through discussion.

For fact extraction, annotators judge whether each fact is (i) entailed by the dialogue context and (ii) correctly normalized. For conflict resolution, they assess whether the resulting memory evolution chain is coherent and logically consistent. For memory dependency, annotators determine whether the gold tool invocation remains underdetermined given the user query alone (i.e., would be infeasible to infer without access to long-term memory). As shown in Table 2, we obtain high validation rates across all stages, indicating that MEM2ACTBENCH provides faithful memory states and that its tool-use tasks intrinsically require memory rather than surface-level reasoning.

Aspect	#Samples	Validated (%)
Fact Extraction Accuracy	200	96.5
Conflict Resolution Quality	150	86.7
Memory Dependency Validity	200	91.3

Table 2: Expert verification on randomly sampled instances from three stages of the MEM2ACTBENCH pipeline.

4 Experiment

4.1 Experimental Setup

Datasets. We selected only the conversation histories that contain all the necessary QA evidence, ensuring the original order is preserved, with a total of 429 sessions used.

Baselines. We evaluate the following representative agent memory systems on MEM2ACTBENCH: Long-term Memory (RAG), Generative Agents (Park et al., 2023), SCM(Wang et al., 2023), Langmem(LangChain, 2025), MemTree(Rezazadeh et al., 2024), Mem0(Chhikara et al., 2025) and A-Mem(Xu et al., 2025). To control for backbone model capacity, we conduct experiments using three model scales of the Qwen2.5 family, namely **Qwen2.5-7B-Instruct**, **Qwen2.5-32B-Instruct**, and **Qwen2.5-72B-Instruct**(Team, 2024), as the inference backbone for all memory systems. All models are evaluated under fixed decoding settings (temperature = 0.0) to ensure result stability and comparability across scales. For memory systems involving retrieval, we use **BGE-m3**(Chen et al., 2024) as the embedding model.

Evaluation Metrics. We evaluate memory-based tool-call generation with three metrics: **F1** for parameter-level precision/recall, **BLEU-1** for unigram overlap with the reference, and **Tool Accuracy (TA)**, which is True only if the correct tool is used and all parameters match exactly. In the main results, we provide the ground-truth tool

[1] https://huggingface.co/Qwen/Qwen3-Next-80B-A3B-Instruct

[2] https://huggingface.co/moonshotai/Kimi-K2-Thinking

to control for tool-selection errors and focuses the comparison on memory-based parameter grounding.

4.2 Main Results

Table3 suggests that TA stays tightly clustered (∼87–97%) and changes little from 32B to 72B on average, indicating that most remaining errors are not structural but semantic. In contrast, argument grounding shows clear headroom: mean F1 increases from 20.4 (7B) to 28.9 (72B), with diminishing returns beyond 32B (≈ +3.5 F1 from 32B→72B), implying that scaling mainly improves post-retrieval composition. The method ranking reveals that A-mem and LTMemory form the top cluster (72B F1=35.9/35.3) and nearly converge at scale, while MemoryTree remains competitive (33.2) but retains a ∼2–3 F1 gap, suggesting structured memory helps but does not fully resolve parameter assembly. Notably, the largest scaling gain appears in weaker memory managers (e.g., Mem0: +14.7 F1 from 7B→72B), consistent with larger models compensating via stronger cross-turn inference when memory organization is suboptimal.

5 Discussions

5.1 Retriever Analysis

To probe whether performance is mainly limited by memory retrieval rather than reasoning, we conduct a controlled comparison across three retrieval conditions (no retrieval, passive retrieval with standard retrievers, and oracle retrieval with ground-truth memories. As shown in Table 4, the best passive retrieval result is achieved by the hybrid retriever at k=5, reaching $F1 \approx 30.7$. In contrast, oracle retrieval boosts performance to $F1 \approx 53.8$, creating a gap of over 23 F1 points. This large margin suggests that the dominant bottleneck is evidence hitting/retrieval quality, rather than the models pure reasoning once the correct supporting memories are available. This finding aligns with our benchmark goal of evaluating memory application for parameter grounding under underspecified requests: improving performance require stronger evidence-hitting mechanisms (e.g., better indexing, retriever training, and query formulation) rather than simply scaling the backbone model.

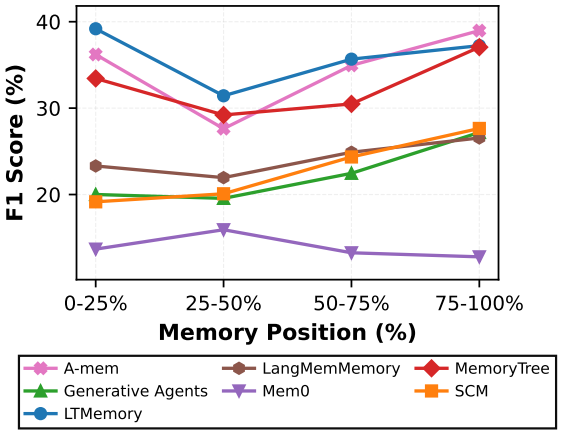

Figure 3: F1 score versus the normalized position of the earliest supporting memory.

