MAPLE: Multi-Agent Adaptive Planning with Long-Term Memory for Table Reasoning

Ye Bai[1], Minghan Wang[1], Thuy-Trang Vu[1]
[1]Department of Data Science & AI, Monash University
`{firstname.lastname}`@monash.edu

Abstract

Table-based question answering requires complex reasoning capabilities that current LLMs struggle to achieve with single-pass inference. Existing approaches, such as Chain-of-Thought reasoning and question decomposition, lack error detection mechanisms and discard problem-solving experiences, contrasting sharply with how humans tackle such problems. In this paper, we propose MAPLE (**M**ulti-agent **A**daptive **P**lanning with **L**ong-term m**E**mory), a novel framework that mimics human problem-solving through specialized cognitive agents working in a feedback-driven loop. MAPLE integrates 4 key components: (1) a Solver using the ReAct paradigm for reasoning, (2) a Checker for answer verification, (3) a Reflector for error diagnosis and strategy correction, and (4) an Archiver managing long-term memory for experience reuse and evolution. Experiments on WikiTQ and TabFact demonstrate significant improvements over existing methods, achieving state-of-the-art performance across multiple LLM backbones.[1]

1 Introduction

Tables represent one of the most prevalent forms of semi-structured data, organizing information systematically across domains ranging from scientific research to business analytics (Dong and Wang, 2024). However, answering questions over tables presents unique challenges, requiring multi-step reasoning over structured data, recognition of implicit relationships between cells, and precise contextual interpretation (Lu et al., 2025). These challenges make table-based question answering (QA) particularly difficult for Large Language Models (LLMs), as they must navigate tabular data structure while performing sophisticated reasoning to derive accurate answers, capabilities that current LLMs struggle to achieve with single-pass inference.

Existing table reasoning frameworks exhibit several limitations. Single-forward-pass methods (Cheng et al., 2023; Ye et al., 2023) lack error detection mechanisms, allowing mistakes to propagate through solutions. ReAct-based approaches (Wang et al., 2024b; Zhang et al., 2023) provide environmental feedback but lack systematic verification. On the other hand, multi-agent approaches primarily focus on output refinement rather than comprehensive reasoning improvement (Ye et al., 2023; Yu et al., 2025b). Additionally, current systems discard problem-solving experiences after completion, preventing transferable knowledge accumulation across tasks. It contrasts with human problem-solving: **when tackling complex tabular problems, humans methodically work through solutions, verify results, reflect on mistakes, and accumulate experiences for future strategies.**

To address these limitations, we propose MAPLE (**M**ulti-agent **A**daptive **P**lanning with **L**ong-term m**E**mory), a novel framework that mimics human problem-solving through specialized cognitive agents in a feedback-driven loop. As illustrated in Figure 1, MAPLE decomposes reasoning into distinct stages: reasoning, verification, reflection, and memory evolution, each managed by a dedicated agent. Our framework implements a feedback-driven cycle with the Solver conducting iterative reasoning, the Checker performing quality assessment, the Reflector diagnosing errors and suggesting improvements, and the Archiver managing long-term memory for cross-task learning. This architecture enables dynamic adaptation both within tasks and across similar problems, mirroring human cognitive processes.

Experiments on WikiTQ and TabFact demonstrate that MAPLE significantly outperforms existing methods across multiple LLM back-

---

[1] Dataset and code are available at `https://github.com/bettyandv/MAPLE-table-reasoning`

Proceedings of The 23rd Annual Workshop of the Australasian Language Technology Association, pages 132–158
November 26-28, 2025 ©2025 Association for Computational Linguistics

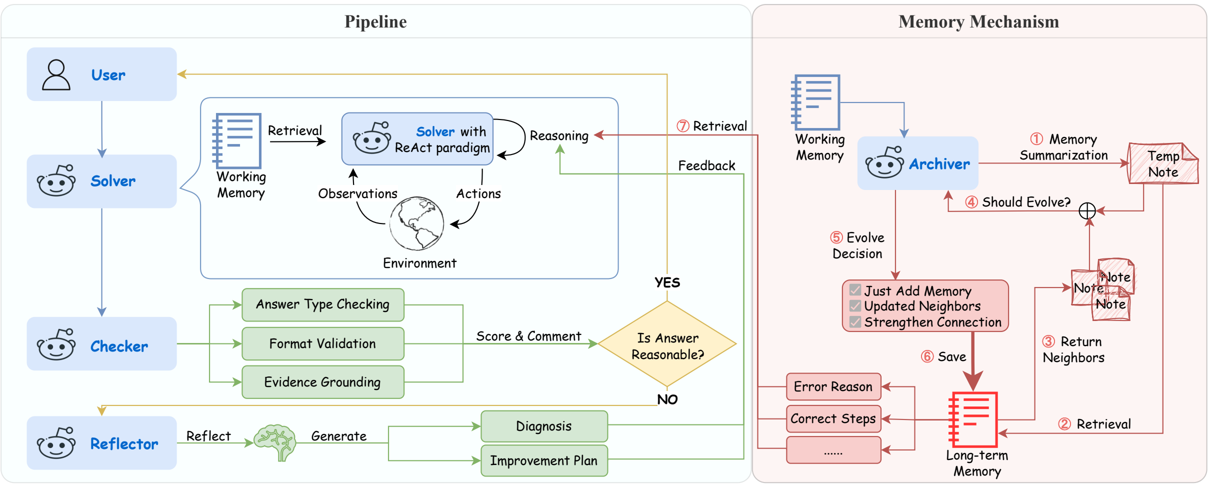


Figure 1: The MAPLE framework pipeline. 4 agents work collaboratively in a feedback loop: the Solver conducts iterative reasoning using ReAct, the Checker evaluates answer quality, the Reflector diagnoses errors and suggests improvements, and the Archiver manages an evolving long-term memory. This architecture enables dynamic adaptation both within tasks and across similar problems, mirroring human cognitive problem-solving processes.

bones. Ablation studies confirm each component substantially contributes to the framework's effectiveness. Our memory analysis reveals that logical reasoning errors and numerical operation failures account for nearly 80% of remaining challenges, providing valuable insights for future research and serving as both a performance enhancer and diagnostic tool.

Our contributions include: **(1)** a multi-agent framework implementing adaptive planning through feedback-driven reasoning; **(2)** a specialized verification and reflection mechanism providing targeted diagnostic feedback; **(3)** a structured long-term memory system that distills experiences and categorizes errors; and **(4)** state-of-the-art performance on WikiTQ and TabFact benchmarks. These innovations address fundamental limitations in current approaches, creating a system that mirrors human cognitive processes while improving performance on complex table reasoning tasks.

2 Related Work

Multi-agent System Multi-agent systems (MAS) have gained attention for leveraging collective intelligence in complex tasks. Current applications span problem-solving domains such as software development (Li et al., 2023a; Du et al., 2023; Qian et al., 2024; Huang et al., 2024), embodied robotic coordination (Dasgupta et al., 2023; Mandi et al., 2023; Zhang et al., 2024; Yu et al., 2025a), and scientific Debate (Du et al., 2023; Xiong et al., 2023; Tang et al., 2024). World simulation applications include social behavior modeling (Park et al., 2022; Gao et al., 2023; Chen et al., 2025), policy simulation (Xiao et al., 2023; Hua et al., 2024), economic forecasting (Horton, 2023; Li et al., 2023b; Zhao et al., 2024), and gaming (Light et al., 2023; Wang et al., 2023b; Xu et al., 2024). Several MAS frameworks, including MetaGPT (Hong et al., 2024), CAMEL (Li et al., 2023a), and AutoGen (Wu et al., 2023), have emerged to facilitate the implementation of these systems.

Memory Mechanisim for Agent Memory enables agents to perform coherent, context-aware reasoning across extended tasks. In multi-agent systems, memory serves as the cognitive foundation for retaining observations, decisions, and interaction histories—critical elements for consistent collaboration and adaptation over time (Sumers et al., 2024).

Memory formats determine how content is stored and utilized, with several prominent approaches emerging in recent work. Natural language representations offer semantic richness and interpretability, as demonstrated in Reflexion (Shinn et al., 2023), Voyager (Wang et al., 2023a), and Generative Agents (Park et al., 2023). Vector embeddings enable efficient similarity-based retrieval, a technique central to systems like MemoryBank (Zhong et al., 2023), A-MEM (Xu et al., 2025), and ChatDev (Qian et al., 2024). Meanwhile, structured formats support symbolic reasoning and precise queries, approaches adopted by ChatDB (Hu et al., 2023) and DB-GPT (Zhou et al., 2023b). Beyond storage formats, recent advances have introduced innovative management strategies, including complete interaction storage (Zhong et al., 2023; Modarressi et al., 2024), cache-like designs (Packer et al., 2024), and controller-based architectures (Wang et al., 2025) that dynamically prioritize and maintain relevant information during extended reasoning processes.

Despite these advances, few existing systems integrate all these memory dimensions within a coherent architecture specifically designed for complex reasoning tasks like table-based QA, where both structured knowledge and flexible retrieval are essential for effective performance.

Table Reasoning Research on table reasoning can be broadly classified into fine-tuning-based and prompting-based methods. Fine-tuning approaches like TAPAS (Herzig et al., 2020), Pasta (Gu et al., 2022), TUTA (Wang et al., 2021), and TAPEX (Liu et al., 2022) adapt pre-trained language models to encode table semantics through specialized training objectives. Other works focus on improving alignment between natural language queries and structured data (Eisenschlos et al., 2020; Jiang et al., 2022). Despite their effectiveness, these methods typically require extensive annotated data and feature static reasoning processes without adaptive correction mechanisms.

Prompting-based methods leverage LLMs with minimal training data requirements. Techniques like Chain-of-Thought (Wei et al., 2023), Least-to-Most (Zhou et al., 2023a) and Dater (Ye et al., 2023) perform reasoning by decomposing tasks into explicit steps. Table-specific adaptations include Binder (Cheng et al., 2023), Chain-of-Table (Wang et al., 2024b), ReAcTable (Zhang et al., 2023), and Table-Critic (Yu et al., 2025b), incorporating agent collaboration or ReAct-style reasoning.

Existing table reasoning methods exhibit significant limitations across key design dimensions. While some implement multi-agent architectures or ReAct-based reasoning, none integrates all critical components: dynamic planning, reflection mechanisms, self-refinement, and long-term memory. A detailed comparison of these methods is provided in Appendix A. Our proposed MAPLE framework addresses these gaps by combining collaborative verification, adaptive planning, and evolving memory structures to achieve more robust and accurate table reasoning.

3 MAPLE Framework

3.1 Overview

Inspired by human problem-solving processes, we propose MAPLE (**M**ulti-agent **A**daptive **P**lanning with **L**ong-term m**E**mory), a novel framework that addresses a critical limitation in existing systems: their inability to adapt, reflect, and learn from experience. As illustrated in Figure 1 and formalized in Algorithm 1, MAPLE decomposes reasoning into four distinct stages, each managed by a dedicated agent that fulfills a specialized cognitive function.

The Solver conducts progressive reasoning using the ReAct paradigm, enabling dynamic interaction with the table environment. The Checker performs multi-dimensional verification across answer type, format, and evidence grounding. The Reflector diagnoses reasoning errors and generates targeted improvement plans when verification fails. Finally, the Archiver manages long-term memory, facilitating experience reuse across similar problems. To more concretely illustrate the flow of our framework, we present a representative case study in Appendix D.

A key innovation in MAPLE is its feedback-driven, multi-round reasoning cycle that enables continuous refinement of reasoning strategies through deliberate planning and adaptation. This allows agents to correct errors and improve solutions through multiple attempts, **mimicking how humans iteratively improve their problem-solving approaches.**

Additionally, while existing systems discard reasoning experiences after completion, our approach implements selective integration and strategic evolution of memory. The system filters redundant experiences, distills valuable problem-solving patterns into structured notes, and evolves the memory base through semantic clustering. This integration of adaptive planning with evolving memory enables MAPLE to leverage past experiences, avoid repeated errors, and continuously improve reasoning capabilities across similar problem types.

3.2 Agent Roles

Table 1 summarizes the specialized responsibilities and input-output specifications of each agent in our framework. Below, we formally define how these agents interact within the MAPLE architecture to create a cohesive reasoning system that surpasses traditional single-pass approaches.

