🎯 AIM: Constructing Adaptive Structured Memory via Agentic Information Management

Anonymous ACL submission

Abstract

Long-term memory remains a critical bottleneck for LLM agents in managing historical user interactions and task-related resources. Existing approaches typically rely on flat, predefined memory structures that lack adaptability and sophisticated reasoning capabilities. To address this, we propose **Agentic Information Management (AIM)**, a novel framework that leverages the LLM's inherent code generation and reasoning abilities to adaptively construct and manage structured memory. Unlike existing summarize-then-retrieve paradigms, AIM regards agent memory as an information management problem. AIM employs an agent to adaptively organize heterogeneous sources into well-defined structured representations. Furthermore, recognizing the limitations of existing memory benchmarks, which often use non-agentic, LLM-generated dialogues, we introduce **VibeCodeMem**, a high-quality, human-annotated long-horizon QA benchmark. It spans realistic working sessions with a coding agent across four software applications and 16 diverse features. Experiments using both open-source and proprietary LLMs demonstrate that AIM consistently outperforms strong baselines, achieving a **10.53%** absolute improvement on LoCoMo and **2.07%** on VibeCodeMem with the `Claude-4.5-haiku` model. Our results highlight the promise of agentic, self-structured memory for long-term agent capabilities. Our code and data will be released upon acceptance.

1 Introduction

Large language model (LLM)-based agents have demonstrated strong capability across a wide range of tasks, from daily chat (Li et al., 2024) to natural language instructed software development (i.e., vibe coding (Sapkota et al., 2025)). With the increasing complexity of these long-horizon tasks, the historical logs of agents have been expanded rapidly, resulting in an urgent need to develop *long-term memory* in agents (Hu et al., 2025), where LLM agents require continuous management of vast amounts of diverse information from both the internal dialogues with users (Guo et al., 2023) and the external environments (Wang et al., 2024).

Existing memory systems generally apply a summarize-then-retrieve paradigm. They store summaries of dialogue chunks in a notebook and retrieve the top-k summaries at query time via an external dense retriever (Chhikara et al., 2025; Xu et al., 2025). However, as shown in Table 1, there are two main issues with these methods. First is the oversimplified memory structure. The flat storage, such as a playbook or a collection of notes, reduces the efficiency and effectiveness of retrieval, especially for scaling to larger logs. Second is the lack of adaptability in structure memory. They heavily rely on the predefined requirements to extract fixed types of information (e.g., time, location, episodic memory, and semantic memory (Wang et al., 2025)). By prescribing the structure ahead of time, it limits the ability to accommodate new tasks or heterogeneous data sources.

To this end, we propose Agentic Information Management (AIM), a training-free framework for agent memory management that adaptively construct and manage structured memory (§5.5). The core idea is reformulating agent memory from a text-retrieval perspective to a structured data management paradigm. The LLM agent acts as a database engineer that infers relational schemas from the input data (e.g., interaction sessions), extracts salient information as records, and continuously evolves these schemas as data sources change over time; all the while, the relational database (RDBMS) maintains and retrieves the resulting persistent memory (Figure 1).

Different from existing memory systems, which assume the memory structure is predefined by humans, AIM instead enables *the agent to redesign its own memory structure and determine: What information should be stored, how it should be or-*

	Storage	Org.	Auto	No-Ret.
MemGPT (Packer et al., 2023)	Playbook	Low	×	×
Mem0 (Chhikara et al., 2025)	Summaries	Low	×	×
A-Mem (Xu et al., 2025)	Notes	Medium	×	×
LightMem (Fang et al., 2025a)	Note points	Medium	×	×
MemoryOS (Li et al., 2025b)	Text Chunks	Medium	×	×
AIM (ours)	Relational DB	High	✓	✓

Table 1: Comparison between AIM and existing memory frameworks. Org. indicates the organization level of the memory structure. Ret. indicates the method does not include dense-retriever.

	Speakers	Age.	Hum.	#	Source
LoCoMo (Maharana et al., 2024)	User-User	×	✓	1680	Dial.
LongMemEval (Wu et al., 2024)	AI-AI	×	×	500	Dial.
PrefEval (Zhao et al., 2025)	User-AI	×	✓	1000	Dial.
MemoryBank (Zhong et al., 2024)	User-AI	×	✓	194	Dial.
VibeCodeMem (ours)	User-AI	✓	✓	250	Dial., tools, files

Table 2: Comparison between VibeCodeMem and existing memory benchmarks. Age. indicates the task is agentic. Hum. indicates that both the dataset source and questions are manually annotated. # is the number of samples in the dataset. Dial. indicates dialogues.

ganized, and how the representation should evolve over time. Besides, different from prevalent RAG-based retrieval, AIM avoids the uncertain performance of additional retrievers in the system, and naturally benefits from two advantages of RDBMS: 1) an agent can choose from broad structure coverage of DB systems, from single-table to complex schema; 2) scalability to durability, prevention of data loss, and fast access and execution.

AIM is designed to manage memory from heterogeneous data inputs. However, existing agent-memory benchmarks largely focus on conversation histories or LLM-simulated multi-turn inputs (Table 2). To better evaluate memory systems in realistic agentic settings, we propose **VibeCodeMem**, the first human-annotated dataset of vibe-coding sessions in four different software development environments: a video game, e-notebook, CLI, and writing assistant. VibeCodeMem contains diverse input sources, such as tools, files, debugging history, and user preferences. Four graduate-level annotators built 16 new features in the four applications and annotated 250 questions that span factual, temporal, and user preference questions from single- and multi-session sources.

We conduct extensive experiments on long memory datasets, including VibeCodeMem, and compare with two classes of strong baselines: state-of-the-art RAG approaches, and existing memory frameworks (e.g., Mem0 (Chhikara et al., 2025), A-Mem (Xu et al., 2025), LightMem (Fang et al., 2025a), and GAM (Yan et al., 2025)). AIM achieves a **10.53%** absolute improvement on LoCoMo (Maharana et al., 2024) and **2.07%** on VibeCodeMem with the `Claude-4.5-haiku`. Our analysis of the generated schema and query demonstrates the advantage of the agentic memory self-management with its strong SQL coding capability.

Our main contributions are: 1) We propose AIM, a simple but effective framework that surpasses state-of-the-art baselines; 2) We propose a dataset to evaluate AIM on real-world agentic tasks, leveraging vibe coding as a typical agentic scenario; 3) Our analysis further shows that AIM enables agents to autonomously construct effective memory schemas and perform multi-step reasoning through executable SQL queries.

