INTRINSIC MEMORY AGENTS: HETEROGENEOUS MULTI-AGENT LLM SYSTEMS THROUGH STRUCTURED CONTEXTUAL MEMORY

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-agent systems built on Large Language Models (LLMs) show exceptional promise for complex collaborative problem-solving, yet they face fundamental challenges stemming from context window limitations that impair memory consistency, role adherence, and procedural integrity. This paper introduces Intrinsic Memory Agents, a framework that addresses these limitations through agent-specific memories that evolve intrinsically with agent outputs, maintaining role-aligned memory that preserves specialised perspectives while focusing on task-relevant information. We evaluate our approach against six published memory designs and a no-memory control on seven benchmark datasets spanning knowledge reasoning (FEVER, HotpotQA), embodied action (ALFWorld, SciWorld, BabyAI), planning (PDDL) and interactive fiction (Jericho), at ten seeds per condition. Intrinsic memory is strongest on knowledge reasoning, taking the top three places on both FEVER and HotpotQA. A hand-written memory schema scores higher than an automatically generated one on five of the seven benchmarks. An additional evaluation is performed on a complex data pipeline design task, and we demonstrate that our approach produces higher quality designs across 5 metrics: scalability, reliability, usability, cost-effectiveness, and documentation, plus additional qualitative evidence of the improvements. Our findings suggest that addressing memory limitations through intrinsic approaches can improve the capabilities of multi-agent LLM systems on knowledge reasoning and structured planning tasks.

1 INTRODUCTION

Recent advances in large language models (LLMs) have enabled their application as autonomous or semi-autonomous agents capable of complex reasoning and decision-making (Huang et al., 2024). Multi-agent LLM systems, where multiple LLM instances interact to solve problems collaboratively, have shown particular promise for tasks requiring diverse expertise (Park et al., 2023; Qian et al., 2025). These systems leverage the complementary capabilities of specialized agents to address challenges that would be difficult for single-agent approaches to resolve effectively.

Despite their theoretical advantages, multi-agent LLM systems face several implementation challenges that limit their practical effectiveness, from coordination overhead, to the consistency in role adherence among the agents (Li et al., 2024c). Most critically, the fixed-size context windows of LLMs restrict their ability to maintain long-term conversational context, an issue that is exacerbated in multi-agent frameworks with multiple agents in a single conversation. This leads to issues such as perspective inconsistency, forgetting key requirements, and procedural drift. Current solutions such as Retrieval-Augmented Generation (RAG) (Lewis et al., 2020; Gao et al., 2024) and agentic memory approaches (Packer et al., 2024; Xu et al., 2025; Chhikara et al., 2025) are designed for single-agent and user interaction scenarios, which do not account for the volume of information growing with the number of agents.

To address these challenges, we introduce Intrinsic Memory Agents, a novel multi-agent architecture that uses agent-specific memories aligned with conversational objectives. Unlike previous approaches, our system updates memories that are specific to each agent, ensuring heterogeneity and memories that reflect both historical context and recent developments while preserving agent-specific perspectives.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

The intrinsic nature of memory updates, derived directly from agent outputs rather than external summarization, ensures unique memories that maintain consistency with agent-specific reasoning patterns and domain expertise. We evaluate our approach through benchmarking and through a specific data pipeline design case study to show its practical usage. On the case study the approach yields improvements in solution quality on all five judged metrics without increasing the number of conversation turns. On the benchmarks, Intrinsic Memory places highest on the knowledge benchmarks and comes joint first when ranked across all benchmarks.

In summary, our contributions are three-fold. Firstly, Intrinsic Memory updates are derived from agent outputs rather than external summarisation. Secondly, the memory is agent-specific, creating independent memories maintained for each agent to preserve perspective autonomy. Thirdly, memory is written into a structured template format which keeps the memory maintainable and role-specific.

2 Related work

Recent years have seen significant progress in the development of multi-agent systems powered by LLMs. These systems have been applied in various domains, such as software development, scientific experimentation, gaming, and social simulation (Li et al., 2024c). For example, in software development, multi-agent systems enable concurrent consideration of architectural design, security, user experience, and performance optimization (Hong et al., 2024). We review the problem of memory from general to specific use cases. First, we introduce works for memory in large language models, and cover the two general approaches: retrieval from an external storage, and memory that the model writes for itself. Then, we investigate agentic memory, where models maintain their own memory. Finally, we look at multi-agent systems, where many current solutions have several agents sharing a single memory system.

Memory for large language models. Memory is what allows an agent to maintain context, learn from past interactions and act on what it has learned. It supports conversational consistency and role-playing in single agent systems (Zhang et al., 2024), and coordination, communication and collaborative problem-solving in multi-agent systems (Guo et al., 2024). The simplest form of memory is the context window itself. Commercial LLMs such as GPT-4o (OpenAI, 2024) and Claude (Anthropic, 2024) are able to process large contexts of over 1 million tokens in its context window. However, the hard limit of the context window size remains, and increasing the context length does not necessarily increase reasoning or learning capabilities of the LLM (Li et al., 2024b).

Memory that must persist beyond a context window is conventionally stored externally, either in a database and retrieved on demand using RAG (Lewis et al., 2020; Gao et al., 2024), or in the form of persistent markdown files used by coding harnesses such as Claude Code (**?**). This lifts the capacity limit at the cost of retrieval noise, the engineering of the retrieval system itself, latency and storage (Asai et al., 2024; Yu et al., 2024). Those costs scale with the volume of text to be indexed, which in a multi-agent system grows with the number of agents (Li et al., 2024a), creating expensive maintenance where memory is most useful.

Agentic memory Alternatively, the model can maintain its own memory. Agentic memory condenses conversation history into sequential or hierarchical summaries that capture key decisions (Wang et al., 2025a; Chen et al., 2024), sometimes storing those summaries for later retrieval (Xu et al., 2022) or paging them in and out of context under the model's own control (Packer et al., 2024; Xu et al., 2025). Reflexion turns a failed attempt into a verbal lesson carried into the next one (Shinn et al., 2023), ExpeL distils insights from past trajectories (Zhao et al., 2024), Agent Workflow Memory induces reusable workflows (Wang et al., 2025b), and Memp treats such procedures as a memory type in their own right (Fang et al., 2026). MIPRO and GEPA optimise the instructions of a multi-stage LLM program (Opsahl-Ong et al., 2024; Agrawal et al., 2026), while Dynamic Cheatsheet and Agentic Context Engineering let the model accumulate and curate notes of its own, both reporting that a self-written, self-revised context beats a fixed one (Suzgun et al., 2025; Zhang et al., 2026). These designs share the premise that a memory may hold learned instructions rather than purely factual knowledge, but each accumulates across episodes and serves a single agent, whereas the Intrinsic Memory proposed here is maintained separately per agent and, in its default configuration, reset at each task.

Memory for multi-agent systems. Most of the memory designs that multi-agent frameworks use were not written for multi-agent settings. ChatDev keeps a phase-local dialogue history (Qian et al., 2024) and MetaGPT passes structured artefacts between roles (Hong et al., 2024), while Voyager's skill library (Wang et al., 2023), the retrieval and reflection stream of Generative Agents (Park et al., 2023) and MemoryBank's forgetting-curve store (Zhong et al., 2024) were each proposed for a single agent and adopted as multi-agent baselines afterwards. G-Memory is the closest work to ours, maintaining a three-tier hierarchy of insight, query and interaction graphs shared across a team and supplying the harness in which these designs can be compared (Zhang et al., 2025). As these approaches all share memory between the agents in the system, the context retrieved is the same whichever role asks for it. That homogeneity works against the property multi-agent systems are built for, the integration of distinct specialised perspectives toward a cohesive solution (He et al., 2025), and recent surveys still list multi-agent memory as an open frontier (Hu et al., 2025).

Our proposed Intrinsic Memory Agents similarly uses an agentic memory approach to summarise and store information. There are three main properties that Intrinsic Memory addresses compared to existing approaches. First, the memory is written by each agent themselves instead of an external summarises. Second, we introduce heterogeneous memory for each agent rather than a single shared memory across the system. And finally, we utilise a structured templated approach to each agent ensuring cohesive memory throughout. This addresses the limitations of existing memory mechanisms by ensuring that each agent maintains its own memory, reflecting both historical context and new information while maintaining heterogeneous agent-specific perspectives and expertise.

