arXiv:2605.22721v1 [cs.MA] 21 May 2026

Self-Evolving Multi-Agent Systems via Decentralized Memory

Guangya Hao[1] **Yunbo Long**[1] **Zhuokai Zhao**[2]
[1]University of Cambridge [2]University of Chicago

Abstract

Self-evolving multi-agent systems (MAS) have emerged as a promising route to LLM agents that continually improve from experience, with persistent memory at their foundation. However, existing designs almost exclusively adopt a *centralized* repository shared across agents, incurring communication and coordination overhead, raising privacy concerns, and collapsing agent diversity. We propose **DECENTMEM**, a *decentralized* memory framework in which each agent maintains its own dual-pool memory — an exploitation pool of consolidated past trajectories and an exploration pool of LLM-generated candidates for unseen contexts. The two pools are reweighted online based on stage-wise feedback from an LLM-as-a-judge. Theoretically, we prove that this design guarantees global reachability of the solution space and achieves $O(\log T)$ cumulative regret, matching the stochastic bandit lower bound up to constants. In practice, across three MAS frameworks (AutoGen, DyLAN, AgentNet), three Qwen3 backbones (4B/8B/14B), two Gemma4 backbones (E2B/E4B) and five benchmarks spanning math, code, QA, and embodied tasks, DECENTMEM improves average accuracy by up to 23.8% over the strongest centralized memory baseline and by up to 52.5% over the no-memory baseline, while reducing token usage by up to 49%.

1 Introduction

Memory enables large language model (LLM) based multi-agent systems (MAS) to accumulate experience across tasks and improve over time. The dominant design pattern in current MAS is a *centralized* memory — a single shared repository that every agent reads from and writes to — inherited directly from single-agent memory architectures [1, 2, 3]. For example, MetaGPT's global message pool [4] and G-Memory's hierarchical shared graph [5] both default to sharing everything across agents. We argue, however, that centralization is the wrong default for multi-agent memory, and that *decentralized* agent-private memory is both better motivated and empirically superior, as illustrated in Fig. 1.

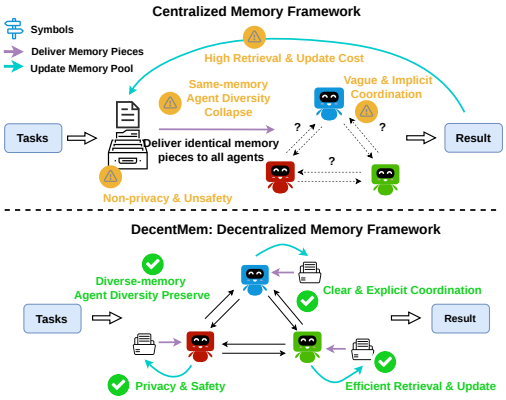

Figure 1: Compared to traditional centralized memory frameworks, DECENTMEM provides the first decentralized memory framework with adaptability to various MAS frameworks.

A key limitation of centralized MAS memory is that it is incompatible with self-evolution [6, 7, 8]. In multi-agent settings, self-evolution should preserve the role-complementary specialization

Correspondence: gh540@cam.ac.uk and zhuokai@uchicago.edu

that makes collaboration valuable in the first place, rather than drive the system toward a single shared strategy. Centralized memory directly undermines this objective: when every agent retrieves from the same pool, their behaviors homogenize over time, and the system collapses toward a single dominant strategy regardless of how distinct the agents' initial roles were [9]. The diversity that motivates instantiating multiple agents is gradually lost. Centralization also imposes well-documented system-level costs, including high communication cost [10, 11], synchronization and coordination tax [9, 12], and privacy risks [13].

Motivated by these limitations, we propose **DECENTMEM**, a decentralized self-evolving memory framework for LLM-based MAS. In DECENTMEM, each agent maintains its own dual-pool memory, which consists of an *exploitation pool (E-pool)* of consolidated trajectories from past tasks, and an *exploration pool (X-pool)* of LLM-generated candidates for unseen contexts. The two pools serve complementary functions, where the E-pool drives a similarity-based local walk that reuses prior successful strategies, while the X-pool injects fresh probability mass into regions of the strategy space the agent has never visited, breaking the local-optimum trap that traps pure exploitation. An LLM-as-a-judge evaluates each stage of the solution trajectory and re-weights the two pools online, allowing each agent to learn its own exploitation–exploration balance from feedback rather than committing to a fixed schedule. Decentralization preserves agent diversity because the consolidated experience never leaves the agent that earned it, and the dual-pool structure preserves the capacity to escape that experience when it is not useful in the current task or scenario.

We evaluate DECENTMEM both theoretically and empirically. Theoretically, we model multi-agent self-evolving search as a graph-structured random walk with heuristic teleportation and cast online routing between the two memory pools as a stochastic bandit problem. Under this formulation, we prove that DECENTMEM achieves global reachability over each agent's local solution subspace and attains $\mathcal{O}(\log T)$ cumulative regret, order-optimal against the $\Omega(\log T)$ bandit lower bound [14]. Empirically, across three MAS frameworks (AutoGen [15], DyLAN [16], AgentNet [13]), five LLM backbones spanning dense and mixture-of-experts (MoE) architectures (Qwen3-4B/8B/14B [17], Gemma4-E2B/E4B [18]), and five benchmarks including mathematical reasoning, code generation, question answering, and embodied decision-making, we show that DECENTMEM improves average accuracy by up to 23.8% over the strongest centralized memory baseline (G-Memory [5]) and by up to 52.5% over the no-memory baseline, while reducing token usage by up to 49%. Notably, the performance improvement widens as collaboration becomes more stochastic — moving from AutoGen's pre-designed workflows [15] to AgentNet's opportunistic coordination [13] — indicating that decentralized memory preserves the divergent trajectories that centralized memory would otherwise collapse.

Our main contributions are summarized as follows:

- *A decentralized dual-pool memory with online routing.* We introduce DECENTMEM, the first decentralized memory framework for LLM-based MAS, in which each agent maintains a private exploitation pool of consolidated trajectories and a private exploration pool of LLM-generated candidates. An online router, supervised by an LLM-as-a-judge, re-weights the two pools from stage-wise feedback so that each agent learns its own exploitation–exploration balance.
- *Theoretical guarantees.* We prove that DECENTMEM achieves global reachability over each agent's local solution subspace, ensuring no agent is permanently trapped in a locally suboptimal region. We further prove that online pool routing attains $\mathcal{O}(\log T)$ cumulative regret, matching the $\Omega(\log T)$ bandit lower bound up to constants and converging to the optimal exploitation–exploration balance at an order-optimal rate.
- *Comprehensive empirical validation.* Across three MAS frameworks, five LLM backbones, and five benchmarks, we demonstrate that DECENTMEM outperforms the strongest centralized baselines by 8.6% on average and uses up to 49% fewer tokens, with the largest gains in the most stochastic collaboration regimes.

2 Related Work

Single-agent memory. Memory is a central mechanism that enables LLM agents to accumulate experience and improve through repeated interaction with their environments. Early systems mainly extended the effective context window by retrieving past interactions, as in MemoryBank [1] and MemGPT [2]. More recent work has moved beyond simple retrieval and introduced higher-level memory abstractions, including reflective memory formation in Generative Agents [19], scalable personalized memory in Mem0 [3], dynamic memory organization in A-Mem [20], lightweight

multi-stage memory management in LightMem [21], and efficient lifelong memory compression in SimpleMem [22]. Despite these advances, existing memory designs are still largely developed for single-agent settings, where interaction histories are assumed to be linear and individually maintained. As a result, they are not well suited to the multi-threaded, collaborative, and interdependent dynamics of multi-agent systems.

LLM-based multi-agent systems. The deployment of multiple LLM agents has enabled collaborative capabilities beyond single-agent systems. Early frameworks, including AutoGen [15], CAMEL [23], and AgentVerse [24], established multi-agent collaboration through predefined roles, fixed interaction patterns, or manually designed coordination topologies. MetaGPT [4] and ChatDev [25] further formalized role-based workflows, while DyLAN [16] introduced inference-time agent selection and Agent Importance Score for adaptive collaboration. Recent methods such as GPTSwarm [26], AFlow [27], AgentNet [13], and Mixture-of-Minds [28] further automate the design or optimization of coordination structures. However, these approaches still focus primarily on task-level or benchmark-level coordination design, rather than persistent memory-driven improvement across tasks. As a result, most existing systems remain limited to static or one-shot adaptation and do not support genuine self-evolution through learning from past collaborations over time.

Memory in multi-agent systems. The intersection of memory and multi-agent systems remains relatively underexplored [9, 29]. Many existing MAS frameworks either omit dedicated memory components or incorporate only rudimentary within-trial memory, such as retaining the immediate conversation history [15, 16]. Existing cross-session memory designs often store condensed task-level artifacts rather than rich collaboration trajectories. For example, ChatDev mainly preserves distilled task summaries or past solutions, while discarding most fine-grained inter-agent interaction details [25]. MetaGPT employs a global shared message pool as a centralized communication and memory substrate [4], and G-Memory further extends this line of work with a hierarchical graph-based shared memory that stores insights, queries, and interaction traces [5]. However, such centralized memory repositories introduce synchronization and consistency overhead, access-control and privacy risks, and scalability bottlenecks [9, 30]. Moreover, exposing all agents to the same shared memory may homogenize agent behavior and weaken the emergence of specialized, role-complementary expertise [9, 13]. These limitations motivate decentralized memory architectures in which each agent maintains its own private experience while selectively sharing useful knowledge [9, 13, 30].

3 Preliminary

We first introduce the notation and formalize the basic concepts of a multi-agent system. Let $\mathcal{X} = \{x_1, \ldots, x_J\}$ denote the set of tasks, and let $\mathcal{A} = \{a_1, \ldots, a_M\}$ denote the set of collaborative agents for solving tasks in $\mathcal{X}$. Each agent $a_m \in \mathcal{A}$ is characterized by $a_m = \big(\mathrm{Base}_m, \mathrm{Role}_m, \mathcal{M}_m, \mathrm{Tool}_m\big)$, where Base_m denotes the underlying LLM, Role_m denotes the functional role or persona of the agent, $\mathcal{M}_m = \{z_1, z_2, \ldots, z_j\}$ denotes its memory space, and Tool_m denotes the set of tools accessible to the agent. Each memory piece $z_i \in \mathcal{M}_m$ is represented as $z_i = (\xi_i, r_i^\star)$, where ξ_i is the context prototype and $r_i^\star$ is the associated action prototype. Depending on the memory framework, $\mathcal{M}_m$ may refer either to memory retrieved from a centralized repository or to the private memory space maintained by agent a_m in a decentralized setting.