Agent	Input	Output	Function
Solver	$\mathcal{T}$ or $t', q, \tau, r, \mathcal{M}$	t' or a_s	Progressive reasoning with real-time environmental feedback
Checker	$\mathcal{T}, q, a_s$	$\mathcal{F}$	Verifies answer type, format and evidence grounding
Reflector	$\mathcal{T}, q, \tau, a_s, \mathcal{F}$	d, p	Diagnoses errors and generates targeted improvement plans
	$\mathcal{T}, q, a_m, a_g, \tau, d, p$	m	Distills experiences into structured memory notes
Archiver	$\mathcal{T}, q, m, \mathcal{M}_l, k, \delta$	$\mathcal{N}$	Retrieves contextually relevant experiences for current tasks
	$m, \mathcal{N}$	e	Evolves memory through semantic clustering and connection

Table 1: Overview of specialized agents in MAPLE. Each agent performs distinct cognitive functions with specific input-output patterns. This modular design allows for verification, reflection, and experience reuse across tasks.

3.2.1 Solver

The Solver agent ($\mathcal{S}$) leverages the **ReAct** paradigm (Yao et al., 2023) to establish genuine environmental interaction with tabular data. After each table operation, the Solver reassesses the environment to determine whether to perform additional manipulations or derive an answer. This iterative reasoning-acting cycle enables strategic adaptation based on real-time feedback from the manipulated table state.

Formally, the Solver operates according to the following process, given input $(I, q, \tau, r, \mathcal{M})$:

$$\lambda = \pi_{\text{solver}}(I, q, \tau, r, \mathcal{M}) \tag{1}$$

where $\lambda \in \{t', a_s\}$, which can either be the operated table t' (if the reasoning is not done yet) or the final answer (a_s) of the Solver, $I \in \{\mathcal{T}, t'\}$ represents the current environment (original table $\mathcal{T}$ or intermediate table t'), q is the question, τ denotes previous Solver operation history, and r is the remaining attempts.

A distinctive feature of our approach is the integration of dual memory systems. The memory input is defined as $\mathcal{M} = \{\mathcal{M}_w, \mathcal{M}_l\}$, where working memory $\mathcal{M}_w$ contains Reflector feedback (d, p) providing diagnostic insights when prior attempts failed. Long-term memory $\mathcal{M}_l$ retrieves relevant historical experiences ($\mathcal{N}_{\text{solver}}$), including similar questions, proven strategies and common pitfalls, creating a knowledge repository that enriches the reasoning process.

After each interaction, the updated environment I is fed back to the Solver, enabling continuous adaptation based on the evolving table state. Through this feedback loop, the Solver can progressively refine its understanding and approach until reaching a satisfactory answer. The complete prompt is provided in Appendix F.

3.2.2 Checker

The Checker agent ($\mathcal{C}$) introduces a critical verification layer based on structured feedback principles. Given table $\mathcal{T}$, question q, and Solver's answer a_s, the Checker evaluates the output according to three essential criteria:

1. **Answer Type Checking:** Evaluates whether the answer matches the expected type implied by the question. For instance, if a question asks "How many medals did the country win?", a numerical value like "5" is expected, not a country name like "USA".
2. **Format Validation:** Assesses conformity to prescribed formatting rules. If the expected output is a single numerical value ("24"), but the answer includes calculation steps ("4 × (1 + 2 + 3) = 24"), this violates format requirements that mandate only the final result.
3. **Evidence Grounding:** Verifies that the answer is properly supported by evidence in the table data. If a question asks "Which country won the most gold medals?" and the table only lists "USA", "China", and "Japan", an answer of "Germany" would violate evidence grounding as it does not appear in the table.

For each criterion $i \in \{\text{type}, \text{format}, \text{evidence}\}$, the Checker assigns a score $s_i \in \{0, 1, 2\}$ with an explanatory comment c_i, where 0 indicates the requirement is not met, 1 indicates it is partially met, and 2 indicates it is fully met or not applicable. Formally, this evaluation process is defined as:

$$\{(s_i, c_i)\}_{i=1}^{3} = \pi_{\text{checker}}(\mathcal{T}, q, a_s) \tag{2}$$

where π_{checker} represents the evaluation function mapping the input triplet to aspect-specific scores and comments. This structured feedback $\mathcal{F}$ includes a total score $s_{\text{total}} = \sum_{i=1}^{3} s_i$ and an aggregated summary, enabling the Reflector agent to diagnose errors and generate improvement strategies.

The Checker forms an integral component of the reasoning cycle, creating a feedback loop that drives continuous improvement. By systematically evaluating answers across multiple dimensions, it helps identify specific weaknesses in the reasoning process rather than merely flagging incorrect answers. The complete prompt is provided in Appendix F.

3.2.3 Reflector

The Reflector agent ($\mathcal{R}$) implements a metacognitive capability essential for advanced reasoning systems, analyzing failures, diagnosing root causes, and generating strategic corrections. This agent drives continuous improvement through deliberate adaptation, providing the critical link between error detection and strategy refinement.

Given the table $\mathcal{T}$, question q, Solver's reasoning trace τ, Solver's answer a_s, and Checker feedback $\mathcal{F}$, the Reflector analyzes reasoning deficiencies and formulates targeted remediation strategies:

$$(d, p) = \pi_{\text{reflector}}(\mathcal{T}, q, \tau, a_s, \mathcal{F}) \tag{3}$$

where d represents a concise diagnostic summary identifying critical reasoning errors, and p outlines an actionable improvement plan with step-by-step corrections for subsequent attempts.

This reflection mechanism creates a powerful feedback loop: the Solver adapts based on precise diagnosis and targeted suggestions, rather than blindly attempting alternatives. Without such directed feedback, traditional systems often repeat the same errors, unable to identify reasoning flaws. The Reflector examines not just answer correctness, but the entire reasoning trajectory, pinpointing where logical connections faltered, operations were misapplied, or question intent was misunderstood.

By implementing this metacognitive layer, MAPLE progressively refines its reasoning strategies across multiple attempts, achieving within-task learning that static reasoning systems cannot replicate. The prompt template detailed in Appendix F.

3.2.4 Archiver

The Archiver agent ($\mathcal{A}$) introduces experiential learning through structured memory management. This agent implements knowledge retention, retrieval, and evolution mechanisms that enable continual improvement across reasoning tasks, with detailed working memory and long-term memory mechanisms described in Appendix B.1. The Archiver operates through three complementary modules:

Memory Summarization. Given the current table $\mathcal{T}$, question q, model-predicted answer a_m, ground truth a_g, Solver's reasoning history τ, and Reflector's outputs (d, p), the Archiver distills this rich context into a compact, structured memory note:

$$m = \pi_{\text{archiver-sum}}(\mathcal{T}, q, a_m, a_g, \tau, d, p) \tag{4}$$

This summarization extracts critical semantic features like keywords, tags, required operations, error types, correct and incorrect steps, creating an informative memory note m that captures the problem-solving episode's essence. These notes are stored in long-term memory $\mathcal{M}_l$ to guide future reasoning.

Memory Retrieval. The system implements dual retrieval modes for different contexts:

1. **Solver-time retrieval**: During question answering, the system retrieves relevant memory notes based on the current table and question. The top-k semantically similar results within threshold δ are returned for prompt augmentation:

$$\mathcal{N}_{\text{solver}} = \pi^{\text{solver}}_{\text{mem-retrieve}}(\mathcal{T}, q \mid \mathcal{M}_l, k, \delta) \tag{5}$$

2. **Archiver-time retrieval**: During memory management, the system identifies neighboring notes for new candidates to inform evolution decisions:

$$\mathcal{N}_{\text{archiver}} = \pi^{\text{archiver}}_{\text{mem-retrieve}}(m \mid \mathcal{M}_l, k, \delta) \tag{6}$$

This dual architecture enables both experiential guidance during active reasoning and strategic memory refinement during maintenance, creating a dynamic knowledge ecosystem that continuously improves.

Memory Evolution. Our memory system actively evolves. Given the newly created memory m and a set of retrieved neighbor memories $\mathcal{N}_{\text{archiver}}$, the Archiver determines whether the memory base should evolve:

$$e = \pi_{\text{archiver-evo}}(m, \mathcal{N}_{\text{archiver}}) \tag{7}$$

Algorithm 1 : MAPLE's Adaptive Reasoning Loop with Multi-Agent Feedback

```
Input: Table $\mathcal{T}$, Question $q$, Memory $\mathcal{M} \in \{\mathcal{M}_w, \mathcal{M}_l\}$, Remaining Attempts $r$, Neighbor Limit $k$,
    Similarity Threshold $\delta$
Output: Final Answer $a_m$
$Finished \leftarrow$ False                                          ▷ Initialization flag
$\tau \leftarrow []$                                                 ▷ Initialize Solver operation history
while $\neg Finished$ and $r > 0$ do
    $\mathcal{N}_{\text{solver}} \leftarrow \pi^{\text{solver}}_{\text{mem-retrieve}}(\mathcal{T}, q \mid \mathcal{M}_l, k, \delta)$      ▷ Retrieve neighbor memories, Eq. 5
    $t'$ or $a_s \leftarrow \pi_{\text{solver}}(I, q, \tau, r, \mathcal{M})$          ▷ Operated table or answer of Solver, Eq. 1
    $\tau$.append(LogSolverOperation($\mathcal{M}_w$))                  ▷ Log current operation in operation history
    if $a_s \neq$ <NOT_READY> then
        $\mathcal{F} \leftarrow \pi_{\text{checker}}(\mathcal{T}, q, a_s)$                   ▷ Evaluate answer, Eq. 2
        if $\mathcal{F}$.total_score == FULL_SCORE then
            $a_m \leftarrow a_s$                                     ▷ Accept Solver answer
            $Finished \leftarrow$ True
        else
            $(d, p) \leftarrow \pi_{\text{reflector}}(\mathcal{T}, q, \tau, a_s, \mathcal{F})$     ▷ Diagnose and provide fix plan, Eq. 3
            $\mathcal{M}_w \leftarrow$ UpdateWorkingMemory$(d, p)$          ▷ Inject feedback for adaptive refinement
        end if
    else
        Continue ReAct reasoning on updated table $t'$               ▷ Continue ReAct loop
    end if
    $r \leftarrow r - 1$                                             ▷ Decrease remaining attempts
end while
return $a_m$                                                         ▷ Output model prediction
```

where e specifies evolution decisions, including whether to evolve and what specific actions (strengthening connections or updating memory metadata) to take. This process enhances semantic clustering of related experiences, enabling more contextually relevant knowledge retrieval in subsequent tasks.

Through this sophisticated memory management approach, MAPLE transcends traditional systems that restart reasoning from scratch on each task. Instead, our framework builds an evolving knowledge repository that improves performance across similar problems through continuous refinement. For a more detailed algorithm describing this memory evolution process, please see Appendix C. The complete Archiver prompt is provided in Appendix F.

Having defined the specialized roles and interactions of each agent in our framework, we now formalize the complete reasoning procedure that orchestrates their collaborative operation. Algorithm 1 presents the adaptive reasoning loop of MAPLE, illustrating how multiple agents coordinate through a feedback-driven cycle to progressively refine reasoning strategies. This algorithm demonstrates several key innovations absent in traditional approaches: (1) iterative refinement through verification and reflection, (2) dynamic adaptation based on structured feedback, and (3) integration of experiential knowledge from similar problems.