3 Intrinsic Memory Agents

The various agentic memory approaches are all designed in single-agent scenarios to remember crucial details when interacting with an end-user. Due to the multi-turn long conversations between agents, a direct implementation of single-agent agentic memory becomes complicated and resource-intensive, with each agent requiring retrieval systems and distinct contextual updates.

We propose Intrinsic Memory Agents, a framework for multi-agent LLM systems that maintains agent-specific memories aligned with conversational objectives. Figure 1 illustrates the architecture of our Intrinsic Memory Agents framework. In this approach a query is made by the user, the first agent makes a comment based on its role description, the conversation is updated, followed by a memory update for the agent that commented, there is a check for consensus and the cycle starts again. The context in this case is made up of both the agent's intrinsic memory and the conversation, meaning that as the conversation continues the agents increasingly diverge in their interpretation of that context.

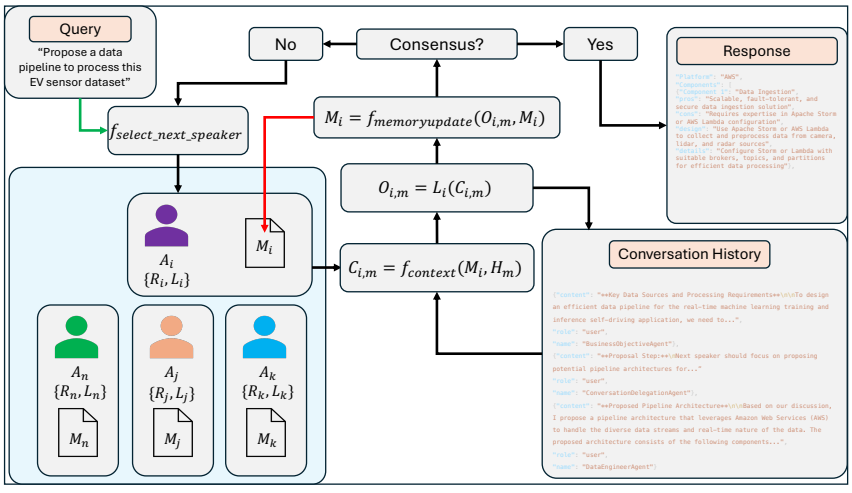

Figure 1: Intrinsic Memory Agents Framework. For n agents and m conversation turns, each agent A_n contains its own role description R_n and language model L_n. Its memory $M_{n,m}$ is updated based on the input context $C_{n,m}$ and output $O_{n,m}$.

3.1 Framework Definition

Let us define the multi-agent system $\mathcal{A} = \{A_1, A_2, ..., A_N\}$ consisting of N agents. Each agent $A_n = \{R_n, M_n, LLM_n\}$ is characterized by a role specification R_n that defines the agent's expertise domain and objectives, a memory M_n that evolves throughout the conversation, and an LLM instance LLM_n, which may share parameters between agents.

The conversation consists of a sequence of turns $T = t_1, t_2, ..., t_M$ where each turn t_m involves an agent selection function $\sigma(t_m) \rightarrow A_n$ that determines which agent speaks, an input context $C_{n,m}$ constructed for the selected agent, an output $O_{n,m}$ generated by the selected agent, and a memory update operation for the selected agent.

Critically, our framework separates the input context construction and memory update processes, allowing for agent-specific memory maintenance while preserving a shared conversation space.

3.2 Memory Update mechanism

For each agent in the system, we maintain a memory M_n that evolves over time. Let $M_{n,m}$ represent the memory of agent n after m conversation turns. The memory update process works as follows:

Agent A_n receives input context $C_{n,m}$ consisting of relevant conversation history H_m and previous memory $M_{n,m-1}$,

$$C_{n,m} = f_{\text{context}}(H_m, M_{n,m-1}); \tag{1}$$

and agent A_n generates output $O_{n,m}$ using the underlying LLM L_n,

$$O_{n,m} = L_n(C_{n,m}). \tag{2}$$

Then with the generated output $O_{n,m}$ and the previous memory $M_{n,m-1}$, we update the slot content using a memory update function,

$$M_{n,m} = f_{\text{memory_update}}(M_{n,m-1}, O_{n,m}). \tag{3}$$

The memory update function $f_{\text{memory_update}}$ is implemented as a prompted LLM operation. Specifically, for the previous memory $M_{n,m-1}$ at turn $m-1$ and agent output $O_{n,m}$ at turn m, the update function constructs the prompt as shown in Figure 7. The LLM's response to this prompt becomes the updated memory $M_{n,m}$. The context construction function f_{context} presented in equation 1 determines what information is provided to an agent when generating a response. The algorithm takes the existing conversation history and agent memory, appending both to the context and using the remaining tokens to include the rest of the conversation history. The full algorithm pseudo-code is displayed in the Appendix A Algorithm 1.

This algorithm prioritizes: 1) The initial task description to maintain objective alignment. 2) The agent's structured memory to preserve role consistency. 3) The most recent conversation turns to maintain immediate context.

By prioritizing memory inclusion over exhaustive conversation history, the algorithm ensures that agents maintain role consistency and task alignment even when conversation length exceeds context window limitations. We test three variations on the Intrinsic memory template, a manual hand-crafted template $\mathcal{T}_{\text{hand}}$, a generic template $\mathcal{T}_{\varnothing}$, and an LLM self-generated template $\mathcal{T}_{\text{LLM}}$. Details of the variations can be found in appendix B. Two configurations are tried: resetting the memory at every task, and carrying it forward across the benchmark's tasks.

4 Empirical results

To evaluate our approach, we test our memory agents against seven benchmarks as shown in Table 1: PDDL (Planning Domain Definition Language), FEVER (Fact Extraction and VERification) Thorne et al. (2018), ALFWorld Shridhar et al. (2021), SciWorld (Wang et al., 2022), HotpotQA (Yang et al., 2018), BabyAI (Chevalier-Boisvert et al., 2019), and Jericho (Hausknecht et al., 2020). PDDL involves structured planning tasks from AgentBoard (Ma et al., 2024), where the agents generate executable plans for abstract problem domains. FEVER and HotpotQA are datasets for evidence-based claim verification, requiring agents to retrieve and reason over textual evidence fetched via

Table 1: The seven quantitative benchmarks used. The trial budget is the per-task step limit. Tasks is the number of tasks in each dataset.

Benchmark	Domain	Tasks	Trials	Metric
FEVER	Knowledge	200	30	Exact match
HotpotQA	Knowledge	200	30	Exact match
ALFWorld	Embodied	134	50	Success rate
SciWorld	Embodied	90	30	Progress rate
BabyAI	Embodied	200	30	Success rate
PDDL	Planning	60	30	Progress rate
Jericho	Game	55	100	Progress rate

Table 2: Mean reward per memory system, with the standard error taken across seeds ($n = 10$ throughout). Best per benchmark in bold. Backbone `gpt-oss-120b`. On Jericho, there is no strong significance between the memory systems.