5.2 Impact of Memory Distance

While the main results demonstrate the overall capability of memory models in tool-use tasks, a critical question remains: does the physical distance between the relevant memory and the current query affect the model's reasoning performance? Existing research on long-context LLMs suggests a "lost-in-the-middle" phenomenon or performance degradation as key information recedes further into the history. To investigate this in the context of MEM2ACTBENCH, we conducted a fine-grained analysis of model performance relative to the "memory distance". For each sample, we identify the earliest turn that provides evidence for any required tool parameter, denoted t_{earliest}, within a conversation of length L. We use the normalized position $P_{\text{mem}} = t_{\text{earliest}}/L$ and bucket samples into four quartiles (0–25%, 25–50%, 50–75%, 75–100%).

Findings. Figure 3 reveals a pronounced positional bias: most baselines achieve higher F1 when the supporting evidence appears in the early (0–25%) or recent (75–100%) context, but drop sharply when it lies in the mid-history (25–50%), forming a clear mid-context valley. AgenticMemory falls from ∼36% to ∼25% in the 25–50% bin, echoing a "lost-in-the-middle"-like failure in tool-use. By contrast, LTMemory is more position-robust, with F1 remaining above 30% across bins. These results directly support that even with retrieval, long-term memory is not reliably *applied* for parameter grounding when evidence is buried in the middle of lengthy interaction histories, leaving mid-context usage as a key bottleneck for

Method	*Qwen2.5-72B-Instruct*			*Qwen2.5-32B-Instruct*			*Qwen2.5-7B-Instruct*			**Average**		
	F1	**BLEU**	**TA**	**F1**	**BLEU**	**TA**	**F1**	**BLEU**	**TA**	**F1**	**BLEU**	**TA**
LTMemory	35.32	67.93	92.00	**33.87**	67.71	90.20	26.71	64.07	87.25	31.97	66.57	89.82
SCM(Wang et al., 2023)	22.73	62.04	96.25	17.35	59.37	95.75	14.99	57.37	91.00	18.36	59.59	94.33
Generative Agents(Park et al., 2023)	22.38	62.58	95.50	17.81	59.76	96.24	14.44	55.59	89.75	18.21	59.31	93.83
MemTree(Rezazadeh et al., 2024)	33.21	**68.17**	94.25	31.89	67.69	94.50	24.60	63.91	88.50	29.90	66.59	92.42
Mem0(Chhikara et al., 2025)	28.95	66.47	**96.75**	24.52	64.36	97.00	14.21	57.37	93.97	22.56	62.73	95.91
Langmem(LangChain, 2025)	24.01	63.06	96.25	18.72	58.27	**97.25**	17.06	58.93	**96.50**	19.93	60.09	**96.67**
A-mem(Xu et al., 2025)	**35.93**	67.93	93.25	33.72	**67.92**	92.25	**30.99**	**66.47**	94.75	**33.55**	**67.44**	93.42

Table 3: Experimental results for different memory methods across multiple model sizes.

Table 4: Performance comparison under varying top-k retrieval settings. Shading in the **Recall@k** column indicates retrieval depth (darker denotes more retrieved documents). Best results among passive retrieval methods are highlighted in bold.

Retrieval Strategy	Recall@k	F1	BLEU	TSA
No Retrieval	–	10.0	8.9	73.8
Passive Retrieval				
BM25	1	29.0	28.0	87.0
	5	26.9	26.2	90.2
	10	27.6	26.7	87.5
Dense	1	25.6	25.3	83.0
	5	29.9	29.1	90.0
	10	28.7	28.3	88.8
Hybrid	1	24.9	24.3	85.0
	5	**30.7**	**29.7**	86.0
	10	30.3	29.5	88.8
Perfect Retrieval				
Oracle	–	**53.8**	**53.7**	88.2

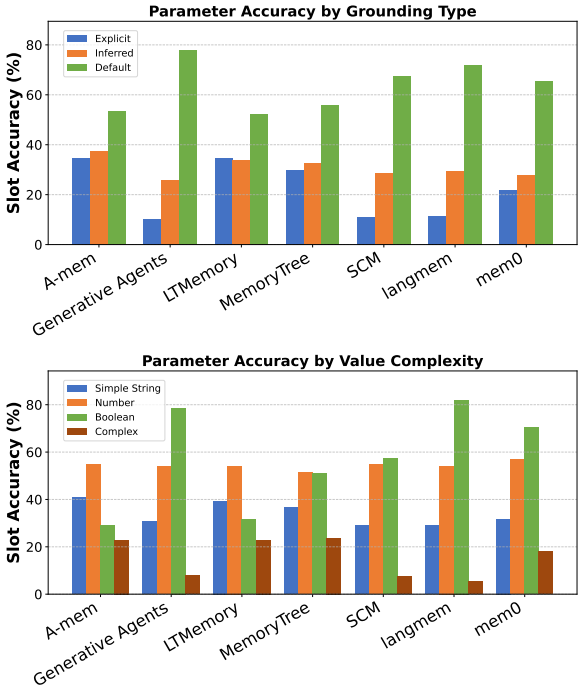

Figure 4: Breakdown of Slot Accuracy by Value Complexity (top) and Grounding Type (bottom).

memory-centric tool agents.

5.3 Parameter Grounding and Complexity

To figure out where argument grounding fails, we report **Slot Accuracy**, as the exact match of each individual argument value, and break it down by grounding type and value complexity.

