

LatentMem: Customizing Latent Memory for Multi-Agent Systems

Muxin Fu^{1,2}, Xiangyuan Xue³, Yafu Li³, Zefeng He⁴,
Siyuan Huang⁵, Xiaoye Qu^{2‡}, Yu Cheng^{6‡}, Yang Yang^{5‡}

¹Tongji University ²Shanghai AI Laboratory

³The Chinese University of Hong Kong ⁴Nanjing University

⁵Shanghai Jiao Tong University ⁶Nanyang Technological University

Email: boonkana10@gmail.com [‡]Corresponding authors

Abstract

Large language model (LLM)-powered multi-agent systems (MAS) demonstrate remarkable collective intelligence, with multi-agent memory serving as a pivotal mechanism for continual adaptation. However, existing multi-agent memory designs remain constrained by two fundamental bottlenecks: (i) *memory homogenization* caused by limited role-aware customization, and (ii) *information overload* induced by exposing agents to excessively fine-grained textual memories. To address these limitations, we propose **LatentMem**, a learnable multi-agent memory framework that customizes agent-specific experiential memories while reducing textual context. Specifically, **LatentMem** comprises an *experience bank* for lightweight storage of raw interaction trajectories and a *memory composer* that synthesizes compact latent memories conditioned on retrieved experience and agent-specific contexts. Further, we introduce *Latent Memory Policy Optimization* (LMPO), which propagates task-level optimization signals through latent memories to learn compact, high-utility latent representations. Extensive experiments across six benchmarks and four mainstream MAS frameworks show that **LatentMem** improves vanilla settings by up to 19.36% and consistently outperforms existing memory architectures, without requiring modifications to the agent-layer workflows. Our code is available at <https://github.com/KANABOON1/LatentMem>.

1 Introduction

Large Language Model (LLM)-powered multi-agent systems (MAS) have emerged as a powerful framework for solving complex tasks through collaboration (Ye et al., 2025; Zhang et al., 2025d; Yue et al., 2025) or competition (Zhang et al., 2026c; Yang et al., 2025). A key driver of their adaptability is multi-agent memory (Wu et al., 2025b; Hu et al., 2025), which allows agents to retain and reuse interaction experience for more coherent

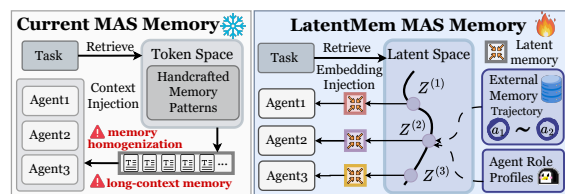


Figure 1: The paradigm comparison between existing multi-agent memory and **LatentMem**. Instead of relying on handcrafted memory units, **LatentMem** extracts agent-specific memories from the latent space by combining raw trajectories with agent profiles.

coordination. Recent memory systems therefore store experiences at multiple granularities, including raw MAS trajectories (Qian et al., 2025; Wang and Chen, 2025), distilled semantic insights (Liu et al., 2025; Zhu et al., 2025), and orchestrable skill schemas (Han et al., 2025; Zhang et al., 2025c).

Despite this progress, existing memory systems remain constrained by two coupled limitations, as illustrated in Figure 1. (i) *Memory homogenization* arises because most methods apply one-size-fits-all memories to functionally heterogeneous agents, weakening role adherence and amplifying correlated errors (Cemri et al., 2026). (ii) *Information overload* arises because MAS already involve long interaction contexts (Zhang et al., 2025d), while multi-granularity repositories further expose agents to large volumes of stored entries (Wang and Chen, 2025; Zhang et al., 2025a). Given the aforementioned challenges, a natural question arises:

Given long and complex contexts in MAS, can we design a learnable memory that is both role-aware and token-efficient, without extensive manual engineering?

To address these challenges, we propose **LatentMem**, a latent multi-agent memory framework that customizes compact memories for individual agents. **LatentMem** maintains a lightweight **experience bank** of raw MAS trajectories and uses

a **memory composer** to transform retrieved experience, conditioned on each agent profile, into role-aware latent memories that guide decision making without adding long textual contexts. We further introduce **Latent Memory Policy Optimization (LMPO)**, which optimizes the memory composer rather than the agent backbones: it computes rollout-level advantages, applies token-level policy objectives to the memory-augmented output distribution, and backpropagates through the differentiable latent memory interface.

As a novel attempt in latent MAS memory, **LatentMem** offers three advantages. **(I)** It conditions memory generation on agent role profiles, mitigating memory homogenization. **(II)** It compresses retrieved trajectories into compact latent memories, reducing information overload. **(III)** It uses task-level rewards to learn memory generation directly, obviating the need for meticulously engineered memory architectures.

Extensive experiments across six benchmarks and four mainstream MAS frameworks demonstrate that **LatentMem** achieves: **(I) high performance**, improving state-of-the-art MAS by up to 16.20% and 18.45% in knowledge QA and code generation tasks, respectively; **(II) high efficiency**, using 50% fewer tokens and reducing inference time to $\sim 2/3$ compared to mainstream memory designs; and **(III) strong generalization**, with out-of-domain datasets such as PDDL showing a 7.10% improvement, and unseen MAS such as CAMEL exhibiting a 7.90% gain compared to the vanilla setting. These results establish **LatentMem** as a novel and effective framework for MAS memory.

2 Related Works

Memory in Multi-Agent Systems. Task-solving MAS memory largely operates in language space. Existing systems retain raw trajectories or shared interaction pools (Hong et al., 2024; Chen et al., 2024; Yin et al., 2023; Qian et al., 2024; Reza-zadeh et al., 2025), distill them into semantic summaries (Zhu et al., 2025; Wu et al., 2025a; Tang et al., 2025), or organize procedural memories for reuse (Wang and Chen, 2025). While these designs improve continual adaptation (Xu et al., 2026; Hu et al., 2025), they expose agents to long textual memories and rarely customize memory by role. **LatentMem** instead composes compact latent memories conditioned on role profiles, addressing information overload and memory homogenization.

Latent Memory and Reasoning. Recent work

uses latent representations to compress reasoning, retrieved documents, or experience. SoftCoT (Xu et al., 2025a) replaces explicit reasoning tokens with latent states, while PISCO (Louis et al., 2025) and OSCAR (Louis et al., 2026) compress retrieved documents into continuous representations for efficient RAG. MemGen (Zhang et al., 2026a) learns generative latent memory for self-evolving agents, and LatentSeek (Li et al., 2025) supports test-time latent deliberation. In MAS settings, LatentMAS (Zou et al., 2025) investigates latent communication among agents. These methods focus on single-agent computation, document compression, or transient communication rather than persistent memory from interleaved heterogeneous MAS trajectories. **LatentMem** instead composes retrieval-conditioned, role-aware latent memories from historical trajectories and optimizes them using task-level multi-agent outcomes; Section A.2 further summarizes the differences.