Memory	FEVER	HotpotQA	ALFWorld	SciWorld	BabyAI	PDDL	Jericho
No memory	0.694 ± 0.006	0.461 ± 0.005	0.145 ± 0.002	0.807 ± 0.004	0.568 ± 0.005	0.726 ± 0.005	0.029 ± 0.001
ChatDev	0.696 ± 0.005	0.462 ± 0.004	0.145 ± 0.002	0.803 ± 0.005	0.560 ± 0.007	0.715 ± 0.006	0.031 ± 0.001
MetaGPT	0.696 ± 0.003	0.466 ± 0.003	$\mathbf{0.157} \pm 0.002$	0.819 ± 0.003	0.573 ± 0.005	0.735 ± 0.008	0.031 ± 0.001
Voyager	0.704 ± 0.004	0.467 ± 0.005	0.155 ± 0.002	0.835 ± 0.004	0.572 ± 0.007	0.728 ± 0.007	0.030 ± 0.001
Generative Agents	0.704 ± 0.003	0.469 ± 0.005	0.154 ± 0.002	0.829 ± 0.004	0.554 ± 0.005	0.730 ± 0.012	0.031 ± 0.001
MemoryBank	0.692 ± 0.005	0.462 ± 0.004	0.147 ± 0.002	0.776 ± 0.007	0.562 ± 0.005	0.701 ± 0.006	0.023 ± 0.001
G-Memory	0.694 ± 0.004	0.453 ± 0.004	0.156 ± 0.002	$\mathbf{0.839} \pm 0.005$	$\mathbf{0.585} \pm 0.005$	0.763 ± 0.008	0.030 ± 0.001
Intrinsic memory, per-task reset							
$\mathcal{T}_{\text{hand}}$	0.704 ± 0.004	$\mathbf{0.480} \pm 0.003$	0.140 ± 0.002	0.812 ± 0.003	0.581 ± 0.004	0.737 ± 0.005	0.029 ± 0.001
$\mathcal{T}_{\text{LLM}}$	0.709 ± 0.004	0.473 ± 0.009	0.138 ± 0.003	0.812 ± 0.004	0.526 ± 0.007	0.715 ± 0.006	0.032 ± 0.002
$\mathcal{T}_{\varnothing}$	0.710 ± 0.002	0.478 ± 0.003	0.128 ± 0.003	0.825 ± 0.004	0.562 ± 0.008	0.728 ± 0.007	0.032 ± 0.003
Intrinsic memory, carry-forward across tasks							
$\mathcal{T}_{\text{hand}}$ + carry	$\mathbf{0.712} \pm 0.004$	0.468 ± 0.005	0.137 ± 0.002	0.801 ± 0.007	0.570 ± 0.004	$\mathbf{0.766} \pm 0.008$	0.025 ± 0.001
$\mathcal{T}_{\text{LLM}}$ + carry	0.693 ± 0.005	0.459 ± 0.005	0.134 ± 0.004	0.782 ± 0.006	0.523 ± 0.007	0.656 ± 0.019	0.025 ± 0.002
$\mathcal{T}_{\varnothing}$ + carry	0.705 ± 0.004	0.451 ± 0.006	0.127 ± 0.004	0.803 ± 0.003	0.538 ± 0.005	0.709 ± 0.007	0.029 ± 0.002

Wikipedia lookup and assess a given factual claims. ALFWorld is a text-based interactive environment which simulates household tasks. It tests an agent's ability to navigate and execute complex sequential actions to complete tasks. Similarly, SciWorld is a text-based environment of elementary science experiments, where an agent must carry out a procedure across connected locations holding the apparatus it needs. BabyAI is a gridworld of synthetic language instructions of graded complexity, testing grounded instruction following and navigation under partial observation, where the agent sees only the cells around it. Finally, Jericho is a suite of human-written interactive fiction games, which combine long horizons, sparse rewards and an action space of free-form natural language.

A trial budget of 30 steps is used across the majority of benchmarks with two exceptions: ALFWorld, where 50 steps is the ALFWorld default, and Jericho, where 100 steps ensure none of the games will be stuck at a reward of 0.

For numerical benchmarks, we follow the same experimental methodology as G-Memory (Zhang et al., 2025), another memory framework for multi-agent systems. We re-run the G-Memory framework [1] as we cannot directly compare to the published G-Memory results which were benchmarked with GPT-4o-mini as the base language model. We chose these seven benchmarks to cover a range of structured planning, comprehension, and reasoning tasks, all of which are aspects aligned with the data pipeline case study detailed in Section 5.

Backbone and coverage We serve a `openai/gpt-oss-120b` model locally with vLLM on a single node of four NVIDIA GH200 GPUs at temperature 0.1 (vLLM 0.15.1, 16,384-token context). Every experiment is run at the ten seeds on each of seven benchmarks.

4.1 Benchmarking results

Table 2 reports mean reward for every memory design on every benchmark.

[1] https://github.com/bingreeky/GMemory

Table 3: League table over the six benchmarks. A score of 1 is given for the highest mean reward over its ten seeds, with equal means sharing a place. Points award $14 -$ place per benchmark, so a first place is worth 13 and a last place 1, and the total runs from 6 to 78. Ranking this way rewards placing well on many benchmarks rather than winning any one of them.

Memory	FEVER	HotpotQA	ALFWorld	SciWorld	BabyAI	PDDL	Mean	Points
$\mathcal{T}_{\text{hand}}$	7	1	8	6	2	3	4.5	57
Voyager	6	6	3	2	4	6	4.5	57
MetaGPT	8	7	1	5	3	4	4.7	56
G-Memory	11	12	2	1	1	2	4.8	55
Generative Agents	5	4	4	3	10	5	5.2	53
$\mathcal{T}_{\text{hand}}$ + carry	1	5	10	11	5	1	5.5	51
$\mathcal{T}_{\varnothing}$	2	2	12	4	8	7	5.8	49
$\mathcal{T}_{\text{LLM}}$	3	3	9	7	12	9	7.2	41
No memory	10	10	6	8	6	8	8.0	36
ChatDev	9	8	6	9	9	10	8.5	33
MemoryBank	13	9	5	13	7	12	9.8	25
$\mathcal{T}_{\varnothing}$ + carry	4	13	13	10	11	11	10.3	22
$\mathcal{T}_{\text{LLM}}$ + carry	12	11	11	12	13	13	12.0	12

Ranked across benchmarks, the hand-written template is level with the best baseline. Table 3 ranks the thirteen conditions within each of the six benchmarks that discriminate and totals the places. $\mathcal{T}_{\text{hand}}$ and Voyager lead jointly on 57 points of a possible 78, a mean place of 4.5, ahead of MetaGPT on 56, G-Memory on 55 and Generative Agents on 53. $\mathcal{T}_{\text{LLM}}$ + carry, totals 12, placing eleventh or worse on every benchmark.

The intrinsic conditions divide sharply on the ranking. $\mathcal{T}_{\text{hand}}$ comes joint first on the league table and $\mathcal{T}_{\text{hand}}$ + carry is sixth, while $\mathcal{T}_{\text{LLM}}$ is eighth and the two remaining carry-forward conditions are last but one and last. The hand-written schema is therefore the only intrinsic variant that ranks with the baselines, and it is 16 points clear of the LLM-written one, which is the widest gap between any two of the three per-task-reset templates.

The hand-written template beats the LLM-written one. Paired by seed and Holm-corrected across the six benchmarks that rank, $\mathcal{T}_{\text{hand}}$ is ahead of $\mathcal{T}_{\text{LLM}}$ on five — significantly on BabyAI ($+0.055$, $t = +6.99$, $p_{\text{H}} < 0.001$) and PDDL ($+0.022$, $t = +3.87$, $p_{\text{H}} = 0.019$), and by margins too small to separate on ALFWorld ($+0.001$), SciWorld ($+0.001$) and HotpotQA ($+0.007$). The exception is FEVER, where it trails by 0.005 ($t = -0.89$, $p = 0.40$). A schema written by hand is therefore never significantly worse here than one that is freely generated by the LLM, and is sometimes markedly better.

Carrying memory across tasks does not help. As the other memory systems write their memory centrally to be used in future tasks, the same is tested for Intrinsic Memory as the carry-forward variant. Of the twenty-one reset/carry-forward pairs, nineteen favour the per-task reset. Only $\mathcal{T}_{\text{hand}}$ gains from carrying anything forward, on PDDL ($+0.029$, $p_{\text{H}} = 0.046$) and non-significantly on FEVER ($+0.008$).

The knowledge benchmarks are where intrinsic memory places highest, which is consistent with FEVER and HotpotQA being tasks in which reasoning rather than retained state matters more. The embodied benchmarks are where it performs worse than the baselines. For example, in ALFWorld, where the memory is an addition to the agents' own token spend rather than a substitution for it, and the memory systems that perform well are the retrieval baselines which better store the state of the world.