2 Related Work

2.1 Memory for LLM Agents

Early agent memory systems maintain memory streams with sequential experience buffers, sometimes combined with controllers, reflection mechanisms, or hierarchical organization to improve memory selection (Liang et al., 2023; Park et al., 2023; Packer et al., 2023; Zhong et al., 2024; Salama et al., 2025; Fang et al., 2025b). Then, some works further explore dense retrieval, read-write memory, cache-inspired memory management, and self-controlled memory mechanisms for long-term information use (Zhong et al., 2024; Modarressi et al., 2023; Packer et al., 2023; Liang et al., 2023). Beyond sequential memory stores, recent work has investigated more structured representations of agent memory: tree-based, graph-based, and temporal knowledge-based approaches organize memories as interconnected units rather than independent text snippets (Rezazadeh et al., 2025; Chhikara et al., 2025; Rasmussen et al., 2025; Xu et al., 2025; Zhang et al., 2025). Another line of work decomposes memory into multiple functional components, such as episodic, semantic, working, and procedural memory, in order to support richer agent behaviors (Kang et al., 2025; Li et al., 2025a; Wang and Chen, 2025; Nan et al., 2025). Different from these works which have oversimplified, and predetermined structures (such as a tree), AIM adopts a relational database and allows the agents to decide the memory structure adaptively and dynamically, making memory access more flexible and controllable for structured task execution.

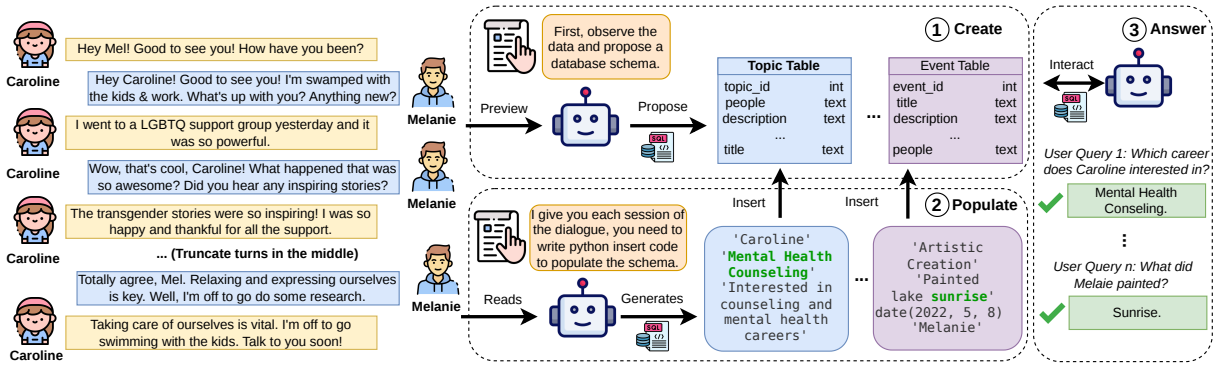


Figure 1: Overall framework of AIM. In Stage 1, the agent creates database schemas after previewing the raw data sources. In Stage 2, it generates entries from the dialogues and populates the self-constructed database. In Stage 3, the agent interacts with the database to answer user queries.

2.2 Retrieval-Augmented Generation

Retrieval-Augmented Generation (RAG) provides LLMs with non-parametric access to external information by retrieving relevant evidence and incorporating it into the generation context (Lewis et al., 2020; Borgeaud et al., 2022; Gao et al., 2023). Standard RAG pipelines typically split documents into chunks, index them with lexical or dense representations, retrieve relevant passages for a given query, and condition generation on the retrieved text (Yu et al., 2023; Wang et al., 2023). Subsequent work improves this pipeline through query rewriting, reranking, context filtering, retrieval-aware training, and other pre- or post-retrieval techniques (Lin et al., 2023; Ilin, 2023). More recent work extends RAG from a fixed retrieval pipeline to an agentic process. In agentic RAG, models can decide whether retrieval is needed, generate or revise search queries, evaluate intermediate evidence, and interleave retrieval with multi-step reasoning (Asai et al., 2023; Jiang et al., 2023; Trivedi et al., 2022; Shao et al., 2023). Differently, AIM does not utilize a RAG retriever to retrieve from the histories. Rather than retrieving semantically similar text chunks, AIM stores task-relevant information in relational tables and lets the agent issue SQL queries. This shifts memory access from passage retrieval to symbolic querying.

2.3 Memory Benchmarks

Recent agent memory benchmarks move closer to agent memory by evaluating information accumulated over conversations or episodes. LoCoMo (Maharana et al., 2024) and LongMemEval (Wu et al., 2025) assess whether models can answer questions based on long-term conversational histories. MemoryBench (Ai et al., 2025) and EvoMem (Wei et al., 2025) further study memory retention, preference following, continual learning, and self-evolving memory behavior. These benchmarks capture important aspects of long-term memory because the relevant information is distributed across previous interactions rather than provided in a single input. Different from them, which rely heavily on the synthetic data generated by LLMs or extracted from external resources, VibeCodeMem focuses on the agentic tasks with real-world annotations.

3 Agentic Information Management

Agentic Information Management (AIM) follows classic data management principles through an agentic lens. The AIM agent treats e.g., dialogue history simply as a data source: it analyzes the data, organizes it into a relational schema, and populates a relational database. This gives the agent freedom to dynamically decide how to model, construct, and access memory. Figure 1 shows the overview of AIM, comprising schema initialization, population, and querying. The full algorithm is listed in Algorithm 1.

3.1 Stage 1: Initialize Schema

We build on top of relational SQL databases because they can express a broad range of existing agent memory representations, from free-text notes to knowledge graphs to relational databases. In addition, considerable research has been devoted to improving LLM interactions with SQL databases (Lei et al., 2025). Thus, we introduce a simple agent that automatically constructs schemas by interacting with a relational database via SQL code. Given a set of input documents or dialogue

sessions $D = \{D_1, D_2, \cdots, D_n\}$, the agent randomly reads l chunks in the source $D_{ob} \in D$ and proposes a schema over it:

$$\text{Schema} \leftarrow \text{Agent}\big(\text{SampleChunks}(D, l),\ I_{\text{GenerateSchema}}\big)$$

where I_x is instruction text for x (all prompts listed in Table 7). The Schema is then instantiated using SQL.