Grounding Type Analysis. We categorize parameters by how their values are supported in the dialogue history: **Explicit** (directly stated, e.g., "New York"), **Inferred** (needs a semantic conversion, e.g., "upcoming week" $\rightarrow$ `days=7`), and **Default** (not mentioned and should be filled from the tool schema).

For 72B-scale models, Explicit and Inferred show a small gap, indicating that once the right evidence is retrieved, semantic transformation is not the main difficulty. The largest errors come from Default values: models often fail to notice that a value was never specified, and instead generate a plausible default, sometimes guided by distractors in long histories (Figure 4).

Value Complexity. We further bucket values as **Simple String** ($\leq$30 chars), **Number**, **Boolean**, and **Complex** (long strings, specialized identifiers such as URLs/addresses, or nested structures).

Slot Accuracy decreases as values become more complex. Models handle Simple Strings and Numbers reasonably well, but performance drops sharply on **Complex** values, which points to weak lossless retention (e.g., truncation or character-level corruption of identifiers). Boolean accuracy also varies across frameworks, indicating that it depends on both the grounding context and how each system enforces tool constraints (Figure 4).

5.4 Tool Selection Robustness

We stress-test tool choice by increasing the candidate set to $N \in \{1, 2, 5\}$, where each query is paired with $N-1$ distractor tools. Distrac-

Candidate size N	1	2	5
Random negatives			
TSA (↑)	94.50	95.50	93.50
EM (↑)	18.25	18.00	17.00
Arg_F1 (↑)	29.88	29.98	28.27
Hard negatives			
TSA (↑)	94.50	78.00	69.75
EM (↑)	18.25	16.50	14.25
Arg_F1(↑)	29.88	27.17	22.64

Table 5: Tool selection robustness under different candidate tool set sizes ($N \in \{1, 2, 5\}$).

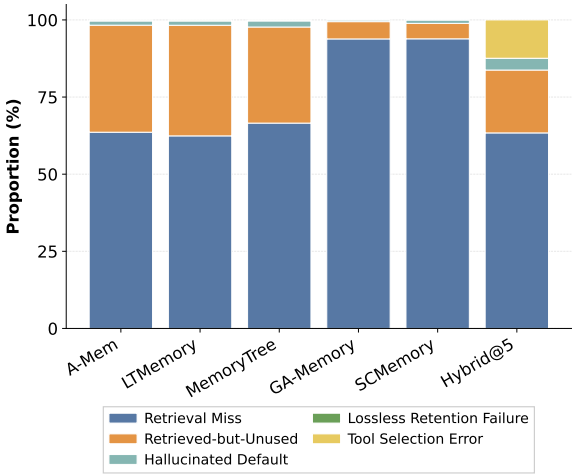

Figure 5: Distribution of failure modes across different memory frameworks.

tors are sampled either **randomly** (uniformly from the tool library) or as **hard negatives** (distractor tools most semantically similar to the ground-truth tool), which tests fine-grained intent separation (e.g., *search* vs. *book*). We report Tool Selection Accuracy (TSA) and end-to-end Exact Match (EM) (correct tool and all arguments). For diagnosis, we also report Arg_F1 conditioned on selecting the correct tool.

Table 5 shows that with random negatives, TSA remains high and nearly unchanged (93.50–95.50%) as N increases, suggesting that models handle unrelated distractors well. In contrast, hard negatives cause a steep drop in TSA, from 94.50% (N=1) to 69.75% (N=5), indicating difficulty when tool semantics overlap. EM stays low in all settings (14.25–18.25%), even when TSA is above 93%, which suggests that argument grounding is the main bottleneck. This is also reflected in Arg_F1: under hard negatives (given the correct tool), it decreases from 29.88 to 22.64, implying that similar distractors can also hurt parameter extraction and inference.

5.5 Error Mode Diagnosis

To characterize why agents fail on memory-grounded parameter filling, we conduct a fine-grained attribution analysis over erroneous predictions. We categorize failures into five types: *(i) Retrieval Miss*, required evidence is absent from retrieved context; *(ii) Retrieved-but-Unused*, evidence is retrieved but not utilized; *(iii) Hallucinated Default*, schema defaults are incorrectly overridden or fabricated; *(iv) Lossless Retention Failure*, long/structured values are corrupted (e.g., truncation or character-level errors); and *(v) Tool Selection Error*, an incorrect tool is selected.

Findings. Figure 5 shows that: (i) As memory frameworks become stronger, failures shift from *accessibility* (retrieval misses dominating weak baselines) to *post-retrieval reasoning* (retrieved-but-unused rising substantially), suggesting retrieval alone is insufficient. (ii) Tool selection is largely robust: agentic frameworks exhibit negligible tool selection errors, while a passive retrieval baseline shows a noticeable fraction of tool-selection failures. (iii) Retrieval Miss remains the largest category even for strong systems, highlighting the enduring difficulty of locating sparse but critical evidence under implicit queries.

6 Conclusion

In this paper, we introduce MEM2ACTBENCH for evaluating whether tool-augmented agents can effectively apply long-term memory to drive task execution. We construct a benchmark through an automated pipeline that simulates real-world interrupted interactions, generating memory-dependent tool calls. Our experiments reveal a significant gap in current memory frameworks, particularly in parameter grounding with mid-context memories often overlooked. These results highlight the limitations of current systems in proactively utilizing long-term memory, especially when tasks are underspecified. Future research should focus on enhancing the active utilization of memory, particularly in scenarios that require reasoning over dispersed, incomplete information.