3 Preliminary

Notations. Consider a multi-agent system $\mathcal{X}$ containing N agents $\mathcal{A} = \{a_1, a_2, \dots, a_N\}$, and equipped with a global memory module $\mathcal{M}$ that stores and retrieves shared information among agents. Formally, the system can be represented as the tuple:

$$\mathcal{X} = (\mathcal{A}, \mathcal{G}, \mathcal{M}) \quad (1)$$

Each agent $a_k = (\gamma_k, \pi_{\theta_k})$ is defined by a role profile γ_k and a policy π_{θ_k} parameterized by θ_k . During execution, an agent receives an input prompt p and a retrieved memory m from $\mathcal{M}$, and produces a response o , denoted as $o = a_k(p, m)$. The execution graph $\mathcal{G}$ governs the topological order in which agents operate. Depending on the system architecture, $\mathcal{G}$ can be instantiated as either a static predefined topology (Qian et al., 2025) or a centralized dynamic regulation mechanism (Wang et al., 2025b).

Problem Formulation. Our objective is to find the memory module $\mathcal{M}$ that maximizes the expected performance of MAS $\mathcal{X}$, which is formally defined as:

$$\max_{\mathcal{M}} \mathbb{E}_{q \sim \mathcal{D}, \tau \sim \mathcal{X}(q)} [R(\tau)] \quad (2)$$

where $\mathcal{D}$ denotes the dataset and q is a query sampled from it. The system $\mathcal{X}$ processes the query q to produce a reasoning trajectory τ , and the reward function R extracts the final answer from τ and evaluates its correctness.

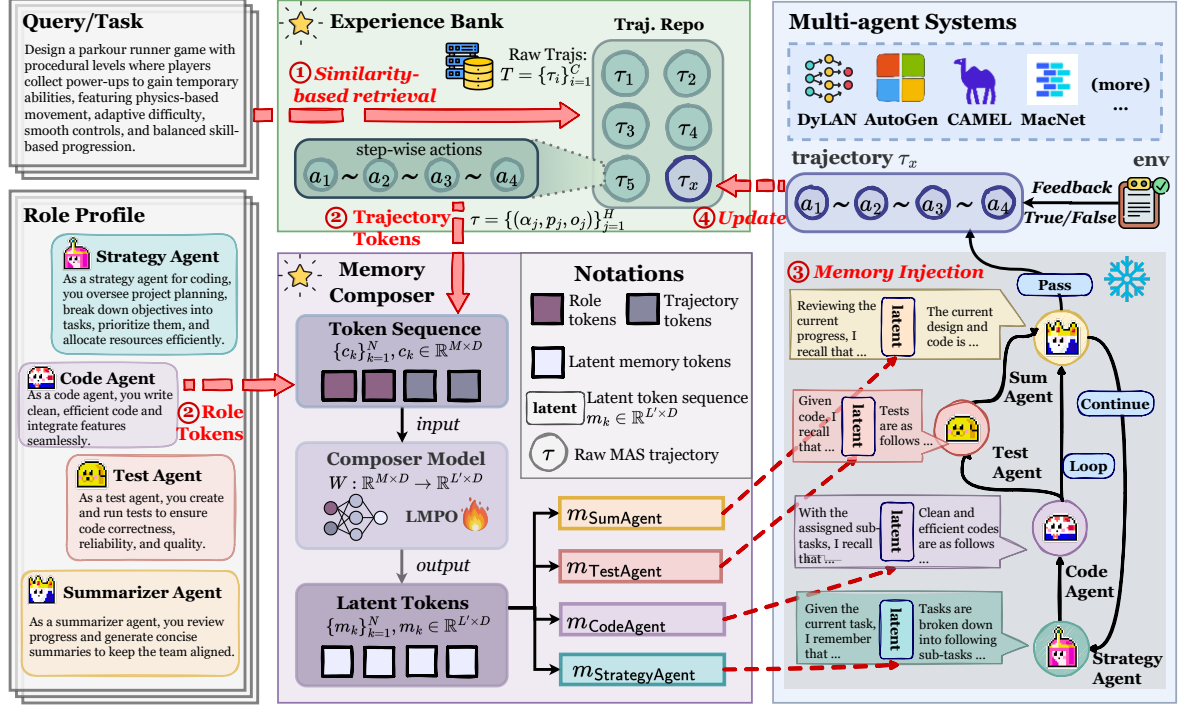


Figure 2: Overview of LatentMem. The framework proceeds as follows: (1) retrieve relevant trajectories from the experience bank; (2) compress them with agent role profiles into latent memories via the LMPO-trained memory composer; (3) inject these memories into agent reasoning processes without altering the agent architectures; and (4) store new trajectories for continual improvement.

This formulation is agnostic to memory architectures, ranging from handcrafted symbolic systems to learnable, parametric counterparts. Conventional memory systems rely on predefined patterns to accumulate experiences, while our approach adopts a learnable memory module that generates compact, role-aware latent representations for dynamic integration into each agent’s reasoning.

4 Methodology

In this section, we first present the pipeline of the LatentMem framework (▷ Section 4.1). Then we detail each module within the framework, including the experience bank (▷ Section 4.2) and the memory composer (▷ Section 4.3). Finally, we introduce Latent Memory Policy Optimization (LMPO), which enables end-to-end optimization of the memory composer through task-level feedback via a reinforcement learning objective (▷ Section 4.4).

4.1 Overall Pipeline

As illustrated in Figure 2, LatentMem consists of a lightweight experience bank $\mathcal{B}$ and a learnable memory composer $\mathcal{C}$. Given a query, LatentMem retrieves relevant historical trajectories from $\mathcal{B}$, uses $\mathcal{C}$ to distill them with each agent’s role profile into compact latent memories, and injects these

memories as additional latent tokens during agent reasoning. After task completion, the new trajectory is appended to $\mathcal{B}$, enabling continual experience accumulation.

This procedure forms a self-improving loop, allowing LatentMem to continuously refine agent reasoning, support long-horizon coordination, and enhance continual adaptation. Moreover, the injection of latent memories maintains end-to-end differentiability of the entire forward process, facilitating efficient RL-based post-training (Qu et al., 2025) without incurring the heavy computation of retraining foundation models.

4.2 Experience Bank

To accurately record historical MAS trajectories for reuse, we construct an extremely lightweight experience bank $\mathcal{B}$. In line with the principle that scalable systems should rely on general learning mechanisms rather than hand-crafted knowledge (Sutton, 2019), this bank stores and retrieves only raw trajectories, without introducing any human priors such as trajectory condensation (Wang and Chen, 2025) or insight extraction (Zhao et al., 2024).

Initialization. We initialize the experience bank with trajectories collected from the training data, with the concrete data sources and collec-

tion protocol described in Section 5.1. The resulting initialized bank is denoted as $\{\tau_i\}_{i=1}^C$, where C specifies its initial capacity. Each trajectory $\tau = \{(\alpha_j, p_j, o_j)\}_{j=1}^H$ records, at each step, the index of the active agent α_j along with its input prompt p_j and corresponding output o_j , where H denotes the trajectory horizon.