Given an input task $x \in \mathcal{X}$, the system evolves through a finite sequence of stages $\mathcal{T} = \{t_1, \ldots, t_N\}$, where each stage corresponds to one step of collaborative problem solving, such as decomposition, reasoning, verification, or integration. At stage t, let $\mathcal{A}^{(t)} \subseteq \mathcal{A}$ denote the set of active agents. Multiple agents may be active simultaneously at the same stage. For each active agent $a_m \in \mathcal{A}^{(t)}$, we define its local context as $c_m^{(t)} = \big(x,\ \phi_m^{(t)},\ \rho_m^{(t)}\big)$, where $\phi_m^{(t)}$ denotes the information induced by previous actions of neighboring or predecessor agents, and $\rho_m^{(t)}$ denotes the memory retrieved from $\mathcal{M}_m$. Conditioned on the local context $c_m^{(t)}$, agent a_m produces an action or intermediate output $v_m^{(t)} = \pi_m\big(c_m^{(t)}\big)$, where π_m denotes the policy of agent a_m. After all active agents produce their outputs, the system aggregates them as $s^{(t)} = \mathrm{Agg}\big(\{v_m^{(t)}\}_{a_m \in \mathcal{A}^{(t)}}\big)$, and outputs the final answer.

4 DecentMem

In this section, we present the key design of DecentMem, a *decentralized* memory mechanism for multi-agent systems. Each agent is equipped with dual memory pools that store collaboration-aware memory pieces. At execution time, an online router retrieves task-relevant experience from one

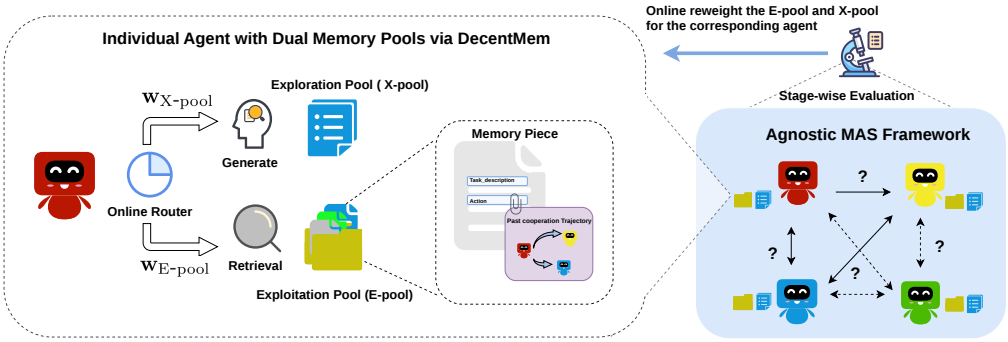

Figure 2: Overview of DECENTMEM. Each agent maintains a private dual-pool memory (left): an *exploitation pool (E-pool)* of consolidated trajectories from past tasks and an *exploration pool (X-pool)* for generating novel candidates in unseen contexts. At each stage, an online router selects between the two pools with probability proportional to their weights w_{E-pool} and w_{X-pool}, retrieving from the E-pool or generating from the X-pool to produce the agent's action. DECENTMEM is agnostic to the underlying MAS framework (right): agents collaborate under any topology, from pre-designed workflows to opportunistic coordination. After execution, an external LLM-as-a-judge evaluates the trajectory stage by stage and reweights each agent's E-pool and X-pool based on stage-wise feedback.

pool or generates new memory from the other for exploration; after execution, the two pools are reweighted online based on stage-wise feedback from an LLM-as-a-judge.

4.1 Dual-Memory Pools

For each agent a_m, DECENTMEM partitions the accessible memory into two pools:

$$\mathcal{M}_m = \mathcal{M}_{m,\text{E-pool}} \cup \mathcal{M}_{m,\text{X-pool}}. \tag{1}$$

The exploitation pool (E-pool), denoted by $\mathcal{M}_{m,\text{E-pool}}$, stores consolidated experience from previous tasks and supports retrieval-based exploitation over historical memory pieces that encode past collaboration trajectories. In contrast, the exploration pool (X-pool) serves as a temporary buffer for the current task and is used to generate novel experiences. As will be shown in §5, this dual-memory design helps the system achieve global reachability over the solution space. Specifically, the E-pool enables retrieval-based exploitation through a local-walk mechanism, whereas the X-pool realizes a heuristic teleportation mechanism induced by the LLM prior. Together, the two pools define a mixed search process that balances local exploitation with global exploration while preventing the search from being trapped in the local optima.

Continue with the definitions from §3, each memory piece in $\mathcal{M}_{m,\text{E-pool}}$ is represented as

$$z = (\xi, r^\star), \qquad r^\star = (r_{\text{trajectory}}, r_{\text{comment}})$$

where $r_{\text{trajectory}}$ includes specific action such as task decomposition and direct answering, together with cooperation trajectory passed to subsequent agents for the next stage induced by the action, and r_{comment} denotes the agent's action-level self-commentary, including its rationale for choosing the corresponding action in past interactions. Unlike standard MAS memory, our memory piece z records not only *what* was solved, but also *how* the task was solved and *who* executed each sub-task. Retrieved memory therefore serves as both an action prior and a coordination-trajectory prior, making each memory piece valuable not only for retrieval and policy reuse, but also for directly learning from past successful cooperation experience.

4.2 Memory Retrieval for New Tasks

DECENTMEM is designed as a plug-in module that can be integrated into mainstream MAS frameworks. When a task x arrives, it is first dispatched to an active agent a_n according to the underlying scheduling policy. Suppose that at stage t, an active agent a_m encounters an intermediate sub-task $x^{(t)}$. The online router of a_m first selects one memory pool from the dual-memory structure according to the corresponding weights $w_{m,\text{E-pool}}$ and $w_{m,\text{X-pool}}$. The probability of selecting the E-pool is defined as

$$\alpha_{m,\text{E-pool}} = \frac{w_{m,\text{E-pool}}}{w_{m,\text{E-pool}} + w_{m,\text{X-pool}}}, \qquad \alpha_{m,\text{X-pool}} = 1 - \alpha_{m,\text{E-pool}}. \tag{2}$$

If the E-pool is selected, the agent retrieves memory pieces according to similarity:

$$\rho_m^{(t)} = \text{Top-K}\Big(\mathcal{M}_{m,\text{E-pool}};\ \text{sim}\big(e(x^{(t)}), e(\xi_z)\big)\Big), \tag{3}$$

subject to a similarity threshold τ. If no sufficiently similar memory is found, the online router falls back to the X-pool. This threshold avoids mismatched reuse from the E-pool when the current task deviates substantially from previously consolidated experience, thereby promoting faster adaptation to novel tasks. If the X-pool is selected directly, the agent instantiates an exploratory memory piece z_{new} for the current context. Either the retrieved set $\rho_m^{(t)}$ or exploratory memory piece z_{new} is then provided to the LLM for executable action generation:

$$v_m^{(t)} = \Pi_{\text{act}}\big(x^{(t)}, \phi_m^{(t)}, \rho_m^{(t)}/z_{\text{new}}\big). \tag{4}$$

Notably, when a retrieved E-pool memory piece contains a compatible collaboration trajectory, its historical allocation can be reused as a coordination prior for the next stage. After all active agents produce their outputs, the system aggregates them as

$$s^{(t)} = \text{Agg}\big(\{v_m^{(t)}\}_{a_m \in \mathcal{A}^{(t)}}\big), \tag{5}$$

and outputs the final answer after the last stage.

4.3 Dual-Memory Update

After execution, the full solution trajectory is evaluated stage by stage by an LLM evaluator. Rather than scoring only the final answer, the evaluator assesses each stage in terms of *correctness*, *allocation quality*, *intermediate coherence*, and *final integration*. Specifically, let q_{prev} and q_{curr} denote the scores of two consecutive stages, and define $\Delta_t = \mathbb{I}[q_{\text{curr}} > q_{\text{prev}}]$. The online router is then updated according to the memory pool used at stage t. Let $\alpha = 0.5$ and $\beta = 0.5$, if the E-pool is used, its weight is increased when $\Delta_t = 1$, and otherwise decayed:

$$w_{m,\text{E-pool}} \leftarrow \begin{cases} w_{m,\text{E-pool}} + \alpha, & \text{if } \Delta_t = 1, \\ \max(1.0, \beta\, w_{m,\text{E-pool}}), & \text{otherwise.} \end{cases} \tag{6}$$

In contrast, if the X-pool is used, the update is reversed:

$$w_{m,\text{E-pool}} \leftarrow \begin{cases} \max(1.0, \beta\, w_{m,\text{E-pool}}), & \text{if } \Delta_t = 1, \\ w_{m,\text{E-pool}} + \alpha, & \text{otherwise.} \end{cases} \tag{7}$$

Meanwhile, $w_{m,\text{X-pool}} = 1.0$ remains fixed. In this way, successful exploitation increases reliance on the E-pool, while successful exploration prevents the router from over-committing to past experience.

After the task is completed, all memory pieces in the X-pool are consolidated into the E-pool, and the X-pool is reset:

$$\mathcal{M}_{m,\text{E-pool}} \leftarrow \mathcal{M}_{m,\text{E-pool}} \cup \mathcal{M}_{m,\text{X-pool}}, \qquad \mathcal{M}_{m,\text{X-pool}} \leftarrow \varnothing. \tag{8}$$

5 Theoretical Analysis

In this section, we formalize self-evolving multi-agent systems as strategy search over a graph-structured solution space and cast the learning objective as cumulative regret minimization. We then show that DECENTMEM guarantees global reachability over the decentralized search space and achieves $O(\log T)$ cumulative regret.

5.1 Problem Formulation

We model the full solution space as a graph $\mathcal{G} = (V, \mathcal{E})$, where each node $v \in V$ denotes a candidate solution strategy, and each edge $(v, v') \in \mathcal{E}$ indicates that the two strategies are highly similar and can be transformed into one another. For each agent a_m, we define an agent-specific solution subspace

$$\mathcal{G}_m = (V_m, \mathcal{E}_m), \qquad V_m \subseteq V,\ \mathcal{E}_m \subseteq \mathcal{E}, \qquad \mathcal{G} = \mathcal{G}_1 \cup \mathcal{G}_2 \cup \cdots \cup \mathcal{G}_M.$$

Thus, $\mathcal{G}_m$ characterizes the local geometry of the strategy space accessible to agent a_m. Unlike centralized memory, which searches directly over the global space $\mathcal{G}$, DECENTMEM enables each agent to search within its own subspace $\mathcal{G}_m$. Given a task x, let $\tau(x) = \{v_m^{(t)} \mid t = 1, \ldots, N,\ a_m \in \mathcal{A}^{(t)}\}$ denote the joint trajectory induced by all active agents, $R_x\big(\tau(x), \hat{y}(x)\big)$ the task reward, which evaluates both the collaborative process and the final output, and $R_x^\star = \max_{\tau,\hat{y}} R_x(\tau, \hat{y})$ the optimal achievable reward for task x, the cumulative regret over the task set $\mathcal{X}$ and the objective are therefore

$$\text{Regret}(\mathcal{X}) = \sum_{x \in \mathcal{X}} \big(R_x^\star - \mathbb{E}\big[R_x\big(\tau(x), \hat{y}(x)\big)\big]\big), \quad \text{Objective} = \min_{\{\pi_m\}_{m=1}^M} \text{Regret}(\mathcal{X}).$$

5.2 Global Reachability

We analyze the search behavior of each agent as a dynamic random walk over its graph-structured solution subspace. For a given agent a_m, let $\mathcal{Q}_m = \{q_{m,1}, q_{m,2}, \ldots, q_{m,L_m}\}$ denote the sequence of local subproblems encountered by a_m during execution, where $q_{m,\ell}$ is the ℓ-th subproblem assigned to agent a_m. Note this sequence is agent-specific and should be distinguished from the global stage sequence used to describe the overall multi-agent collaboration process.