Limitations

MEM2ACTBENCH is designed to evaluate memory-grounded parameterization in tool-based tasks under controlled conditions. However, it is limited to offline tool-call generation, using a fixed backbone model family, which does not reflect the diversity of real-world models. The benchmark also excludes interactive execution settings, where agents adapt to feedback over time. Additionally, while automated task generation helps scale the dataset, it may not fully capture the complexities of real-world dialogues. Lastly, human verification introduces potential biases in the validation process, especially in edge cases.

Ethical considerations

MEM2ACTBENCH is constructed by synthesizing interaction histories from publicly available datasets, including task-oriented tool-use data from ToolACE and BFCL, and conversational content from OpenAssistant (OASST1). No new data were collected from end users or through human-subject experiments. Following the ethical practices described by these source datasets and the ACL ethics guidance, we take steps to reduce privacy risks by releasing only processed benchmark instances necessary for evaluating memory-grounded tool use, and by applying automated redaction or rewriting to remove obvious personally identifiable information (e.g., emails, phone numbers, account identifiers) when encountered.

References

Jianlyu Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024. M3-embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 2318–2335.

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

Maarten Grootendorst. 2022. Bertopic: Neural topic modeling with a class-based tf-idf procedure. *arXiv preprint arXiv:2203.05794*.

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

Jiho Kim, Woosog Chay, Hyeonji Hwang, Daeun Kyung, Hyunseung Chung, Eunbyeol Cho, Yeonsu Kwon, Yohan Jo, and Edward Choi. 2025. Dialsim: A dialogue simulator for evaluating long-term multi-party dialogue understanding of conversational agents. *Preprint*, arXiv:2406.13144.

Andreas Köpf, Yannic Kilcher, Dimitri Von Rütte, Sotiris Anagnostidis, Zhi Rui Tam, Keith Stevens, Abdullah Barhoum, Duc Nguyen, Oliver Stanley, Richárd Nagyfi, and 1 others. 2023. Openassistant conversations-democratizing large language model alignment. *Advances in neural information processing systems*, 36:47669–47681.

LangChain. 2025. Langmem (langchain-ai/langmem). `https://github.com/langchain-ai/langmem`. Version 0.0.30; accessed 2026-01-03.

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

Weiwen Liu, Xu Huang, Xingshan Zeng, xinlong hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong WANG, Yuxian Wang, Wu Ning, Yutai Hou, Bin Wang, Chuhan Wu, Wang Xinzhi, Yong Liu, Yasheng Wang, and 8 others. 2025. Toolace: Winning the points of llm function calling. In *International Conference on Representation Learning*, volume 2025, pages 41359–41381.

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

Ali Modarressi, Ayyoob Imani, Mohsen Fayyaz, and Hinrich Schütze. 2024. Ret-llm: Towards a general read-write memory for large language models. *Preprint*, arXiv:2305.14322.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2024. Memgpt: Towards llms as operating systems. *Preprint*, arXiv:2310.08560.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pages 1–22.

Shishir G. Patil, Huanzhi Mao, Charlie Cheng-Jie Ji, Fanjia Yan, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. 2025. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*.

Alireza Rezazadeh, Zichao Li, Wei Wei, and Yujia Bao. 2024. From isolated conversations to hierarchical schemas: Dynamic tree memory representation for llms. *arXiv preprint arXiv:2410.14052*.

Qwen Team. 2024. Qwen2.5: A party of foundation models.

Bing Wang, Xinnian Liang, Jian Yang, Hui Huang, Shuangzhi Wu, Peihao Wu, Lu Lu, Zejun Ma, and Zhoujun Li. 2023. Scm: Enhancing large language model with self-controlled memory framework. *arXiv e-prints*, pages arXiv–2304.

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

Jing Xu, Arthur Szlam, and Jason Weston. 2022. Beyond goldfish memory: Long-term open-domain conversation. In *Proceedings of the 60th annual meeting of the association for computational linguistics (volume 1: long papers)*, pages 5180–5197.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. 2025. A-mem: Agentic memory for llm agents. *Preprint*, arXiv:2502.12110.

Sanae Yamashita, Koji Inoue, Ao Guo, Shota Mochizuki, Tatsuya Kawahara, and Ryuichiro Higashinaka. 2023. Realpersonachat: A realistic persona chat corpus with interlocutors own personalities. In *Proceedings of the 37th Pacific Asia Conference on Language, Information and Computation*, pages 852–861.

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

A Data Processing Details

A.1 Pipeline Implementation

We employ a standardized pipeline to unify heterogeneous data sources into a consistent multi-turn format. The processing specifics for each source are as follows:

- **ToolACE Processing:** We parse raw interaction traces using a custom stack-based algorithm to handle nested bracket structures (e.g., `[Function(args...)]`). These parsed traces are then refined into natural language dialogues using `Qwen/Qwen3-Next-80B-A3B-Instruct` (temperature=0.0) to ensure conversational fluidity while preserving execution logic.
- **BFCL v3 Synthesis:** To transform static query-response pairs into dynamic interactions, we utilize the same LLM engine (temperature=0.0) to synthesize multi-round histories. This involves expanding single-turn ground truths into coherent contexts containing user clarifications and sequential tool invocations.
- **OASST1 Formatting:** We reconstruct full conversation threads by tracing leaf nodes to the root, filtering for high-quality responses (`rank=0`). The data is further processed by deduplicating based on the longest conversation path per prompt and translating non-English samples to English via the Google Translate API to maintain linguistic consistency.