5 Data Pipeline Design Case Study

As a practical case study to evaluate our approach, we applied our memory agents to a collaborative data pipeline design task. The agents must discuss the best data pipeline for ingesting and processing real-time LiDAR and video data from a fleet of vehicles to train self-driving algorithms. The goal is to discuss and find consensus on how to set up the data pipeline, including each component in the

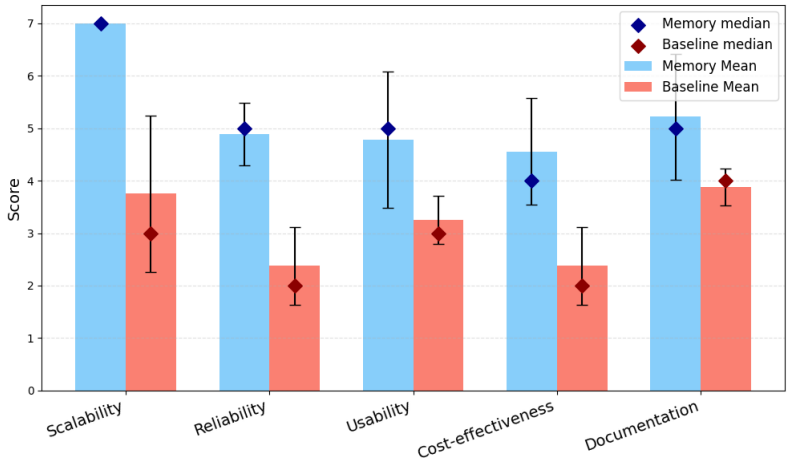

Figure 2: LLM-as-a-Judge metrics for the Data Pipeline design case study.

Table 4: Mean efficiency and LLM-as-a-Judge metrics after 10 independent runs, with p-values calculated using a Wilcoxon ranked sum test. Usability and number of conversation turns is highlighted in italics as the metrics that do not show statistical significance between the baseline and our Intrinsic Memory approach.

Metric	**Baseline Autogen**	**Intrinsic Memory**	**p-value**
Tokens	36077	47830	0.0195
Conversation turns	14.3	16	*0.2632*
Scalability	3.75	7	0.0004
Reliability	2.37	4.9	0.0003
Usability	3.25	4.9	0.0093
Cost-effectiveness	2.37	4.7	0.001
Documentation	3.87	5.4	0.0077

data pipeline. This is a complex task requiring multiple perspectives. We run 10 independent outputs with eight specialized agents.

The full prompts, task description, system configuration, and evaluation process can be found in Appendix D.3.

The output requirements include a concise summary, high-level plan, resource estimates, and a structured JSON specification. We compared a baseline multi-agent system in Autogen against one using Intrinsic Memory. We evaluated system performance against scalability, reliability, usability, cost-effectiveness, and documentation quality, using an LLM-as-a-judge approach.

5.1 Data pipeline design performance

The median and standard deviation of each quality metric is presented in Figure 2. The Intrinsic Memory system shows consistent improvement on all metrics compared to the baseline Autogen.

The Documentation quality focuses on the clarity and how well-justified the design choices are. While Intrinsic Memory helps to boost the Documentation score over the baseline, the score is still relatively low at a mean of 5.4. This suggests that retaining memory of the conversation alone does not guarantee good justification, and while some context and attributes of each component are remembered, the reasons for choosing the components are not. This could be a problem with the training corpus, and a requirement for better annotated training data. Similarly, the Usability score is low with means of 3.32 and 4.9 for the baseline and Intrinsic Memory, respectively.

The improved quality comes at a cost of additional tokens outlined in Table 4. Intrinsic Memory uses on average 32% more tokens than the baseline as its outputs are more descriptive on average, although the number of conversation turns is similar and not statistically significant. This indicates that the addition of a memory module costs additional token overhead to maintain, but does not increase the number of conversation turns between agents.

5.2 Qualitative analysis of data pipeline outputs

Figure 3 shows snippets for one component from the highest-scoring outputs for the intrinsic memory agent system and baseline Autogen system.

```
"Component 1":  "Data Ingestion (Amazon S3)"
"AWSName":  "AmazonS3",
"Pros":  ["Scalable", "durable", "secure storage for raw data"],
"Cons":  ["Additional cost for storing large amounts of data"],
"Design":  "Use S3 as a central repository for all data sources,
with separate buckets for each source if needed.",
"Details":  "Implement S3 event notifications to trigger
processing workflows upon new data arrival."
```

(a) Intrinsic Memory Agent system sample from highest-scoring output. This data pipeline received scores of Scalability: 7, Reliability: 5, Usability: 6, Cost-effectiveness: 6, Documentation: 7

```
"Component 1":
 "Name":  "Data Ingestion",
 "Description":  "Ingest data from various sources (camera,
lidar, radar) at high speeds",
 "Implementation difficulties":  7,
 "Maintainability difficulties":  6
```

(b) Baseline Autogen system system sample from highest-scoring output. This data pipeline received scores of Scalability: 5, Reliability: 4, Usability: 3, Cost-effectiveness: 3, Documentation: 2

Figure 3: Snippets of one component within the data pipeline design from both systems. The full outputs can be found in the appendix in Figures 10 and 12.

The Intrinsic Memory Agent system outperforms the baseline system across the five quality metrics. In terms of scalability, the Intrinsic Memory Agent system is capable of providing an overall assessment of scalability, specifically around varying data volumes, whereas the baseline system encapsulates that measure only in the form of "maintenance difficulty" for each component of the pipeline. In terms of reliability, the Intrinsic Memory Agent provides considerations for each component, such as AWS Kinesis's secure streaming capabilities and considerations as well as the use of Docker containers within amazon SageMaker to improve stability and reproducibility of ML pipelines. The Intrinsic Memory Agent provides a more descriptive Usability output of the Intrinsic and a clearer pathway to implementation. In terms of cost, the Intrinsic Memory Agent makes specific calculations and observations for the cost-effectiveness and resource requirements, including reasoning behind each component choice, whereas the baseline system limits itself to overall evaluations of implementation and maintainability difficulties.

Finally, the Intrinsic Memory Agent ultimately provides justification and documents its recommendation under each component, including pros and cons for each component choice.

Overall, the Intrinsic Memory Agent provides a more descriptive answer and more value to engineers by specifying tools, configurations and trade-offs. For example, its Data Streaming design recommends Amazon Kinesis, whereas the baseline simply states "Ingest data from various sources (camera, lidar, radar) at high speeds." Similarly, the IMA cites the specific connections between components that must be implemented (for example, Amazon S3 => Amazon EC2 through API) . Although some precise configuration settings remain unspecified, the baseline merely names each component without offering implementation details or alternatives.

The components specified by the Intrinsic Memory Agent are more relevant to the problem specification. The data pipeline design task explicitly specifies the input data contains lidar and radar data sources, in which SageMaker and Kinesis are particularly relevant, specifically used for lidar data processing. This contrasts the vague "Lidar data processing" component of the baseline, which only contains a general description to process the data without providing any details.

6 Discussion and Limitations

Although the Intrinsic Memory approach shows improved performance across the data pipeline generation task and the selected benchmarks, further validation is required across a broader set of complex tasks, potentially with varying number of agents, and models. The benchmark results and the case study point in different directions. On the benchmarks Intrinsic Memory places highest on the knowledge datasets but does not beat the strongest retrieval baselines on the embodied ones, while on the data pipeline design task it improves every judged metric. The two settings differ in the work the memory has to do. The benchmarks are short task-execution episodes where the useful thing to remember is the state of the environment, which the retrieval baselines store directly. The case study is an extended design discussion between eight specialised agents, where what matters is that each agent holds on to its own position and expertise as the discussion develops. This is the setting the design was built for, and it is where it performs best.

There are several limitations that remain. The benchmark results come from a single backbone, and the case study is one design task over ten runs, so neither establishes how far the approach generalises. The approach's performance and consistency comes at the cost of increased token usage, due to the additional update calls. Further work to reduce the number of update calls, updating only when necessary, will help to alleviate the additional usage.

The results demonstrate that a movement towards heterogeneity of agents leads to an improvement in performance of the multi-agent system, allowing agents to focus more specifically on an area of the design. This indicates that methods to provide additional heterogeneity, such as the ability to fine-tune agents towards their specialisation, might see additional performance gains, alongside the personalization of memories focused on individual experience.