Let $p_{m,\ell} \in \mathbb{R}^{|V_m|}$ denote the state distribution of agent a_m when solving subproblem $q_{m,\ell}$, where $p_{m,\ell}$ lies in the probability simplex over V_m. In DECENTMEM, the two memory pools induce two complementary search operators. The exploitation pool (E-pool) defines a local walk governed by a similarity-based transition matrix T_m, which captures the reuse of historically relevant strategies. The exploration pool (X-pool) defines a teleportation mechanism induced by an LLM prior $h_m \in \Delta^{|V_m|}$, where $(h_m)_i > 0$ for every feasible state $i \in V_m$. The resulting transition is

$$p_{m,\ell+1} = M_{m,\ell} p_{m,\ell} = \left(\alpha_{m,\ell} T_m + (1 - \alpha_{m,\ell})\, h_m \mathbf{1}^\top\right) p_{m,\ell}, \tag{9}$$

where $\alpha_{m,\ell} = \frac{w_{m,\text{E-pool}}}{w_{m,\text{E-pool}} + w_{m,\text{X-pool}}}$. Eq. (9) shows that DECENTMEM combines local exploitation with global exploration within the local subspace $\mathcal{G}_m$: the first term preserves local structural bias, while the second term injects nonzero probability mass into every feasible region.

Theorem 1 (Global Reachability)**.** *Assume that $w_{m,\text{X-pool}} > 0$ throughout the search process, so that $\alpha_{m,\ell} < 1$ for all ℓ, and assume that $(h_m)_i > 0$ for every feasible state $i \in V_m$. Then the transition matrix*

$$M_{m,\ell} = \alpha_{m,\ell} T_m + (1 - \alpha_{m,\ell})\, h_m \mathbf{1}^\top$$

is strictly positive. Consequently, the induced Markov chain over V_m is irreducible and aperiodic, and the search process of agent a_m is globally reachable over its entire subspace $\mathcal{G}_m$.

Since this argument holds for every agent a_m, DECENTMEM ensures global reachability across the decentralized solution space $\mathcal{G} = \bigcup_{m=1}^{M} \mathcal{G}_m$. The detailed proof is deferred to Appendix A.1.

5.3 Logarithmic Cumulative Regret

We next analyze the routing efficiency of DECENTMEM from an online optimization perspective. At round t, the online router of agent a_m selects between the E-pool and the X-pool according to their current weights. Let $r_{\text{E}}(\alpha)$ and $r_{\text{X}}(\alpha)$ denote the expected rewards of the E-pool and X-pool under routing probability α, respectively. The resulting expected reward is $r(\alpha) = \alpha\, r_{\text{E}}(\alpha) + (1-\alpha)\, r_{\text{X}}(\alpha)$.

Assumption 1. *The function $r : [0.5, 1] \to [0, 1]$ is strictly concave, twice continuously differentiable, and admits a unique maximizer $\alpha^\star \in (0.5, 1)$.*

For a routing sequence $\{\alpha_t\}_{t=1}^{T}$, we define the cumulative regret as $R(T) = \sum_{t=1}^{T} \Big(r(\alpha^\star) - \mathbb{E}[r(\alpha_t)]\Big)$. The update rule of DECENTMEM induces an adaptive recursion on α_t: successful exploitation increases α_t, whereas successful exploration suppresses over-dominance of the E-pool and decreases α_t. The router steers the system toward the optimal exploitation–exploration balance.

Theorem 2 (Logarithmic Regret)**.** *Under Assumption 1, the routing policy of* DECENTMEM *satisfies $\mathbb{E}[R(T)] = O(\log T)$. Hence,* DECENTMEM *achieves $O(\log T)$ cumulative regret.*

The proof is detailed in Appendix A.2. Moreover, stochastic bandit theory implies that, for some problem instances, any policy incurs expected cumulative regret at least $\Omega(\log T)$ [14]. Therefore, the regret bound of DECENTMEM is order-optimal up to constant factors.

6 Experiment

6.1 Experimental Setup

Datasets. We evaluate DECENTMEM on five public benchmarks spanning four task categories. These cover mathematical reasoning (AIME25 [31], AIME24 [32]), code generation (MBPP-Plus [33]), question answering (BBH [34]), and embodied decision-making (ALFWorld [35]). AIME25 and AIME24 are combined into a single dataset and reported as AIME25&24 in the tables.

Baselines. We compare DECENTMEM against three centralized memory baselines including MetaGPT [4], ChatDev [25], and G-Memory [5], and a no-memory baseline that runs the underlying MAS framework without any cross-session memory.

MAS Frameworks and LLM Backbones. A central design property of DECENTMEM is that it is agnostic to both the underlying MAS framework and the LLM backbone. To validate this, we deliberately span the experimental matrix along both axes. We integrate DECENTMEM and all baselines into three MAS frameworks chosen to cover a broad spectrum of coordination structures: **AutoGen** [15], which represents pre-designed workflow-based collaboration; **DyLAN** [16], which captures more dynamic and partially stochastic interaction patterns; and **AgentNet** [13], which reflects unstructured and opportunistic coordination. The degree of stochasticity increases progressively from AutoGen to AgentNet, allowing us to test whether DECENTMEM's gains are robust across coordination regimes. And, as we will show in §6.2, the gains in fact widen as stochasticity grows. We instantiate each MAS framework with five open-source LLM backbones spanning two model families and various sizes: QWEN3-4B, QWEN3-8B, QWEN3-14B [17], GEMMA4-E2B-IT, and GEMMA4-E4B-IT [18]. This range lets us verify that DECENTMEM's benefits do not depend on a particular model family, scale, or architecture. We instantiate the QWEN series models locally via Ollama [36], and the GEMMA series models locally using the Hugging Face Transformers library [37].

6.2 Main Results

DECENTMEM consistently outperforms all baselines. As shown in Table 1 and 2, DECENTMEM achieves the best average accuracy in 14 of 15 (backbone, framework) cells, with an average relative improvement of 8.6% over the strongest centralized baseline and 26.1% over the no-memory baseline (up to 52.5% on Qwen3-4B + AgentNet). Beyond steady-state accuracy, DECENTMEM also exhibits stronger self-evolution with accumulated experience, reaching strong performance with substantially fewer tasks (§7.1).

The advantage of decentralization grows with coordination stochasticity. As stochasticity increases from AutoGen (pre-designed workflows) to DyLAN (partially stochastic) to AgentNet (opportunistic), the relative margin widens monotonically on both families: 2.7% / 9.2% / 23.1% on Qwen3 and 1.7% / 3.9% / 6.8% on Gemma4. This is direct evidence for the paper's central claim — when coordination is less predictable, centralized memory homogenizes agent behavior more aggressively, and preserving per-agent experience matters more.

7 Analysis and Ablation Study

7.1 Self-Evolution with Experience

Fig. 3 traces cumulative accuracy on MBPP-Plus across the three MAS frameworks under Qwen3-4B. DECENTMEM reaches strong performance substantially faster than all three centralized baselines, with the convergence-speed advantage most pronounced on DyLAN (~2.5× faster). On AgentNet, DECENTMEM also surpasses every baseline's final accuracy, indicating that under stochastic coordination it not only learns faster but discovers strategies centralized memory cannot recover.

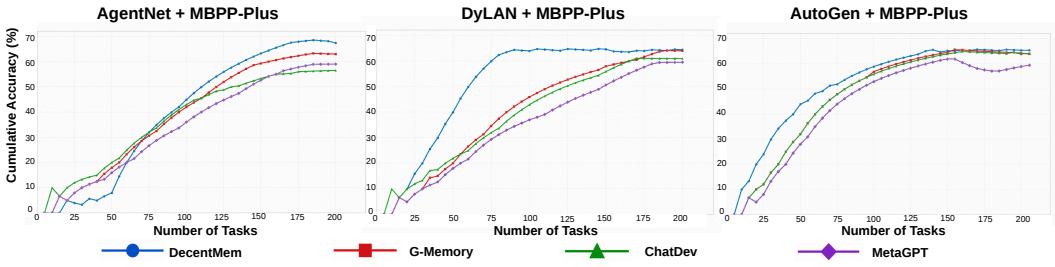

Figure 3: Cumulative accuracy vs. number of tasks on MBPP-Plus across three MAS frameworks under Qwen3-4B. DECENTMEM exhibits stronger self-evolution than all three centralized baselines.

7.2 Cost Analysis

Beyond accuracy, a practical memory framework must scale efficiently in token usage. Fig. 4 plots performance against token cost on BBH across the three MAS frameworks under Qwen3-8B. DECENTMEM sits at the upper-left corner of all three plots — highest accuracy at the lowest token cost. Relative to G-Memory, the strongest centralized baseline, DECENTMEM reduces token consumption by 32%, 49%, and 47% on AgentNet, DyLAN, and AutoGen respectively (43% on average) while improving accuracy.

Table 1: Main results on the Qwen3 family across three MAS frameworks and five datasets. Bold marks the best result and underline marks the second-best within each (backbone, framework) block.