Data Schema. All processed data is serialized into a unified JSONL format compatible with standard chat completion APIs. Table 6 illustrates a representative sample structure.

A.2 Fact Extraction, Semantic Clustering, and Local Conflict Resolution

Our pipeline transforms raw dialogue sessions into a coherent memory structure through three sequential stages: extracting atomic facts, clustering them by semantic topic, and resolving inconsistencies within each local group.

Fact Extraction. We employ LLM to process each dialogue session and extract structured facts formatted as triplets: `(attribute, fact, source_id)`. The `attribute` functions as a normalized category label (e.g., "Dietary Preference"), while the `fact` encapsulates the specific atomic statement derived from the user's input. To ensure precision and reproducibility in the extraction process, we set the generation temperature to 0.0.

Semantic Clustering. To unify scattered references to the same topic across disjointed sessions, we utilize **BERTopic** for semantic aggregation. First, we generate dense vector representations for all extracted attributes using the **BAAI/bge-m3** embedding model. These embeddings are normalized using the L_2 norm to ensure consistent distance measures. For the clustering backend, we employ **HDBSCAN** to identify semantically related attribute groups. Based on our implementation, we configure the algorithm with a minimum

Table 6: Unified JSON schema used for training, aligned with standard tool-use formats.

```
{
  "id": "toolace_sample_01",
  "tools": [
    {
      "type": "function",
      "function": {
        "name": "search_api",
        "description": "Search for information
        online.",
        "parameters": { ... }
      }
    }
  ],
  "conversation_history": [
    {
      "role": "user",
      "content": "Check the weather in NY."
    },
    {
      "role": "assistant",
      "content": "I will check the forecast for
      New York.",
      "tool_calls": [
        {
          "id": "call_abc123",
          "type": "function",
          "function": {
            "name": "weather_api",
            "arguments": "{\"location\": \"New
            York, NY\"}"
          }
        }
      ]
    },
    {
      "role": "tool",
      "tool_call_id": "call_abc123",
      "name": "weather_api",
      "content": "{\"temp\": \"20C\",
      \"condition\": \"Sunny\"}"
    }
  ]
}
```

cluster size of 2 (`min_cluster_size=2`) to capture even sparse thematic connections. We utilize the `euclidean` metric for distance calculation and set the cluster selection method to `leaf` with an epsilon threshold of 0.01, favoring finer-grained clusters over broad generalizations. Post-clustering, all attributes within a cluster are mapped to a single canonical representative, ensuring a unified namespace for subsequent processing.

Local Conflict Resolution. Once facts are grouped by their canonical attributes, we perform local conflict resolution to establish a consistent timeline for each topic. We prompt LLM to analyze each cluster. The model performs three key tasks:

1. **Chronological Ordering:** It determines the logical sequence of events, producing a sorted list of source IDs (`sorted_source_ids`) that reflects the true evolution of the user's status.

2. **Conflict Elimination:** It identifies and explicitly discards source IDs (`discarded_source_ids`) containing obsolete, redundant, or contradictory information that does not fit the coherent narrative.

3. **Narrative Synthesis:** It generates a natural language summary and a reasoning trace, explaining how the state evolved (e.g., a change in preference) to facilitate interpretability.

This hierarchical approach ensures that individual topic histories are locally consistent before they are integrated into the global memory evolution chain. Example shown in Tabel 7.

A.3 Algorithm for Global Evolution Sequence Construction

In this section, we provide the detailed pseudocode for constructing the Global Evolution Sequence. We employ a modified topological sorting algorithm based on Kahn's algorithm. To handle cyclic dependencies (conflicts) arising from merging heterogeneous data sources, we introduce a deterministic heuristic mechanism.

The core of our conflict resolution strategy lies in the *Cycle Breaking* step. When the topological sort stalls due to a cycle (i.e., no nodes have an in-degree of zero), we explicitly identify the set of deadlocked nodes. From this set, we select a candidate to discard based on the following priority:

1. **Maximum Out-Degree:** We prioritize removing the node with the highest out-degree. A high out-degree implies that the fact imposes ordering constraints on many subsequent facts. Removing it relaxes the graph structure most effectively, allowing the sorting process to resume.

2. **Lexicographical Order (Tie-breaker):** If multiple nodes share the same maximum out-degree, we select the one with the lexicographically smallest identifier. This ensures the algorithm is strictly deterministic and reproducible.

The complete procedure is outlined in Algorithm 1.

Table 7: Memory Evolution Example

Item	Content
Attribute Group	Dietary Preference (Vegan)
Sorted Source IDs	`[oss_5924, oss_9685, oss_8154, toolace_1099]`
Discarded Source IDs	`[toolace_518]`
Narrative Summary	The user has evolved from a vegetarian diet that included croissants and wine to a strictly vegan lifestyle, with a strong preference for vegetables and high-protein meals. All non-vegan foods, including fish, are no longer included.
Reasoning	The preference for fish and vegetables (toolace_518) conflicts with the current vegan requirement (oss_5924) and is therefore discarded. The vegetarian status (toolace_1099) is retained as historical context, reflecting an earlier dietary stage. The current vegan requirement, preference for vegetables, and high-protein intake are retained to represent the complete dietary evolution.