3.2 Stage 2: Populate Database

While stage 1 proposed an initial schema to model the data source, populating the schema with records extracted from each data source D_i may also require updating the schema. Thus, D_i follows 3 subtasks. **Schema Update**: the agent decides whether the current schema (tables, column names, data types, constraints) needs to be updated to store the information in D_i. The updates are *additive*, so the schema changes do not invalidate or lose past data. **Conflict Resolution**: in case the new records already exist, the agent is asked to include update logic if needed. **Entry Extraction**: the agent autonomously extracts the useful information—entities like speakers or events, preferences, and relationships—as records that are inserted via SQL statements:

$$\text{Schema}' \leftarrow \text{Agent}(\text{GetInfo}(\text{Schema}), D_i, I_{\text{UpdateSchema}}),$$
$$\text{Database} \leftarrow \text{Agent}(D_i, \text{Schema}', I_{\text{CheckConflict}}, I_{\text{GenerateEntry}}).$$

To handle errors, the agent can `retry` up to K_r times. Each time, it reflects on the execution logs, revises the code, and executes again. This reduced the error rate from $30\% \rightarrow 0\%$ when using Qwen.

3.3 Stage 3: Generate & Answer Task

Rather than a RAG retrieval interface, AIM exposes a Text2SQL interface to the executor agent that answers task Q. In our implementation, the executor is provided the database schema, and can iteratively execute up to K_e SQL queries. The results of all SQL queries are used to generate the task's answer:

$$\text{Result} \leftarrow \text{Executor}^{K_e}\big(Q, \text{DB}, I_{\text{GenerateQuery}}\big),$$
$$\text{Answer} \leftarrow \text{Executor}\big(\text{Result}, I_{\text{AnswerFormat}}\big)$$

Algorithm 1 Agentic Information Management

Require: Unstructured data $D = \{D_1, \ldots, D_n\}$, query Q, executor agent $Executor$
Define S as Schema, DB as Database
Stage 1: Initialize Schema
$S \leftarrow Agent(\text{SampleChunks}(D, l), I_{\text{GenerateSchema}})$
Stage 2: Populate Database
for each $D_i \in D$ **do** // Expand schema if needed
  $S' \leftarrow Agent(\text{GetInfo}(S), D_i, I_{\text{UpdateSchema}})$
  $DB \leftarrow \text{retry}^{K_r}(Agent(D_i, S', I_{\text{CheckConflict}}, I_{\text{GenerateEntry}}))$
end for
Stage 3: Query Generation and Answering
$Results \leftarrow Executor^{K_e}(Q, DB, I_{\text{GenerateQuery}})$
$Answer \leftarrow Executor(Results, I_{\text{AnswerFormat}})$
return $Answer$

4 VibeCodeMem Benchmark

To evaluate a memory agent in the real-world and agent-centric task, we propose a new dataset, VibeCodeMem (Figure 2), developed based on two principles. **High Quality**: unlike previous work, we ensure each part of the data is human-generated for an important real-world application. **Challenging**: we created questions that are both difficult and that test the diverse capabilities of memory agents, such as cross-session questions and multi-hop questions, over millions of tokens and diverse data sources.

4.1 Data Collection and Annotation

Vibe Coding Dialogue Collection. We collect real-world interaction traces from four diverse repositories categorized into two scenarios. The first category includes Zoro (LLM-CLI tool) and DocAssist (a sandboxed Google Docs that provides LLM writing assistance), both of which are large-scale research code bases. The second category comprises two from-scratch projects: NoteSense (an e-notebook with Google Cal integration) and SnakeGame (A full-stack browser game) which are designed for personal use. Unlike existing synthetic benchmarks, these dialogues were generated organically using Cline with Claude 4.6 Sonnet model. Cline is an open-source coding agent widely adopted in professional development workflows. The developer steered the agent through under-specified, context-dependent instructions across long-horizon tasks, reflecting authentic vibe coding practices. The resulting sessions span 500 to 5,000 lines of history per task, covering complex UI logic, database and API integrations, and system-level AI functionality.

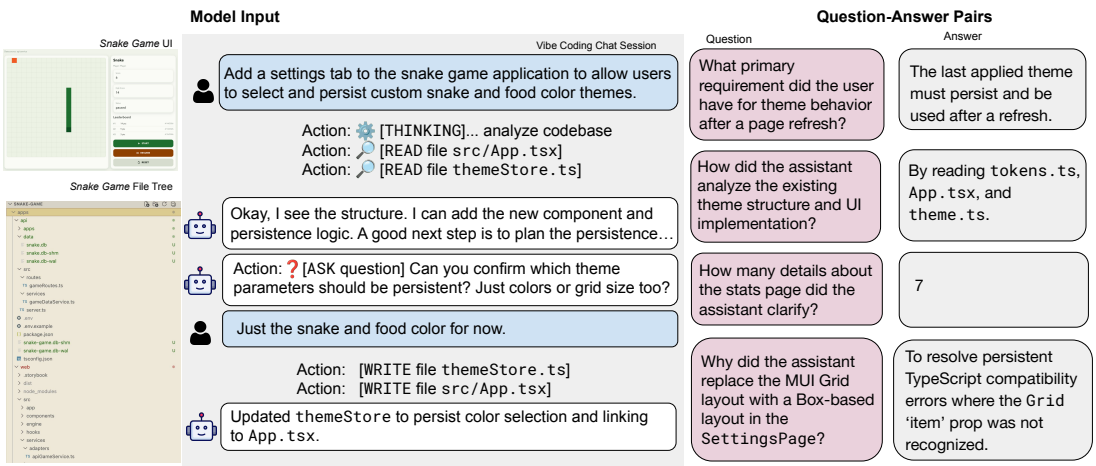


Figure 2: Overview of VibeCodeMem dataset.

Question Annotation. To ensure high evaluation standards, three graduate-level annotators followed a rigorous three-step protocol to curate the questions. First, they performed a comprehensive "trace-reading" phase to internalize the multi-session evolution of the code bases. Second, they designed challenging queries that require reasoning across both single-session and multi-session contexts, spanning user-specific coding preferences, temporal factual updates, and cross-file architectural constraints. Finally, for each question-answer pair, annotators were required to provide "ground-truth evidence"—specific snippets from the dialogue or database state that justify the answer.