Backbone	Framework	Memory	AIME25&24	MBPP-Plus	ALFWorld	BBH	Avg.
QWEN3-4B	AgentNet	No memory	13.33	53.20	46.12	28.13	35.20
		MetaGPT	8.33	59.14	50.47	40.86	39.70
		ChatDev	18.33	56.40	49.23	33.67	39.41
		G-Memory	10.00	63.18	52.14	48.14	43.37
		DECENTMEM	23.33	67.53	64.45	59.43	53.69
	DyLAN	No memory	16.67	54.72	48.54	41.17	40.28
		MetaGPT	15.00	58.60	51.17	45.23	42.50
		ChatDev	21.67	61.23	48.56	44.73	44.05
		G-Memory	18.33	63.93	52.30	43.97	44.63
		DECENTMEM	26.67	64.50	57.25	52.19	50.15
	AutoGen	No memory	18.33	58.14	54.27	44.43	43.79
		MetaGPT	11.67	58.92	59.53	54.19	46.08
		ChatDev	20.00	64.72	63.87	53.37	50.49
		G-Memory	15.00	63.87	69.17	61.53	52.39
		DECENTMEM	21.67	65.63	67.73	63.80	54.71
QWEN3-8B	AgentNet	No memory	23.33	62.94	61.40	35.13	45.70
		MetaGPT	13.34	66.94	66.51	57.14	50.98
		ChatDev	31.67	64.99	64.03	42.11	50.70
		G-Memory	16.67	70.94	70.19	62.18	55.00
		DECENTMEM	36.67	77.93	80.29	76.19	67.77
	DyLAN	No memory	26.67	65.91	63.19	54.68	52.61
		MetaGPT	23.33	67.97	68.21	61.58	55.27
		ChatDev	31.67	70.94	64.64	59.61	56.72
		G-Memory	30.00	74.49	69.43	58.62	58.14
		DECENTMEM	40.00	72.99	74.42	65.52	63.23
	AutoGen	No memory	28.33	66.94	78.81	61.88	58.99
		MetaGPT	20.00	68.99	83.25	76.12	62.09
		ChatDev	28.33	71.05	89.14	71.14	64.91
		G-Memory	25.00	73.92	92.11	82.09	68.28
		DECENTMEM	31.67	75.97	91.54	85.07	71.06
QWEN3-14B	AgentNet	No memory	26.67	68.72	66.20	42.61	51.05
		MetaGPT	20.00	68.81	71.32	65.48	56.40
		ChatDev	36.67	76.78	68.50	48.92	57.72
		G-Memory	23.33	71.29	75.61	68.52	59.69
		DECENTMEM	41.66	82.27	83.59	84.30	72.95
	DyLAN	No memory	30.00	72.10	68.82	60.52	57.86
		MetaGPT	26.67	76.95	73.11	68.86	61.40
		ChatDev	35.00	79.17	69.43	65.28	62.22
		G-Memory	31.67	85.75	74.24	66.84	64.62
		DECENTMEM	40.00	82.39	80.11	72.43	68.73
	AutoGen	No memory	30.00	74.21	81.51	67.33	63.26
		MetaGPT	23.33	76.37	89.43	81.68	67.70
		ChatDev	33.33	80.36	94.89	79.28	71.97
		G-Memory	33.33	85.23	96.69	89.20	76.11
		DECENTMEM	35.00	84.74	93.13	90.50	75.84

7.3 Ablation Study

To isolate the contribution of the online weighting, we compare it against three fixed routing strategies sharing the same dual-memory architecture: *Exploitation Only* ($\alpha = 1$), *Exploration Only* ($\alpha = 0$), and *Fixed Weight* ($\alpha = 0.5$). As shown in Table 3, replacing it with the strongest fixed policy (*Exploitation Only*) costs 6.93%, 3.51%, and 3.38% on AgentNet, DyLAN, and AutoGen respectively, with larger gaps against the other two. Notably, *Exploration Only* collapses on BBH but remains competitive on AIME, showing that exploration alone suffices for fresh reasoning problems but cannot substitute for accumulated experience on tasks with reusable structure.

Table 2: Main results on the Gemma4 family across three MAS frameworks and five datasets. Bold marks the best result and underline marks the second-best within each (backbone, framework) block.

Backbone	Framework	Memory	AIME25&24	MBPP-Plus	ALFWorld	BBH	Avg.
GEMMA4-E2B	AgentNet	No memory	23.33	55.67	60.43	21.32	40.19
		MetaGPT	18.33	57.99	63.37	30.85	42.64
		ChatDev	26.67	57.00	62.27	27.11	43.26
		G-Memory	20.00	62.13	68.47	38.15	47.19
		DECENTMEM	25.00	64.08	70.80	42.56	50.61
	DyLAN	No memory	23.33	57.62	62.29	28.67	42.98
		MetaGPT	21.66	60.15	64.51	35.42	45.44
		ChatDev	28.33	62.81	63.00	31.47	46.40
		G-Memory	25.00	69.89	68.11	36.39	49.85
		DECENTMEM	28.33	66.79	69.53	43.47	52.03
	AutoGen	No memory	25.00	59.77	66.38	35.15	46.58
		MetaGPT	28.33	62.93	67.49	40.83	49.89
		ChatDev	30.00	65.57	74.59	46.45	54.15
		G-Memory	28.33	64.82	66.73	42.17	50.51
		DECENTMEM	33.33	69.28	73.43	41.87	54.48
GEMMA4-E4B	AgentNet	No memory	33.33	70.23	75.67	31.92	52.79
		MetaGPT	25.00	74.49	77.47	48.14	56.28
		ChatDev	38.33	72.18	81.59	35.74	56.96
		G-Memory	30.00	78.39	82.39	55.38	61.54
		DECENTMEM	43.33	77.17	81.46	59.69	65.41
	DyLAN	No memory	36.67	72.49	76.28	45.17	57.65
		MetaGPT	31.67	74.69	80.49	52.41	59.82
		ChatDev	38.33	75.37	77.23	50.69	60.41
		G-Memory	36.67	82.45	84.49	55.83	64.86
		DECENTMEM	41.67	81.49	84.21	60.84	67.05
	AutoGen	No memory	35.00	75.52	82.73	49.53	60.70
		MetaGPT	36.67	77.69	86.27	58.37	64.75
		ChatDev	41.67	83.04	88.99	55.46	67.29
		G-Memory	38.33	81.17	89.17	63.72	68.10
		DECENTMEM	40.00	79.39	92.28	68.09	69.94

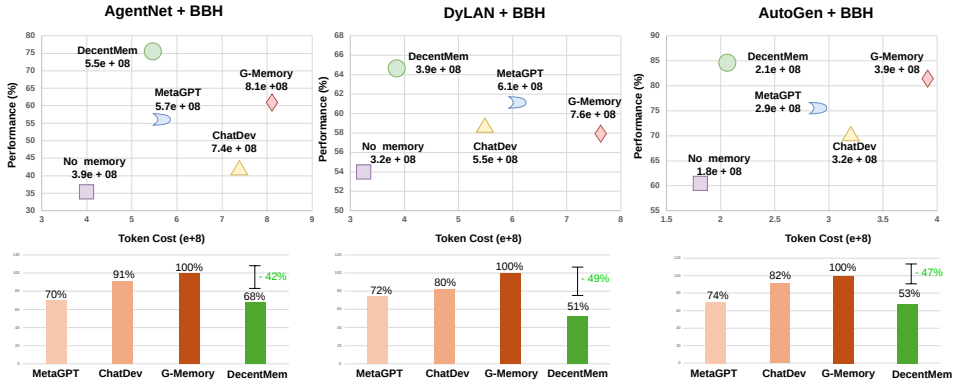

Figure 4: Token cost vs. performance on BBH across three MAS frameworks under QWEN3-8B.

Table 3: Ablation of the online router. All variants share the same dual-pool architecture; only the routing policy differs. **Bold** indicates the best performance.

		AgentNet				DyLAN				AutoGen			
		Qwen3-8B		Gemma4-E4B		Qwen3-8B		Gemma4-E4B		Qwen3-8B		Gemma4-E4B	
Routing	α	BBH	AIME	BBH	AIME	BBH	AIME	BBH	AIME	BBH	AIME	BBH	AIME
Exploit	1	74.27	26.67	57.23	30.00	63.59	31.67	58.72	40.00	84.29	28.33	65.37	33.33
Explore	0	35.13	23.33	31.92	33.33	54.68	26.67	45.17	36.67	61.88	28.33	49.53	35.00
Fixed	0.5	65.28	33.33	53.18	40.00	60.21	35.00	55.38	**43.33**	78.34	**31.67**	63.38	38.33
Online	–	**76.19**	**36.67**	**59.69**	**43.33**	**65.52**	**40.00**	**60.84**	41.67	**85.07**	**31.67**	**68.09**	**40.00**

8 Conclusion and Limitation

In this paper, we argued that centralized memory is the non-optimal default for multi-agent systems, as sharing a single pool erodes the role-complementary specialization that motivates having multiple agents at all, and proposed DECENTMEM, a decentralized dual-pool memory with online routing between exploitation and exploration. The construction admits an $O(\log T)$ regret guarantee under a graph-walk-with-teleportation reading, and the empirical margin over the strongest centralized baseline grows monotonically as coordination becomes more stochastic.

Limitation. Although DECENTMEM is evaluated across four domains, broader validation on more diverse and high-stakes tasks, such as legal and medical reasoning, would further strengthen its empirical soundness. We leave this direction for future work.

References

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

[2] Charles Packer, Vivian Fang, Shishir G. Patil, Kevin Lin, Sarah Wooders, and Joseph E. Gonzalez. Memgpt: Towards llms as operating systems, 2023.

[3] Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory. In *European Conference on Artificial Intelligence*, 2025. URL https://api.semanticscholar.org/CorpusID:278165315.

[4] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zi Hen Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. Metagpt: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations*, 2023. URL https://api.semanticscholar.org/CorpusID:265301950.

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

[6] Noah Shinn, Federico Cassano, Beck Labash, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems 36*, 2023. URL https://api.semanticscholar.org/CorpusID:258833055.

[7] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi (Jim) Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *ArXiv*, abs/2305.16291, 2023. URL https://api.semanticscholar.org/CorpusID:258887849.

[8] Zhaorun Chen, Zhuokai Zhao, Kai Zhang, Bo Liu, Qi Qi, Yifan Wu, Tarun Kalluri, Sara Cao, Yuanhao Xiong, Haibo Tong, Huaxiu Yao, Hengduo Li, Jiacheng Zhu, Xian Li, Dawn Song, Bo Li, Jason Weston, and Dat Huynh. Scaling agent learning via experience synthesis, 2025. URL https://arxiv.org/abs/2511.03773.

[9] Zhongming Yu, Naicheng Yu, Hejia Zhang, Wentao Ni, Mingrui Yin, Jiaying Yang, Yujie Zhao, and Jishen Zhao. Multi-agent memory from a computer architecture perspective: Visions and challenges ahead, 2026. URL https://api.semanticscholar.org/CorpusID:286457695.

[10] Jeremy Hadfield, Barry Zhang, Kenneth Lien, Florian Scholz, Jeremy Fox, and Daniel Ford. How we built our multi-agent research system. https://www.anthropic.com/engineering/multi-agent-research-system, June 2025. Anthropic Engineering Blog. Accessed: 2026-04-27.

[11] Kelly Hong, Anton Troynikov, and Jeff Huber. Context rot: How increasing input tokens impacts llm performance. Technical report, Chroma, July 2025. URL https://trychroma.com/research/context-rot.

[12] Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, et al. Why do multi-agent llm systems fail? *arXiv preprint arXiv:2503.13657*, 2025.

[13] Yingxuan Yang, Huacan Chai, Shuai Shao, Yuanyi Song, Siyuan Qi, Renting Rui, and Weinan Zhang. Agentnet: Decentralized evolutionary coordination for llm-based multi-agent systems. *ArXiv*, abs/2504.00587, 2025. URL https://api.semanticscholar.org/CorpusID:277468263.

[14] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47:235–256, 2002. URL https://api.semanticscholar.org/CorpusID:207609497.

[15] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First conference on language modeling*, 2024.