Retrieval. Upon receiving a new user query q , **LatentMem** performs similarity-based retrieval over $\mathcal{B}$ to obtain top- K relevant trajectories $\mathcal{T}_q$:

$$\mathcal{T}_q = \text{top-}K_{\tau_i \in \mathcal{B}}(\text{sim}(\mathbf{v}(q), \mathbf{v}(\tau_i))) = \{\tau_i\}_{i=1}^K, \quad (3)$$

where $\mathbf{v}(\cdot)$ maps queries or trajectories into a latent embedding space, e.g., using MiniLM (Wang et al., 2020), and $\text{sim}(\cdot, \cdot)$ denotes the cosine similarity. The retrieved trajectories will be subsequently processed by the memory composer, which distills them into latent memories to guide subsequent MAS reasoning tasks.

Update. Once a task is completed, the new trajectory τ_{new} is appended to the experience bank $\mathcal{B}$ for future reuse:

$$\mathcal{B} \leftarrow \mathcal{B} \cup \{\tau_{\text{new}}\}. \quad (4)$$

This streamlined update mechanism allows **LatentMem** to incrementally accumulate experiences online during inference, facilitating continual adaptation and cross-task coordination without the need for retraining.

Unlike single-agent latent memory methods such as MemGen (Zhang et al., 2026a), which evolve memory around one agent’s experience, the retrieved MAS trajectories contain interleaved decisions, role-dependent contributions, and cross-agent error-correction patterns. Directly feeding these trajectories to agents is suboptimal: it can overwhelm LLMs with excessive context (Cemri et al., 2026) and fails to distill reusable coordination signals (Subramaniam et al., 2025). We therefore introduce the memory composer $\mathcal{C}$, which transforms shared raw trajectories into compact, high-level, role-aware latent memories.

4.3 Memory Composer

After identifying the relevant raw trajectories $\mathcal{T}_q$, we introduce the memory composer $\mathcal{C}$, which provides each agent with generalizable memories. Formally, $\mathcal{C}$ is instantiated as a deep neural network σ_ϕ parameterized by ϕ . At each reasoning step j , σ_ϕ takes as input the retrieved trajectories $\mathcal{T}_q$ and the

role profile γ_{α_j} of the active agent a_{α_j} , producing a fixed-length, agent-aware latent memory matrix:

$$m_j = \sigma_\phi(\gamma_{\alpha_j}, \mathcal{T}_q) \in \mathbb{R}^{L' \times D}, \quad (5)$$

where L' is the latent memory length and D is the hidden dimension of the foundation model. We implement this mapping using a Transformer-based Composer initialized from the agent backbone. Specifically, given the retrieved trajectories and role profile, we take the final L' hidden states from its last Transformer layer as the latent memory m_j for integration into the active agent’s hidden-state sequence.

To conduct reasoning, the active agent a_{α_j} first encodes its input prompt p_j into a sequence of hidden state vectors $h_j = (h_j^{(1)}, \dots, h_j^{(L)}) \in \mathbb{R}^{L \times D}$. The agent’s latent memory m_j is then concatenated to h_j to form an extended input shaped $\mathbb{R}^{(L+L') \times D}$, resulting in a memory-augmented policy:

$$\tilde{\pi}_{\theta_{\alpha_j}}(p_j, m_j) = \pi_{\theta_{\alpha_j}}(\text{concat}(h_j, m_j)), \quad (6)$$

where $\tilde{\pi}_{\theta_{\alpha_j}}$ is a wrapped version of $\pi_{\theta_{\alpha_j}}$ that integrates memory injection at the model level, remaining transparent to the agent layer and requiring no system architecture modifications.

4.4 Latent Memory Policy Optimization

To enable end-to-end optimization while preserving generalization, we propose Latent Memory Policy Optimization (LMPO), a GRPO-based objective (Shao et al., 2024) that trains the memory composer through MAS rollouts while keeping agent backbones frozen. Unlike isolated single-agent memory evolution, LMPO uses task-level rewards to shape memories according to cross-agent coordination outcomes. In implementation, we further use Sequential Agent Gradient Accumulation (SAGA) for memory-efficient multi-agent optimization, as detailed in Appendix B.4.3.

Parametric Dependency. We first describe the gradient flow during LMPO, in which the learning signal propagates through the latent memories to optimize the memory composer $\mathcal{C}$, while keeping the agent backbones $\{\theta_k\}_{k=1}^N$ frozen. Formally, given a query q and the retrieved trajectories $\mathcal{T}_q$ from the experience bank $\mathcal{B}$, the generation of a new trajectory $\tau = \{(\alpha_j, p_j, o_j)\}_{j=1}^H$ is factorized sequentially as:

$$\mathbb{P}(\tau \mid q, \mathcal{T}_q; \phi, \{\theta_k\}_{k=1}^N) = \prod_{j=1}^H \mathbb{P}(o_j \mid p_j, m_j; \theta_{\alpha_j}). \quad (7)$$

Crucially, the latent memory $m_j = \sigma_\phi(\gamma_{\alpha_j}, \mathcal{T}_q)$, as defined in Equation (5), serves as a differentiable interface through which ϕ influences the autoregressively generated output o_j of the active agent a_{α_j} at reasoning step j :

$$\mathbb{P}(o_j | p_j, m_j; \theta_{\alpha_j}) = \prod_{t=1}^T \tilde{\pi}_{\theta_{\alpha_j}}(o_j^{(t)} | p_j, o_j^{(<t)}, m_j). \quad (8)$$

Since the composite policy $\tilde{\pi}_{\theta_{\alpha_j}}$ is conditioned on m_j , the gradient of any task-level objective can be backpropagated through the agent’s forward pass to refine ϕ . This dependency ensures that the memory composer can be optimized end-to-end to produce high-quality latent memories, forming the basis of our policy optimization strategy.

Policy Optimization. Building on the differentiable path above, LMPO leverages task-level feedback through latent memories to directly optimize memory composer attached to the MAS, encouraging it to distill high-utility, agent-specific memories from retrieved raw trajectories and thereby enhance reasoning quality and overall performance.

Formally, given a query q and retrieved trajectories $\mathcal{T}_q$, we sample a group of G trajectories:

$$\{\hat{\tau}_i\}_{i=1}^G \sim \mathbb{P}(\cdot | q, \mathcal{T}_q; \phi, \{\theta_k\}_{k=1}^N). \quad (9)$$

Each trajectory is evaluated using reward $R(\hat{\tau}_i)$, and its relative quality is captured by the group-based advantage:

$$\hat{A}_i = \frac{R(\hat{\tau}_i) - \text{mean}(\{R(\hat{\tau}_i)\}_{i=1}^G)}{\text{std}(\{R(\hat{\tau}_i)\}_{i=1}^G) + \epsilon}. \quad (10)$$