7 Conclusion

This paper introduces Intrinsic Memory Agents, a novel multi-agent LLM framework that constructs agent-specific heterogeneous memories to enhance multi-agent collaboration in discussion and planning tasks.

Evaluated over seven benchmarks at ten different seeds, our Intrinsic Memory framework places highest on the two knowledge benchmarks, with the variant using the handcrafted templates ranking joint first among all memory systems, with a mean place of 4.5. It reaches parity with the best baselines on BabyAI, and loses on ALFWorld and SciWorld. The hand-written template is also the strongest of the three variants, never significantly worse than one the agent generates for itself

Results on the data pipeline case study further show the Intrinsic Memory Agents' enhanced ability to collaborate on complex tasks. The Intrinsic Memory Agent system outperforms the baseline system across all quality measures of scalability, reliability, usability, cost-effectiveness, and documentation, as well as an ability to more closely follow the task specification, providing more actionable recommendations by suggesting specific tools and frameworks, as well as trade-off details of each component in the pipeline.

Taken together, the two evaluations suggest that agent-specific memory is most useful where agents must sustain distinct positions over a long discussion, and least useful where a single record of the environment would serve every agent equally well.

AI use statement

Generative AI was used in this work for coding assistance and for editing prose. All AI-assisted code was read, run and tested by the authors, and every number reported in this paper is produced by the released experiment scripts. No part of the research ideation, experimental design or analysis was

delegated to a generative model, and no text was included without an author reading and revising it. We take responsibility for the final content of this work.

Ethics statement

The benchmarks used here are public task suites with no human subjects and no personal data. The data pipeline case study evaluates synthetic design documents produced by the agents themselves. We report the negative results alongside the positive ones, including the benchmarks on which the proposed method loses to the baselines.

Reproducibility Statement

We have taken care to ensure that our experiments and results are transparent and reproducible by detailing the models, computational setup, code, statistical tests, and prompts used in our experiments. The LLM model used is cited, named, and referenced in the main text. The computational infrastructure used, including the GPU model names inference engine verions are specified in section 4 of the main text. For code, the names and versions of relevant Python libraries are specified within the supplementary code files. A Wilcoxon rank-sum test is used to test statistical significance for the data pipeline case study. P-values and standard deviation measures are included in the performance analysis in section 5.1. Every benchmark condition is run at the ten seeds, one experiment per seed, and we report results across all seeds. The result row of every experiment, from which all seven benchmark tables are generated, is released with the code, together with the script that generates each table. All code for running the data pipeline case study and numeric benchmarks are included in the supplementary materials. Finally, the selected prompts of the multi-agent and Intrinsic memory architecture are shown in Appendix D.3.

References

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, et al. Gepa: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://arxiv.org/abs/2507.19457.

Anthropic. Claude 3.5 sonnet. Technical announcement, 2024. URL https://www.anthropic.com/news/claude-3-5-sonnet. Available via Claude.ai and API.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *The 12th International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=hSyW5go0v8.

Nuo Chen, Hongguang Li, Juhua Huang, Baoyuan Wang, and Jia Li. Compress to impress: Unleashing the potential of compressive memory in real-world long-term conversations, 2024. URL https://arxiv.org/abs/2402.11975.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. Babyai: A platform to study the sample efficiency of grounded language learning. In *International Conference on Learning Representations (ICLR)*, 2019. URL https://arxiv.org/abs/1810.08272.

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

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. Memp: Exploring agent procedural memory. In *Findings of the Association for Computational Linguistics: ACL 2026*, 2026. URL https://arxiv.org/abs/2508.06433.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey, 2024. URL https://arxiv.org/abs/2312.10997.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. Large language model based multi-agents: A survey of progress and challenges, 2024. URL https://arxiv.org/abs/2402.01680.

Matthew Hausknecht, Prithviraj Ammanabrolu, Marc-Alexandre Côté, and Xingdi Yuan. Interactive fiction games: A colossal adventure. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020. URL https://arxiv.org/abs/1909.05398.

Junda He, Christoph Treude, and David Lo. Llm-based multi-agent systems for software engineering: Literature review, vision and the road ahead, 2025. URL https://arxiv.org/abs/2404.04834.

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. Metagpt: Meta programming for a multi-agent collaborative framework, 2024. URL https://arxiv.org/abs/2308.00352.

Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, Senjie Jin, Jiejun Tan, et al. Memory in the age of ai agents. *arXiv preprint arXiv:2512.13564*, 2025. URL https://arxiv.org/abs/2512.13564.

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

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

Junyou Li, Qin Zhang, Yangbin Yu, Qiang Fu, and Deheng Ye. More agents is all you need, 2024a. URL https://arxiv.org/abs/2402.05120.

Tianle Li, Ge Zhang, Quy Duc Do, Xiang Yue, and Wenhu Chen. Long-context llms struggle with long in-context learning, 2024b. URL https://arxiv.org/abs/2404.02060.

Xinyi Li, Sai Wang, Siqi Zeng, Yu Wu, and Yi Yang. A survey on llm-based multi-agent systems: workflow, infrastructure, and challenges. *Vicinagearth*, 1(1):9, October 2024c. ISSN 3005-060X. doi: 10.1007/s44336-024-00009-2. URL https://doi.org/10.1007/s44336-024-00009-2.

Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. Agentboard: An analytical evaluation board of multi-turn llm agents, 2024.

OpenAI. Gpt-4o system card, 2024. URL https://arxiv.org/abs/2410.21276.

Krista Opsahl-Ong, Michael J Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. Optimizing instructions and demonstrations for multi-stage language model programs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2024. URL https://arxiv.org/abs/2406.11695.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. Memgpt: Towards llms as operating systems, 2024. URL https://arxiv.org/abs/2310.08560.

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

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, et al. Chatdev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024. URL https://arxiv.org/abs/2307.07924.

Chen Qian, Zihao Xie, YiFei Wang, Wei Liu, Kunlun Zhu, Hanchen Xia, Yufan Dang, Zhuoyun Du, Weize Chen, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Scaling large language model-based multi-agent collaboration, 2025. URL https://arxiv.org/abs/2406.07155.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://arxiv.org/abs/2303.11366.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning Text and Embodied Environments for Interactive Learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021. URL https://arxiv.org/abs/2010.03768.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. *arXiv preprint arXiv:2504.07952*, 2025. URL https://arxiv.org/abs/2504.07952.

James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. FEVER: a large-scale dataset for fact extraction and VERification. In Marilyn Walker, Heng Ji, and Amanda Stent (eds.), *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pp. 809–819, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1074. URL https://aclanthology.org/N18-1074/.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023. URL https://arxiv.org/abs/2305.16291.

Qingyue Wang, Yanhe Fu, Yanan Cao, Shuai Wang, Zhiliang Tian, and Liang Ding. Recursively summarizing enables long-term dialogue memory in large language models, 2025a. URL https://arxiv.org/abs/2308.15022.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. Scienceworld: Is your agent smarter than a 5th grader? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2022. URL https://arxiv.org/abs/2203.07540.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025b. URL https://arxiv.org/abs/2409.07429.

Jing Xu, Arthur Szlam, and Jason Weston. Beyond goldfish memory: Long-term open-domain conversation. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5180–5197, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.356. URL https://aclanthology.org/2022.acl-long.356/.

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

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018. URL https://arxiv.org/abs/1809.09600.

Tian Yu, Shaolei Zhang, and Yang Feng. Auto-rag: Autonomous retrieval-augmented generation for large language models, 2024. URL https://arxiv.org/abs/2411.19443.

Guibin Zhang, Muxin Fu, Guancheng Wan, Miao Yu, Kun Wang, and Shuicheng Yan. G-memory: Tracing hierarchical memory for multi-agent systems, 2025. URL https://arxiv.org/abs/2506.07398.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, et al. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://arxiv.org/abs/2510.04618.

Zeyu Zhang, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Quanyu Dai, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model based agents, 2024. URL https://arxiv.org/abs/2404.13501.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024. URL https://arxiv.org/abs/2308.10144.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena, 2023. URL https://arxiv.org/abs/2306.05685.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024. URL https://arxiv.org/abs/2305.10250.

A ALGORITHMS

This section contains the context construction algorithm presented in Section 3 and the finalisation algorithm presented in D.1.