[16] Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. Dynamic llm-agent network: An llm-agent collaboration framework with agent team optimization. *arXiv preprint arXiv:2310.02170*, 2023.

[17] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

[18] Clement Farabet and Olivier Lacombe. Gemma 4: Byte for byte, the most capable open models. https://blog.google/innovation-and-ai/technology/developers-tools/gemma-4/, April 2026. Google Blog, The Keyword. Accessed: 2026-04-27.

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

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

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

[22] Jiaqi Liu, Yaofeng Su, Peng Xia, Siwei Han, Zeyu Zheng, Cihang Xie, Mingyu Ding, and Huaxiu Yao. Simplemem: Efficient lifelong memory for llm agents. *ArXiv*, abs/2601.02553, 2026. URL https://api.semanticscholar.org/CorpusID:284512931.

[23] G. Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for "mind" exploration of large language model society. *Advances in Neural Information Processing Systems 36*, 2023. URL https://api.semanticscholar.org/CorpusID:268042527.

[24] Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, et al. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *The Twelfth International Conference on Learning Representations*, 2023.

[25] Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, et al. Chatdev: Communicative agents for software development. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pages 15174–15186, 2024.

[26] Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. Gptswarm: Language agents as optimizable graphs. In *Forty-first International Conference on Machine Learning*, 2024.

[27] Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. Aflow: Automating agentic workflow generation. *arXiv preprint arXiv:2410.10762*, 2024.

[28] Yuhang Zhou, Mingrui Zhang, Ke Li, Mingyi Wang, Qiao Liu, Qifei Wang, Jiayi Liu, Fei Liu, Serena Li, Weiwei Li, et al. Mixture-of-minds: Multi-agent reinforcement learning for table understanding. *arXiv preprint arXiv:2510.20176*, 2025.

[29] Zeyu Zhang, Quanyu Dai, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model-based agents. *ACM Transactions on Information Systems*, 43:1 – 47, 2024. URL https://api.semanticscholar.org/CorpusID:269293320.

[30] Alireza Rezazadeh, Zichao Li, Ange Lou, Yuying Zhao, Wei Wei, and Yujia Bao. Collaborative memory: Multi-user memory sharing in llm agents with dynamic access control. *ArXiv*, abs/2505.18279, 2025. URL https://api.semanticscholar.org/CorpusID:278904585.

[31] math ai. Aime 2025 dataset. https://huggingface.co/datasets/math-ai/aime25, 2025.

[32] Maxwell-Jia. Aime 2024 dataset. https://huggingface.co/datasets/Maxwell-Jia/AIME_2024, 2024.

[33] EvalPlus. Mbppplus. https://huggingface.co/datasets/evalplus/mbppplus, 2024. Hugging Face dataset. Accessed: 2026-04-27.

[34] Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed Chi, Denny Zhou, et al. Challenging big-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 13003–13051, 2023.

[35] Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2020.

[36] Ollama. https://github.com/ollama/ollama.

[37] Hugging face transformers. https://github.com/huggingface/transformers.

[38] Vivek S. Borkar. *Stochastic Approximation: A Dynamical Systems Viewpoint*, volume 48 of *Texts and Readings in Mathematics*. Springer Singapore, 2 edition, 2024. ISBN 978-981-99-8277-6. doi: 10.1007/978-981-99-8277-6. URL https://doi.org/10.1007/978-981-99-8277-6.

A Additional Theoretical Analysis

In this appendix, we provide a rigorous theoretical foundation for the dual-memory design of **DECENTMEM**. By modeling each agent's search process as a random walk with heuristic teleportation over a graph-structured solution subspace, we show that the adaptive dual-memory mechanism guarantees global reachability and yields superior search efficiency in terms of asymptotic regret.

A.1 Global Reachability

We formalize the search process of each agent as a random walk over its graph-structured solution subspace. Since agents make decisions independently conditioned on their local context, it suffices to analyze a single agent. For agent a_m, let

$$\mathcal{G}_m = (V_m, \mathcal{E}_m)$$

denote its local solution graph, where each node in V_m represents a candidate solution strategy and each edge in $\mathcal{E}_m$ encodes a local similarity relation between strategies.

During execution, agent a_m encounters a sequence of local subproblems

$$\mathcal{Q}_m = \{q_{m,1}, q_{m,2}, \ldots, q_{m,L_m}\},$$

where $q_{m,\ell}$ denotes the ℓ-th subproblem assigned to a_m. Let

$$p_{m,\ell} \in \Delta^{|V_m|}$$

denote the state distribution of agent a_m when solving $q_{m,\ell}$, where $\Delta^{|V_m|}$ is the probability simplex over V_m.

In **DECENTMEM**, the two memory pools induce two complementary search operators. The exploitation pool (E-pool) defines a local walk governed by a column-stochastic transition matrix

$$T_m \in \mathbb{R}^{|V_m| \times |V_m|},$$

where $(T_m)_{ij}$ denotes the probability of moving from state j to state i according to historical similarity. The exploration pool (X-pool) defines a teleportation mechanism induced by an LLM prior

$$h_m \in \Delta^{|V_m|}, \qquad (h_m)_i > 0, \ \forall i \in V_m.$$

The resulting transition when moving from subproblem $q_{m,\ell}$ to $q_{m,\ell+1}$ is

$$p_{m,\ell+1} = M_{m,\ell} p_{m,\ell} = \left(\alpha_{m,\ell} T_m + (1 - \alpha_{m,\ell})\, h_m \mathbf{1}^\top\right) p_{m,\ell}, \tag{10}$$

where

$$\alpha_{m,\ell} = \frac{w_{m,\text{E-pool},\ell}}{w_{m,\text{E-pool},\ell} + w_{m,\text{X-pool},\ell}} \in [0, 1). \tag{11}$$

Equation (10) shows that **DECENTMEM** combines local exploitation with global exploration. The term $\alpha_{m,\ell} T_m$ preserves the local geometry induced by past experience, while the rank-one term $(1 - \alpha_{m,\ell}) h_m \mathbf{1}^\top$ injects nonzero probability mass into every feasible state.

Theorem 3 (Global Reachability)**.** *Assume that the exploration pool remains active throughout the search process, i.e.,*

$$w_{m,\text{X-pool},\ell} > 0 \qquad \textit{for all } \ell,$$

so that $\alpha_{m,\ell} < 1$, and assume that the LLM prior has full support over the feasible state space:

$$(h_m)_i > 0, \qquad \forall i \in V_m.$$

Then the transition matrix

$$M_{m,\ell} = \alpha_{m,\ell} T_m + (1 - \alpha_{m,\ell}) h_m \mathbf{1}^\top$$

is strictly positive. Consequently, the induced Markov chain is irreducible and aperiodic. In particular, from any initial state, the search process can reach any region of the solution subspace with nonzero probability.

Proof. For any pair of states (i, j), the (i, j)-th entry of $M_{m,\ell}$ is

$$(M_{m,\ell})_{ij} = \alpha_{m,\ell}(T_m)_{ij} + (1 - \alpha_{m,\ell})(h_m)_i. \tag{12}$$

Since $(T_m)_{ij} \geq 0$, $\alpha_{m,\ell} < 1$, and $(h_m)_i > 0$, we have

$$(M_{m,\ell})_{ij} > 0, \qquad \forall i, j.$$

Hence $M_{m,\ell}$ is a strictly positive stochastic matrix.

Any strictly positive stochastic matrix is irreducible and aperiodic. Therefore, the Markov chain induced by $M_{m,\ell}$ admits a unique stationary distribution with strictly positive support on all states. Equivalently, for any two states in V_m, there exists a path of nonzero probability connecting them under the dynamics of **DECENTMEM**. Thus, the search process cannot be permanently trapped in a local absorbing region and remains globally reachable over the entire solution subspace $\mathcal{G}_m$. □

Remark 1 (Boundary Cases)**.** *The condition $\alpha_{m,\ell} \in (0, 1)$ is essential.*

Pure exploitation *($\alpha_{m,\ell} = 1$). In this case, the transition reduces to*

$$M_{m,\ell} = T_m.$$

If T_m contains a closed communicating class corresponding to a locally consistent but suboptimal region, then the chain is reducible. Once the process enters that region, the escape probability is zero, and global reachability is lost.

Pure exploration *($\alpha_{m,\ell} = 0$). In this case, the transition reduces to*

$$M_{m,\ell} = h_m \mathbf{1}^\top, \qquad p_{m,\ell+1} = h_m.$$

The chain remains ergodic, but the dynamics become memoryless: the next state no longer depends on the current state or the local geometry encoded by T_m. Hence the process loses the ability to accumulate and refine historical experience.

These two boundary cases clarify why **DECENTMEM** *requires an adaptive mixture of the two operators: E-pool alone may get trapped locally, whereas X-pool alone discards structured reuse. Their combination yields both global reachability and meaningful self-evolution.*

A.2 Superior Search Efficiency

We now establish the search efficiency of **DECENTMEM**. We show that its online routing mechanism achieves logarithmic cumulative regret, which is order-optimal, and asymptotically outperforms any fixed routing policy.

Theorem 4 (Logarithmic Regret)**.** *Under Assumption 1, the routing policy of* **DECENTMEM** *satisfies*

$$\mathbb{E}[R(T)] = O(\log T).$$

Hence, **DECENTMEM** *achieves $O(\log T)$ cumulative regret.*

Proof. We first analyze the online router introduced in Section 4.3. For notational simplicity, we suppress the agent index m when no ambiguity arises. Recall that

$$\alpha_\ell = \frac{w_\ell}{w_\ell + 1}, \tag{13}$$

where

$$w_\ell := w_{m,\text{E-pool},\ell}, \qquad w_{m,\text{X-pool},\ell} = 1.$$

Thus, $\alpha_\ell \in [0.5, 1)$ is the probability of selecting the E-pool when solving the ℓ-th local subproblem.

At local step ℓ, the router samples

$$I_\ell \in \{0, 1\},$$

where $I_\ell = 1$ denotes selecting the E-pool and $I_\ell = 0$ denotes selecting the X-pool. The routing probabilities are

$$\mathbb{P}(I_\ell = 1 \mid \mathcal{F}_\ell) = \alpha_\ell, \qquad \mathbb{P}(I_\ell = 0 \mid \mathcal{F}_\ell) = 1 - \alpha_\ell,$$

where $\mathcal{F}_\ell$ denotes the filtration generated by the routing history up to step ℓ. After the routing decision, the system observes a binary reward

$$R_\ell \in \{0, 1\},$$

indicating whether the stage-wise score improves.