Original Facts	`{`
	`"toolace_518": {`
	`"attribute": "Diet Preference (Fish and Vegetables)",`
	`"fact": "User prefers to include fish and vegetables in their diet"`
	`},`
	`"toolace_1099": {`
	`"attribute": "Dietary Preference (Vegetarian)",`
	`"fact": "User is vegetarian and enjoys croissants and wine"`
	`},`
	`"oss_5924": {`
	`"attribute": "Dietary Preference (Vegan)",`
	`"fact": "User requires all lunch options to be vegan"`
	`},`
	`"oss_8154": {`
	`"attribute": "Food Preferences (Vegetables)",`
	`"fact": "User really likes vegetables"`
	`},`
	`"oss_9685": {`
	`"attribute": "Dietary Preference (Vegetarian with High Protein)",`
	`"fact": "User is vegetarian and consumes a lot of protein"`
	`}`
	`}`

B Quality Control for Reverse Query Generation

To ensure that the generated user queries (Q) strictly rely on long-term memory (M) to resolve tool parameters (P), we implement a filtering pipeline. This process eliminates both surface-level parameter leakage and semantic redundancy where the task is solvable without memory context.

Lexical Leakage Filtering. We first apply a rule-based filter to detect explicit mentions of ground-truth values in Q. This check specifically targets parameters derived from memory (marked as *explicit* or *inferred*), ignoring generic schema defaults. The filter rejects Q if it contains: (1) **Exact Matches** of parameter strings (case-insensitive with boundary checks); (2) **Numeric Values** appearing in the query (e.g., price constraints); (3) **Token Overlap** for compound entities (length $>$ 4), ensuring that distinctive parts of a name (e.g., "California" in "Hotel California") are not leaked; and (4) **Structured Identifiers** (e.g., IDs, emails) detected via substring matching.

Solvability Discriminator. Lexical rules cannot detect semantic leakage (e.g., describing "NYC" as "the Big Apple"). To address this, we employ a **Blinded LLM Discriminator**. The discriminator is presented with the query Q and tool schema but *denied access* to the memory context M. It attempts to predict the tool arguments solely from Q. If the discriminator successfully infers

Algorithm 1 Global Evolution Sequence Construction with Deterministic Conflict Resolution

Require: Set of local sequences $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$
Ensure: Globally sorted sequence G, set of discarded facts D

Initialize:
Construct directed graph $\mathcal{G}(V, E)$ from $\mathcal{S}$
Compute in-degree $deg_{in}(v)$ and out-degree $deg_{out}(v)$ for all $v \in V$
$Q \leftarrow \{v \in V \mid deg_{in}(v) = 0\}$ ▷ Initialize queue with source nodes
Sort Q lexicographically ▷ Ensure determinism
$G \leftarrow []$, $D \leftarrow []$
while $|G| + |D| < |V|$ **do**
  if Q is not empty **then**
    $u \leftarrow Q.\text{pop}()$
    $G.\text{append}(u)$
    for each neighbor v of u **do**
      $deg_{in}(v) \leftarrow deg_{in}(v) - 1$
      if $deg_{in}(v) = 0$ **then**
        $Q.\text{push}(v)$
      end if
    end for
  else ▷ **Cycle detected:** heuristic conflict resolution
    $V_{\text{remain}} \leftarrow V \setminus (G \cup D)$
    $v_{\text{drop}} \leftarrow$ NULL, $max\_out \leftarrow -1$
    for v in V_{remain} **do**
      if $deg_{out}(v) > max\_out$ **then**
        $v_{\text{drop}} \leftarrow v$, $max\_out \leftarrow deg_{out}(v)$
      else if $deg_{out}(v) = max\_out$ **and** $v < v_{\text{drop}}$ **then**
        $v_{\text{drop}} \leftarrow v$ ▷ Lexicographical tie-breaker
      end if
    end for
    $D.\text{append}(v_{\text{drop}})$ ▷ Discard the conflicting node
    for each neighbor v of v_{drop} **do**
      $deg_{in}(v) \leftarrow deg_{in}(v) - 1$
      if $deg_{in}(v) = 0$ **then**
        $Q.\text{push}(v)$
      end if
    end for
  end if
end while
return G, D

the correct parameters, the query is deemed *"Solvable Without Memory"* and rejected. This counterfactual evaluation guarantees that the final samples strictly require memory integration.

C Human Verification Guidelines

To ensure the reliability of MEM2ACTBENCH, a rigorous human verification process was implemented, involving five expert annotators with experience in NLP and agent-evaluation tasks. Each annotator holds advanced degrees in fields such as Computational Linguistics, Computer Science, or Artificial Intelligence, and has prior experience in evaluating AI models. The verification process was divided into three stages: fact extraction, conflict resolution, and memory dependency verification. Annotators cross-checked each item, and disagreements were resolved through discussion to ensure accuracy and consistency.