4.2 Quality Controls

For annotator recruitment, we ensure that the annotators have Vibe coding and native English fluency. A pilot test is done for each annotator to judge if they are qualified to annotate samples. To ensure inter-annotator agreement, a cross-verification step was implemented where the author reviewed 10 examples generated by the different annotators and ensured the consistency of annotation question types and styles, resulting in a finalized set of 250 high-entropy data points. After annotation, the annotator who generated the vibe coding dialogue was asked to check the quality of each sample and leave comments for further review.

4.3 Data Statistics

Figure 2 shows the categories of the dataset. It covers diverse types of questions, including user preference, knowledge-facts, temporal, and open-domain questions. Notably, it contains challenging cross-session and multi-hop questions.

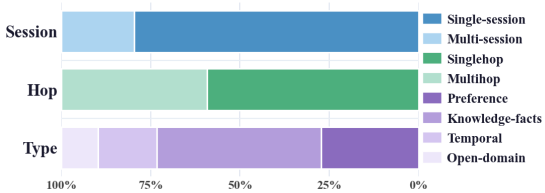


Figure 3: Statistics of different data categories.

Statistic	Snake	Note	Zoro	DocAssist	Overall
QA Pairs	100	100	30	20	250
Sessions	5	6	3	2	16
Max Q Tokens	25	19	17	19	25
Max A Tokens	34	31	27	31	34
Avg Tokens/Session	71k	69k	47k	284k	84k
Avg Turns	83	76.3	69	518	132.2
Avg Q Tokens	16.1	10.9	12	13.7	13.3
Avg A Tokens	16.7	10.4	10.6	11.7	13.1

Table 3: Dataset Statistics of VibeCodeMem.

5 Experiments and Results

5.1 Experiment Setting

Datasets. We employ the following benchmarks in our experimental studies. 1) LoCoMo. It is a widely used memory benchmark for conversational settings, designed to evaluate an agent's ability to maintain and recall information across extended multi-session dialogues. We adopt its single-hop, multi-hop, temporal-reasoning, and open-domain tasks in our experiments. 2) VibeCodeMem, the dataset we proposed for the evaluation of real-world agentic tasks.

Model	Method	LoCoMo										VibeCode.	
		Single Hop		Multi Hop		Temporal		Open Domain		Average		Average	
		F1	BLEU	F1	BLEU	F1	BLEU	F1	BLEU	F1	BLEU	F1	BLEU
Qwen3-32B	RAG-8k	29.48	24.58	20.67	18.38	23.16	20.22	16.28	15.53	22.40	19.68	13.17	11.13
	Mem0	29.18	23.08	13.64	11.17	25.13	20.64	10.10	7.40	19.51	15.58	13.19	9.93
	A-Mem	33.24	31.63	20.24	20.11	24.00	20.59	17.89	9.49	23.84	20.45	17.17	13.08
	LightMem	42.10	38.43	30.48	32.48	38.06	33.32	20.38	11.04	32.76	28.82	12.78	9.29
	GAM	41.67	40.75	26.40	29.04	43.23	39.17	20.17	11.35	32.87	30.08	17.01	13.16
	AIM	**43.70**	**41.59**	**31.40**	**32.89**	**45.41**	**41.03**	**25.61**	**13.15**	**36.53**	**32.17**	**19.34**	**14.36**
Claude Haiku	RAG-8k	29.13	25.61	18.26	17.92	28.55	27.18	14.66	9.18	22.65	19.97	20.19	17.35
	Mem0	27.40	26.88	23.83	25.22	23.53	20.49	17.11	5.95	22.97	19.64	22.29	18.64
	Mem0^g	26.51	23.35	24.85	26.18	27.77	23.79	11.28	5.10	22.61	19.58	–	–
	A-Mem	33.56	31.56	20.96	20.61	30.14	32.57	19.44	11.98	26.03	24.18	26.22	20.45
	LightMem	33.31	28.36	29.97	33.34	36.84	37.95	20.27	16.94	30.09	29.15	18.40	13.65
	AIM	**40.99**	**36.50**	**41.32**	**36.65**	**58.42**	**56.39**	**21.75**	**17.61**	**40.62**	**36.79**	**28.94**	**21.15**

Table 4: Performance comparison on LoCoMo, and VibeCodeMem.

Baselines. We consider the following baselines in our experiments. 1) Retrieval-augmented generation (RAG). For the RAG baseline, the input is uniformly partitioned into segments of 300 tokens, and the top-20 retrieved segments are used to perform the downstream task. 2) Memory-based methods, including A-Mem, Mem0, GAM and LightMem. These approaches construct specialized memory structures to store historical information, addressing memory-related tasks at runtime.

Implementation Details. In our experiments, we adopt `Claude-4.5-haiku` (Claude model for short) and `Qwen3-32B-A3B` (Qwen model for short) as the backbone models for both AIM and all baselines. Both LLMs offer a long-context window of 128K tokens. We use state-of-the-art model `bge-large-en-v1.5` (Xiao et al., 2023) as the default dense retriever. For AIM's detailed configuration, we set the maximum query steps to 3 and the maximum number of retries for coding to 3. The input context is segmented into 8k token pages for populating the data into the schema.

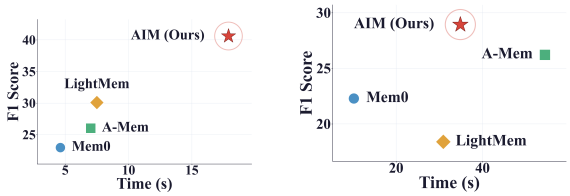
5.2 Overall Results

LoCoMo. Table 4 shows the overall results of our experiments. As can be seen, on the LoCoMo dataset, AIM outperforms all baselines with both Qwen and Claude models. Specifically, *Claude-4.5-Haiku surpasses previous state-of-the-art by a large margin (10.53% F1 and 7.64% BLEU scores), demonstrating the effectiveness of the proposed AIM*. Furthermore, Qwen 3 model outperforms baselines by 3.66% F1 and 2.09% BLEU scores, showing the generalizability of AIM on different families of models. It can be observed that the Claude model yields a larger improvement over the Qwen 3 model. Based on log analysis, we found that Claude has a lower coding error rate than Qwen 3, such as syntax errors, wrong memorization of schema, and instruction following. This reveals the strong coding capability of Claude and how it helps AIM to boost performance.