Let $r_{\mathrm{E}}(\alpha)$ and $r_{\mathrm{X}}(\alpha)$ denote the expected rewards of the E-pool and X-pool under routing probability α, respectively. The induced expected reward is

$$r(\alpha) = \alpha\, r_{\mathrm{E}}(\alpha) + (1-\alpha)\, r_{\mathrm{X}}(\alpha), \qquad \alpha \in [0.5, 1). \tag{14}$$

Under Assumption 1, r is strictly concave and admits a unique maximizer

$$\alpha^\star \in (0.5, 1).$$

One-step recursion. Ignoring the projection $\max(1, \cdot)$ for asymptotic analysis, the update rule in Section 4.3 becomes

$$w_{\ell+1} = \begin{cases} w_\ell + \frac{1}{2}, & \text{if } (I_\ell = 1, R_\ell = 1) \text{ or } (I_\ell = 0, R_\ell = 0), \\ \frac{1}{2} w_\ell, & \text{if } (I_\ell = 1, R_\ell = 0) \text{ or } (I_\ell = 0, R_\ell = 1). \end{cases} \tag{15}$$

Using (13), we obtain

$$\alpha_{\ell+1} = \begin{cases} \phi_+(\alpha_\ell) := \dfrac{1+\alpha_\ell}{3-\alpha_\ell}, & \text{if } (I_\ell = 1, R_\ell = 1) \text{ or } (I_\ell = 0, R_\ell = 0), \\ \phi_-(\alpha_\ell) := \dfrac{\alpha_\ell}{2-\alpha_\ell}, & \text{if } (I_\ell = 1, R_\ell = 0) \text{ or } (I_\ell = 0, R_\ell = 1). \end{cases} \tag{16}$$

Define

$$q(\alpha) := \alpha\, r_{\mathrm{E}}(\alpha) + (1-\alpha)\big(1 - r_{\mathrm{X}}(\alpha)\big), \tag{17}$$

which is the probability of taking the first branch in (16). Then

$$\mathbb{E}[\alpha_{\ell+1} - \alpha_\ell \mid \mathcal{F}_\ell] = g(\alpha_\ell), \tag{18}$$

where

$$g(\alpha) = q(\alpha)\left(\frac{1+\alpha}{3-\alpha} - \alpha\right) + \big(1 - q(\alpha)\big)\left(\frac{\alpha}{2-\alpha} - \alpha\right). \tag{19}$$

Since r is strictly concave with unique maximizer $\alpha^\star$, the induced mean dynamics are locally contractive around $\alpha^\star$. In particular, there exists $\lambda > 0$ such that

$$g(\alpha)(\alpha - \alpha^\star) \le -\lambda(\alpha - \alpha^\star)^2 \tag{20}$$

for all α in a neighborhood of $\alpha^\star$.

Stochastic approximation. After standard time rescaling, (16) can be written in Robbins–Monro form:

$$\alpha_{\ell+1} = \alpha_\ell + \frac{1}{\ell} g(\alpha_\ell) + \frac{1}{\ell} \xi_{\ell+1}, \tag{21}$$

where ℓ denotes the update scaling parameter, and $\{\xi_\ell\}$ is a martingale difference sequence satisfying

$$\mathbb{E}[\xi_{\ell+1} \mid \mathcal{F}_\ell] = 0, \qquad \sup_\ell \mathbb{E}\big[\|\xi_{\ell+1}\|^2 \mid \mathcal{F}_\ell\big] < \infty \quad \text{a.s.}$$

By standard stochastic approximation results [38, Theorem 2.2],

$$\mathbb{E}\big[(\alpha_\ell - \alpha^\star)^2\big] = O(1/\ell). \tag{22}$$

By strong concavity of r, there exists $\mu > 0$ such that

$$r(\alpha^\star) - r(\alpha) \le \mu(\alpha - \alpha^\star)^2, \qquad \forall \alpha \in [0.5, 1). \tag{23}$$

Therefore,

$$\mathbb{E}[R(T)] \le \mu \sum_{\ell=1}^{T} \mathbb{E}\big[(\alpha_\ell - \alpha^\star)^2\big]. \tag{24}$$

Substituting (22) into (24) yields

$$\mathbb{E}[R(T)] \le \mu \sum_{\ell=1}^{T} O(1/\ell) = O(\log T).$$

□

The cumulative regret over T local decisions is

$$R(T) = \sum_{\ell=1}^{T} \Big(r(\alpha^\star) - \mathbb{E}[r(\alpha_\ell)] \Big). \tag{25}$$

Since stochastic bandit lower bounds imply that $\Omega(\log T)$ regret is unavoidable in general [14], an $O(\log T)$ upper bound is order-optimal.

Corollary 1 (Fixed Routing Is Suboptimal)**.** *For any fixed routing probability $\bar{\alpha} \neq \alpha^\star$, the cumulative regret is linear:*

$$R_{\text{fixed}}(T) = \Theta(T).$$

Consequently, **DECENTMEM** *is asymptotically superior to any fixed routing policy.*

Proof. Let $\bar{\alpha} \neq \alpha^\star$ be fixed. Since r is strictly concave and attains its unique maximum at $\alpha^\star$, the per-step gap

$$\delta := r(\alpha^\star) - r(\bar{\alpha})$$

is a strictly positive constant. Therefore,

$$R_{\text{fixed}}(T) = \sum_{\ell=1}^{T} \delta = \delta T = \Theta(T).$$

By Theorem 4, **DECENTMEM** satisfies

$$\mathbb{E}[R(T)] = O(\log T).$$

Hence,

$$\lim_{T \to \infty} \frac{\mathbb{E}[R(T)]}{R_{\text{fixed}}(T)} = 0,$$

which implies the claim. □

Remark 2. *In particular, the balanced fixed routing choice $\bar{\alpha} = 0.5$, which corresponds to assigning equal selection probability to the two memory pools at every step, is a special case of a fixed policy and is therefore asymptotically dominated by* **DECENTMEM**.

B Experiment Details

B.1 Baseline Setup

In this section, we provide detailed descriptions of the centralized memory baselines used in our comparisons.

MetaGPT. This memory design is derived from MetaGPT [4] and focuses exclusively on within-trial memory, i.e., information stored and shared internally during the resolution of a single task by multiple agents.

ChatDev. This memory design is adapted from ChatDev [25], which incorporates both within-trial and cross-trial memory. The within-trial memory is passed from the central or initiating agent at the beginning of each round to guide subsequent interactions based on prior context. The cross-trial memory is relatively simple, storing past solutions to previous queries for future retrieval.

G-Memory. This memory design is directly adopted from G-Memory [5], which manages long multi-agent interaction histories through a three-tier graph hierarchy consisting of insight, query, and interaction graphs.

B.2 Multi-agent system setup

In this section, we detail the setups of our three adopted MAS framework, AgentNet, DyLAN and AutoGen.

AgentNet. AgentNet [13] is a decentralized multi-agent framework characterized by unstructured and opportunistic coordination, where collaboration is not governed by a fixed workflow or a central controller, but instead emerges dynamically from local agent decisions. Specifically, each agent is able to decide whether to forward a task, decompose it into subtasks, or execute it locally based on

the current task state and its own capability. It then performs the assigned subtask and produces task-specific outputs. This flexible structure enables scalable collaboration without relying on predefined interaction patterns. In our implementation of AgentNet, three agents participate in a three-stage collaboration process.

DyLAN. DyLAN [16] is a debate-style framework similar to LLM-Debate, but it incorporates a more efficient agent-wise early-stopping mechanism during multi-turn interactions. DyLAN further employs an agent selection algorithm based on an unsupervised metric, namely the Agent Importance Score, to identify the most contributive agents through a preliminary trial tailored to the target task. In our implementation of DyLAN, three agents participate in the debate, while an additional ranker agent evaluates their relative importance.

AutoGen. AutoGen [15] is a widely used multi-agent orchestration framework for coordinating interactions among specialized agents in problem-solving tasks. Specifically, we adopt its A3: Decision Making structure, which consists of: (1) a Solver Agent, responsible for generating solutions and initialized with the system prompt "You are a smart agent designed to solve problems."; (2) a Ground Truth Agent, which critically evaluates the solver's output and identifies potential errors against a reference standard; and (3) an Executor Agent, which translates validated solutions into executable commands. This modular design enables transparent, verifiable, and actionable multi-agent collaboration.

C Additional Experiment Result

C.1 Case Study

In this section, we use G-Memory as a representative centralized memory baseline on the AutoGen framework to explain why centralized memory incurs higher token cost. The main overhead comes from three stages: memory retrieval, memory reading, and memory update.

In centralized memory, all agents access a shared memory repository. When a new task arrives, each active agent retrieves relevant memory from the global memory space, which contains memory pieces accumulated by the entire multi-agent system. Therefore, even if only a few memory pieces are finally selected, the retrieval process still operates over a large and heterogeneous repository containing experiences from different agents and roles. In contrast, DECENTMEM decentralizes memory at the agent level. Each agent maintains its own private dual-memory pools, consisting of an exploitation pool and an exploration pool. During memory retrieval, an agent only accesses its own local memory space, which is much smaller and more role-specific. This reduces irrelevant retrieval and avoids repeatedly exposing agents to memory pieces generated by other agents.

The same difference also appears in memory reading and update. In centralized memory, agents may repeatedly read overlapping global memories, and after task completion the system needs to summarize and update the shared repository. In DECENTMEM, each agent reads only its personalized memory and updates only its own dual-memory pools. Thus, no system-wide memory synchronization or global memory rewriting is required.

Overall, the high token cost of centralized memory mainly comes from its global access pattern: agents retrieve, read, and update memory against the whole shared repository. By localizing these operations to agent-private dual-memory pools, DECENTMEM reduces token consumption while preserving role-specific experience.

Case Study (DecentMem)

Task

It is not always easy to see which chemicals are contained in our consumer products. The following argument pertains to this question. First, vitamin A is an ingredient of LIQUID EYELINER. Second, every ingredient of BC Eye Cream 15 ml is not an ingredient of Mango Lip Butter or not an ingredient of LIQUID EYELINER. Therefore, it is not the case that vitamin A is an ingredient of BC Eye Cream 15 ml.

Is the argument, given the explicitly stated premises, deductively valid or invalid?

[Task Manager]
Now I have task received: determine whether the chemical-ingredient argument is deductively valid or invalid.

[Online router | Dual Memory Pool Access]
Memory pattern: DecentMem

Query:
"BBH deductive validity; natural-language logic; ingredient relation; disjunction; valid/invalid option format"

Memory Search Process

```
Accessed pool:
agent_id = 0
stage_pool = agent_0_stage_pool["1"]
E_pool_weight = 1.50
X_pool_weight = 1.00

Online router: E-pool
Candidate local memories:
AP-f1-old-09 | sim=0.74 | prior task: "check whether conclusion follows only from stated premises"
AP-f1-old-12 | sim=0.46 | prior task: "multiple-choice answer must be one of the given options"
AP-f1-new-03 | sim=0.59 | recent note: "avoid importing external product knowledge"

Selected compact memory packet for Solver Agent 0:
```

Memory Reading Process

```
MemoryFragment(
  fragment_id = "agent_0_L1_mem_171642_old_49",
  stage = "1",
  pool_type = "old",
  similarity_to_current_task = 0.74,

  environment =
      "problem": "Determine whether the argument is deductively valid or invalid.",
      "task_type": "bbh/chemical_logical_deduction",
      "answer_space": ["valid", "invalid"]
      ,

  rule_definition = RuleDefinition(
    rule_name = "C1",
    description = "Initial-stage agent for checking whether the conclusion follows from the stated
    ↪  premises."
  ),

  action = ActionRecord(
    action_type = "direct_answer",
    direct_answer = "invalid",
    thought_summary = "Use only stated premises. If a counterexample exists, answer invalid."
  ),

  task_allocation = [],
  result_quality = 9.0,
  created = 18
)

prompt_injected_to_solver:
"Past experience: For deductive validity tasks, separate the premises from the conclusion,
use only the stated premises, and answer invalid if a counterexample is possible."
```

[#1] Solver Agent 0
I receive the task plus the compact local memory packet.