4 Experiments

4.1 Experimental Setup

Datasets. We evaluate our approach on two standard benchmarks: (1) WikiTableQuestions (WIKITQ) (Pasupat and Liang, 2015): A widely used benchmark dataset for studying question answering over structured tables. It contains 14,149 question-answer pairs for training and 4,344 for testing, collected from 421 Wikipedia tables. The questions require different levels of reasoning, and the answers can be single values, lists of values, or derived results that are not explicitly present in the table. (2) TABFACT (Chen et al., 2020): A benchmark for fact verification over tabular data, consisting of natural language statements paired with tables from diverse domains. Each statement

Dataset	WikiTQ				TabFact			
Model	**GPT-4o-mini**	**LLaMA3.3-70B**	**Qwen2.5-72B**	**Average**	**GPT-4o-mini**	**LLaMA3.3-70B**	**Qwen2.5-72B**	**Average**
End-to-End QA	46.64	45.58	35.93	42.72	71.69	75.49	63.49	70.22
Few-Shot QA	57.16	58.54	37.66	51.12	70.16	74.73	67.49	70.79
Chain-of-Thought	62.96	65.75	63.12	63.94	71.94	75.89	77.67	75.17
Binder (Cheng et al., 2023)	60.24	65.26	63.64	63.05	75.45	74.54	80.20	76.73
Dater (Ye et al., 2023)	61.13	62.18	61.94	61.75	75.39	81.12	78.46	78.32
Chain-of-Table (Wang et al., 2024b)	62.98	64.80	67.87	65.22	79.05	83.25	81.96	81.42
ReAcTable (Zhang et al., 2023)	62.79	63.07	63.74	63.20	79.18	81.49	82.06	80.91
MAPLE (Ours)	**67.13** ↑ 4.15	**74.01** ↑ 8.26	**73.39** ↑ 5.52	**71.51** ↑ 6.29	**81.62** ↑ 2.44	**90.66** ↑ 7.41	**86.02** ↑ 3.96	**86.10** ↑ 4.68

Table 2: Table reasoning accuracy on WikiTQ and TabFact using Qwen2.5-72B, LLaMA3.3-70B, and GPT-4o-mini. **Bold** denotes the best performance and underline denotes the second-best performance in each column. Red arrows indicate improvements over the strongest baseline.

is labeled as either entailed ("yes") or refuted ("no") based on the table content. The test set includes 2,024 statements across 298 tables, requiring models to perform complex reasoning to verify factual accuracy.

Baselines. We compare our multi-agent framework against three categories of baseline approaches: (1) **Standard Reasoning**, where the model directly generates answers from the table and question. This includes End-to-End QA, which outputs the answer in a single step. Few-Shot QA, which adds example (Table, Question, Answer) triplets to guide the model. Chain-of-Thought (Wei et al., 2023), which encourages the model to explain its reasoning process before answering. (2) **Program-Based Reasoning**, which guide the model to produce executable code for answering. Binder (Cheng et al., 2023) prompts the model to generate Python or SQL code. Dater (Ye et al., 2023) breaks down the question and table into smaller parts for easier processing. (3) **ReAct-Based Reasoning**: This approach integrates reasoning and acting in an iterative process, using external tools to assist decision-making. Chain-of-Table (Wang et al., 2024b) dynamically constructs intermediate tables to support reasoning. ReAcTable (Zhang et al., 2023) follows this paradigm by integrating SQL and Python executions to generate intermediate results and refine reasoning steps.

Implementation Details. We conduct our experiments using 3 state-of-the-art LLMs: GPT-4o-mini[2], LLaMA3.3-70B-Instruct (Grattafiori et al., 2024)[3] and Qwen2.5-72B-Instruct (Qwen et al., 2025)[4]. All models run on two NVIDIA A100 GPUs. The tabular input is converted into markdown format before being passed to the LLMs. We use in-context prompting by including task-specific examples, which are provided in Appendix F. Default decoding parameters are used throughout. For all baseline methods, we follow their original settings to ensure optimal performance.

Metrics. For WikiTQ, we compute denotation accuracy by measuring whether the predicted answer matches the gold answer, regardless of surface form. For TabFact, where the task is framed as binary classification ("yes" or "no"), we evaluate model predictions using exact string matching against the ground truth labels.

4.2 Main Results

Table 2 presents the performance comparison on WikiTQ and TabFact across LLaMA3.3-70B, GPT-4o-mini and Qwen2.5-72B. Our proposed method, MAPLE, consistently outperforms all baselines across both datasets and model backbones. On WikiTQ, MAPLE achieves 74.01% and 73.39% accuracy with respective models, while on TabFact, it reaches 90.66% and 86.02%. These results represent substantial gains of up to +8.26% on WikiTQ and +7.41% on TabFact over the strongest baseline methods.

Compared to recent specialized frameworks like Chain-of-Table and ReAcTable, MAPLE demonstrates consistent improvements across both datasets. The gains are particularly pronounced on WikiTQ (+5.52% with Qwen2.5-72B), aligning with our framework's strength in handling compositional reasoning tasks that require progressive refinement. For TabFact, the improvements confirm that our approach remains effective even in binary classification settings. For analysis of how table size affects reasoning performance and the impact of multi-round reasoning, see Appendix E.

---

[2] https://platform.openai.com/docs/models/gpt-4o-mini

[3] https://huggingface.co/meta-llama/Llama-3.3-70B-Instruct

[4] https://huggingface.co/Qwen/Qwen2.5-72B-Instruct

Settings	LLAMA3.3-70B
Baseline	45.58
+ Solver	63.81
+ Solver & Checker	65.91
+ Solver & Checker & Reflector	71.09
+ Solver & Checker & Reflector & Archiver	74.01

Table 3: Ablation study showing the incremental contribution of each agent in MAPLE.

These results validate the core design principles of MAPLE: dynamic adaptive planning through multi-round feedback loops, specialized agent roles with distinct cognitive functions, and progressive knowledge accumulation through long-term memory.

4.3 Evaluating Agent Contributions

To assess the contribution of each agent in our framework, we conduct an ablation study using the LLaMA3.3-70B model on WikiTQ. As shown in Table 3, we incrementally introduce agents into the MAPLE pipeline. Starting from a baseline that directly predicts answers without reasoning, we observe that adding the Solver agent alone leads to a substantial performance boost of +18.2 points, confirming the effectiveness of our ReAct-style multi-step reasoning. Incorporating the Checker further improves accuracy (+2.1), suggesting that verifying answer quality plays a crucial role in reducing erroneous outputs. Introducing the Reflector yields an additional significant gain (+5.2), highlighting the importance of iterative reflection and error correction. Finally, equipping the system with the Archiver enables long-term memory utilization, resulting in peak accuracy of 74.01.

4.4 Memory Analysis and System Behavior

Error Distribution in LLM Table Reasoning. Figure 2 shows the distribution of error types on WikiTQ, based on all errors stored in our memory system after multiple rounds of reasoning and verification. The two most dominant categories are Logical Reasoning Errors (40.4%) and Counting & Aggregation Errors (38.7%), together accounting for nearly 80% of all failures. These are followed by Format & Temporal Interpretation Errors (11.0%), Incomplete Information Extraction (5.8%), and Calculation & Comparison Errors (4.1%).

Notably, this distribution provides valuable insights into persistent challenges even after multi-round verification and reflection. The relatively low proportion of basic computational errors (4.1%) suggests that our iterative verification process effectively eliminates many simpler mistakes. However, the predominance of logical reasoning and aggregation errors indicates two critical directions for future improvement: (1) enhancing the fundamental reasoning capabilities of LLMs to address the 40.4% of logical errors, and (2) integrating specialized external tools for precise counting and aggregation operations, which could potentially resolve the 38.7% of errors related to handling large tables with numerous entities. This analysis demonstrates how **our memory-based error categorization not only provides diagnostic information but also guides strategic research priorities for advancing table-based reasoning capabilities.**

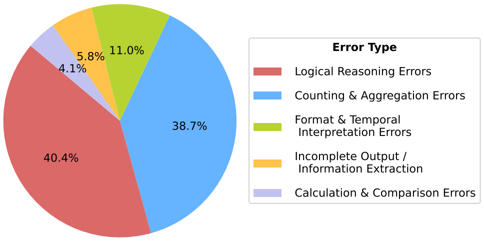


Figure 2: Distribution of error types identified through MAPLE's memory system on WikiTQ.

Memory Dynamics and Similarity Threshold Analysis. To understand how similarity thresholds affect memory system behavior, we analyze memory statistics across different distance thresholds (δ) on both datasets, with results presented in Table 4.

Memory Filtering Effects. As δ increases from 0.3 to 1.0, total memory size decreases dramatically, from 4,078 to 191 notes for WikiTQ and 1,882 to 78 for TabFact. This demonstrates our selective integration approach in action, preventing memory explosion at low thresholds while becoming increasingly selective at higher thresholds. At $\delta = 0.3$, nearly all experiences (93.9%) are added to memory, creating potential redundancy, while at $\delta = 1.0$, only highly unique experiences (4.4%) are retained, representing two extremes of the memory gradient phenomenon.

Optimal Evolution Dynamics. The evolution ratio (percentage of memories that undergo evolution) follows an interesting pattern, peaking at approximately $\delta = 0.7$ for both datasets (65.2% for WikiTQ, 74.7% for TabFact). Similarly, evolu-

Dataset	**Threshold (δ)**	**Memory Count**	**Memory Ratio (%)**	**Evolution Count**	**Evolution Ratio (%)**	**# Evolved Memories**	**Evolution Efficiency ↑**	**Med. Strengthen Distance**	**Med. Update Distance**	**Accuracy (%) ↑**
WikiTQ	0.1	4343	100.0%	0	00.0%	0	0.00	NA	NA	73.64
	0.3	4078	93.9%	843	20.7%	981	1.16	0.25	0.25	74.01
	0.5	2615	60.2%	1269	48.5%	1504	1.19	0.45	0.44	71.18
	0.7	1023	23.5%	667	65.2%	820	1.23	0.64	0.63	72.28
	0.9	347	8.0%	224	64.6%	254	1.13	0.81	0.81	71.82
	1	191	4.4%	112	58.6%	133	1.19	0.89	0.88	71.82
TabFact	0.1	2024	100.0%	0	00.0%	0	0.00	NA	NA	85.77
	0.3	1882	93.0%	719	38.2%	787	1.09	0.24	0.24	85.70
	0.5	1108	54.7%	710	64.1%	813	1.15	0.42	0.41	90.66
	0.7	427	21.1%	319	74.7%	372	1.17	0.60	0.58	86.29
	0.9	151	7.5%	100	66.2%	106	1.06	0.78	0.70	85.79
	1	78	3.9%	53	67.9%	59	1.11	0.86	0.82	85.40

Table 4: Memory system dynamics across different similarity thresholds (δ) on WikiTQ and TabFact datasets.

tion efficiency, measured as evolved memories per evolution operation, reaches its maximum around the same threshold (1.23 for WikiTQ, 1.17 for TabFact). This suggests that moderate similarity thresholds create ideal conditions for knowledge evolution.

Optimal Retrieval Thresholds. While evolution efficiency peaks at $\delta = 0.7$, optimal thresholds for accuracy show distinct patterns: WikiTQ achieves highest performance (74.01%) at $\delta = 0.3$, while TabFact reaches peak accuracy (90.66%) at $\delta = 0.5$. This divergence reflects the different functional requirements: during problem-solving, lower thresholds retrieve highly relevant, directly applicable memories, whereas evolution benefits from moderate thresholds balancing similarity with sufficient diversity. Performance stabilizes beyond $\delta = 0.9$, indicating a saturation point where retrieving increasingly dissimilar memories provides little additional value. These findings reveal that **$\pi^{\text{solver}}_{\text{mem-retrieve}}$ requires stricter relevance criteria (lower δ values of 0.3-0.5) for effective reasoning guidance, while $\pi^{\text{archiver}}_{\text{mem-retrieve}}$ operates optimally at $\delta = 0.7$ for efficient memory organization.**

Cross-Dataset Consistency. Both datasets exhibit remarkably similar memory dynamics despite their different task characteristics, suggesting that these patterns reflect fundamental properties of knowledge organization rather than dataset-specific artifacts.

Theoretical Significance. These findings align with the "approximate learning" theory in cognitive science, which posits that **optimal knowledge acquisition occurs when new information is related to existing knowledge that is neither too similar nor too different** (Gentner and Smith, 2013). Our empirical results showing peak evolution at moderate distances ($\delta \approx 0.7$) provide computational evidence for this cognitive principle.

This analysis reveals that memory dynamics in MAPLE follow a nuanced optimization pattern across different operational modes. For memory evolution, a "Goldilocks principle" applies—with too little filtering ($\delta < 0.5$), the system becomes overwhelmed with redundant information; with excessive filtering ($\delta > 0.9$), it lacks sufficient knowledge connections for meaningful evolution. The optimal range for evolution ($\delta \approx 0.7$) balances memory diversity and coherence. Meanwhile, accuracy optimization benefits from more stringent relevance criteria ($\delta = 0.3$-0.5), ensuring that only the most applicable experiences inform reasoning. This dual-threshold approach enables MAPLE to simultaneously optimize both knowledge organization and problem-solving performance.