C.1 Stage 1 & 2: Fact Extraction and Conflict Resolution

Annotators verify whether extracted facts are faithful to the dialogue and whether memory updates (conflicts) are resolved logically. We instruct annotators to strictly distinguish between *updates* (which overwrite old values) and *refinements* (which coexist). Table 9 illustrates the adjudication logic for edge cases.

C.2 Stage 3: Memory Dependency Verification

Annotators apply *Information Necessity*: if a competent agent can infer *all* target arguments from the query and the tool schema alone (without consulting long-term memory), the sample is rejected as leakage. In practice, we reject (i) direct/partial leakage that exposes gold values or distinctive substrings, (ii) semantic leakage where the query uniquely identifies the value and becomes solvable without memory, and (iii) unsupported constraints that are grounded in neither memory nor the local query context. Table 8 provides end-to-end filtering examples from the automated pipeline.

C.3 Disagreement Resolution Strategy

We employ a tiered strategy to resolve disagreements between the two initial annotators:

1. **Deterministic Verification (for Facts):** Disagreements on extracted values (e.g., dates, numbers) are resolved by a third expert checking the raw text. This is treated as an objective truth problem.

2. **Strict-Recall Principle (for Dependency):** For ambiguous reasoning cases (e.g., whether "Call Mom" implies a specific number), we apply the *Strict-Recall Principle*. If the tool API requires a specific value (e.g., a phone number string) that is not in the query, it is marked as **MEMORY-DEPENDENT**, even if the intent seems obvious.

3. **Automated Leakage Check:** We utilize a rule-based filter as an auxiliary judge. If a query contains exact string matches for memory parameters, it is automatically flagged as **LEAKAGE**, overriding human oversight errors.

Error Type	Memory Context	Generated Query (Q)	Target Param	Verdict
Direct Leakage	User wants to visit *Seattle*.	"Help me book a flight to **Seattle**."	`city="Seattle"`	Reject (Rule: Exact Match)
Partial Leakage	Account ID is *AX-9920*.	"Check my account status, it ends in **9920**."	`id="AX-9920"`	Reject (Rule: Token Split)
Solvable w/o Memory	User prefers *Italian* food.	"Find me a place that serves **pasta and pizza**."	`cuisine="Italian"`	Reject (Discriminator: Semantic Leak)
Hallucination	(No mention of time).	"Book the ticket for **tomorrow morning**."	`time="09:00"`	Reject (Discriminator: Invalid Constraint)
Valid Instance	User mentioned budget is *$500* in Turn 1.	"Find a flight that fits **the budget I mentioned earlier**."	`price=500`	Accept

Table 8: Examples of the filtering process. **Direct/Partial Leakage** is caught by the rule-based filter . **Solvable/Hallucination** errors are caught by the LLM Discriminator. Only queries that strictly require memory resolution are accepted.

Dialogue Context & Memory State	Pipeline Action / Result	Verdict & Rationale
Case: Preference Update History: "I prefer aisle seats." New Turn: "Actually, change that to window."	**Action:** `aisle` $\rightarrow$ `discarded` **Result:** `seat_pref: window`	**VALID.** Explicit user correction requires overwriting the previous state (Temporal Obsolescence).
Case: Scope Distinction History: "Budget for hotel is $200." New Turn: "Budget for flight is $500."	**Action:** `Merge attempt` **Result:** `budget: $500`	**INVALID.** Different entity scopes (Hotel vs. Flight). Both facts must be retained as separate entries.
Case: Specificity Refinement History: "I want Asian food." New Turn: "Let's go for Sushi."	**Action:** `Coexist` **Result:** `cuisine: Asian, dish: Sushi`	**VALID.** The new fact refines the old one without contradiction. Both are useful for tool retrieval.

Table 9: Guidelines for validating **Conflict Resolution**. Annotators must determine if the pipeline correctly identified whether to overwrite, merge, or keep facts based on logical consistency.

D Prompt Templates

Prompt for Fact Extraction

```
You are an expert information extraction system specializing in processing dialogues for fact
    clustering. Your primary goal is to extract key factual statements, events, and plans from
    the provided text.

Your output must be strictly a JSON array of objects. Do not include any other text, explanations
    , or formatting outside of the JSON array.

Each object in the array must contain exactly these three keys:

1. **"attribute"**: This is the **clustering key**. It must be a **specific, normalized category
    label** that includes the main topic and a key entity or detail. This key is used to group
    similar, specific facts.

   * *Example 1*: For the text "Hello, I am going to travel to Japan", the `attribute` should be
     **'Travel Planning (Japan)'**.
   * *Example 2*: For "I need help setting up my Azure account", the `attribute` should be **'
    Account Setup (Azure)'**.
2. **"facts"**: This is a concise description of the specific fact, event, or statement.

   * *Example*: For the text "Hello, I am going to travel to Japan", the `facts` description
    should be **'User is planning to travel to Japan'**.
3. **"source\_text"**: This is the exact dialogue snippet from the original text that supports
    the extracted fact.

   * *Example*: **'Hello, I am going to travel to Japan'**.

Key Instructions:

* **Attribute Normalization is Critical**: The `attribute` is the most important field. Strive
    for consistent, specific labels. The format should generally be `Topic (Entity)` or `Topic (
    Detail)`. (e.g., `Travel Planning (Japan)`, `Account Setup (Azure)`).
* **Extract Explicit Facts**: Only extract information that is explicitly stated. Do not infer or
     add information that is not present in the text.
* **Be Concise**: The `facts` description should be a clear, simple statement.
* **Focus on Substance**: Ignore generic conversational phrases (e.g., "hello", "thank you", "how
     are you") and only extract substantive facts, plans, or statements.
* **Multiple Facts**: There may be multiple facts. Please carefully extract all of them.
* **JSON Only**: The output must be a valid JSON list and nothing else.

Output example:
[
    \{\{
        "attribute": "Topic (Detail)",
        "facts": "A concise description of the fact.",
        "source_text": "The exact source text snippet."
    \}\}
]

---
Now, process the following text:
{text_content}
```