Among four types of questions, including single-hop, multi-hop, temporal, and open domain questions, AIM improves the baseline scores by a large margin for all types. Surprisingly, it improves the temporal scores by 21.58% F1 and 18.74% BLEU scores, showing a significant capability in processing time-sensitive questions. We further explored the reasons and found that databases are efficient for storing and leveraging time information. Agents can easily save and retrieve time-related information. This potentially boosts the performance of models. Besides, Claude also improves other types by a large margin, with around 10% gain on single-hop and multi-hop questions. *For the most difficult open domain questions, AIM can still outperform the baselines.*

Besides, we evaluate the Mem0^g performance as an additional comparison. Mem0^g adopts a graph database as storage format. As shown in the table, although some types of the question in LoCoMo are improved, the general accuracy still decreases. This means general GraphDB with Mem0 summarize-then-retrieve structure cannot obtain a significant improvement.

VibeCodeMem Table 4 also displays the performance of VibeCodeMem datasets on both models.



(a) LoCoMo (b) VibeCodeMem

Figure 4: Performance-efficiency comparison across memory methods.

Table	Representative Entry
Achievements	achievement_id: 1; user_id: 1; achievement_type: Dance Competition Win; achievement_description: Won first place in local dance competition with crew; achievement_date: 2022-01-20.
BusinessVentures	venture_id: 1; user_id: 1; business_type: Dance Studio; business_description: Dance studio – currently completing business plan and seeking investors; start_date: 2023-06-20; status: Full-time.
Travels	travel_id: 1; user_id: 1; destination: Paris; travel_date: 2023-01-28; mentioned_in_session: 2.

Table 5: Representative rows from selected AIM-generated tables(154) in the LoCoMo example.

	LoCoMo	
	F1	BLEU
AIM	**40.62**	**36.79**
w/ GraphDB	24.52	17.14
w/ JSON	30.13	21.58
w/o preview	33.15	30.17
w/o update	37.15	34.33

Table 6: Ablation study of AIM on LoCoMo dataset with Claude model.

First, the average scores of both models are less than 30%, demonstrating the challenge of the proposed dataset. Then, comparing AIM with all baselines, Qwen 3 beats the baseline by 2.33% F1 while Claude surpasses the strongest baselines by 2.72% F1. *This demonstrates that on agentic tasks with complex source documents and dialogues, AIM also yields the highest performance.*

It is worth noting that, for the Claude model, LightMem obtains the second-best performance on LoCoMo while yielding the lowest performance on VibeCodeMem. This demonstrates that the existing work is difficult to adapt to new domains due to its fixed memory design. AIM has persistent performance on both datasets, thanks to its adaptation capability to diverse sources.

5.3 Cost and Complexity

Figure 4 compares different methods in terms of time cost and performance. As shown in the figure, AIM achieves both higher performance and lower latency than A-Mem on the VibeCodeMem dataset. On the LoCoMo dataset, although AIM incurs slightly higher latency, it generally demonstrates a better or comparable performance–efficiency trade-off via calculating the value of performance divided by cost divided. The additional latency mainly comes from the database interaction interface provided to the executor agent, where the executor is allowed to perform multiple iterative operations. If the executor is restricted to a single interaction step, the latency on LoCoMo can be reduced by approximately 10 seconds.

5.4 Ablation Study

To test different design choices and evaluate the effectiveness of critical components, we conduct an ablation study on the LoCoMo dataset with the Claude model. As shown in Table 6, changing the relation database to other structures greatly hurts the performance, decreasing the performance from 40% to 20-30%. This shows that a relational database is more suitable for the LLMs to store memory, among other choices. We found that using GraphDB will lead to a higher error rate, so that the model cannot efficiently manage the database, while using JSON format makes it difficult to store the structural information and has difficulty in retrieving according to queries.

Besides, as shown in the table, if the agent does not preview the input source and directly generates the scheme, the performance drops to 33.15%, showing that the agent has specific schema designs for different dialogues that adapt to the data. If schemas, including the table and columns, are not updated during the data population, it also hurts the performance, degrading from 40.62% to 37.15%. This demonstrates that the schema updates can help the model tailor to new information.

5.5 Case Study on Self-Proposed Schemas

Figure 5 shows one database schema of AIM on the LoCoMo dataset. The schema contains nine relational tables, such as Achievements, Planned Activities, and Travels. Each table stores a different number of rows and is connected to other tables via foreign keys. Interestingly, these tables are self-designed by LLMs without any human guidance. To explore the content inside the tables, we further display some entries in the tables. As shown in Ta-

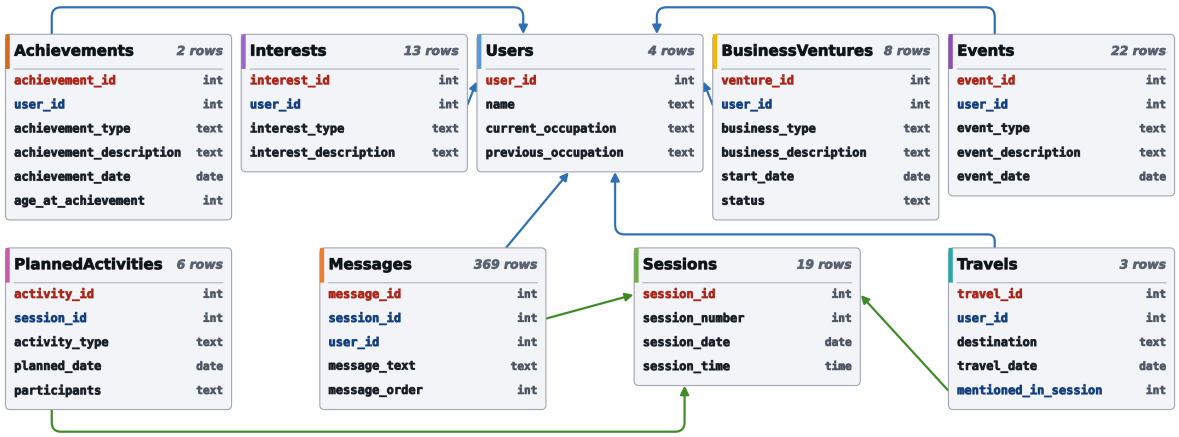


Figure 5: AIM database schema on the LoCoMo dataset, example 154.