While standard reinforcement learning (Zhang et al., 2025e) often employs trajectory-level objectives, such approaches treat all sequences equally, causing generated output tokens in longer MAS interactions to contribute disproportionately less to the gradient (Yu et al., 2026). This makes it difficult for the memory composer to capture critical coordination patterns within long-horizon tasks. Instead, we adopt a token-level surrogate objective over the agents’ generated outputs:

$$\mathcal{J}_{\text{LMPO}}(\phi) = \mathbb{E}_{q \sim \mathcal{D}, \{\hat{\tau}_i\}_{i=1}^G \sim \mathbb{P}(\cdot | q, \mathcal{T}_q)} \left[\frac{1}{|\{\hat{\tau}_i\}_{i=1}^G|} \sum_{i,j,t} \mathcal{L}_{i,j,t}(\phi) \right], \quad (11)$$

where $|\{\hat{\tau}_i\}_{i=1}^G|$ is the total number of generated

tokens within the trajectory group and $\mathcal{L}_{i,j,t}(\phi)$ is:

$$\mathcal{L}_{i,j,t}(\phi) = \min \left(r_{i,j,t}(\phi) \hat{A}_i, \text{clip}(r_{i,j,t}(\phi), 1 - \varepsilon, 1 + \varepsilon) \hat{A}_i \right), \quad (12)$$

and the token-level importance sampling ratio

$$r_{i,j,t}(\phi) = \frac{\tilde{\pi}_\theta(o_{i,j}^{(t)} | p_{i,j}, o_{i,j}^{(<t)}, \sigma_\phi(\gamma_{\alpha_{i,j}}, \mathcal{T}_q))}{\tilde{\pi}_\theta(o_{i,j}^{(t)} | p_{i,j}, o_{i,j}^{(<t)}, \sigma_{\phi_{\text{old}}}(\gamma_{\alpha_{i,j}}, \mathcal{T}_q))} \quad (13)$$

measures how the policy of agent $a_{\alpha_{i,j}}$ at reasoning step j and token t is modulated by the memory.

5 Experiments

5.1 Experimental Setup

Datasets and Benchmarks. Our evaluation covers six benchmarks across four domains: (1) **Knowledge-intensive QA:** TriviaQA (Joshi et al., 2017) and PopQA (Mallen et al., 2023); (2) **Coding:** KodCode (Xu et al., 2025b) and Big-CodeBench (Zhuo et al., 2025); (3) **Reasoning QA:** StrategyQA (Geva et al., 2021); and (4) **Symbolic Planning:** PDDL (Silver et al., 2024). See Appendix B.1 for benchmark details.

Baselines. Besides memory-free methods, we select three representative single-agent memory baselines: Voyager (Wang et al., 2023), Generative (Park et al., 2023), and JoyAgent (Liu et al., 2025), as well as four multi-agent memory baselines from mainstream MAS frameworks: MetaGPT (Hong et al., 2024), ChatDev (Qian et al., 2024), OAgents (Zhu et al., 2025), and MAS-specific G-Memory (Zhang et al., 2025a). Details are in Appendix B.2.

MAS and LLM Backbones. Four representative multi-agent frameworks are adopted to integrate with LatentMem and the baselines, including AutoGen (Wu et al., 2024), MacNet (Qian et al., 2025), CAMEL (Li et al., 2023) and DyLAN (Liu et al., 2023). More details on the MAS setups are placed in Appendix B.3. To instantiate these MAS frameworks, we adopt two common LLMs with different sizes, i.e., Qwen/Qwen3-4B-Instruct-2507 and meta-llama/Llama-3.1-8B-Instruct.

Training Configurations. We implement the embedding function $\mathbf{v}(\cdot)$ with all-MiniLM-L6-v2 (Wang et al., 2020). The experience bank is initialized with 40,580

Table 1: Performance comparison across diverse memory frameworks on six benchmarks, using Qwen3-4B-Instruct-2507 as the backbone. The **best** and **second best** results are highlighted. TriviaQA, KodCode, StrategyQA, and PopQA are in-domain benchmarks, while BigCodeBench and PDDL are out-of-domain. AutoGen and MacNet are in-distribution MAS frameworks, whereas CAMEL and DyLAN are unseen frameworks introducing new agent roles and collaboration patterns. Additional details are provided in Appendix B.4.