5 Conclusion

This paper presents MAPLE, a multi-agent framework for table reasoning that integrates adaptive planning with long-term memory evolution. By decomposing reasoning into specialized functions handled by distinct agents, our approach enables dynamic strategy refinement through a feedback-driven cycle. Experiments on WikiTQ and TabFact demonstrate significant improvements over existing methods, with ablation studies confirming each component's value. Our memory analysis reveals that logical reasoning errors and counting/aggregation operations account for nearly 80% of remaining mistakes, suggesting 2 promising directions: enhancing fundamental reasoning capabilities and developing specialized numerical tools for complex operations. Beyond table reasoning, the principles demonstrated in MAPLE may benefit knowledge-intensive tasks where verification, reflection, and experience accumulation are crucial.

Limitations

Despite MAPLE's promising results, several limitations should be acknowledged. First, our approach is computationally more intensive than single-pass methods due to its multi-round, multi-agent architecture. Each reasoning attempt requires multiple LLM calls across different agents, increasing both inference time and computational costs. This presents challenges for real-time applications or deployment on resource-constrained systems.

Second, while our memory evolution mechanism demonstrates effectiveness in our experiments, its long-term scalability remains unexplored. As the memory base grows, maintaining coherence and preventing knowledge dilution become increasingly challenging. Future work should examine more sophisticated memory management strategies, including forgetting mechanisms and hierarchical organization of memory notes.

Finally, our framework currently focuses exclusively on table-based reasoning without incorporating external knowledge. This limits its applicability to questions requiring information beyond what's explicitly presented in the table. Enhancing MAPLEwith external knowledge collection capabilities would be a valuable extension to address this limitation.

References

Huaben Chen, Wenkang Ji, Lufeng Xu, and Shiyu Zhao. 2025. Multi-agent consensus seeking via large language models. *Preprint*, arXiv:2310.20151.

Wenhu Chen, Hongmin Wang, Jianshu Chen, Yunkai Zhang, Hong Wang, Shiyang Li, Xiyou Zhou, and William Yang Wang. 2020. Tabfact: A large-scale dataset for table-based fact verification. *Preprint*, arXiv:1909.02164.

Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul Nadkarni, Yushi Hu, Caiming Xiong, Dragomir Radev, Mari Ostendorf, Luke Zettlemoyer, Noah A. Smith, and Tao Yu. 2023. Binding Language Models in Symbolic Languages. *arXiv preprint*. ArXiv:2210.02875 [cs].

Ishita Dasgupta, Christine Kaeser-Chen, Kenneth Marino, Arun Ahuja, Sheila Babayan, Felix Hill, and Rob Fergus. 2023. Collaborating with language models for embodied reasoning. *Preprint*, arXiv:2302.00763.

Haoyu Dong and Zhiruo Wang. 2024. Large language models for tabular data: Progresses and future directions. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '24, page 2997–3000, New York, NY, USA. Association for Computing Machinery.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. 2023. Improving factuality and reasoning in language models through multiagent debate. *Preprint*, arXiv:2305.14325.

Julian Eisenschlos, Syrine Krichene, and Thomas Müller. 2020. Understanding tables with intermediate pre-training. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 281–296, Online. Association for Computational Linguistics.

Chen Gao, Xiaochong Lan, Zhihong Lu, Jinzhu Mao, Jinghua Piao, Huandong Wang, Depeng Jin, and Yong Li. 2023. S3: Social-network simulation system with large language model-empowered agents. *Preprint*, arXiv:2307.14984.

Dedre Gentner and Linsey A. Smith. 2013. 668 analogical learning and reasoning. In *The Oxford Handbook of Cognitive Psychology*. Oxford University Press.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, and et al. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Zihui Gu, Ju Fan, Nan Tang, Preslav Nakov, Xiaoman Zhao, and Xiaoyong Du. 2022. PASTA: Table-Operations Aware Fact Verification via Sentence-Table Cloze Pre-training. *arXiv preprint*. ArXiv:2211.02816 [cs].

Jonathan Herzig, Paweł Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. 2020. TAPAS: Weakly Supervised Table Parsing via Pre-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4320–4333. ArXiv:2004.02349 [cs].

Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024. Metagpt: Meta programming for a multi-agent collaborative framework. *Preprint*, arXiv:2308.00352.

John J. Horton. 2023. Large language models as simulated economic agents: What can we learn from homo silicus? *Preprint*, arXiv:2301.07543.

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

Wenyue Hua, Lizhou Fan, Lingyao Li, Kai Mei, Jianchao Ji, Yingqiang Ge, Libby Hemphill, and Yongfeng Zhang. 2024. War and peace (waragent): Large language model-based multi-agent simulation of world wars. *Preprint*, arXiv:2311.17227.

Dong Huang, Jie M. Zhang, Michael Luck, Qingwen Bu, Yuhao Qing, and Heming Cui. 2024. Agentcoder: Multi-agent-based code generation with iterative testing and optimisation. *Preprint*, arXiv:2312.13010.

Zhengbao Jiang, Yi Mao, Pengcheng He, Graham Neubig, and Weizhu Chen. 2022. OmniTab: Pretraining with natural and synthetic data for few-shot table-based question answering. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 932–942, Seattle, United States. Association for Computational Linguistics.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023a. CAMEL: Communicative agents for "mind" exploration of large language model society. In *Thirty-seventh Conference on Neural Information Processing Systems*.

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

Jonathan Light, Min Cai, Sheng Shen, and Ziniu Hu. 2023. Avalonbench: Evaluating llms playing the game of avalon. *Preprint*, arXiv:2310.05036.

Qian Liu, Bei Chen, Jiaqi Guo, Morteza Ziyadi, Zeqi Lin, Weizhu Chen, and Jian-Guang Lou. 2022. TAPEX: Table Pre-training via Learning a Neural SQL Executor. *arXiv preprint*. ArXiv:2107.07653 [cs].

Weizheng Lu, Jing Zhang, Ju Fan, Zihao Fu, Yueguo Chen, and Xiaoyong Du. 2025. Large language model for table processing: a survey. *Frontiers of Computer Science*, 19(2).

Zhao Mandi, Shreeya Jain, and Shuran Song. 2023. Roco: Dialectic multi-robot collaboration with large language models. *Preprint*, arXiv:2307.04738.

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 C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. *Preprint*, arXiv:2304.03442.

Joon Sung Park, Lindsay Popowski, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2022. Social simulacra: Creating populated prototypes for social computing systems. *Preprint*, arXiv:2208.04024.

Panupong Pasupat and Percy Liang. 2015. Compositional Semantic Parsing on Semi-Structured Tables. *arXiv preprint*. ArXiv:1508.00305 [cs].

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. Chatdev: Communicative agents for software development. *Preprint*, arXiv:2307.07924.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, and 25 others. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

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

Theodore R. Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L. Griffiths. 2024. Cognitive architectures for language agents. *Preprint*, arXiv:2309.02427.

Xiangru Tang, Anni Zou, Zhuosheng Zhang, Ziming Li, Yilun Zhao, Xingyao Zhang, Arman Cohan, and Mark Gerstein. 2024. Medagents: Large language models as collaborators for zero-shot medical reasoning. *Preprint*, arXiv:2311.10537.

Bing Wang, Xinnian Liang, Jian Yang, Hui Huang, Shuangzhi Wu, Peihao Wu, Lu Lu, Zejun Ma, and Zhoujun Li. 2025. Scm: Enhancing large language model with self-controlled memory framework. *Preprint*, arXiv:2304.13343.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023a. Voyager: An open-ended embodied agent with large language models. *Preprint*, arXiv:2305.16291.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. 2024a. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6).

Shenzhi Wang, Chang Liu, Zilong Zheng, Siyuan Qi, Shuo Chen, Qisen Yang, Andrew Zhao, Chaofei Wang, Shiji Song, and Gao Huang. 2023b. Avalon's game of thoughts: Battle against deception through recursive contemplation. *Preprint*, arXiv:2310.01320.

Zhiruo Wang, Haoyu Dong, Ran Jia, Jia Li, Zhiyi Fu, Shi Han, and Dongmei Zhang. 2021. TUTA: Tree-based Transformers for Generally Structured Table Pre-training. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1780–1790. ArXiv:2010.12537 [cs].

Zilong Wang, Hao Zhang, Chun-Liang Li, Julian Martin Eisenschlos, Vincent Perot, Zifeng Wang, Lesly Miculicich, Yasuhisa Fujii, Jingbo Shang, Chen-Yu Lee, and Tomas Pfister. 2024b. Chain-of-Table: Evolving Tables in the Reasoning Chain for Table Understanding. *arXiv preprint*. ArXiv:2401.04398 [cs].

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models. *Preprint*, arXiv:2201.11903.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. 2023. Autogen: Enabling next-gen llm applications via multi-agent conversation. *Preprint*, arXiv:2308.08155.

Bushi Xiao, Ziyuan Yin, and Zixuan Shan. 2023. Simulating public administration crisis: A novel generative agent-based simulation system to lower technology barriers in social science research. *Preprint*, arXiv:2311.06957.

Kai Xiong, Xiao Ding, Yixin Cao, Ting Liu, and Bing Qin. 2023. Examining inter-consistency of large language models collaboration: An in-depth analysis via debate. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, page 7572–7590. Association for Computational Linguistics.

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

Zelai Xu, Chao Yu, Fei Fang, Yu Wang, and Yi Wu. 2024. Language agents with reinforcement learning for strategic play in the werewolf game. *Preprint*, arXiv:2310.18940.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. *Preprint*, arXiv:2210.03629.

Yunhu Ye, Binyuan Hui, Min Yang, Binhua Li, Fei Huang, and Yongbin Li. 2023. Large Language Models are Versatile Decomposers: Decompose Evidence and Questions for Table-based Reasoning. *arXiv preprint*. ArXiv:2301.13808 [cs].

Bangguo Yu, Qihao Yuan, Kailai Li, Hamidreza Kasaei, and Ming Cao. 2025a. Co-navgpt: Multi-robot cooperative visual semantic navigation using vision language models. *Preprint*, arXiv:2310.07937.

Peiying Yu, Guoxin Chen, and Jingjing Wang. 2025b. Table-critic: A multi-agent framework for collaborative criticism and refinement in table reasoning. *Preprint*, arXiv:2502.11799.

Hongxin Zhang, Weihua Du, Jiaming Shan, Qinhong Zhou, Yilun Du, Joshua B. Tenenbaum, Tianmin Shu, and Chuang Gan. 2024. Building cooperative embodied agents modularly with large language models. *Preprint*, arXiv:2307.02485.

Yunjia Zhang, Jordan Henkel, Avrilia Floratou, Joyce Cahoon, Shaleen Deep, and Jignesh M. Patel. 2023. ReAcTable: Enhancing ReAct for Table Question Answering. *arXiv preprint*. ArXiv:2310.00815 [cs].

Qinlin Zhao, Jindong Wang, Yixuan Zhang, Yiqiao Jin, Kaijie Zhu, Hao Chen, and Xing Xie. 2024. Competeai: Understanding the competition dynamics in large language model-based agents. *Preprint*, arXiv:2310.17512.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. 2023. Memorybank: Enhancing large language models with long-term memory. *Preprint*, arXiv:2305.10250.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and Ed Chi. 2023a. Least-to-most prompting enables complex reasoning in large language models. *Preprint*, arXiv:2205.10625.

Xuanhe Zhou, Guoliang Li, and Zhiyuan Liu. 2023b. Llm as dba. *Preprint*, arXiv:2308.05481.

Appendix

A Related Work

As illustrated in Table 5, existing table reasoning methods exhibit significant limitations across key design dimensions. While some implement multi-agent architectures or ReAct-based reasoning, none integrates all critical components: dynamic planning, reflection mechanisms, self-refinement, and long-term memory. Our proposed MAPLE framework addresses these gaps by combining collaborative verification, adaptive planning, and evolving memory structures to achieve more robust and accurate table reasoning.

B Cognitive Architecture

B.1 Memory Module

To enable multi-step reasoning, verification, and reflection, MAPLE organizes internal information across two complementary memory modules: a short-term working memory and a long-term memory. The working memory enables flexible planning and adaptation during reasoning by dynamically maintaining intermediate states, while the long-term memory provides stable knowledge accumulated across tasks to guide future decisions. Together, these memory structures allow different agents to persist, access, and manipulate relevant information throughout and across problem-solving sessions.