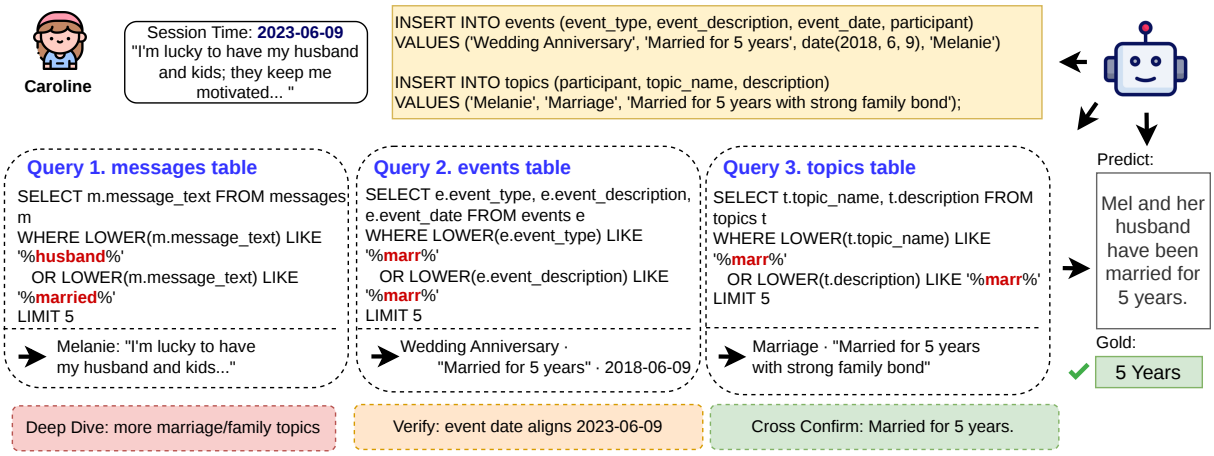


Figure 6: A case study of reasoning traces. The question is How long have Mel and her husband been married? and the gold answer is 5 years.

ble 5, these tables store valid information about the achievements and travels, including the person's ID, title, and detailed descriptions. By contrast, Figure 7 shows a schema created by agents for other dialogues. The schema structures are relatively simple, with only six tables, spanning basic events, sessions, etc. This demonstrates that the agents can adaptively and effectively change their designs of the schema and propose the schemas that best fit the needs.

5.6 Case Study on A Reasoning Trace

Figure 6 shows one example of how the dialogue is memorized and used for answering the question in AIM. As seen, the agent first reads the Caroline's dialogue, and stores the general information and the time it happens (session time). Then the agent saves the information in both event and topics. Next, in the stage 3, the agent needs to ask three questions before answering the question. The agent does not know what the database contains, so it first try to search the information about the husbands and marries in the database. After finding the useful information, the agent verifies the information by querying the events table, and found the marry date is aligned. Finally, the agent already knows the answer to the question, it still cross confirms the information from topic tables.

6 Conclusion

We propose AIM, a framework for long term agentic memory that utilizes the agent to adaptively construct the structured memory. Results demonstrate the strong improvement of AIM on both LoCoMo and a newly proposed VibeCodeMem datasets. Future work can further trains the agent to better construct the memory layer, and we believe future agents may autonomously construct and evolve their own information systems.

Limitations

Although AIM outperforms the baselines by a large margin, there are still many points that AIM can improve. First, we have not tested the performance of AIM on multi-modal tasks, which includes the memory of images or even videos. Second, AIM relies on the coding capabilities of the models, using SQL as a tool. Thus, it is important that the target model needs the strong capability of SQL writing to ensure the performance. Third, we have not tested on many real-world agentic tasks and VibeCodeMem only serves as a novel dataset and preliminary study on this direction. We call on more agentic task memory datasets to better evaluate the agent memory systems. Last but not least, AIM also relies on a DB systems, increasing the potential security issues such as SQL injection attack, or other pervasive attacks in DB domain.

References

Qingyao Ai, Yichen Tang, Changyue Wang, Jianming Long, Weihang Su, and Yiqun Liu. 2025. Memorybench: A benchmark for memory and continual learning in llm systems. *arXiv preprint arXiv:2510.17281*.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2023. Self-rag: Learning to retrieve, generate, and critique through self-reflection. *arXiv preprint arXiv:2310.11511*.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, and 1 others. 2022. Improving language models by retrieving from trillions of tokens. In *International conference on machine learning*, pages 2206–2240. PMLR.

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

Jizhan Fang, Xinle Deng, Haoming Xu, Ziyan Jiang, Yuqi Tang, Ziwen Xu, Shumin Deng, Yunzhi Yao, Mengru Wang, Shuofei Qiao, and 1 others. 2025a. Lightmem: Lightweight and efficient memory-augmented generation. *arXiv preprint arXiv:2510.18866*.

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. 2025b. Memp: Exploring agent procedural memory. *Preprint*, arXiv:2508.06433.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*.

Jing Guo, Nan Li, Jianchuan Qi, Hang Yang, Ruiqiao Li, Yuzhen Feng, Si Zhang, and Ming Xu. 2023. Empowering working memory for large language model agents. *arXiv preprint arXiv:2312.17259*.

Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, and 1 others. 2025. Memory in the age of ai agents. *arXiv preprint arXiv:2512.13564*.

I. Ilin. 2023. Advanced rag techniques: An illustrated overview.

Zhengbao Jiang, Frank F Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Active retrieval augmented generation. *arXiv preprint arXiv:2305.06983*.

Jiazheng Kang, Mingming Ji, Zhe Zhao, and Ting Bai. 2025. Memory os of ai agent. *ArXiv*, abs/2506.06326.

Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, and 1 others. 2025. Spider 2.0: Evaluating language models on real-world enterprise text-to-sql workflows. In *International Conference on Learning Representations*, volume 2025, pages 28691–28735.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, and 1 others. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474.

Yuanchun Li, Hao Wen, Weijun Wang, Xiangyu Li, Yizhen Yuan, Guohong Liu, Jiacheng Liu, Wenxing Xu, Xiang Wang, Yi Sun, and 1 others. 2024. Personal llm agents: Insights and survey about the capability, efficiency and security. *arXiv preprint arXiv:2401.05459*.

Zhiyu Li, Shichao Song, Hanyu Wang, Simin Niu, Ding Chen, Jiawei Yang, Chenyang Xi, Huayi Lai, Jihao Zhao, Yezhaohui Wang, Junpeng Ren, Zehao Lin, Jiahao Huo, Tianyi Chen, Kai Chen, Ke-Rong Li, Zhiqiang Yin, Qingchen Yu, Bo Tang, and 3 others. 2025a. Memos: An operating system for memory-augmented generation (mag) in large language models. *ArXiv*, abs/2505.22101.