Output:
invalid

.....

Memory Update Process

```
Agent Pool Local Write | After L1
Write a compact stage result to agent_0_stage_pool["1"].E_pool:

memory_type = "deductive_validity_short_trace"
action = "initial_answer"
answer = "invalid"
support = "uses premise"

Agent Pool Weight Update | L1
Update only the local L1 pool. The retrieved strategy receives a higher local weight because it
↪  supported a high-scoring answer. No centralized memory merge is triggered.

Agent Dual Pool Memory Update:
Store compact local traces in the assigned agents' stage-specific pools.
```

Case Study (G-Memory)

Task

It is not always easy to see which chemicals are contained in our consumer products. The following argument pertains to this question. First, vitamin A is an ingredient of LIQUID EYELINER. Second, every ingredient of BC Eye Cream 15 ml is not an ingredient of Mango Lip Butter or not an ingredient of LIQUID EYELINER. Therefore, it is not the case that vitamin A is an ingredient of BC Eye Cream 15 ml.

Is the argument, given the explicitly stated premises, deductively valid or invalid?

[Task Manager]
Now BBH task received: determine whether the chemical-ingredient argument is deductively valid or invalid.

[G-Memory Controller | Centralized Memory Pool Access]
Memory pattern: G-Memory.

Query:
"BBH deductive validity; natural-language logic; ingredient relation; disjunction; valid/invalid option format"

Memory Search Process

```
Initialize task context:
task_id = bbh_logical_deduction_case
framework = AutoGen
stage = [solver, coordinator, executor]
answer_space = {valid, invalid}

Create centralized working state:
current_task_node
stage_message_buffer
retrieved_success_pool
retrieved_failure_pool
retrieved_insight_pool
agent_message_graph

[G-memory Query Builder]
Build a higher query from the whole task:

semantic keywords = "deductive validity, ingredient relation, universal quantifier, disjunction,
↪  countermodel"
structural pattern = "premise P(x); premise forall x, B(x)->not M(x) or not P(x); conclusion not
↪  B(a)"
expected reasoning tool = "counterexample / model construction"

[G-memory Retrieval | Search Multiple Central Pools]
Search spaces:
```

```
successful_trajectory_pool
failed_trajectory_pool
task_insight_pool
evaluator_feedback_pool
agent_message_graph

Retrieved successful memories:
GM-S-017 | sim=0.88 | weight=1.42 | "For validity tasks, construct a countermodel if the
↪  conclusion is not forced."
GM-S-042 | sim=0.84 | weight=1.31 | "Disjunction in a universal premise leaves multiple satisfying
↪  branches."
GM-S-068 | sim=0.80 | weight=1.18 | "Translate natural-language relations into predicates before
↪  deciding."

Retrieved failed memories:
GM-F-011 | sim=0.78 | weight=1.25 | failure: "Mistook A -> (not B or not C) as directly implying
↪  not B."
GM-F-028 | sim=0.73 | weight=1.10 | failure: "Imported external commonsense instead of using
↪  explicitly stated premises."

Retrieved task-level insights:
GM-I-008 | sim=0.88 | "A single satisfying countermodel is enough to prove invalidity."
GM-I-014 | sim=0.77 | "Final answer must be exactly one of the provided options."

[G-memory Maintenance Pass | Before Solver]
The centralized memory manager performs extra work:

1. Normalize retrieved records into a common schema.
2. Deduplicate near-identical memories.
3. Convert failed trajectories into negative constraints.
4. Re-rank memories by similarity, success weight, recency, and stage relevance.
5. Compress the selected records into a solver-specific memory packet.

Memory packet injected to Solver Agent 0:

Positive guidance:
Formalize the premises with predicates.
Try to construct a countermodel.
If one model makes all premises true and the conclusion false, answer invalid.

Negative guidance:
Do not treat "not M(x) or not E(x)" as "not B(x)".
Do not use outside facts about cosmetics or ingredients.
```

Memory Reading Process

```
CentralMemoryRecord(
  memory_id = "GM-S-017",
  pool_type = "successful_trajectory_pool",
  memory_type = "successful_task_trajectory",
  similarity_to_current_task = 0.88,
  retrieval_weight = 1.42,
  source_framework = "AutoGen",
  source_task_family = "BBH deductive validity",

  task_signature = {
    "input_modality": "natural_language_argument",
    "answer_format": "binary valid/invalid",
    "logic_pattern": "universal conditional with disjunction",
    "recommended_tool": "predicate formalization + countermodel construction"
  },

  successful_trajectory = {
    "L1_solver": {
      "role": "solver",
      "action": "formalize premises",
      "intermediate_result": "identify predicates and map the conclusion to a target formula",
      "quality_score": 9.0
    },
    "L2_ground_truth": {
      "role": "validator",
      "action": "construct countermodel",
      "intermediate_result": "find an assignment where all premises hold and the conclusion fails",
      "quality_score": 10.0
    }
  },

  prompt_injection_text =
```

```
  "Use predicate formalization and countermodel search. A single model satisfying the premises
  and falsifying the conclusion is sufficient to prove invalidity."
)

CentralMemoryRecord(
  memory_id = "GM-F-011",
  pool_type = "failed_trajectory_pool",
  memory_type = "failure_case_with_correction",
  similarity_to_current_task = 0.79,
  retrieval_weight = 1.25,

  failed_task_signature = {
    "logic_pattern": "conditional premise containing disjunction",
    "surface_error": "agent treated a weak disjunctive condition as a strong exclusion",
    "observed_wrong_answer": "valid",
    "correct_answer": "invalid"
  },

  prompt_injection_text =
  "Be careful with disjunction. From B(x)->not M(x) or not E(x), E(x) does not imply not B(x),
  because not M(x) may still satisfy the premise."
)

CentralMemoryRecord(
  memory_id = "GM-I-008",
  pool_type = "task_insight_pool",
  memory_type = "reusable_high_level_insight",
  similarity_to_current_task = 0.86,

  insight_statement =
  "Deductive invalidity can be established by one countermodel.",

  insight_scope = [
    "valid/invalid classification",
    "formal logic tasks",
    "natural-language arguments",
    "universal or conditional premises"
  ],

  why_it_matters =
  "The solver does not need to derive the opposite conclusion. It only needs to show that the
  stated premises leave at least one possible world where the conclusion is false.",

  prompt_injection_text =
  "Do not over-prove. If one possible assignment makes the premises true and the conclusion false,
  classify the argument as invalid."
)

CentralMemoryRecord(
  memory_id = "GM-E-024",
  pool_type = "evaluator_feedback_pool",
  memory_type = "evaluator_preference_and_scoring_signal",
  similarity_to_current_task = 0.72,

  evaluator_feedback = {
    "high_score_pattern": "Answer defines predicates, writes the formal premises, and provides a
    ↪  concrete countermodel.",
    "low_score_pattern": "Answer only outputs valid/invalid without explaining whether premises
    ↪  entail the conclusion.",
    "preferred_final_style": "Concise final label after enough validation evidence has been
    ↪  accumulated."
  },

  prompt_injection_text =
  "For the validation stage, include predicate definitions and a concrete countermodel.
  For the final executor stage, compress the result to the output label."
)
```

Memory Reading Process

```
GMemoryMergedPacket(
  target_framework = "AutoGen",
  target_stage = "L1",
  target_role = "solver",

  retrieved_records = [
    "successful": ["GM-S-017"],
    "failed": ["GM-F-011"],
```

```
    "insights": ["GM-I-008"],
    "evaluator_feedback": ["GM-E-024"]
  ],

  centralized_memory_summary = {
    "task_family": "deductive validity",
    "dominant_success_strategy": "predicate formalization + countermodel search",
    "dominant_failure_mode": "over-strengthening disjunctive premises",
    "answer_format_constraint": "must output valid or invalid"
  },

  positive_strategy = [
    "Map ingredient relations to predicates.",
    "Write the first premise as E(VitaminA).",
    "Write the second premise as forall x, B(x) -> not M(x) or not E(x).",
    "Write the conclusion as not B(VitaminA).",
    "Try the assignment E(VitaminA)=true, B(VitaminA)=true, M(VitaminA)=false."
  ],

  negative_constraints = [
    "Do not use external knowledge about actual cosmetic ingredients.",
    "Do not infer not B(VitaminA) merely from E(VitaminA).",
    "Do not collapse a disjunction into a conjunction or a stronger exclusion."
  ],

  validation_checklist = [
    "Premise 1 is true under the candidate model.",
    "Premise 2 is true because not M(VitaminA) is true.",
    "The conclusion not B(VitaminA) is false.",
    "Therefore, a countermodel exists and the argument is invalid."
  ],

  prompt_injected_to_solver =
  "Central memory suggests that this task is a deductive-validity problem.
  Use only the stated premises. Formalize the predicates and search for a countermodel.
  Be especially careful with the disjunction in the second premise:
  E(VitaminA) being true does not force B(VitaminA) to be false, because not M(VitaminA)
  can still satisfy the premise."
)
```

Memory packet injected to Solver Agent 0:

Positive guidance:

Formalize the premises with predicates.

Try to construct a countermodel.

If one model makes all premises true and the conclusion false, answer invalid.

Negative guidance:

Do not treat “not M(x) or not E(x)” as “not B(x)”.

Do not use outside facts about cosmetics or ingredients.

[#1] Solver Agent 0

I receive the task plus the G-memory packet. I reason explicitly over the premises.

Output:

The argument is invalid.