Algorithm 1 presents the context construction algorithms, which takes the current conversation history, memory of the agent, and maximum number of tokens. It appends the most recent conversation turn and agent memory to the context first, before using the remainder of the tokens to append the rest of the conversation history, ensuring the memory and most recent output is always included.

Algorithm 1 Context Construction

Input: conversation history H, agent memory M, token budget B
$C \leftarrow [\,]$
append H_0 to C
append M to C
$R \leftarrow B - \textsc{TokenCount}(C)$
$T \leftarrow [\,]$
for each turn h in reverse order of $H_{1:}$ **do**
  $t \leftarrow \textsc{TokenCount}(h)$
  if $t \leq R$ **then**
    prepend h to T
    $R \leftarrow R - t$
  else
    break
  end if
end for
append T to C
return C

Algorithm 2 is the finalisation algorithm that specifies the order of agents speaking. It is a modified round-robin discussion between the agents: The discussion begins with each of the worker agents (BOA, DEA, MLA, IA) contributing to the conversation, with each worker's turn being followed by the conversation delegation agent (CDA). Once the workers have each had their turn, the knowledge integration agent and evaluation agent make their contributions, and the cycle begins again. The CDA is programmed to dedicate a certain number of turns to discussion, proposals, and consensus. The number of turns dedicated to each conversation stage is tracked, and once the consensus round is reached, each agent is asked to confirm if they agree with the proposed solution or not. If all agree on the proposed solution as being acceptable, the CDA will emit a "FINALIZE" response, triggering the documentation joining agent (DJE) to compile the agreed response and format it according to the task requirements.

B INTRINSIC MEMORY VARIATIONS

There are three variations on Intrinsic Memory to understand the structure of the templates that the agents use to update their memory. The three approaches are:

1. **Manual template** $\mathcal{T}_{\text{hand}}$ - Manual hand-crafted templates for each type of dataset and problem, based on the kind of task the agents have to solve.

2. **Generic template** $\mathcal{T}_{\varnothing}$ - A generic template to store updated memory without any reference to specific problems or fields.

3. **LLM-generated template** $\mathcal{T}_{\text{LLM}}$ - We prompt the agent LLM to self-generate a template based on the instructions of the task, allowing the agent to dynamically create templates it deems suitable for each task.

Algorithm 2 Finalisation Algorithm

Require: worker set $W = [w_1, w_2, w_3, w_4]$, last speaker s, message list M
Require: global counters *turn_counter*, *worker_counter*
Ensure: next speaker n
  turn_counter $\leftarrow$ *turn_counter* $+ 1$
  if CONTAINS(M_{last}, "FINALIZATION") **then**
    return DJE
  end if
  if $s =$ CDA **then**
    $n \leftarrow W[$*worker_counter* $\bmod |W|]$
    worker_counter $\leftarrow$ *worker_counter* $+ 1$
    return n
  else if $s \in W$ **then**
    if *worker_counter* $\bmod |W| = 0$ **then**
      return KIA
    else
      return CDA
    end if
  else if $s =$ KIA **then**
    return ERA
  else if $s =$ ERA **then**
    return CDA
  end if

B.0.1 MANUAL TEMPLATE

In the manual template approach, a strict structured template is used that agents must follow to update their internal memory. The manual templates are created to track the relevant information of the given task, with specific fields.

B.0.2 GENERIC TEMPLATE

For the generic approach, we use a universal template structure, giving only general fields to be filled in with the current task description and trajectory, allowing the model to use any form of memory representation, and not strictly requiring the same structured format to be used each time.

B.0.3 LLM-GENERATED TEMPLATE

Finally, we leverage each agent's own capabilities to create a template during initialisation which it deems suitable for the task at hand. This approach removes the need to create manual templates for each type of task while allowing for memory templates specific to the task. The template generation prompt is as follows:

```
You are a MEMORY UPDATER for a PDDL-style planning agent.
Your job:  - Maintain a compact JSON memory capturing stable,
reusable information across tasks and domains.  - Only store
information that improves future planning:  common strategies,
mistakes, valid action patterns, state-transition insights.
- Do not store long histories.  Keep everything concise and
deduplicated.
Inputs you receive each update call:  - currentmemory:  the
previous memory as json (may be empty re-init).  - latestturn:  the
agent's most recent Thought/Action/Observation.  - currenttask:
one of blockworld, barman, gripper, tyreworld.  - goal:  current
goal description.
OUTPUT: - Return ONLY the updated memory as valid JSON following
the template below.  - No extra commentary.
----------------- MEMORY TEMPLATE (ALWAYS FOLLOW)
"task summary":  "brief description of PDDL planning setting",
"global strategies":  [ "high-level reusable planning heuristics
across domains" ], "domains":    "blockworld":    "valid action
patterns":  ["pickup X", "putdown X", "stack X Y", "unstack X Y"],
"good strategies":  ["free target block before stacking"], "invalid
patterns":  ["wrong think format", "stack without clear base"],
"mistakes":  ["attempting pickup while arm full"] , "barman":
"valid action patterns":  ["hand grasp glass", "fill-shot ...",
"pour-shot-to-clean-shaker ..."], "good strategies":  ["ensure
hand availability before filling"], "invalid patterns":  ["fill
without holding glass"], "mistakes":  ["grasp with occupied hand"]
, "gripper":    "valid action patterns":  ["move R1 R2", "pick O
Room Gripper", "drop O Room Gripper"], "good strategies":  ["carry
multiple items before moving rooms"], "invalid patterns":  ["drop
object in wrong room"], "mistakes":  ["pick while gripper full"]
, "tyreworld":    "valid action patterns":  ["open X", "fetch O
C", "loosen N H", "jack-up H"], "good strategies":  ["open boot
early to access tools"], "invalid patterns":  ["loosen nut without
wrench"], "mistakes":  ["inflate wheel without pump"]  , "tasks":
[  "id":  "identifier or hash of goal", "goal":  "exact goal text",
"status":  "pending|solved", "helpful observations":  ["short state
insights from valid steps"], "invalid actions":  ["summaries of
failed attempts"], "progress notes":  ["short planning insights for
this task"]  ]
----------------- UPDATE INSTRUCTIONS
1.  Parse current memory.  - If empty or invalid, initialize using
the template above.
2.  Update the domain-specific sections:  - From latest turn, add
new useful action patterns, invalid patterns, or mistakes.  - Keep
lists short, deduplicated, and generalisable.
3.  Update global strategies if the latest turn reveals a robust
cross-domain heuristic.
4.  Update the relevant task entry:  - If no entry exists for this
goal, create one.  - Add helpful observations if new actionable
state insights appear.  - Add invalid actions if latest turn
shows an invalid move.  - Add progress notes for general reasoning
improvements.  - If task finished, mark status = "solved".
5.  Return ONLY the updated JSON memory, nothing else.
```

Figure 4: Manually created prompt for the LLM agent in the PDDL task.

```
Use your latest response to populate and update the current memory
with factual information to solve the task based on the task
description.

## Task Description:

{task_description}

## Current Task Trajectory:

{task_trajectory}

## Current Memory:

{current_memory}
```

Figure 5: Prompt for the LLM agent to update its memory

```
I have an AI agent that has to complete a task.
The agent has a memory that is updated each time the LLM responds
by comparing the latest response and the existing memory, and
adding any new important information.  The memory should be
templated based on the nature of the task following a json-style
format.  The memory update is conducted as a prompted LLM call to
update the memory.  Provide the instructions to the agent for such
an update operation, as well as the generic memory template for
this particular task.  Provide the full answer as a single prompt.
Only include the most crucial details to the updating instructions
to preserve token usage.  Do not explain or describe the prompt,
simply return the prompt and nothing more.
This is the task description:  {task_description}
```

Figure 6: Prompt for the LLM agent to generate its own memory template.

Table 5: Completion tokens *per task*, in thousands, split into what the agents spend solving the task (*Ag.*) and what the Intrinsic Memory spends on its updates (*Mem.*) Backbone `gpt-oss-120b`.