Working Memory. The working memory $(\mathcal{M}_w)$ temporarily stores all information related to the current task instance, implementing a Shared Message Pool architecture (Hong et al., 2024) for agent communication. As shown in Figure 3, it maintains the original table $\mathcal{T}$, the question q, the Solver's operation history (including intermediate tables and tentative answers), Checker feedback (scores and comments), Reflector analysis (diagnosis and suggestions), and task-level metadata such as the number of remaining attempts.

Unlike centralized or hierarchical communication structures, our Shared Message Pool enables all agents to asynchronously publish information to and subscribe from a common memory space. This architecture facilitates flexible many-to-many interactions without predefined communication pathways, allowing emergent collaboration patterns based on informational dependencies rather than rigid control flow. For example, the Reflector can simultaneously observe both Solver reasoning steps and Checker feedback, synthesizing insights that would be difficult to achieve in strictly layered or peer-to-peer architectures.

To directly facilitate communication with large language models during multi-turn interactions, the working memory is represented entirely in natural language format. Each agent, Solver, Checker, Reflector, and Archiver, reads from and writes to this shared memory throughout the reasoning cycle, ensuring that context is consistently updated and accessible at every decision point.

Long-term Memory. The long-term memory $(\mathcal{M}_l)$ captures accumulated knowledge across tasks, supporting continual improvement and experience reuse. Inspired by frameworks like A-MEM (Xu et al., 2025), we adapt their approaches specifically for table-based reasoning challenges.

Our memory structure is tailored for table QA, with each memory note containing fields crucial for reasoning diagnostics: question type, required operations, correct/wrong steps, error types, reasons and additional metadata enables tracking of memory dynamics over time. We implement a hybrid storage format—structured metadata for embedding-based retrieval and natural language descriptions for interpretability.

A key innovation is our selective integration mechanism, which differs from frameworks like A-MEM that accumulate all experiences indiscriminately. While A-MEM addresses long-range conversations where preserving every detail is essential, table QA questions often exhibit high similarity and structural homogeneity, making comprehensive retention inefficient. Our approach implements density-based filtering: when a new memory note closely resembles multiple existing entries, it is not added to the repository, preventing memory saturation while preserving diverse reasoning patterns.

Similarly, while adopting the concept of memory evolution from prior work, we introduce strategic timing for evolution operations, performing them selectively rather than universally, to maximize coherence without unnecessary computational overhead.

This specialized approach achieves an optimal balance between knowledge retention and efficiency for table reasoning tasks. During operation, the Archiver retrieves relevant memories to assist the Solver, and after task completion, strategically updates the memory base by either inserting

Model	**Multi-Agent System**	**External Tool Use**	**ReAct**	**Reflection**	**Self-Refinement**	**Long-Term Memory**	**Dynamic Planning**
Binder (Cheng et al., 2023)	✗	✓	✗	✗	✗	✗	✗
Dater (Ye et al., 2023)	✓	✓	✗	✗	✗	✗	✗
Chain-of-Table (Wang et al., 2024b)	✗	✗	✓	✗	✗	✗	✗
ReAcTable (Zhang et al., 2023)	✗	✓	✓	✗	✗	✗	✗
Table-Critic (Yu et al., 2025b)	✓	✗	✗	✓	✓	✓	✗
MAPLE (Ours)	✓	✗	✓	✓	✓	✓	✓

Table 5: Comparison of representative table reasoning methods across key design dimensions.

Algorithm 2 : MAPLE's Dynamic Memory Evolution Process

```
Input: Working Memory $\mathcal{M}_w = \{\mathcal{T}, q, a_m, a_g, \tau, d, p\}$, Long-term Memory $\mathcal{M}_l$, Distance Threshold $\delta$, Neighbor Limit $k$, Required Minimum Neighbors $k_{min}$
Output: Updated Long-term Memory $\mathcal{M}_l$
for each sample in working memory do
    $m \leftarrow \pi_{\text{archiver-sum}}(\mathcal{M}_w)$    ▷ Distill experience into memory note, Eq. 4
    $\mathcal{N}_{\text{archiver}} \leftarrow \pi^{\text{archiver}}_{\text{mem-retrieve}}(m, \mathcal{M}_l, k, \delta)$    ▷ Retrieve $\leq k$ similar memories within $\delta$, Eq. 6
    if $|\mathcal{N}_{\text{archiver}}| < k_{min}$ then    ▷ Filter redundant memories
        $e \leftarrow \pi_{\text{archiver-evo}}(m, \mathcal{N}_{\text{archiver}})$    ▷ Decide evolution actions, Eq. 7
        if $e$.should_evolve == True then
            if strengthen $\in e$.actions then
                ADDLINKS($m$, $e$.suggested_connections)    ▷ Create semantic connections
            end if
            if update_neighbor $\in e$.actions then
                UPDATEMETADATA($\mathcal{N}_{\text{archiver}}$, $e$.new_info_neighbor    ▷ Refine neighbor metadata
            end if
            $m$.tags $\leftarrow e$.tags_to_update    ▷ Update semantic tags
        end if
        ADDMEMORY($\mathcal{M}_l, m$)    ▷ Persist new experience
    end if
end for
return $\mathcal{M}_l$
```

unique experiences or evolving existing entries to strengthen their utility.

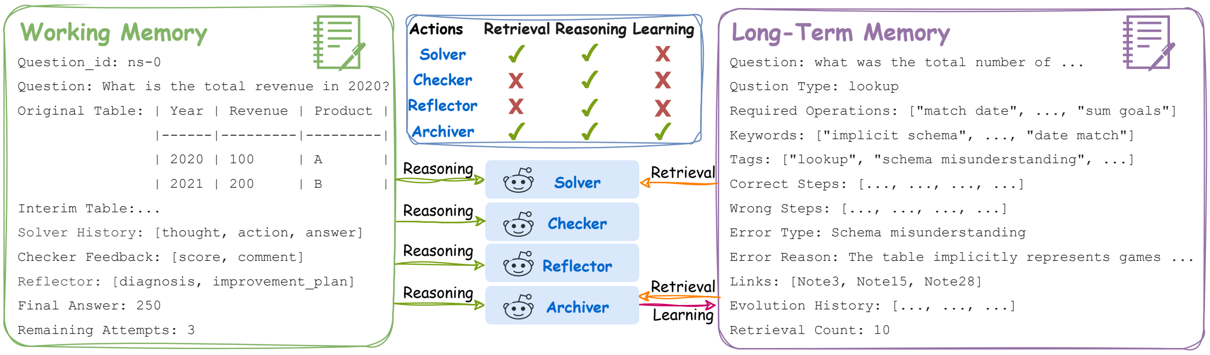
B.2 Action Module

The action module is responsible for translating an agent's decisions into specific outcomes. Positioned at the most downstream point of the architecture, it directly interacts with the environment and is influenced by the memory and planning modules.

Following the categorization proposed in CoALA Framework (Sumers et al., 2024), action spaces can be broadly divided into two categories: *external actions* and *internal actions*. External actions involve interactions with the external environment, such as controlling a robot, communicating with a human, or navigating a website. As our framework focuses on table reasoning tasks, we do not involve external actions.

Instead, our method operates entirely within internal action spaces, where actions are directed toward interacting with internal memory systems. Internal actions can be further classified into three types based on their interaction with memory: **Retrieval** involves reading from long-term memory to access relevant past experiences. **Reasoning** refers to updating the short-term working memory through LLM-based processing of current information. **Learning** denotes writing new information into long-term memory for future use.

These fundamental actions rarely occur in isolation, instead, they form characteristic sequences and combinations that enable sophisticated reasoning patterns. For example, the Solver typically engages in iterative reasoning cycles punctuated by occasional retrieval operations, while the Archiver combines retrieval and learning to maintain mem-



```
Question_id: ns-0
Question: What is the total revenue in 2020?
Original Table: | Year | Revenue  | Product  |
                |------|----------|----------|
                | 2020 | 100      | A        |
                | 2021 | 200      | B        |
Interim Table:...
Solver History: [thought, action, answer]
Checker Feedback: [score, comment]
Reflector: [diagnosis, improvement_plan]
Final Answer: 250
Remaining Attempts: 3
```

Actions	Retrieval	Reasoning	Learning
Solver	✓	✓	✗
Checker	✗	✓	✗
Reflector	✗	✓	✗
Archiver	✓	✓	✓

```
Question: what was the total number of ...
Qustion Type: lookup
Required Operations: ["match date", ..., "sum goals"]
Keywords: ["implicit schema", ..., "date match"]
Tags: ["lookup", "schema misunderstanding", ...]
Correct Steps: [..., ..., ..., ...]
Wrong Steps: [..., ..., ..., ...]
Error Type: Schema misunderstanding
Error Reason: The table implicitly represents games ...
Links: [Note3, Note15, Note28]
Evolution History: [..., ..., ...]
Retrieval Count: 10
```

Figure 3: Overview of the memory structures and information flows in MAPLE. The green arrows (→) represent reasoning processes, where agents **read and update** working memory during multi-step problem solving. The orange arrows (←) represent **retrieval** operations from long-term memory to support current reasoning. The red arrows (→) denote learning operations, where new knowledge is **written back** into the long-term memory.

ory coherence. The power of our multi-agent approach emerges from these diverse action patterns, allowing different agents to specialize in distinct cognitive operations while maintaining a cohesive problem-solving process.

Figure 3 summarizes the internal actions associated with each agent in our system. Both the Solver and Checker agents primarily engage in reasoning actions: they read from the working memory, process information according to their designated roles, and write the updated reasoning steps back into the working memory. The Reflector agent performs both reasoning and learning, as it not only updates the working memory but also contributes insights to the long-term memory. The Archiver agent engages in all three action types: retrieval to access relevant experiences, reasoning to analyze current tasks, and learning to evolve the memory base with new knowledge.

B.3 Planning Module

Effective planning is crucial for solving multi-step reasoning tasks, where the sequence and selection of actions can significantly impact final outcomes. Following the categorization proposed by Wang et al. (2024a), planning approaches can be broadly classified into two categories: planning without feedback and planning with feedback.

Planning without Feedback. Traditional reasoning systems typically employ static planning, where the entire reasoning trajectory is predetermined before execution. For instance, standard Chain-of-Thought prompting generates all reasoning steps in a single forward pass without adjusting to intermediate results. Whether using single-path reasoning (where each step leads to exactly one subsequent step) or multi-path reasoning (where reasoning steps form tree-like structures), these approaches struggle with complex tasks where initial plans require revision based on unexpected discoveries during execution. The fundamental limitation is their inability to iteratively refine strategies based on execution outcomes—a critical capability in human problem-solving.

Planning with Feedback. In contrast, MAPLE implements dynamic planning with dual-source feedback, enabling adaptive reasoning that more closely mirrors human cognition:

Environmental Feedback enables the Solver to observe changes in the table state after each operation and decide whether to continue manipulation or derive an answer. Similar to approaches like ReAct (Yao et al., 2023), our framework incorporates thought-action-observation triplets, allowing the Solver to adapt its reasoning trajectory based on real-time observations of how table manipulations affect the environment state. This environmental grounding prevents the accumulation of reasoning errors that plague single-pass methods.

Model Feedback from specialized verification agents (Checker and Reflector) provides structured evaluation of reasoning quality. Unlike self-reflection approaches where the same model instance both generates and evaluates its own solutions, our architecture implements a clear separation of concerns, dedicated agents with specialized prompts and evaluation criteria perform verification tasks. This functional modularity enables more objective assessment, as the Checker evaluates answers without access to the generation process, and the Reflector provides targeted diagnosis rather than mere self-justification. This division of cognitive labor creates a system of checks and balances that significantly reduces the confirmation bias inherent in single-model reflection approaches.

As illustrated in Figure 1, this feedback-driven planning eliminates the need for predefined reasoning sequences. Instead, the exact path through the reasoning space emerges dynamically from agent interactions: the Solver adjusts based on intermediate table states and Reflector diagnostics, the Checker determines when reasoning quality meets acceptance criteria, and the Archiver retrieves relevant experiences to guide initial approaches. This distributed, adaptive planning architecture creates an output-feedback-refinement loop that iteratively improves reasoning quality—a capability fundamental to robust problem-solving but absent in traditional single-pass systems.