Zhiyu Li, Chenyang Xi, Chunyu Li, Ding Chen, Boyu Chen, Shichao Song, Simin Niu, Hanyu Wang, Jiawei Yang, Chen Tang, and 1 others. 2025b. Memos: A memory os for ai system. *arXiv preprint arXiv:2507.03724*.

Xinnian Liang, Bing Wang, Huijia Huang, Shuangzhi Wu, Peihao Wu, Lu Lu, Zejun Ma, and Zhoujun Li. 2023. Scm: Enhancing large language model with self-controlled memory framework.

Xi Victoria Lin, Xilun Chen, Mingda Chen, Weijia Shi, Maria Lomeli, Rich James, Pedro Rodriguez, Jacob Kahn, Gergely Szilvasy, Mike Lewis, and 1 others. 2023. Ra-dit: Retrieval-augmented dual instruction tuning. *arXiv preprint arXiv:2310.01352*.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. 2024. Evaluating very long-term conversational memory of llm agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13851–13870.

Ali Modarressi, Ayyoob Imani, Mohsen Fayyaz, and Hinrich Schütze. 2023. Ret-llm: Towards a general read-write memory for large language models. *arXiv preprint arXiv:2305.14322*.

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

Charles Packer, Vivian Fang, Shishir_G Patil, Kevin Lin, Sarah Wooders, and Joseph_E Gonzalez. 2023. Memgpt: towards llms as operating systems.

Joon Sung Park, Joseph C. 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, UIST 2023, San Francisco, CA, USA, 29 October 2023- 1 November 2023*, pages 2:1–2:22. ACM.

Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. 2025. Zep: a temporal knowledge graph architecture for agent memory. *arXiv preprint arXiv:2501.13956*.

Alireza Rezazadeh, Zichao Li, Wei Wei, and Yujia Bao. 2025. From isolated conversations to hierarchical schemas: Dynamic tree memory representation for llms. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

Rana Salama, Jason Cai, Michelle Yuan, Anna Currey, Monica Sunkara, Yi Zhang, and Yassine Benajiba. 2025. Meminsight: Autonomous memory augmentation for llm agents. *ArXiv*, abs/2503.21760.

Ranjan Sapkota, Konstantinos I Roumeliotis, and Manoj Karkee. 2025. Vibe coding vs. agentic coding: Fundamentals and practical implications of agentic ai. *arXiv preprint arXiv:2505.19443*.

Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. 2023. Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy. *arXiv preprint arXiv:2305.15294*.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. *arXiv preprint arXiv:2212.10509*.

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

Yu Wang, Ryuichi Takanobu, Zhiqi Liang, Yuzhen Mao, Yuanzhe Hu, Julian McAuley, and Xiaojian Wu. 2025. Mem-{\alpha}: Learning memory construction via reinforcement learning. *arXiv preprint arXiv:2509.25911*.

Zhiruo Wang, Jun Araki, Zhengbao Jiang, Md Rizwan Parvez, and Graham Neubig. 2023. Learning to filter context for retrieval-augmented generation. *arXiv preprint arXiv:2311.08377*.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. 2024. Agent workflow memory. *arXiv preprint arXiv:2409.07429*.

Tianxin Wei, Noveen Sachdeva, Benjamin Coleman, Zhankui He, Yuanchen Bei, Xuying Ning, Mengting Ai, Yunzhe Li, Jingrui He, Ed H Chi, and 1 others. 2025. Evo-memory: Benchmarking llm agent test-time learning with self-evolving memory. *arXiv preprint arXiv:2511.20857*.

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*.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. 2025. Longmemeval: Benchmarking chat assistants on long-term interactive memory. In *The Thirteenth International Conference on Learning Representations*.

Shitao Xiao, Zheng Liu, Peitian Zhang, and Niklas Muennighoff. 2023. C-pack: Packaged resources to advance general chinese embedding. *Preprint*, arXiv:2309.07597.

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

BY Yan, Chaofan Li, Hongjin Qian, Shuqi Lu, and Zheng Liu. 2025. General agentic memory via deep research. *arXiv preprint arXiv:2511.18423*.

Wenhao Yu, Hongming Zhang, Xiaoman Pan, Kaixin Ma, Hongwei Wang, and Dong Yu. 2023. Chain-of-note: Enhancing robustness in retrieval-augmented language models. *arXiv preprint arXiv:2311.09210*.

Guibin Zhang, Muxin Fu, Guancheng Wan, Miao Yu, Kun Wang, and Shuicheng Yan. 2025. G-memory: Tracing hierarchical memory for multi-agent systems. *arXiv preprint arXiv:2506.07398*.

Siyan Zhao, Mingyi Hong, Yang Liu, Devamanyu Hazarika, and Kaixiang Lin. 2025. Do llms recognize your preferences? evaluating personalized preference following in llms. *arXiv preprint arXiv:2502.09597*.

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 Other AIM Cases

Figure 7 shows the schema created by Claude model on LoCoMo. Figure 8 demonstrates the reasoning trace for another sample on LoCoMo dataset.