	Method	TriviaQA	Held-in				Held-out		Average
			KodCode	StrategyQA	PopQA		BigCodeBench	PDDL	
Held-in	AutoGen	No-memory	60.31	68.40	58.25	38.78	79.53	16.39	53.61
		ChatDev	57.34 $\downarrow$ 2.97	68.55 $\uparrow$ 0.15	59.18 $\uparrow$ 0.93	33.24 $\downarrow$ 5.54	79.15 $\downarrow$ 0.38	15.22 $\downarrow$ 1.17	52.11 $\downarrow$ 1.50
		MetaGPT	60.35 $\uparrow$ 0.04	70.05 $\uparrow$ 1.65	60.92 $\uparrow$ 2.67	33.80 $\downarrow$ 4.98	80.40 $\uparrow$ 0.87	11.95 $\downarrow$ 4.44	52.91 $\uparrow$ 0.70
		Generative	59.65 $\downarrow$ 0.66	70.90 $\uparrow$ 2.50	62.66 $\uparrow$ 4.41	40.37 $\uparrow$ 1.59	81.14 $\uparrow$ 1.61	13.94 $\downarrow$ 2.45	54.78 $\uparrow$ 1.17
		Voyager	57.50 $\downarrow$ 2.81	68.95 $\uparrow$ 0.55	60.48 $\uparrow$ 2.23	33.56 $\downarrow$ 5.22	79.40 $\downarrow$ 0.13	13.62 $\downarrow$ 2.77	52.25 $\downarrow$ 1.36
		G-Memory	60.56 $\uparrow$ 0.25	71.40 $\uparrow$ 3.00	63.89 $\uparrow$ 5.64	42.67 $\uparrow$ 3.89	80.96 $\uparrow$ 1.43	17.06 $\uparrow$ 0.67	56.09 $\uparrow$ 2.48
		JoyAgent	59.44 $\downarrow$ 0.87	70.90 $\uparrow$ 2.50	62.33 $\uparrow$ 4.08	41.89 $\uparrow$ 3.11	80.52 $\uparrow$ 0.99	14.26 $\downarrow$ 2.13	54.89 $\uparrow$ 1.28
		OAGents	59.85 $\downarrow$ 0.46	70.80 $\uparrow$ 2.40	62.45 $\uparrow$ 4.20	40.70 $\uparrow$ 1.92	81.32 $\uparrow$ 1.79	16.70 $\uparrow$ 0.31	55.30 $\uparrow$ 1.69
		LatentMem	76.51 $\uparrow$ 16.20	76.80 $\uparrow$ 8.40	65.48 $\uparrow$ 7.23	52.70 $\uparrow$ 13.92	81.49 $\uparrow$ 1.96	23.49 $\uparrow$ 7.10	62.75 $\uparrow$ 9.14
	MacNet	No-memory	53.77	70.40	56.44	24.89	78.15	20.73	50.73
		ChatDev	57.29 $\uparrow$ 3.52	70.50 $\uparrow$ 0.10	60.22 $\uparrow$ 3.78	35.33 $\uparrow$ 10.44	79.17 $\uparrow$ 1.02	15.85 $\downarrow$ 4.88	53.06 $\uparrow$ 2.33
		MetaGPT	61.16 $\uparrow$ 7.39	71.50 $\uparrow$ 1.10	61.14 $\uparrow$ 4.70	33.98 $\uparrow$ 9.09	78.33 $\uparrow$ 0.18	17.81 $\downarrow$ 2.92	53.99 $\uparrow$ 3.26
		Generative	59.89 $\uparrow$ 6.12	71.15 $\uparrow$ 0.75	61.79 $\uparrow$ 5.35	43.39 $\uparrow$ 18.50	79.47 $\uparrow$ 1.32	16.81 $\downarrow$ 3.92	55.42 $\uparrow$ 4.69
		Voyager	58.19 $\uparrow$ 4.42	69.80 $\downarrow$ 0.60	61.35 $\uparrow$ 4.91	35.38 $\uparrow$ 10.49	80.53 $\uparrow$ 2.38	14.75 $\downarrow$ 5.98	53.33 $\uparrow$ 2.60
		G-Memory	62.43 $\uparrow$ 8.66	72.50 $\uparrow$ 2.10	62.20 $\uparrow$ 5.76	43.88 $\uparrow$ 18.99	80.44 $\uparrow$ 2.29	21.82 $\uparrow$ 1.09	57.21 $\uparrow$ 6.48
		JoyAgent	61.33 $\uparrow$ 7.56	70.80 $\uparrow$ 0.40	62.34 $\uparrow$ 5.90	43.22 $\uparrow$ 18.33	79.80 $\uparrow$ 1.65	21.20 $\uparrow$ 0.47	56.45 $\uparrow$ 5.72
		OAGents	60.63 $\uparrow$ 6.86	71.30 $\uparrow$ 0.90	61.79 $\uparrow$ 5.35	41.90 $\uparrow$ 17.01	78.86 $\uparrow$ 0.71	22.83 $\uparrow$ 2.10	56.22 $\uparrow$ 5.49
		LatentMem	65.98 $\uparrow$ 12.21	78.90 $\uparrow$ 8.50	64.46 $\uparrow$ 8.02	44.14 $\uparrow$ 19.25	81.49 $\uparrow$ 3.34	25.13 $\uparrow$ 4.40	60.02 $\uparrow$ 9.29
Held-out	CAMEL	No-memory	56.96	70.70	58.04	32.38	79.14	22.10	53.22
		ChatDev	57.55 $\uparrow$ 0.59	68.20 $\downarrow$ 2.50	59.04 $\uparrow$ 1.00	36.78 $\uparrow$ 4.40	78.12 $\downarrow$ 1.02	18.58 $\downarrow$ 3.52	53.04 $\downarrow$ 0.18
		MetaGPT	59.06 $\uparrow$ 2.10	69.90 $\downarrow$ 0.80	59.38 $\uparrow$ 1.34	37.47 $\uparrow$ 5.09	80.53 $\uparrow$ 1.39	22.55 $\uparrow$ 0.45	54.82 $\uparrow$ 1.60
		Generative	57.63 $\uparrow$ 0.67	70.65 $\downarrow$ 0.05	61.20 $\uparrow$ 3.16	35.32 $\uparrow$ 2.94	80.35 $\uparrow$ 1.21	17.29 $\downarrow$ 4.81	53.74 $\uparrow$ 0.52
		Voyager	56.57 $\downarrow$ 0.39	69.85 $\downarrow$ 0.85	58.27 $\uparrow$ 0.23	36.25 $\uparrow$ 3.87	79.91 $\uparrow$ 0.77	23.65 $\uparrow$ 1.55	54.08 $\uparrow$ 0.86
		G-Memory	59.20 $\uparrow$ 2.24	70.40 $\downarrow$ 0.30	60.12 $\uparrow$ 2.08	38.04 $\uparrow$ 5.66	79.28 $\uparrow$ 0.14	24.56 $\uparrow$ 2.46	55.27 $\uparrow$ 2.05
		JoyAgent	58.10 $\uparrow$ 1.14	70.20 $\downarrow$ 0.50	61.34 $\uparrow$ 3.30	37.50 $\uparrow$ 5.12	79.45 $\uparrow$ 0.31	20.65 $\downarrow$ 1.45	54.54 $\uparrow$ 1.32
		OAGents	58.33 $\uparrow$ 1.37	71.40 $\uparrow$ 0.70	60.09 $\uparrow$ 2.05	31.99 $\downarrow$ 0.39	79.28 $\uparrow$ 0.14	13.89 $\downarrow$ 8.21	52.50 $\downarrow$ 0.72
		LatentMem	68.74 $\uparrow$ 11.78	77.75 $\uparrow$ 7.05	64.20 $\uparrow$ 6.16	47.23 $\uparrow$ 14.85	80.70 $\uparrow$ 1.56	28.12 $\uparrow$ 6.02	61.12 $\uparrow$ 7.90
	DyLAN	No-memory	53.87	69.25	57.44	24.89	79.81	23.81	51.51
		ChatDev	56.28 $\uparrow$ 2.41	68.55 $\downarrow$ 0.70	60.23 $\uparrow$ 2.79	39.27 $\uparrow$ 14.38	79.82 $\uparrow$ 0.01	16.55 $\downarrow$ 7.26	53.45 $\uparrow$ 1.94
		MetaGPT	61.78 $\uparrow$ 7.91	71.50 $\uparrow$ 2.25	62.23 $\uparrow$ 4.79	37.00 $\uparrow$ 12.11	78.25 $\downarrow$ 1.56	19.19 $\downarrow$ 4.62	54.99 $\uparrow$ 3.48
		Generative	60.39 $\uparrow$ 6.52	71.70 $\uparrow$ 2.45	60.92 $\uparrow$ 3.48	43.57 $\uparrow$ 18.68	78.86 $\downarrow$ 0.95	18.58 $\downarrow$ 5.23	55.67 $\uparrow$ 4.16
		Voyager	59.06 $\uparrow$ 5.19	70.05 $\downarrow$ 0.80	61.79 $\uparrow$ 4.35	40.40 $\uparrow$ 15.51	80.00 $\uparrow$ 0.19	26.37 $\uparrow$ 2.56	56.28 $\uparrow$ 4.77
		G-Memory	61.23 $\uparrow$ 7.36	71.35 $\uparrow$ 2.10	63.76 $\uparrow$ 6.32	42.28 $\uparrow$ 17.39	80.35 $\uparrow$ 0.54	17.94 $\downarrow$ 5.87	56.15 $\uparrow$ 4.64
		JoyAgent	59.96 $\uparrow$ 6.09	71.35 $\uparrow$ 2.10	62.12 $\uparrow$ 4.68	41.67 $\uparrow$ 16.78	79.89 $\uparrow$ 0.08	25.81 $\uparrow$ 2.00	56.80 $\uparrow$ 5.29
		OAGents	60.98 $\uparrow$ 7.11	71.15 $\uparrow$ 1.90	61.35 $\uparrow$ 3.91	42.36 $\uparrow$ 17.47	78.77 $\downarrow$ 1.04	19.75 $\downarrow$ 4.06	55.73 $\uparrow$ 4.21
		LatentMem	65.55 $\uparrow$ 11.68	78.80 $\uparrow$ 9.55	65.22 $\uparrow$ 7.78	44.25 $\uparrow$ 19.36	81.40 $\uparrow$ 1.59	29.08 $\uparrow$ 5.27	60.72 $\uparrow$ 9.21

trajectories collected by running AutoGen and MacNet on the training splits of TriviaQA, KodCode, StrategyQA, and PopQA. The memory composer $\mathcal{C}$ is realized as a lightweight transformer initialized from the backbone LLM. During LMPO, all agent backbones remain frozen, and training the memory composer takes approximately 15 hours on two NVIDIA H200 GPUs. We set $K = 1$ for retrieval and fix the latent memory length to $L' = 8$, with ablations and full training configurations reported in Section 5.6 and Appendix B.4.