C Memory Evolution Algorithm

In this section, we present the detailed algorithm for MAPLE's memory evolution process (Algorithm 2). While the main text describes the conceptual framework and key innovations of our memory system, this appendix provides the complete algorithmic implementation of how new experiences are evaluated, filtered, and integrated into the long-term memory base. Specifically, the ADDLINKS function corresponds to the *Strengthen* option in Figure 11: it does not alter the metadata of the neighboring memories but instead appends their `memory_id` to the `Links` field of the current memory. In contrast, the UPDATEMETADATA function corresponds to the *Update_Neighbor* option in Figure 11: it summarizes new `tags` and `context` based on both the current and neighboring memories, and updates these fields in each memory accordingly.

D Case Study

To illustrate how MAPLE's agents collaborate to refine reasoning, we present a step-by-step case study. As shown in Figure 4, the Solver begins with an initial attempt based on the input table and question. After generating an intermediate answer, the Checker detects a logical error, prompting the Reflector to diagnose the mistake and provide actionable feedback. Incorporating these insights, the Solver re-attempts the task and successfully derives the correct answer in the next round. The final reasoning trace is then passed to the Archiver, which summarizes it into a memory note and evaluates whether to evolve the memory base. In this case, the Archiver decides that no evolution is necessary.

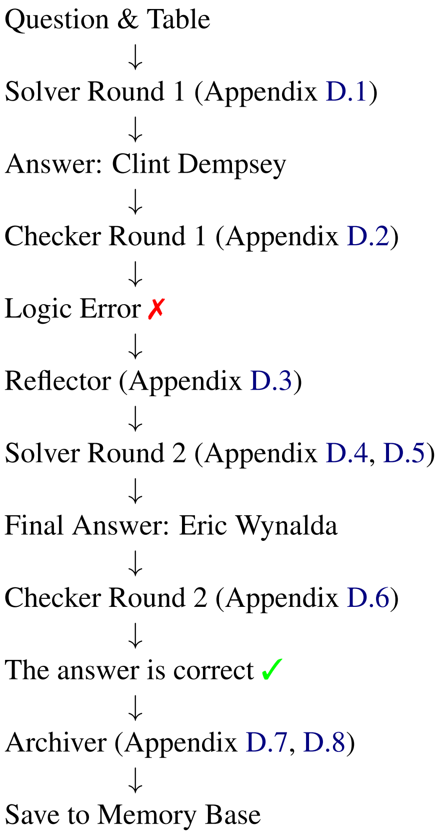

Question & Table
Solver Round 1 (Appendix D.1)
Answer: Clint Dempsey
Checker Round 1 (Appendix D.2)
Logic Error ✗
Reflector (Appendix D.3)
Solver Round 2 (Appendix D.4, D.5)
Final Answer: Eric Wynalda
Checker Round 2 (Appendix D.6)
The answer is correct ✓
Archiver (Appendix D.7, D.8)
Save to Memory Base

Figure 4: Illustrative case study of MAPLE's multi-agent reasoning workflow.

D.1 Round 1 - Solver

D.1.1 User Input

Below is a retrieved memory from a similar question previously answered by the system. You may use it as a reference to inform your reasoning.

<Related Memory>

- Past Question: who scored the most goals?
- Question Type: aggregation
- Required Operations: find maximum, compare
- Correct Reasoning Steps: 1.Identify the column representing the number of goals scored by each player; 2.Find the maximum value in the 'Goals' column; 3.Return the player associated with the maximum number of goals;
- Error Type: none
- Error Reason: none

Now, here is your actual Solver task. This is your 1 attempt. You have 4 attempts remaining.

<Table>

See Table 6

#	Player	Goals	Caps	Career
1	Landon Donovan	57	155	2000–present
2	Clint Dempsey	36	103	2004–present
3	Eric Wynalda	34	106	1990–2000
4	Brian McBride	30	95	1993–2006
5	Joe-Max Moore	24	100	1992–2002
6T	Jozy Altidore	21	67	2007–present
6T	Bruce Murray	21	86	1985–1993
8	Eddie Johnson	19	62	2004–present
9T	Earnie Stewart	17	101	1990–2004
9T	DaMarcus Beasley	17	114	2001–present

Table 6: Original input table used for case study. This is the table provided to the Solver at the beginning of the reasoning process.

<Question>

who was the top goalscorer previous to landon donovan?

D.1.2 LLM Response

- Thought: To find the top goalscorer before Landon Donovan, I need to identify the player with the most goals scored, excluding Landon Donovan. Since the table is already sorted by the number of goals in descending order, I can look at the second row to find the player with the next highest number of goals.
- Action: Identify the player with the second-highest number of goals
- Intermediate table: <NOT CHANGED>
- Answer: Clint Dempsey

D.2 Round 2 - Checker

D.2.1 User Input

<Table>

Original Table (Table 6)

<Question>

who was the top goalscorer previous to landon donovan?

<Answer>

Clint Dempsey

D.2.2 LLM Response

Answer Type Checking

- Score: 2
- Comments: The question asks for a player's name, and the answer is a player's name, so the type matches correctly.

Format Validation

- Score: 2
- Comments: The answer is a single player's name, which follows the expected format for this type of question.

Evidence Grounding

- Score: 0
- Comments: According to the table, before Landon Donovan, the top goalscorer would be the player with the next highest number of goals. Clint Dempsey has 36 goals, but Eric Wynalda has 34 goals and is listed above Dempsey in the table in terms of goals. However, the correct answer should be the one with the highest goals before Donovan, which is not Dempsey but actually the player with the next highest goal count after Donovan is not the correct interpretation. The correct interpretation is to find the player who was the top scorer before Donovan's time or the one with the next highest goals if considering the table's data. Since Donovan has 57 goals, the next in line is Dempsey with 36 goals, but considering the question's phrasing, it seems to ask for the top scorer before Donovan's emergence or record, which would indeed be Eric Wynalda or another player based on the 'previous to' criteria. Given the table, Eric Wynalda had 34 goals and was a top scorer before Donovan's time, making Dempsey not the correct answer based on the 'previous to' interpretation.

Summary

- Total Score: 4
- Final Comments: The answer is incorrect because, based on the logical interpretation of 'previous to Landon Donovan', the correct top goalscorer before Donovan would likely be Eric Wynalda, given the context and data provided in the table. Clint Dempsey does have a significant number of goals but does not fit the 'previous to' criteria as directly as another player might.

D.3 Round 3 - Reflector

D.3.1 User Input

<Question>

who was the top goalscorer previous to landon donovan?

<Table>

Original Table (Table 6)

<Reasoner's Processing History>

1.Identify the player with the second-highest number of goals

<Reasoner's Final answer>

Clint Dempsey

<Checker Feedback>

Same as Round 2 (D.2) Checker's output

D.3.2 LLM Response

- Diagnosis: The reasoner incorrectly identified Clint Dempsey as the top goalscorer before Landon Donovan. The mistake lies in the interpretation of 'previous to', which implies considering the time or emergence of players, not just the next highest goal count. The reasoner should have considered the career span and goals of players who were active before or alongside Donovan's early career to accurately determine the top scorer before him.
- Improvement plan: To improve, the reasoner should re-evaluate the question's phrasing and the table's data. Specifically, it should identify players who were active before Landon Donovan's emergence or during his early career and then determine which of these players had the highest number of goals. In this case, considering Eric Wynalda's career span (1990-2000) and goal count (34), he would be a more appropriate answer as the top goalscorer before Donovan, given the 'previous to' criteria. The reasoner needs to enhance its ability to interpret temporal relationships and contextual clues within the question to provide a more accurate answer.

D.4 Round 4 - Solver

D.4.1 User Input

Below is a retrieved memory from a similar question previously answered by the system. You may use it as a reference to inform your reasoning.

<Related Memory>

Same memory content as in Round 1(D.1) Solver input.

Now, here is your actual Solver task. This is your 1 attempt. You have 4 attempts remaining. (Note: This refers to the inner-round count under the ReAct paradigm, not the outer round of Solver retries.)

<Table>

Original Table (Table 6)

#	Player	Goals	Caps	Career
3	Eric Wynalda	34	106	1990–2000
4	Brian McBride	30	95	1993–2006
5	Joe-Max Moore	24	100	1992–2002
6T	Bruce Murray	21	86	1985–1993
9T	Earnie Stewart	17	101	1990–2004

Table 7: Intermediate table produced by the Solver during the reasoning process. Irrelevant rows have been filtered out based on the question.

<Question>

who was the top goalscorer previous to landon donovan?

If the <Reflector result> is available, it means you previously made a mistake, and the Reflector has summarized the cause of that error. So this time, when solving the question, please take the improvement plan section into account to avoid making the same mistake again.

<Reflector Result>

Same as Round 3(D.3) Reflector's output

D.4.2 LLM Response

- Thought: The question asks for the top goalscorer previous to Landon Donovan. The Reflector result indicates that I should consider the career span and goals of players who were active before or alongside Donovan's early career. Landon Donovan's career started in 2000. I need to identify players who were active before 2000 and find the one with the highest number of goals.
- Action: Filter players whose career ended before 2000 or started before 2000, then find the maximum goals among them
- Intermediate table: see Table 7
- Answer: <NOT READY>

D.5 Round 5 - Solver

D.5.1 User Input

Below is a retrieved memory from a similar question previously answered by the system. You may use it as a reference to inform your reasoning.

<Related Memory>

Same memory content as in Round 1(D.1) Solver input.

Now, here is your actual Solver task. This is your 2 attempt. You have 3 attempts remaining.

<Table>

Intermediate Table(Table 7) from Round 4(D.4)

<Question>

who was the top goalscorer previous to landon donovan?

<Action History>

1.Filter players whose career ended before 2000 or started before 2000, then find the maximum goals among them

<Reflector Result>

Same as Round 3(D.3) Reflector's output

D.5.2 LLM Response

- Thought: The Reflector result indicates that I should consider players who were active before or during Landon Donovan's early career. Given the 'previous to' criteria, I need to identify the top goalscorer among players whose career span ended before or overlapped with Donovan's emergence. Eric Wynalda's career (1990-2000) and high goal count (34) make him a strong candidate. I will re-evaluate the table data to confirm this.
- Action: Identify top goalscorer among players active before or during Landon Donovan's early career
- Intermediate table: <NOT CHANGED>
- Answer: Eric Wynalda

D.6 Round 6 - Checker

D.6.1 User Input

<Table>

Original Table (Table 6)

<Question>

who was the top goalscorer previous to landon donovan?

<answer>

Eric Wynalda

D.6.2 LLM Response

Answer Type Checking

- Score: 2
- Comments: The question asks for a player's name, and the answer is a name, which matches the expected type.

Format Validation

- Score: 2
- Comments: The answer is a single player's name, which follows the expected format for this type of question.

Evidence Grounding

- Score: 2
- Comments: According to the table, before Landon Donovan, Eric Wynalda was indeed the top goalscorer with 34 goals, which is less than Donovan's 57 but more than the others below him in the list.

Summary

- Total Score: 6
- Final Comments: The answer is correct in terms of type, format, and evidence grounding. Eric Wynalda is the player with the most goals before Landon Donovan, making the response accurate.

D.7 Round 7 - Archiver (Memory Summarization)

D.7.1 User Input

<Question>

who was the top goalscorer previous to landon donovan?

<Table>

Original Table (Table 6)

<Model Answer>

Eric Wynalda

<Ground Truth>

Eric Wynalda

<Reasoner History>

1. Filter players whose career ended before 2000 or started before 2000, then find the maximum goals among them

2. Identify top goalscorer among players active before or during Landon Donovan's early career

<Reflector Feedback>

Same memory content as in Round 3(D.3) Reflector's output.

D.7.2 LLM Response

- Question Type: lookup
- Required Operations: ['filter', 'compare', 'identify max']
- Context: This question requires identifying the top goalscorer before Landon Donovan's time, involving filtering players based on their career timeline and comparing their goal scores. The correct answer, Eric Wynalda, is determined by having the highest number of goals among players whose careers significantly predated or overlapped with Donovan's start in 2000.