B Prompt Used in AIM

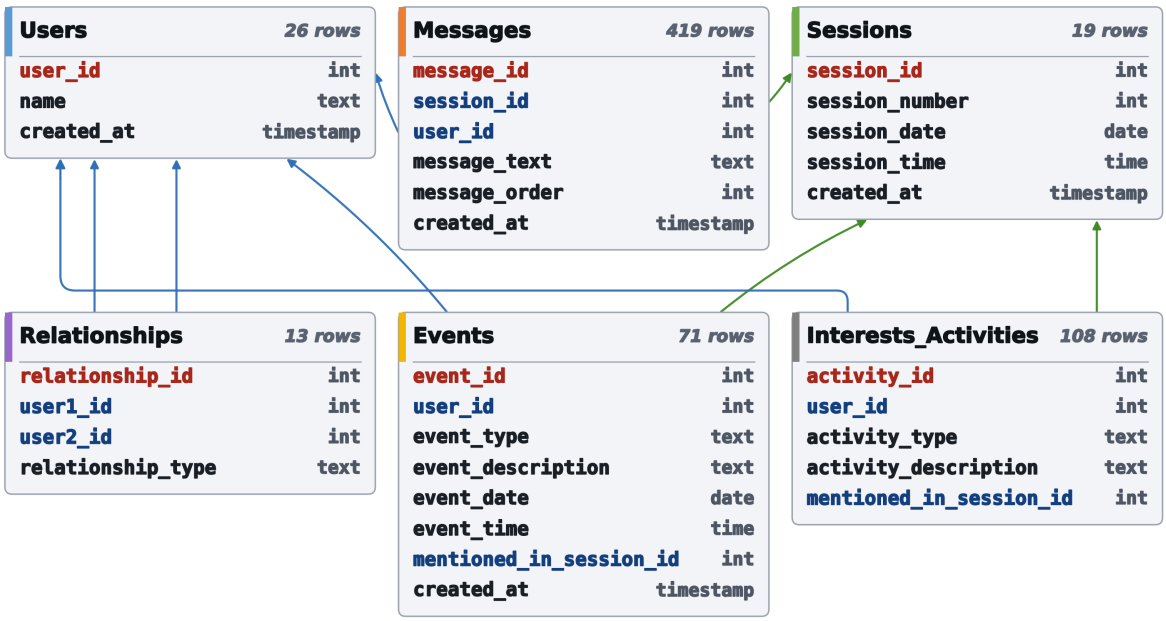


Figure 7: AIM database schema on the LoCoMo dataset.

Instruction Name	Prompt
$I_{\text{GenerateSchema}}$	You will answer the questions based on the long dialogue sessions. You need to build a SQL database and then store the information about the dialogue into the database. First, observe the data and propose a database schema. Please write python code to build the schema. This step does not require inserting the session; just build the empty database. Please write the python code directly and return the code in the format "‘python ... "‘, so that I can execute the code for you. The name for creating the database is `dialogue_database.db`.
$I_{\text{UpdateSchema}}$	You can modify the schema, such as adding tables if you think there should be more tables to store the information.
$I_{\text{CheckConflict}}$	In case there are duplicated IDs, do not hard-code table IDs; use ? instead. Check whether the inserted entry is already in the database.
$I_{\text{GenerateEntry}}$	Above, I give you each session of the dialogue. You need to write Python insert code to populate the schema. Please write the python code directly and return the code in the format "‘python ... "‘.
$I_{\text{GenerateQuery}}$	The question is below. Please write SQL queries in Python code to answer the question. You can try 3 iterations, and each round can contain 1–3 queries. Please write the python code and return the code in the format "‘python ... "‘. The name of the database is `dialogue_database.db`. Also generate the found information in plain text.
$I_{\text{AnswerFormat}}$	Your final answer to the question is? Answer according to the execution returns; do not write Python code at this stage. We are calculating F1 score with the gold label, so the answer should be a simple answer in the format: `### {Answer}`.

Table 7: Prompts used in different instruction stages.

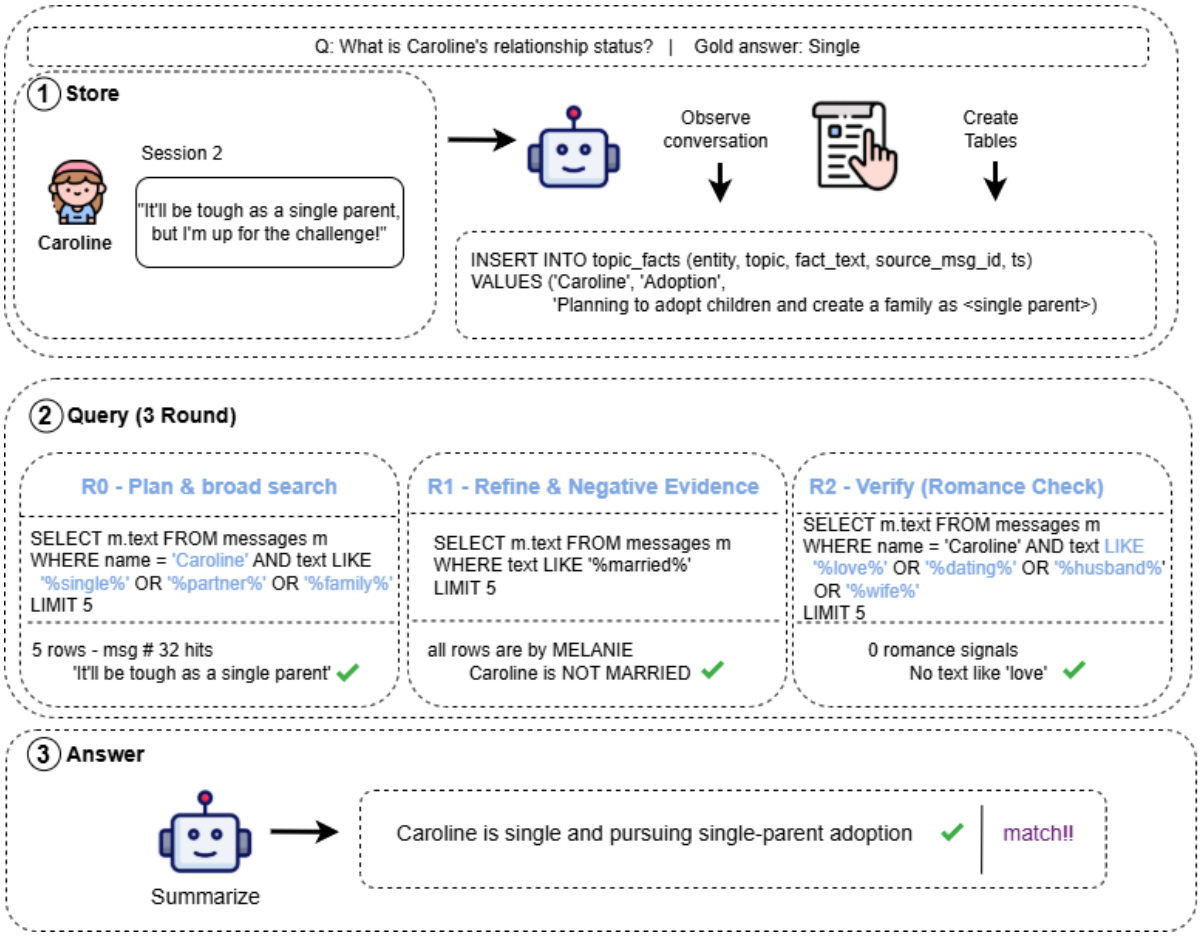


Figure 8: A reasoning trace on LoCoMo, generated by Claude model.