5.2 Main Results

LatentMem Delivers High-Performance Memory Across Domains and MAS Frameworks. As shown in Table 1, when integrated with in-domain MAS frameworks such as AutoGen and MacNet, LatentMem outperforms the state-of-the-art single-

and multi-agent memory baselines by an average of 7.86% and 6.66%, respectively. Across all four MAS frameworks, it consistently achieves the highest average performance over the six benchmarks. Notably, it yields a 16.20% improvement for AutoGen on the TriviaQA benchmark. Furthermore, LatentMem scales effectively to larger backbone models. As shown in Appendix Table 7, it elevates MacNet’s performance on KodCode from 48.50% to 65.50% using Llama-3.1-8B-Instruct.

LatentMem Exhibits Strong Generalization Capability. As shown in Table 1, most memory baselines struggle under distribution shifts across both task domains and MAS frameworks: on PDDL, MetaGPT and Voyager reduce AutoGen performance by up to 4.44% and 2.77%, respectively, while on unseen CAMEL, nearly all baselines decline on KodCode. In contrast,

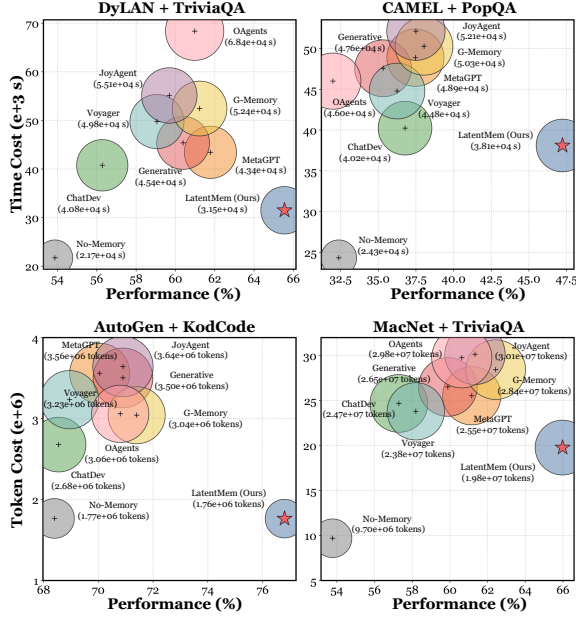


Figure 3: Time and token consumption of **LatentMem**. Each panel shows the trade-off between performance and resource cost under different memory architectures: the top row plots performance versus time, the bottom row plots performance versus token cost. Circle area reflects relative resource consumption.

Table 2: Performance comparison between the multi-agent fine-tuning method MARTI and **LatentMem** on KodCode and TriviaQA across AutoGen and MacNet.

MAS	Method	KodCode	TriviaQA
AutoGen	MARTI	74.20	64.78
	LatentMem	76.80 $\uparrow$ 2.60	76.51 $\uparrow$ 11.73
MacNet	MARTI	73.10	62.31
	LatentMem	78.90 $\uparrow$ 5.80	65.98 $\uparrow$ 3.67

LatentMem improves AutoGen on PDDL by 7.10% and CAMEL on KodCode by 7.05%. This generalization is consistent across unseen MAS frameworks: compared with the no-memory setting, **LatentMem** improves average performance by 7.90% on CAMEL and 9.21% on DyLAN, despite their new agent roles and collaboration patterns. We attribute this gap to its learnable, role-aware memory design, which avoids rigid homogeneous memories and supports adaptation across domains, agent roles, and collaboration patterns.

5.3 Cost Analysis

As shown in Figure 3, **LatentMem** achieves the best performance-cost trade-off among memory-based baselines, with inference cost measured after both retrieval and latent injection. It yields the largest gains on TriviaQA with DyLAN (+11.68%) and KodCode with AutoGen (+8.40%), while

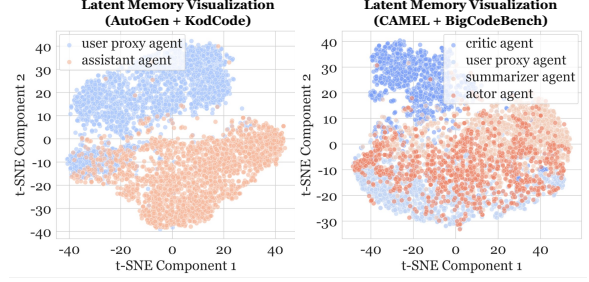


Figure 4: t-SNE visualization of latent memories generated by **LatentMem** across different datasets and MAS frameworks.

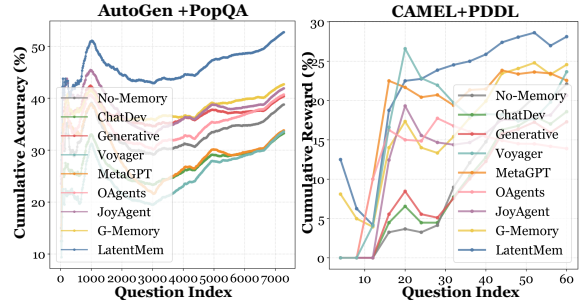


Figure 5: Evolution of cumulative accuracy across question indices. Accuracy at index i is averaged over the first i questions; early fluctuations decrease as more samples accumulate.

substantially reducing time overhead (e.g., $2.16\times$ faster than OAgents) and using even fewer tokens than No-Memory on KodCode.

5.4 Comparison with Multi-Agent Fine-Tuning

To assess memory optimization against direct backbone tuning, we compare **LatentMem** with MARTI (Zhang et al., 2026b), a representative multi-agent fine-tuning baseline trained with GRPO under the same budget and datasets (TriviaQA, KodCode, StrategyQA, and PopQA), with all agents sharing one LLM backbone.

As shown in Table 2, **LatentMem** outperforms direct agent fine-tuning. On AutoGen-TriviaQA, it achieves an 11.73% gain; on MacNet-KodCode, MARTI drops by 1.10% relative to AutoGen, whereas **LatentMem** improves by 2.10%. This suggests that memory-composer optimization better exploits complex MAS structures than direct backbone tuning.

5.5 Framework Analysis

LatentMem Consistently Delivers Role-Aware Memory. As shown in Figure 4, **LatentMem** generates role-specific latent memories across both seen and unseen datasets and MAS frameworks.