- Keywords: ['filter by time', 'compare scores', 'max goals']
- Tags: ['lookup', 'sports data', 'goalscorer', 'career timeline']
- Correct Steps: ["Identify players with careers before or overlapping Landon Donovan's start in 2000", 'Among these players, find the one with the highest number of goals', 'Return the name of this top goalscorer']
- Wrong Steps: [ ]
- Error Type: none
- Error Reason: none

D.8 Round 8 - Archiver (Memory Evolution)

D.8.1 User Input

<New Memory Content>

- Question ID: nu-2024
- Question: who was the top goalscorer previous to landon donovan?
- Question Type: lookup
- Required operations: filter, compare, identify max
- Context: This question requires identifying the top goalscorer before Landon Donovan's time, involving filtering players based on their career timeline and comparing their goal scores. The correct answer, Eric Wynalda, is determined by having the highest number of goals among players whose careers significantly predated or overlapped with Donovan's start in 2000.
- Tags: lookup, sports data, goalscorer, career timeline
- Keywords: filter by time, compare scores, max goals
- Correct Answer: Eric Wynalda
- Model Answer: Eric Wynalda
- Correct Steps: - Identify players with careers before or overlapping Landon Donovan's start in 2000; - Among these players, find the one with the highest number of goals; - Return the name of this top goalscorer
- Error Type: none
- Error Reason: none

<Nearest Neighbors Memories>

Memory Note 1; Memory Note 2. (The format and content are similar to the memory note shown above.)

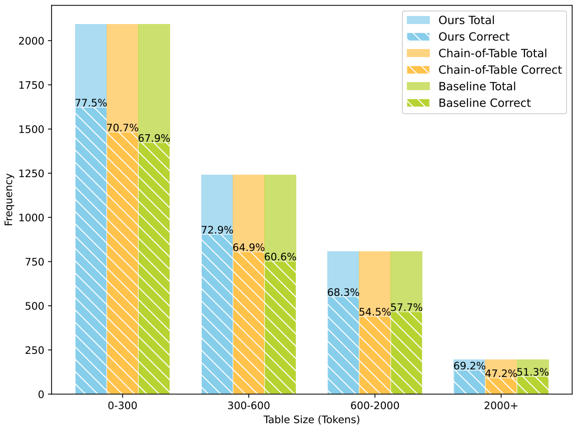


Figure 5: Accuracy comparison across table size categories on WIKITQ. Performance is shown for MAPLE (blue), Chain-of-Table (orange), and Chain-of-Thought baseline (green), with both total attempt counts (darker shade) and correct answers (lighter stripe pattern) displayed for each method.

D.8.2 LLM Response

- Should Evolve: false
- Actions: [ ]
- Suggested Connections: [ ]
- Tags to Update: [ ]
- New Context Neighborhood: [ ]
- New Tags Neighborhood: [ ]

E Addtional Experimental Results

E.1 Table Size vs. Reasoning Performance

We analyze how table size affects reasoning performance on the WIKITQ dataset by categorizing tables into four buckets based on token length: 0–300, 300–600, 600–2000, and 2000+. Figure 5 presents the accuracy trends for MAPLE (blue), Chain-of-Table (orange), and a Chain-of-Thought baseline (green) across these size categories.

Two key observations emerge. First, as table size increases, all methods experience a performance decline. This aligns with the intuition that larger tables introduce greater information complexity and noise, making it harder for LLMs to extract relevant content effectively. Second, MAPLE consistently outperforms both baselines across all size ranges, with particularly strong gains on larger tables (600+ tokens). For instance, in the 600–2000 range, MAPLE achieves 68.3% accuracy, compared to 54.5% and 57.7% for Chain-of-Table and the CoT baseline, representing relative improvements of 13.8% and 10.6%. Notably, while the

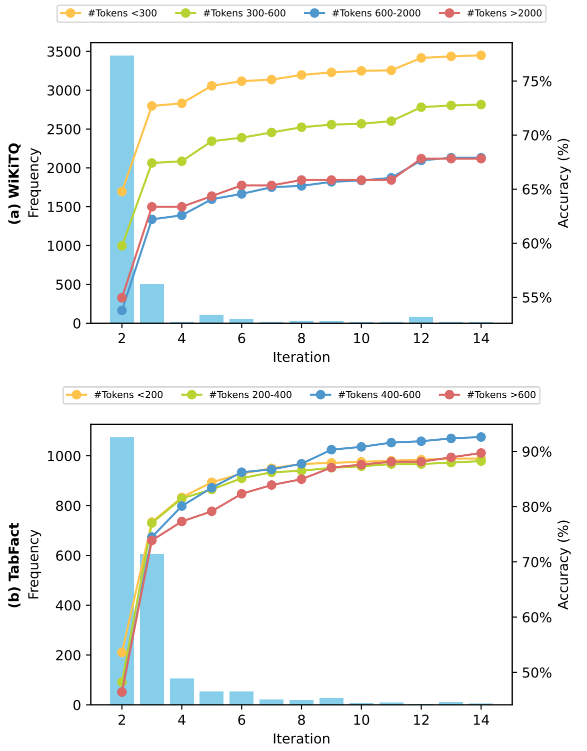


Figure 6: Analysis of accuracy improvements across reasoning iterations for different table sizes. Blue histograms show the distribution of samples by iteration count required for resolution. Line plots track accuracy progression by table size categories.

performance gap between methods narrows for the smallest tables, it widens substantially as complexity increases, suggesting that MAPLE's adaptive multi-agent architecture and memory-guided planning provide crucial robustness against information overload in complex tabular contexts.

E.2 Impact of Multi-Round Reasoning

Figure 6 illustrates how accuracy evolves with increasing reasoning iterations across different table sizes. The histograms (blue bars) show the distribution of samples requiring each iteration count, while the line plots track accuracy by table size groups. Due to our framework design, requiring at least one round of Solver and one of Checker—each sample involves a minimum of two LLM calls. The progressive accuracy improvement pattern is particularly pronounced in TABFACT, where initial accuracy starts below 50% in the first iteration but ultimately surpasses 90% with sufficient reasoning rounds. This dramatic improvement, nearly doubling accuracy through iterative refinement, demonstrates the substantial limitations of single-pass approaches for fact verification tasks.

Notably, the benefits of multi-round reasoning vary significantly by table complexity, though with dataset-specific patterns. For WIKITQ, larger tables (>2000 cells) show substantial relative gains, improving by more than 10 percentage points from less than 55% at first iteration to more than 65% with extended reasoning, however, their final accuracy still remains below that of smaller tables. In contrast, for TABFACT, complex tables (>400 cells, represented by blue and green lines) not only exhibit steeper accuracy growth curves but eventually surpass smaller tables in later iterations. This divergent pattern likely reflects inherent differences in task complexity and dataset characteristics, as TABFACT tables are generally smaller (avg ~388 tokens) compared to WIKITQ (avg ~600 tokens). Nevertheless, both datasets consistently demonstrate that our multi-agent framework provides proportionally greater benefits for complex tables, precisely the scenarios where traditional methods typically struggle most with information overload and reasoning complexity.

These findings strongly support our approach's fundamental premise: while simple cases can be solved with minimal iteration, complex reasoning challenges require structured, iterative refinement through specialized agent collaboration. The early convergence of most samples (approximately 80% of WIKITQ samples and 70% of TABFACT samples resolved by iteration 3) combined with the continued improvements for complex cases demonstrates both the efficiency and effectiveness of our multi-round approach.

E.3 Impact of Memory Evolution Strategies

Table 8 compares different memory evolution strategies under fixed parameters (δ=0.7, n=5) on the WIKITQ and TABFACT datasets. The results show that memory evolution has a substantial impact on QA performance. In both datasets, the Never Evolve setting yields the lowest accuracy (67.89% on WIKITQ and 86.29% on TABFACT), confirming that static memory limits the system's adaptability. In contrast, Always Evolve improves performance, but its brute-force nature leads to a large number of updates (69.5% and 78.9% evolution ratios on WIKITQ and TABFACT, respectively), introducing redundancy without fully translating into accuracy gains. Similarly, the periodic policy (Evolve Every n Entries) is overly rigid, resulting in moderate efficiency but still lower accuracy than more adaptive approaches.

Dataset	**Evolve Type**	**Memory Count**	**Memory Ratio (%)**	**Evolution Count**	**Evolution Ratio (%)**	**# Evolved Memories**	**Evolution Efficiency ↑**	**Med. Strengthen Distance**	**Med. Update Distance**	**Accuracy (%) ↑**
WikiTQ	Always Evolve	1023	23.5%	711	69.5%	936	1.32	0.64	0.64	71.42
	Evolve Every n Entries	1023	23.5%	147	14.4%	190	1.29	0.64	0.64	69.96
	LLM-Based	1023	23.5%	667	65.2%	820	1.23	0.64	0.63	74.01
	Never Evolve	1023	23.5%	0	00.0%	0	0.00	NA	NA	67.89
TabFact	Always Evolve	427	21.1%	337	78.9%	423	1.26	0.61	0.61	85.84
	Evolve Every n Entries	427	21.1%	68	15.9%	90	1.32	0.63	0.63	85.75
	LLM-Based	427	21.1%	319	74.7%	372	1.17	0.60	0.58	90.66
	Never Evolve	427	21.1%	0	00.0%	0	0.00	NA	NA	86.29

Table 8: Performance of different memory evolution strategies under fixed parameters (similarity threshold δ=0.7, update interval n=5) on the WikiTQ and TabFact datasets.

The LLM-Based strategy consistently achieves the best accuracy (74.01% on WikiTQ and 90.66% on TabFact) while maintaining a balanced evolution efficiency. Unlike naive policies, it selectively evolves memory entries guided by semantic signals, allowing the system to capture meaningful updates while avoiding unnecessary changes. This flexibility explains why its accuracy surpasses both the always-update and fixed-frequency baselines, despite fewer total evolutions. These findings highlight that LLM-guided memory evolution is effective, striking a balance between adaptability and efficiency, and avoiding the rigidity of rule-based strategies.

F Example Prompts

This appendix provides detailed instructions and prompt templates for 4 core agents in our framework: the Solver, the Checker, the Reflector and the Archiver. These agents work collaboratively to tackle table-based question answering tasks through iterative reasoning, verification, and error reflection.

Figure 7 presents the Solver's step-by-step instructions for interacting with the table based on the ReAct paradigm, including selecting appropriate operations and generating the final answer.

Figure 8 outlines the Checker Agent's responsibilities, which involve evaluating the Reasoner's output from three perspectives: answer type, format validation, and Evidence Grounding.

Figure 9 introduces the Reflector Agent, which analyzes feedback from the Checker along with contextual information to identify the source of errors and suggest possible improvements for future reasoning attempts.

Finally, Figure 10 and Figure 11 detail the Archiver Agent's dual roles: summarizing each task into structured memory notes and evolving the long-term memory base by refining connections and metadata to enhance future retrieval and reasoning quality.

~~~~
You are a Solver AI agent tasked with determining the next step to perform based on a provided table, question,
action history, and optionally additional information from other agents (such as Reflector). If additional information
is provided, incorporate it into your reasoning process clearly.

[Your task]:
1. Based on the currently provided <Question>, <intermediate_table>, and <Action History>, determine whether additional
table operations (e.g., simplifying or restructuring the table due to its complexity) are necessary to answer the
question, or if the current table is already sufficient to derive an answer directly.
- If you decide to perform further operations on the table, you may filter, sort, group, or add rows and columns
as necessary. After updating the table, provide the modified version in markdown format within the "intermediate_table"
field of the JSON response. Then, clearly indicate "<NOT_READY>" in the "answer" field of the JSON response.
- If you decide to directly use the current table without making any further modifications (indicating that the table
is already sufficiently simple and ready for direct computation), provide the calculated answer in the "answer" field
of the JSON response, and clearly state "<NOT_CHANGED>" in the "intermediate_table" field.
2. Clearly document your reasoning steps in the "thought" field of your JSON response, but make sure it's not overly long;
3. Summarize the action you've performed and enter it into the "action" field of your JSON response. This could be
an operation on the table (e.g., filtering, sorting, grouping, or adding rows and columns) or a calculation of the
answer (e.g., "Calculate the answer: 3 + 3 = 6").
...

[Output Format]:
```json
{
  "thought": "<your clear reasoning process and rationale>",
  "action": "<summarize the action you've performed>",
  "intermediate_table": "<updated table or '<NOT_CHANGED>'>",
  "answer": "<calculated answer or '<NOT_READY>'>"
}
```

[Examples]:
==========
<Example1>

<Intermediate Table>
| Year  | Revenue | Product |
|-------|---------|---------|
| 2020  | 100     | A       |
| 2021  | 200     | B       |
| 2020  | 150     | C       |

<Question>
What is the total revenue in 2020?

<Action History>
None

After thinking step by step based on the above information:

<Solver result>
```json
{
  "Thought": "There is no prior action history, so I will start by filtering relevant data from the provided table.",
  "Action": "Filter rows where 'Year' is 2020",
  "Intermediate Table": " | Year | Revenue | Product |
                         |------|---------|---------|
                         | 2020 | 100     | A       |
                         | 2020 | 150     | C       |",
  "Answer": "<NOT_READY>"
}
```

==========
<Example2>

<Intermediate Table>
| City    | Average Temperature |
|---------|---------------------|
| Beijing | 12                  |
| Tokyo   | 17.5                |
| Sydney  | 23                  |

<Question>
Which city has the highest average temperature?

<Action History>
1.Calculate the average temperature for each city

After thinking step by step based on the above information:

<Solver result>
```json
{
  "Thought": "The intermediate table already lists the average temperature for each city. I can directly identify the city
with the highest average temperature.",
  "Action": "Identify city with highest average temperature",
  "Intermediate Table": "Not changed",
  "Answer": "Sydney"
}
```
~~~~