	FEVER		HotpotQA		ALFWorld		SciWorld		BabyAI		PDDL		Jericho	
Memory	Ag.	Mem.	Ag.	Mem.	Ag.	Mem.	Ag.	Mem.	Ag.	Mem.	Ag.	Mem.	Ag.	Mem.
No memory	0.7	*n/i*	2.7	*n/i*	34.8	*n/i*	7.8	*n/i*	27.4	*n/i*	18.5	*n/i*	28.7	*n/i*
ChatDev	0.8	*n/i*	2.9	*n/i*	35.4	*n/i*	8.6	*n/i*	28.3	*n/i*	19.9	*n/i*	36.2	*n/i*
MetaGPT	0.6	*n/i*	3.0	*n/i*	37.9	*n/i*	7.1	*n/i*	28.3	*n/i*	18.1	*n/i*	30.2	*n/i*
Voyager	1.0	*n/i*	3.5	*n/i*	38.0	*n/i*	7.4	*n/i*	29.2	*n/i*	18.7	*n/i*	30.8	*n/i*
Generative Agents	0.9	*n/i*	3.4	*n/i*	38.6	*n/i*	7.1	*n/i*	28.8	*n/i*	18.2	*n/i*	30.0	*n/i*
MemoryBank	0.9	*n/i*	3.2	*n/i*	12.2	*n/i*	7.3	*n/i*	25.5	*n/i*	19.7	*n/i*	13.9	*n/i*
G-Memory	3.4	*n/i*	6.0	*n/i*	48.6	*n/i*	10.5	*n/i*	30.3	*n/i*	21.2	*n/i*	31.3	*n/i*
$\mathcal{T}_{\text{hand}}$	0.5	2.7	2.6	10.4	27.3	90.5	7.9	25.9	22.6	34.9	16.4	34.3	42.0	149.0
$\mathcal{T}_{\text{hand}}$ + carry	0.6	5.1	2.6	13.4	27.0	92.4	8.8	36.0	22.9	44.8	18.0	69.7	45.1	157.2
$\mathcal{T}_{\text{LLM}}$	0.4	1.2	2.2	2.5	35.0	47.6	7.6	17.8	23.1	19.3	18.8	26.0	35.3	80.8
$\mathcal{T}_{\text{LLM}}$ + carry	0.6	2.4	2.2	4.0	37.3	59.0	9.3	24.4	24.6	40.4	23.2	43.0	37.1	99.7
$\mathcal{T}_{\varnothing}$	0.4	0.7	2.4	2.7	35.0	26.0	7.2	9.8	29.0	18.8	19.2	14.5	34.6	38.1
$\mathcal{T}_{\varnothing}$ + carry	0.4	1.3	2.3	2.9	39.7	36.1	8.4	11.1	34.0	20.7	20.7	15.3	37.3	43.2

C Full benchmarking results

D Data pipeline design case study

As a practical case study to evaluate our approach, we applied our memory agents to a collaborative data pipeline design task. This is a complex task requiring multiple perspectives. We run 10 independent outputs with eight specialized agents: **Evaluation Agent (EA)** (evaluates the output solutions); **Knowledge Integration Agent (KIA)** (summarizes each discussion); **Data Engineer Agent (DEA)** (determines the data processing needs); **Infrastructure Engineer (IA)** (designs the cloud infrastructure); **Business Objective Engineer (BOA)** (checks against business requirements); **Machine Learning Engineer (MLE)** (provides ML implementation); **Conversation Delegation Agent (CDA)** (responsible for facilitating the collaborative process); Documentation Joining Agent (DJE) (responsible for producing final output after consensus is reached among agents).

D.1 System Configurations

We evaluated two system configurations: First, the **Baseline System** which consists of a standard multi-agent implementation without intrinsic memory. It uses standard prompt templates for each agent role, relying exclusively on conversation history for context. Second is our **Intrinsic Memory System** approach with agent-specific memories. It implements agent-specific memories and updates them intrinsically based on agent outputs, and constructs context using both conversation history and agent memories.

Both systems used identical agent roles and task specifications, with Llama-3.2-3b as the underlying LLM. Each agent role was initialized with the same role description and initial instructions across the two system configurations to ensure a fair comparison.

An agent selection function iterates through each worker agent and the conversation delegation agent (CDA), ensuring that all agents are represented in the discussion. Once all agents have accepted a proposed solution, marked through the "ACCEPT" flag, the CDA emits a "FINALIZE" flag, prompting the Documentation Engineer Agent to produce the final data pipeline output. The full algorithm for finalisation and ordering of agents is displayed in the Appendix A Algorithm 2.

D.2 Evaluation Metrics

To evaluate the quality of the data pipeline designs generated by our memory agents, and to compare to the pipeline designs generated from default Autogen, we use an LLM as a judge (Zheng et al., 2023) to score each pipeline design and provide a qualitative analysis to support these scores. We evaluated the multi-agent system performance under the following metrics:

- Scalability: ability for the data pipeline to handle increasing data volumes or user loads.

- Reliability: ability for the data pipeline to handle failures and ensure data integrity.
- Usability: is there enough detail in the data pipeline design for developers to implement the design?
- Cost-effectiveness: balance between costs and benefits of each chosen component.
- Documentation: how well-justified and documented is the choice of elements for the data pipeline?

The scalability and reliability metrics are chosen as core requirements in modern data pipelines. Scalability reflects the ability to grow the pipeline to handle larger volumes of data and users, while reliability ensures the pipeline is consistent and fault-tolerant, both of which are crucial if the pipeline were to be deployed. Usability and documentation metrics reflect the details and design decisions taken. A strong design is not useful if it does not contain enough detail or is too abstract to be practically implemented. Usability measures whether the output designs are detailed and clear enough for engineering teams to implement. Design decisions must be well-documented, with clear justifications and explanations for each component, which reveals the reasoning behind the agents' choices. Finally, the cost-effectiveness metric evaluates whether the pipeline design has considered and balanced the need for computation resources with the cost of those resources. Run-time metrics such as latency and throughput are not included in our evaluation metrics as we only present the design of the data pipelines to be evaluated, and do not implement the designs into code

D.3 Full prompts and example outputs

The full prompts for the data pipeline case study are shown here. The prompts are used to start the conversation in the multi-agent system. The prompt used for LLM-as-a-Judge evaluation is shown in Figure 9. Finally, there are examples of the full outputs by the Instrinsic Memory Agents which scored highest on the data pipeline problem.

```
You are maintaining the memory of an agent working as [ROLE]
in a multi-agent conversation.  Use the old memory and the
newest output by the agent to populate and up- date the
current memory json with factual information.

For context, old memory content:
[MEMORY_CONTENT]

Current content generated by the agent:
[AGENT_OUTPUT]

Update the memory content to incorporate new information
while preserving key historical context.  The updated content
should be concise and focus on information relevant to both
the old memory and the newly generated output.
```

Figure 7: Prompt of the memory update function, where *ROLE* is the agent's role specification R_n; *MEMORY_CONTENT* is the current content $M_{n,m-1}$; *AGENT_OUTPUT* is the agent's output $O_{n,m}$.

```
"` This discussion session is set up to discuss the best data
pipeline for a real time data intensive machine learning training
and inference self driving application.  The goal is to discuss and
find consensus on how to set up the data pipeline, including each
component in the data pipeline.
You can assume that we have access to AWS.

**Data Description:** Real-time data of cars driving in street.
There are 6 camera sources with data in .jpg format; 1 lidar source
in .pcd.bin format; and 5 radar sources with data in .pcd format.

**Discussion and Design:**
- Emphasize comprehensive understanding of the data sources,
processing requirements, and desired outcomes.
- Encourage each other to engage in an open discussion on potential
technologies, components, and architectures that can handle the
diverse data streams and real-time nature of the data.
- Keep the conversation on design and evaluating the pros and
cons of different design choices, considering scalability,
maintainability, and cost-effectiveness.
- The team should agrees on a final architectural design, justifying
the choices made.
- The team should produce the required the document
PIPELINE_OVERVIEW.json.

**Final Output:**
- Produce a concise summary of the agreed-upon pipeline
architecture, highlighting its key components and connections.
- Provide a high-level plan and rationale for the design, explaining
why it is well-suited for the given data and use case.
- Estimate the cloud resources, implementation efforts, and
associated costs, providing a rough breakdown and complexity
rating.
- Generate a 'PIPELINE_OVERVIEW.json' file, detailing the proposed
complete architecture in JSON format with the following fields:
- "Platform":  A cloud service provider's name if the cloud solution
is the best, or "local server" if locally hosted servers are
preferred.
- "Component 1":  The first component in the pipeline framework.
- "Component 2":  The second component in the pipeline framework.
Continue until all required components are listed.
- "Implementation difficulties":  A rating from 1 to 10 (lowest to
highest).
- "Maintenance difficulties":  A rating from 1 to 10 (lowest to
highest).

**Instructions:**
- Remember, this is a collaborative design discussion, not a project
execution.  Refrain from assigning tasks with deadlines.
- Keep the conversation focused on architectural choices,
technologies, and potential challenges.
- Emphasize the importance of a well-thought-out design.  "`
```