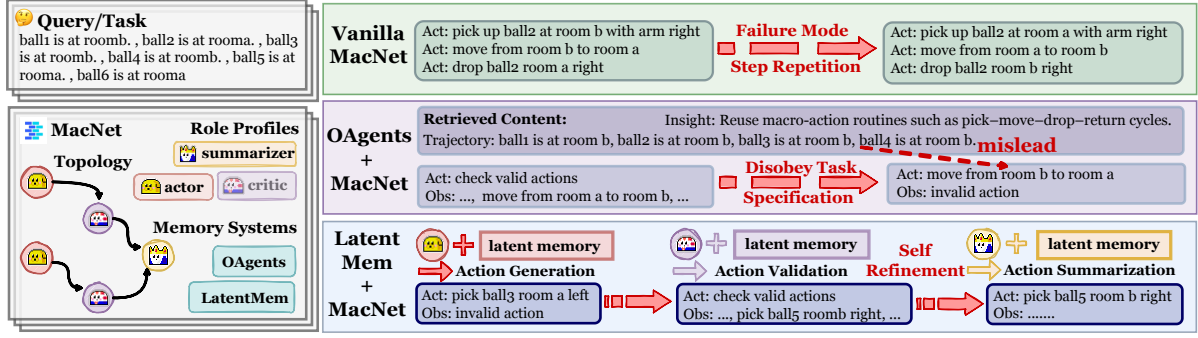


Figure 6: Case study of **LatentMem**. By leveraging role-aware and compact latent memory, **LatentMem** helps correct common MAS failure modes such as step repetition and blindly following retrieved trajectories in this case, while enabling role-aware coordination and self-correction.

In the in-domain KodCode setting with AutoGen, user-proxy and assistant memories form clearly separated clusters; notably, this separation persists on out-of-domain BigCodeBench with unseen CAMEL. This indicates that **LatentMem** avoids homogeneous memory and preserves role-specific representations under novel domains, roles, and collaboration patterns.

LatentMem Scales Efficiently as Experience Accumulates. As shown in Figure 5, **LatentMem** steadily improves as more experiences are collected, surpassing baselines with complex multi-granularity memories. Although early estimates fluctuate due to limited samples, the curve quickly stabilizes and continues to rise, showing that **LatentMem** can distill transferable knowledge from past trajectories and reuse it to support MAS.

LatentMem Benefits from Outcome-Level Optimization and Remains Robust to Retriever Choice. As shown in Appendix Table 5, LMPO consistently outperforms SFT across all evaluated tasks, with a particularly large gain on held-out PDDL. Appendix Table 6 further shows that replacing MiniLM with Qwen3-Embedding-8B (Zhang et al., 2025f) yields only modest improvements, while **LatentMem** remains superior to Generative under both retrievers. Together, these results indicate that the gains primarily arise from outcome-optimized memory composition rather than retriever capacity alone.

5.6 Sensitivity & Ablation Study.

Sensitivity Analysis. We analyze the sensitivity of **LatentMem** to two key hyperparameters: the latent memory length L' and the number of relevant trajectories K . As shown in Figure 7 (Left), performance generally improves with larger L' , but with diminishing returns; balancing accuracy and

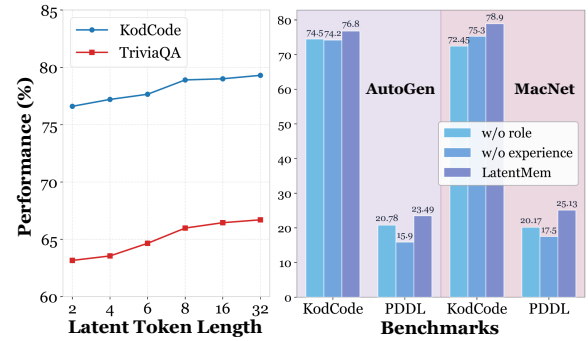


Figure 7: (Left) Sensitivity of model performance to the latent memory length L' . (Right) Ablation results highlighting the impact of the memory composer and the experience bank.

computational cost, we set $L' = 8$. The effect of K is detailed in Appendix C.3: while baselines such as G-Memory (see Appendix B.2 for more details) degrade when $K > 3$ due to *information overload*, **LatentMem** consistently improves on both TriviaQA and KodCode, demonstrating its ability to distill useful information from redundant trajectories via latent memory.

Component Ablation. As shown in Figure 7 (Right), we ablate two core mechanisms: role-aware customization and trajectory reuse. The *without role* variant removes agent profile guidance (γ in Equation (5)), forcing agents to share the same latent memory; the *without experience* variant disables experience bank updates (Equation (4)), limiting trajectory reuse. Removing role information causes a modest drop for AutoGen (2.30% on KodCode) but a larger one for MacNet (6.45%), while disabling experience reuse degrades KodCode performance by 3.60% on MacNet and PDDL by 7.63%. These results confirm the importance of personalized memory customization and historical experience reuse in **LatentMem**.

5.7 Case Study

Figure 6 shows that **LatentMem**, by providing role-aware memory, can help correct common error patterns in MAS. Vanilla MacNet often suffers from step repetition (Cemri et al., 2026), while MacNet with OAgents blindly follows the retrieved trajectories, violating task specifications (Cemri et al., 2026). In contrast, **LatentMem**’s high-level, role-aware latent memory enables agents to reinforce role compliance and coordinate effectively in this case, allowing the MAS to self-correct short-term errors and complete the task successfully. Detailed trajectories and error analyses are in Appendix C.5.

6 Conclusion

In this work, we present **LatentMem**, a latent memory framework for role-aware and token-efficient memory customization in MAS. With a lightweight experience bank and learnable memory composer, **LatentMem** distills raw trajectories into agent-specific latent memories that reinforce role compliance and coordination. We further introduce Latent Memory Policy Optimization to learn transferable, high-utility memories across task domains and MAS frameworks. Experiments on six benchmarks and four MAS frameworks show substantial performance gains, robust generalization, and high efficiency while mitigating memory homogenization and information overload.

Limitations

LatentMem is evaluated primarily in text-based MAS scenarios, including knowledge-intensive QA, coding, reasoning QA, and symbolic planning. This setting does not fully capture real-world multi-agent deployments where agents may need to process visual, audio, or embodied observations. Extending latent memory composition to multi-modal agents, for example by combining LLM agents with vision-language models or captioning modules, is an important direction for making the framework more generally applicable.

LatentMem also assumes access to model representations for latent memory injection, which makes it more suitable for open-weight or otherwise controllable LLM backbones than for purely black-box API agents. Finally, although LMPO provides a practical way to optimize latent memories with task-level feedback, its theoretical properties in long-horizon MAS remain underexplored. A deeper understanding of how latent memories

affect coordination, stability, and policy efficiency is left for future work.

Ethical Considerations

This work studies memory mechanisms for LLM-based multi-agent systems using publicly accessible benchmarks and offline experiments. It does not involve collecting private, personal, or sensitive user information, nor does it conduct human-subject studies or deploy systems to real users. The main ethical concern is dual use: stronger persistent memory and coordination may improve useful agent systems, but could also amplify unsafe behavior if deployed without appropriate safeguards. We therefore frame **LatentMem** as a research prototype and recommend that real-world deployment include privacy-preserving memory policies, filtering of stored experience, safety evaluation, human oversight, and compliance with applicable licenses, laws, and institutional requirements.

Acknowledgments

We thank Guibin Zhang for his patient guidance and valuable advice throughout this work. We also thank the anonymous reviewers for their constructive feedback.