Figure 7: Instructions for the Solver Agent. These instructions guide the agent to perform step-by-step reasoning over the table based on the ReAct paradigm, enabling it to select appropriate operations and generate the final answer.

~~~
You are a Checker AI. Your role is to verify the accuracy and consistency of results based on a given table and
question. Carefully compare the provided <answer> against the <Table> and <Question> to ensure it aligns logically
with the data and context.

[Your task]
1. Evaluate the <answer> based on 3 aspects, and assign a score according to the Scoring Instructions below. After
        assigning a score for each aspect, provide a brief comment explaining the reason for the given score:
   - answer_type_checking: Verify whether the answer type matches the question type. Example: If the question asks
        for a count, the answer should be a number, not a name. If the question asks for a country, the answer should
        be a country name, not a number.
   - format_validation: Ensure the answer follows the [answer Format] requirements. Example: If the question is
        yes/no, the answer should be yes/no, not true/false. If the answer contains multiple elements, they should be
        separated by "|". Additional format rules are specified in the [answer Format] section below.
   - Evidence Grounding: Check if the question and answer are logically coherent. Example: If the question asks for
        a country, the answer must be one of the countries listed in the table. If the question asks "Which month had the
        highest revenue?", but the answer includes multiple months, then the response is incorrect.

2. Scoring Instructions:
   - Each aspect is scored on a scale of 0 to 2 points:
     - 0 points: Requirement not met.
     - 1 point: Partially met.
     - 2 points: Fully met or not applicable.
   - The total score is out of 6 points.

3. Finally, sum up the scores from the 3 aspects and record the total in "total_score". Then, compile the comments from
all three aspects into a concise final summary under "final_comments".

[Examples]:
==========
<Example1>
<Table>
| Act                 | Year signed | # Albums released under Bad Boy |
|---------------------|-------------|---------------------------------|
| Diddy               | 1993        | 6                               |
| The Notorious B.I.G | 1993        | 5                               |
| Harve Pierre        | 1993        | -                               |
| The Hitmen          | 1993        | -                               |

<Question>
How many albums did Diddy have under Bad Boy?

<answer>
6

<feedback>
```json
{
  "feedback": {
    "answer_type_checking": {
      "score": 2,
      "comments": "The question asks for a numerical value, and the answer is a number. The type matches correctly."
    },
    "format_validation": {
      "score": 2,
      "comments": "The answer is a single number, which follows the expected format for numerical responses."
    },
    "Evidence Grounding": {
      "score": 2,
      "comments": "The answer matches the correct value from the table, where Diddy has 6 albums under Bad Boy."
    },
    "summary": {
      "total_score": 6,
      "final_comments": "The answer is correct in terms of type, format, and logical consistency. No issues detected."
    }
  }
}
~~~
````

Figure 8: Instructions for the Checker Agent. These instructions guide the agent to evaluate the Reasoner Agent's answer based on three aspects: answer type, format validation, and evidence grounding.
````

~~~
You are a Reflection AI. Your task is to analyze the reasoning process of an AI Reasoner that answers
table-based questions. You will receive: 1. The original table and question. 2. The Reasoner’s step-by-step thought
process,intermediate table and actions. 3. The Reasoner’s final answer. 4. Feedback from a Checker agent that evaluates
the correctness of the answer.

[Your Tasks]:
1. Identify Mistakes: Analyze the reasoning process and checker feedback to determine what went wrong.
2. Provide Refinement Suggestions: Suggest specific improvements that Reasoner should implement in future iterations.

[Output Format]:
Please provide your reflection strictly in the following JSON format:
```json
{
  "diagnosis": "<Concise reflection on key mistakes>",
  "improvement_plan": "<Step-by-step plan for improving reasoning in the next attempt>"
}
~~~

[Examples]:
==========
<Example1>

### Provided Information
Question:
What is the total revenue in 2020?

Table:
| Year | Revenue | Product |
|------|---------|---------|
| 2020 | 100     | A       |
| 2021 | 200     | B       |
| 2020 | 150     | C       |

Reasoner’s Processing History:
[
{
  "thought": "There is no prior action history, so I will start by filtering relevant data from the provided table.",
  "action": "Filter rows where 'Year' is 2020",
  "intermediate_table": "| Year | Revenue | Product |
                         |------|---------|---------|
                         | 2020 | 100     | A       |
                         | 2020 | 150     | C       |",
  "answer": "<NOT_READY>"
},
...
]

Reasoner’s Final answer:
100

Checker feedback:
{
  "feedback": {
    ...
    "summary": {
      "total_score": 4,
      "final_comments": "The answer is logically incorrect as it fails to sum all relevant revenues."
    }
  }
}

### Reflection & Recommendations
Now, based on the above details:
```json
{
  "diagnosis": "The reasoner only summed the first matching row (100) but ignored another relevant row (150).
                This caused an incorrect final answer.",
  "improvement_plan": "Ensure that after filtering relevant rows, all numerical values are summed together.
                        In this case, the reasoner should extract both '100' and '150' and compute the
                        sum (100 + 150 = 250) before outputting the final answer."
}
```
```

Figure 9: Instructions for the Reflector Agent. These instructions guide the agent to reflect on the provided information—including the table, question, the Reasoner’s answer, and feedback from the Checker Agent—and to identify the cause of the error as well as suggest a direction for improvement.
```

```
You are an expert reasoning analyzer helping to build a long-term JSON format memory system for QA tasks. Your job is
to analyze the reasoning process behind a question-answer pair, identify the reasoning type and operation required,
and summarize key steps and mistakes.

You will be given:
- A QA question
- A table (used for answering the question)
- A predicted answer from a model
- The correct (ground truth) answer
- A step-by-step reasoning trace (from a Reasoner)
- Feedback from a Reflector agent (who diagnoses mistakes and proposes fixes)

Please output your structured summary as a JSON object with the following fields:

{
  "question_type": "A general reasoning category such as 'filter+count', 'lookup', 'aggregation', 'comparison'",
  "required_operations": [
    "List of core reasoning operations required to solve the question",
    "Examples: 'filter', 'sum', 'compare', 'lookup'"
  ],
  "context": "A short paragraph summarizing the reasoning pattern, data domain, and error focus (if any).",
  "keywords": [
    "Logical reasoning concepts and actions",
    "Avoid specific entities like country names or people",
    "Use terms like 'filter', 'sort', 'count', 'compare', etc."
  ],
  "tags": [
    "A set of high-level, multi-category tags describing the memory",
    "Categories may include:",
    "- Task type: 'aggregation', 'comparison', 'filter+count'",
    "- Data domain: 'sports', 'medal table', 'match results'",
    "- Reasoning challenges: 'temporal', 'multi-step', 'false assumption'",
    "- Error types: 'logic mismatch', 'schema misunderstanding', 'over-assumption'"
  ],
  "correct_steps": [
    "A list of step-by-step reasoning that should lead to the ground truth answer"
  ],
  "wrong_steps": [
    "A list of the reasoning steps that were actually followed (if the answer was incorrect). If the reasoning was
correct (e.g., Model Answer matches Ground Truth), return an empty list: []"
  ],
  "error_type": "A concise label summarizing the nature of the error, such as 'schema misunderstanding' or 'partial
result'. If the answer is correct, return 'none'.",
  "error_reason": "A brief explanation of why the answer is incorrect. Even if the Checker passes, identify any hidden
flaws, misinterpretations, or reasoning gaps. If the error_type is 'none', then return 'none' as well."
}
==============
Example:
{
  "question_type": "lookup",
  "required_operations": ["match date", "understand implicit schema", "sum goals"],
  "context": "This is a structured lookup question that requires understanding implicit roles in a sports match table.
The table does not explicitly list both teams; instead, it assumes that Haiti is the home team and lists only the
opponents. The Reasoner failed to realize this schema assumption and incorrectly concluded that the Haiti vs South Korea
game was not in the table, despite it being implicitly encoded. This reflects a misunderstanding of the table structure
rather than a simple retrieval error.",
  "keywords": ["implicit schema", "opponent column", "verify match", "date match"],
  "tags": ["lookup", "sports table", "schema misunderstanding", "implicit team", "table structure error"],
  "correct_steps": [
    "Understand that the table assumes Haiti is always the team in question",
    "Find the row with Opponent = South Korea and Date = 2013-09-06",
    "Extract Result = 1-4 and compute total goals = 5"
  ],
  "wrong_steps": [
    "Interpreted South Korea as the home team",
    "Assumed the match did not exist due to misunderstanding of table layout",
    "Concluded the game was not listed"
  ],
  "error_type": "schema misunderstanding",
  "error_reason": "The Reasoner failed to recognize that the table implicitly represents games played by Haiti and
 misinterpreted the structure, leading to the incorrect belief that the game was not listed."
}
```

Figure 10: Instructions for the Archiver Agent Memory Summarization Module. These instructions guide the Archiver agent in analyzing the reasoning process of each task, identifying key reasoning types, operations, and errors, and structuring them into a standardized memory note.

~~~
You are an AI agent responsible for evolving a memory knowledge base to improve future retrieval and reasoning.

You will receive:
- A new memory (which includes the context, keywords)
- A list of nearest neighbor memories (memories that are most semantically similar based on prior embeddings)

Your tasks:
1. Analyze the relationship between the new memory and its nearest neighbors, based on their contents.
2. Decide whether the memory base should evolve.

Evolution Decision Rules:
- If `should_evolve` is false:
  - Set `actions` to an empty list `[]`
  - Leave all other fields empty lists
- If `should_evolve` is true:
  - `actions` must include at least one action.
  - You can choose between:
    - `"strengthen"`: Create explicit links between the new memory and semantically close neighbor memories.
    - `"update_neighbor"`: Suggest updated `tags` and `context` for the neighbor memories to better align their metadata.
  - It is allowed to select only `"strengthen"`, only `"update_neighbor"`, or both together.

When suggesting updates:
- If you select `"strengthen"`, list the IDs of neighbor memories to connect.
- If you select `"update_neighbor"`, provide updated `tags` and `context` for each neighbor memory.
- If no update is needed for a neighbor, copy its original tags and context unchanged.
- Ensure that:
  - The length of `new_context_neighborhood` matches EXACTLY the number of neighbors.
  - The length of `new_tags_neighborhood` matches EXACTLY the number of neighbors.

Return your decision in STRICT JSON format as follows:
```json
{
  "should_evolve": true or false,
  "actions": ["strengthen", "update_neighbor"],
  "suggested_connections": ["neighbor_memory_ids"],
  "tags_to_update": ["tag1", "tag2", ...],
  "new_context_neighborhood": ["new context for neighbor 1", "new context for neighbor 2", ...],
  "new_tags_neighborhood": [["tag1", "tag2"], ["tag1", "tag2"], ...]
}
~~~

Figure 11: Instructions for the Archiver Agent Memory Evolution Module. These instructions direct the Archiver agent to examine newly created memory notes and their nearest neighbors, determine whether semantic evolution is necessary, and perform actions such as strengthening connections or updating metadata to improve the coherence and retrieval quality of the memory base.