Memory Update Process

```
[G-memory Final Update]
Write the completed trajectory back into centralized memory:
task_signature = "deductive validity with universal disjunctive premise"
task_family = "BBH logical deduction"
framework = "AutoGen"
final_label = "invalid"
final_confidence = high

finalized_task_record = {
  "root_question": "Does the conclusion logically follow from the stated premises?",
  "final_answer": "invalid",
  "successful_strategy": "predicate formalization + countermodel construction",
  "dominant_failure_avoided": "collapsing the disjunction into a stronger exclusion",
```

```
  "key_countermodel": {
    "E(VitaminA)": true,
    "B(VitaminA)": true,
    "M(VitaminA)": false
  },
  "why_invalid": "All premises can be true while the conclusion is false."
}

finalized_stage_trace = [
  {
    "agent_role": "solver",
    "node_id": "GM-current-L1-agent0",
    "main_contribution": "identified that a countermodel may exist",
    "score": 9.0
  },
  {
    "agent_role": "validator",
    "node_id": "GM-current-L2-agent1",
    "main_contribution": "constructed the explicit symbolic countermodel",
    "score": 10.0
  },
  {
    "agent_role": "executor",
    "node_id": "GM-current-L3-agent2",
    "main_contribution": "compressed the validated reasoning into the final label",
    "score": 9.0
  }
]

extracted_reusable_knowledge = {
  "success_rule_1": "Formalize natural-language relations as predicates before deciding validity.",
  "success_rule_2": "For deductive invalidity, one countermodel is sufficient.",
  "success_rule_3": "When a premise contains a disjunction, test whether the alternative disjunct
  ↪  can still satisfy the premise.",
  "failure_rule_1": "Do not infer not B(a) from B(a)->not M(a) or not E(a) merely because E(a) is
  ↪  true."
}

evaluator_scores = {
  "L1": 9.0,
  "L2": 10.0,
  "L3": 9.0,
  "trajectory_average": 9.33
}

Update centralized pools and indexes:
  successful_trajectory_pool += {
    "memory_id": "GM-S-new-021",
    "source_task_signature": "deductive validity with universal disjunctive premise",
    "stored_trace": "solver -> validator -> executor trajectory",
    "stored_countermodel": "E(VitaminA)=true, B(VitaminA)=true, M(VitaminA)=false",
    "stored_label": "invalid"
  }

  failed_trajectory_pool += {
    "memory_id": "GM-F-new-021",
    "stored_failure_pattern": "incorrectly collapsing a disjunctive conditional",
    "stored_negative_constraint": "one false disjunct does not invalidate the whole premise if the
    ↪  other disjunct remains true",
    "link_to_corrected_case": "GM-S-new-021"
  }

  task_insight_pool += {
    "memory_id": "GM-I-new-021",
    "insight": "For premise B(x)->not M(x) or not E(x), E(a) does not imply not B(a), because not
    ↪  M(a) can satisfy the disjunction.",
    "generalization_scope": "natural-language deductive validity tasks with disjunctive premises"
  }

  evaluator_feedback_pool += {
    "memory_id": "GM-E-new-021",
    "preferred_high_score_pattern": "define predicates, write the formal premise, and provide a
    ↪  concrete countermodel",
    "observed_score_trace": "[L1: 9.0, L2: 10.0, L3: 9.0]"
  }

agent_message_graph += {
  "new_nodes": [
    "task_root_node",
    "GM-current-L1-agent0",
    "GM-current-L2-agent1",
```

```
    "GM-current-L3-agent2",
    "countermodel_memory_node",
    "final_label_node"
  ],
  "new_edges": [
    "task_root_node -> GM-current-L1-agent0",
    "GM-current-L1-agent0 -> GM-current-L2-agent1",
    "GM-current-L2-agent1 -> countermodel_memory_node",
    "countermodel_memory_node -> GM-current-L3-agent2",
    "GM-current-L3-agent2 -> final_label_node"
  ]
}

Refresh retrieval weights and summaries:
Increase weight of GM-S-017 because its countermodel strategy directly supported a successful
↪  trajectory.
Increase weight of GM-I-008 because the "one countermodel is sufficient" rule was validated again.
Preserve GM-F-011 as a high-value negative constraint because it matches the core failure mode of
↪  this task.
Regenerate centralized summary so future solver agents see this task family as predicate-first,
↪  countermodel-first, disjunction-sensitive.
```

D Prompt Set

DECENTMEM Prompt

1. Exploration Pool Prompt

When the exploration memory pool is selected, no historical memory fragment is reused. Instead, the agent enters a fresh exploration mode and solves the task through standard workflow prompts, including role definition, approach decision, optional problem decomposition, direct problem solving, and solution integration.

1.1 Role-Definition Prompt — Role selection

```
Given this problem:
{problem_description[:300]}

Select the BEST role from this list to solve this problem:
- Problem Solver: General problem-solving and analysis
- Data Analyst: Analyzing tables, numerical data, patterns
- Reasoning Specialist: Logical deduction and causal reasoning
- Verification Checker: Verifying correctness and validating solutions
- Planning Strategist: Breaking down complex tasks into steps
- Math Solver: Mathematical calculations and equations
- Logic Checker: Checking logical consistency and correctness
- Table Interpreter: Reading and interpreting data tables
- Causal Analyst: Analyzing cause-and-effect relationships
- Navigation Tracker: Tracking positions and movements

Your response MUST be EXACTLY ONE of these role names:
Problem Solver, Data Analyst, Reasoning Specialist, Verification Checker,
Planning Strategist, Math Solver, Logic Checker, Table Interpreter,
Causal Analyst, Navigation Tracker

Only respond with the role name, nothing else.
```

1.2 Approach-Decision Prompt — Routing decision

```
Analyze this problem and decide the BEST approach:

Problem: {problem_desc[:200]}...

{if decomposition_score > 0.4:
  "HINT: This problem shows signs of complexity that would benefit from decomposition"}

Decision criteria for 'problem_decomposition':
- Can be broken into 2-4 clear, independent sub-tasks
```

```
- Requires different types of expertise or analysis
- Has multiple distinct components or phases
- Examples: system design, multi-step processes, building components

Decision criteria for 'direct_answer':
- Can be solved straightforwardly with a single approach
- Doesn't require breaking into sub-parts
- Simple analysis, calculation, or explanation



Choose the approach that will lead to the BEST solution quality.
Respond ONLY with ONE word: problem_decomposition, direct_answer.
```

1.3 Direct-Solving Prompt — Direct answer

```
Solve this problem:
{problem_description}

Provide a clear, direct answer.
```

1.4 Problem-Decomposition Prompt — Sub-task generation

```
You are an expert at breaking down complex problems into manageable sub-tasks.

Original Problem:
{problem_desc}

Please decompose this problem into 2-4 DISTINCT and INTERDEPENDENT sub-problems.

Requirements:
1. Each sub-problem should be focused on a specific aspect or step.
2. Sub-problems should be solvable with different expertise levels.
3. Each must contribute to solving the original problem.
4. Ensure the sub-problems are complementary and cover different angles.

For each sub-problem, provide:
- "id": Sequential ID, such as 1, 2, 3.
- "description": Clear, specific description of the sub-problem.
- "focus": Main focus area, e.g., "Analysis", "Design", "Verification".
- "dependencies": Dependencies on other sub-problems, or an empty list.

Respond ONLY with a valid JSON array like this:
[
  {"id": 1, "description": "...", "focus": "...", "dependencies": []},
  {"id": 2, "description": "...", "focus": "...", "dependencies": [1]}
]
```

2. Exploitation-Pool Prompt with Similarity Matching — Historical memory reuse

When the Exploitation-Pool is selected, the framework does not immediately inject historical memory. Instead, it first applies a similarity-matching mechanism. The current task description is used as the retrieval query, encoded into an embedding, and compared against stored memory fragments. Only fragments whose semantic similarity exceeds a predefined threshold are reused.

2.1 Exploitation-Pool Retrieval Query — Similarity search

```
Use the current task description as the retrieval key and search the  memory pool for semantically
↪  similar historical fragments.

Retrieval query:
{problem_description}

Search scope:
agent_id = {agent_id}
stage_id = {stage_id}
pool_type = "Exploitation-Pool"

Return candidate memory fragments ranked by semantic similarity.
```

2.2 Similarity Filtering Prompt — Threshold selection

```
Given the current task and retrieved memory candidates, decide which historical
memory fragments are relevant enough to reuse.

Current task:
{problem_description}

Candidate memories:
{retrieved_memory_candidates}

Similarity threshold:
{similarity_threshold}

Selection criteria:
- Keep fragments whose semantic similarity exceeds the threshold.
- Prefer fragments that match the task type, reasoning pattern, and expected output format.
- Discard fragments that only share surface-level keywords.
- Discard fragments that may introduce misleading assumptions.

Return ONLY the selected memory fragment IDs as a JSON array:
["memory_id_1", "memory_id_2"]
```

2.3 Exploitation-Pool Prompt

Memory-conditioned solving

```
You are solving a new task with help from relevant historical memory fragments.

Current task:
{problem_description}

Retrieved Exploitation-Pool fragments:
{selected_Exploitation-Pool_fragments}

Use the retrieved memory only as guidance. Do not copy previous answers directly.
Adapt useful reasoning patterns, checks, or constraints to the current task.

Please solve the current task and provide a clear final answer.
```

3. Evaluation Prompt

Stage-level quality scoring

The evaluation framework scores the quality of execution at the stage level. It considers both the integrated solutions and the raw direct LLM answers, and returns structured feedback for subsequent memory updates.

3.1 Evaluation Prompt

Evaluator instruction

```
You are an expert evaluator. Evaluate the overall quality of work done in this stage.

PROBLEM:
{original_problem}

STAGE: {stage} - {stage_name}
- Number of tasks: {num_nodes}
- Task node paths: {node_paths}
- Agents involved: {agents_involved}
- Action types distribution: {action_types_count}

SOLUTIONS PROVIDED (Final integrated solutions):
{solutions_summary}

DIRECT LLM ANSWERS (Raw LLM responses before processing):
{direct_llm_answers_summary}
```

3.2 Stage-Specific Criteria for t_1

Initial stage

```
1. Problem Understanding:
Did the agent properly understand the problem?

2. Decomposition Quality:
If decomposed, is the breakdown logical and complete?

3. Solution Clarity:
Are the solutions clear and well-structured?
```

```
4. LLM Direct Answer Quality:
Is the LLM's direct response accurate and helpful?

5. Foundation:
Did this stage provide good foundation for next stages?
```

3.3 Stage-Specific Criteria for t_2 — Intermediate stage

```
1. Processing Quality:
How well were intermediate tasks solved?

2. Building on Previous:
Did agents effectively use guidance from stage t_1?

3. Task Allocation:
Were tasks appropriately allocated to capable agents?

4. Coherence:
Do the solutions form a coherent middle layer?

5. LLM Answer Consistency:
Do the LLM direct answers align with the integrated solutions?
```

3.4 Stage-Specific Criteria for t_3 — Final stage

```
1. Refinement Quality:
How well were solutions refined?

2. Integration:
How well do the final solutions integrate all previous work?

3. Completeness:
Is the final solution complete and comprehensive?

4. Excellence:
Does the final work meet high quality standards?

5. LLM Answer Quality:
Are the LLM direct answers comprehensive and accurate?
```

3.5 Expected Evaluator Output — Structured feedback

```
{
  "score": <0-10>,
  "stage_quality": "<poor/fair/good/excellent>",
  "reasoning": "<detailed explanation>",
  "solution_quality": "<assessment of the integrated solutions>",
  "llm_answer_quality": "<assessment of the LLM direct answers>",
  "strengths": "<what went well>",
  "weaknesses": "<what could be improved>",
  "agent_coordination": "<how well agents worked together>"
}
```