Prompt for Tool Construction

```
Extract parameter values from user memory to construct a tool call.

Tool Name: {tool.get('name')}
Tool Description: {tool.get('description', '')}

Complete Parameter Schema:
{json.dumps(tool_params_schema, indent=2, ensure_ascii=False)}

User Memory (extract values from here):
{json.dumps(memory_chain, indent=2, ensure_ascii=False)}

CRITICAL REQUIREMENTS:
1. At least one parameter values MUST come from the memory above (Explicit or Inferred).
2. Match parameter types exactly (string/integer/number/boolean/array/object).
3. For nested structures (arrays of objects), follow the schema's items/properties definitions.
4. All required parameters must be filled.
5. PREFER ORIGINAL TEXT: If a parameter can be found in memory, USE THE EXACT ORIGINAL TEXT from
     the memory.

6. PARAMETER SOURCES:
   - Memory (Explicit): Value appears verbatim in memory (e.g., "ID is 123" -> id="123").
   - Memory (Inferred): Value is inferred from context (e.g., "visiting Eiffel Tower" -> city="
     Paris").
   - Schema (Default): Value comes from the tool schema's default value if not found in memory.
   - Multi-hop: Parameters come from multiple different facts.

7. GROUNDING INFO:
   - For each extracted parameter, specify:
     * "source_text": The text in memory used for extraction (or "default value").
     * "type": "explicit", "inferred", or "default".

8. SOURCE IDENTIFICATION:
   - Identify which specific memory items (by their 'source_id') were necessary to derive the
     parameters.
   - List ALL source_ids that contributed to the parameters (including inferred ones).
   - If a parameter comes from a default value, do not include a source_id for it unless it was
     verified against memory.

Output JSON (follow this structure):
{json.dumps(example_output, indent=2, ensure_ascii=False)}
```

Prompt for Reverse Query Generator

```
You are generating a challenging, memory-dependent user query for testing an AI agent's long-term
     memory capabilities.

Context:
- The assistant has access to various tools and can call them to complete user requests.
- You are given a specific tool call that the assistant will make, along with the user's memory
    context.
- Your task is to generate a natural user query that would REQUIRE the assistant to call exactly
    this tool, but crucially, the user MUST OMIT key details that are already present in their
    memory/history.

Tool Call (the assistant will execute this):{json.dumps(tool_call, indent=2, ensure_ascii=False)}

User's Memory Context (History/Preferences):{memory_text}

Requirements for the generated query:
1. MUST BE TOOL-DEPENDENT
- The query CANNOT be answered using the model's parametric knowledge alone.
- It MUST require accessing external data or performing operations.
2. STRICTLY IMPLICIT (NO PARAMETER LEAKAGE)
- You MUST OMIT parameter values that are already established in the memory context.
- If the memory contains the destination "Dallas", the query MUST NOT say "Dallas". It should say
     "my destination" or "there".
- If the memory contains the date "March 13th", the query MUST NOT say "March 13th". It should
    say "that day" or "the date we discussed".
- The query MUST be ambiguous without the memory, but clear with the memory.
3. PROVIDE DOMAIN CONTEXT (CRITICAL)
- While avoiding specific parameter values, you MUST include enough semantic context (topic,
    category, or action nature) so the user knows WHICH memory thread is being referred to.
- AVOID purely generic pronouns like "it", "that", "those numbers" without any category.
- BAD: "Can you check the numbers?" (Too vague, could be anything)
- GOOD: "Can you check the atmospheric numbers?" (Better, domain is clear, but specific gas is
    hidden)
- GOOD: "How is the air quality data looking?" (Good, implies the topic without stating "Nitrous
    Oxide")
- BAD: "Book it." (Too vague)
- GOOD: "Book that flight." (Better, domain is travel)
4. REFLECT TOOL SPECIFICS (DISAMBIGUATION)
- If the tool requires a specific type of identifier (e.g., SecUID vs Username), the query should
     imply that specific type without stating the value.
- Example: If the tool uses 'SecUID', the query should say "using the secure ID I gave you"
    rather than just "my account".
- This ensures the query logically leads to the specific tool selected.
5. NATURAL AND CONVERSATIONAL
- Use phrases like "as usual", "like I said before", "for my trip", "book it", "check that thing
    ".
- Make it sound like a continuing conversation or a user with a long history.
6. NO SYSTEM INTERNALS
- Do NOT mention "tool", "function", "API", "call", "parameter".

Output Format:
Return ONLY a valid JSON object with one field:
{{
"query": "the generated user query here"
}}

Do not include any explanations, comments, or text outside this JSON object.
```