Figure 8: Prompt for the Data Pipeline design. This is the task description provided to all agents in the system.

```
You are an expert in data pipeline design evaluation. Your
task is to evaluate the following data pipeline design based
on its description. For each of the specified metrics,
assign a score from 1 to 10, where 1 is the lowest and 10
is the highest. Provide a brief justification for each
score. Be critical and harsh if the design is poor and give
it a low score. Base your evaluation solely on the provided
description. Do not assume any additional information.

Metrics:
1. Scalability: Ability to handle increasing data volumes
or user loads.
2. Reliability: Ability to handle failures and ensure data
integrity.
3. Usability: Enough detail for developers to implement the
design.
4. Cost-effectiveness: Balance between costs and benefits.
5. Documentation: How well-justified is the choice of
elements for the data pipeline

Provide your evaluation in the following format in a json
dict:
[metric1]: score: [score],
justification: [justification],
[metric2]...
```

Figure 9: Prompt for the LLM-as-a-Judge to evaluate data pipeline designs.

```
"Platform":  "AWS"

"Component 1":  "Amazon S3 (Data Storage)"
"pros":  "Scalable, durable, and secure storage for raw data."
"cons":  "Additional cost for storing large amounts of data."
"design":  "Use S3 as a central repository for all data sources,
with separate buckets for each source if needed."
"details":  "Implement S3 event notifications to trigger data
processing workflows upon new data arrival."

"Component 2":  "Amazon EC2 (GPU Instances) -- Image & Point Cloud
Processing"
"pros":  "Scalable, high-performance computing resources for
complex image and point cloud processing."
"cons":  "Higher cost due to instance usage and GPU pricing."
"design":  "Use EC2 NVIDIA GPU instances for camera source data
processing."
"details":  "Implement Docker-based containerization for
reproducibility and efficient execution."

"Component 3":  "Amazon SageMaker -- Model Training & Processing"
"pros":  "Fully managed, high-performance infrastructure for ML
training and inference."
"cons":  "Higher cost for GPU-backed instances."
"design":  "Use SageMaker for lidar and radar data processing,
including preprocessing and feature extraction."
"details":  "Use Docker containers to ensure stable, reproducible
ML pipelines."

"Component 4":  "Amazon Kinesis -- Real-Time Streaming"
"pros":  "Scalable, fast, and secure streaming and analytical
capabilities."
"cons":  "Additional cost for high-throughput streaming."
"design":  "Use Kinesis for real-time ingestion, analytics, and
event-driven triggers."
"details":  "Implement aggregation, filtering, and visualization
workflows to support decisions."

"Component 5":  "Amazon Redshift -- Batch Analytics"
"pros":  "High-performance analytics engine with strong scaling
capabilities."
"cons":  "Additional cost for warehousing large datasets."
"design":  "Use Redshift for batch analytical workloads across
multi-source data."
"details":  "Implement analytics queries and visualization
pipelines for reporting and BI."
```

Figure 10: Example output from intrinsic memory agent system which scored the highest. The intrinsic memory agent system adheres to the initial prompt well, providing an answer in the correct format, as well as providing sufficient justification for the choice of components, as well as implementation details.

```
"Component 6":  "Amazon IAM -- Identity & Access Control"
"pros":  "Secure, scalable identity and permissions management."
"cons":  "Requires careful design to avoid security
misconfiguration."
"design":  "Implement IAM roles and policies to manage access
across all pipeline components."
"details":  "Use Cognito for user authentication and authorization
where appropriate."

"Component 7":  "Amazon Cognito -- Authentication"
"pros":  "Managed, secure identity federation and authentication."
"cons":  "Additional cost depending on usage tiers."
"design":  "Use Cognito for managing user identity, login, and
token issuance."
"details":  ""
"Connections 1":  "Amazon S3 → Amazon EC2 (GPU instances) (API)"
"Connections 2":  "Amazon EC2 (GPU instances) → Amazon SageMaker
(API)"
"Connections 3":  "Amazon SageMaker → Amazon Kinesis (API)"
"Connections 4":  "Amazon Kinesis → Amazon Redshift (API)"
"Connections 5":  "Amazon IAM → Amazon Cognito (API)"

"Potential Challenge 1":  "Scalability"
"Potential Challenge 2":  "Performance"
"Potential Challenge 3":  "Security"

"Rationale":  "Pipeline is designed to be scalable, maintainable,
and secure using microservices-based architecture."

"Complexity Rating":  "7"

"Estimated Cloud Resource 1":  "S3:  10-20 TB of storage"
"Estimated Cloud Resource 2":  "EC2 (GPU instances):  4-8
GPU-enabled instances"
"Estimated Cloud Resource 3":  "SageMaker:  2-4 GPU-enabled
instances"
"Estimated Cloud Resource 4":  "Kinesis:  2-4 streams"
"Estimated Cloud Resource 5":  "Redshift:  2-4 clusters"

"Implementation Effort":  "12-16 weeks"

"Cost 1":  "S3:  $500-$2,000 per month"
"Cost 2":  "EC2 (GPU instances):  $5,000-$20,000 per month"
"Cost 3":  "SageMaker:  $2,000-$10,000 per month"
"Cost 4":  "Kinesis:  $1,000-$5,000 per month"
"Cost 5":  "Redshift:  $2,000-$10,000 per month"
```

Figure 11: Example output from intrinsic memory agent system which scored the highest. The intrinsic memory agent system adheres to the initial prompt well, providing an answer in the correct format, as well as providing sufficient justification for the choice of components, as well as implementation details. (Continued)

```
"Platform": "AWS",
"Component 1":
"Name": "Data Ingestion",
"Description": "Ingest data from various sources (camera, lidar,
radar) at high speeds",
"Implementation difficulties": 7,
"Maintainability difficulties": 6 ,
"Component 2":
"Name": "Image Processing",
"Description": "Process camera data in real-time to extract
relevant features",
"Implementation difficulties": 8,
"Maintainability difficulties": 7
,
"Component 3":
"Name": "Object Detection",
"Description": "Detect objects of interest (e.g., pedestrians,
cars, lanes) within image frames and track their movement over
time",
"Implementation difficulties": 9,
"Maintainability difficulties": 8
,
"Component 4":
"Name": "Lidar Data Processing",
"Description": "Process lidar data in real-time to estimate
distances, velocities, and other relevant metrics",
"Implementation difficulties": 8,
"Maintainability difficulties": 7
,
"Component 5":
"Name": "Radar Data Processing",
"Description": "Process radar data in real-time to estimate
distances, velocities, and other relevant metrics",
"Implementation difficulties": 8,
"Maintainability difficulties": 7
,
"Component 6":
"Name": "Data Fusion",
"Description": "Fuse the outputs from different sensors (camera,
lidar, radar) to create a more accurate representation of the
environment",
"Implementation difficulties": 9,
"Maintainability difficulties": 8
,
"Component 7":
"Name": "Model Training",
"Description": "Train machine learning models on large datasets
using AWS SageMaker's Training Grounds feature",
"Implementation difficulties": 8,
"Maintainability difficulties": 7 ,
"Component 8":
"Name": "Inference",
"Description": "Perform real-time inference on trained models,
making predictions on new, unseen data",
"Implementation difficulties": 9,
"Maintainability difficulties": 8
```


Figure 12: Example output from baseline Autogen system which scored the highest.