References

- Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, and 1 others. 2026. Why do multi-agent llm systems fail? *Advances in Neural Information Processing Systems*, 38.
- Ma Chang, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. 2024. Agentboard: An analytical evaluation board of multi-turn llm agents. *Advances in neural information processing systems*, 37:74325–74362.
- Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, and 1 others. 2024. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *International Conference on Learning Representations*, volume 2024, pages 20094–20136.
- Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. 2024. Improving factuality and reasoning in language models through multiagent debate. In *Forty-first international conference on machine learning*.

- Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. 2021. Did aristotle use a laptop? a question answering benchmark with implicit reasoning strategies. *Transactions of the Association for Computational Linguistics*, 9:346–361.
- Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V Chawla, Olaf Wiest, and Xiangliang Zhang. 2024. Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint arXiv:2402.01680*.
- Dongge Han, Camille Couturier, Daniel Madrigal Diaz, Xuchao Zhang, Victor Rühle, and Saravan Rajmohan. 2025. Legomem: Modular procedural memory for multi-agent llm systems for workflow automation. *arXiv preprint arXiv:2510.04851*.
- Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Steven Yau, Zijuan Lin, Liyang Zhou, and 1 others. 2024. Metagpt: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations*, volume 2024, pages 23247–23275.
- Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.
- Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, and 1 others. 2025. Memory in the age of ai agents. *arXiv preprint arXiv:2512.13564*.
- Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. 2017. *TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension*. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611, Vancouver, Canada. Association for Computational Linguistics.
- Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023. Camel: Communicative agents for "mind" exploration of large language model society. *Advances in Neural Information Processing Systems*, 36:51991–52008.
- Hengli Li, Chenxi Li, Tong Wu, Xuekai Zhu, Yuxuan Wang, Zhaoxin Yu, Eric Hanchen Jiang, Song-Chun Zhu, Zixia Jia, Ying Nian Wu, and 1 others. 2025. Seek in the dark: Reasoning via test-time instance-level policy gradient in latent space. *arXiv preprint arXiv:2505.13308*.
- Xinyi Li, Sai Wang, Siqi Zeng, Yu Wu, and Yi Yang. 2024. A survey on llm-based multi-agent systems: workflow, infrastructure, and challenges. *Vicnearth*.
- Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Shuming Shi, and Zhaopeng Tu. 2024. Encouraging divergent thinking in large language models through multi-agent debate. In *Proceedings of the 2024 conference on empirical methods in natural language processing*, pages 17889–17904.
- Jiarun Liu, Shiyue Xu, Shangkun Liu, Yang Li, Wen Liu, Min Liu, Xiaoqing Zhou, Hanmin Wang, Shilin Jia, Shaohua Tian, and 1 others. 2025. Joyagent-jdgenie: Technical report on the gaia. *arXiv preprint arXiv:2510.00510*.
- Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. 2023. A dynamic llm-powered agent network for task-oriented agent collaboration. *arXiv preprint arXiv:2310.02170*.
- Maxime Louis, Hervé Déjean, and Stéphane Clinchant. 2025. Pisco: Pretty simple compression for retrieval-augmented generation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 15506–15521.
- Maxime Louis, Thibault Formal, Hervé Déjean, and Stéphane Clinchant. 2026. Oscar: Online soft compression for rag. In *International Conference on Learning Representations*, volume 2026, pages 3102–3121.
- Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605.
- Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9802–9822.
- Joon Sung Park, Joseph O’Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pages 1–22.
- Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, and 1 others. 2024. 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.
- Chen Qian, Zihao Xie, YiFei Wang, Wei Liu, Kunlun Zhu, Hanchen Xia, Yufan Dang, Zhuoyun Du, Weize Chen, Cheng Yang, and 1 others. 2025. Scaling large language model-based multi-agent collaboration. In *International Conference on Learning Representations*, volume 2025, pages 41488–41505.

- Xiaoye Qu, Yafu Li, Zhaochen Su, Weigao Sun, Jianhao Yan, Dongrui Liu, Ganqu Cui, Daizong Liu, Shuxian Liang, Junxian He, and 1 others. 2025. A survey of efficient reasoning for large reasoning models: Language, multimodality, and beyond. *arXiv preprint arXiv:2503.21614*.
- Alireza Rezazadeh, Zichao Li, Ange Lou, Yuying Zhao, Wei Wei, and Yujia Bao. 2025. Collaborative memory: Multi-user memory sharing in llm agents with dynamic access control. *arXiv preprint arXiv:2505.18279*.
- Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.
- Tom Silver, Soham Dan, Kavitha Srinivas, Joshua B Tenenbaum, Leslie Kaelbling, and Michael Katz. 2024. Generalized planning in pddl domains with pretrained large language models. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pages 20256–20264.
- Vighnesh Subramaniam, Yilun Du, Joshua B Tenenbaum, Antonio Torralba, Shuang Li, and Igor Mordatch. 2025. Multiagent finetuning: Self improvement with diverse reasoning chains. In *International Conference on Learning Representations*, volume 2025, pages 10840–10862.
- Richard Sutton. 2019. The bitter lesson. *Incomplete Ideas (blog)*, 13(1):38.
- Xiangru Tang, Tianrui Qin, Tianhao Peng, Ziyang Zhou, Daniel Shao, Tingting Du, Xinming Wei, Peng Xia, Fang Wu, He Zhu, and 1 others. 2025. Agent kb: Leveraging cross-domain experience for agentic problem solving. *arXiv preprint arXiv:2507.06229*.
- Khanh-Tung Tran, Dung Dao, Minh-Duong Nguyen, Quoc-Viet Pham, Barry O’Sullivan, and Hoang D Nguyen. 2025. Multi-agent collaboration mechanisms: A survey of llms. *arXiv preprint arXiv:2501.06322*.
- Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*.
- Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Y Zou. 2025a. Mixture-of-agents enhances large language model capabilities. In *International Conference on Learning Representations*, volume 2025, pages 33944–33963.
- Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in neural information processing systems*, 33:5776–5788.
- Yu Wang and Xi Chen. 2025. Mirix: Multi-agent memory system for llm-based agents. *arXiv preprint arXiv:2507.07957*.
- Zhao Wang, Sota Moriyama, Wei-Yao Wang, Briti Gangopadhyay, and Shingo Takamatsu. 2025b. Talk structurally, act hierarchically: A collaborative framework for llm multi-agent systems. *arXiv preprint arXiv:2502.11098*.
- Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, and 1 others. 2024. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First Conference on Language Modeling*.
- Rong Wu, Xiaoman Wang, Jianbiao Mei, Pinlong Cai, Daocheng Fu, Cheng Yang, Licheng Wen, Xueming Yang, Yufan Shen, Yuxin Wang, and 1 others. 2025a. Evolver: Self-evolving llm agents through an experience-driven lifecycle. *arXiv preprint arXiv:2510.16079*.
- Yaxiong Wu, Sheng Liang, Chen Zhang, Yichao Wang, Yongyue Zhang, Huifeng Guo, Ruiming Tang, and Yong Liu. 2025b. From human memory to ai memory: A survey on memory mechanisms in the era of llms. *arXiv preprint arXiv:2504.15965*.
- Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. 2026. A-mem: Agentic memory for llm agents. *Advances in Neural Information Processing Systems*, 38:17577–17604.
- Yige Xu, Xu Guo, Zhiwei Zeng, and Chunyan Miao. 2025a. Softcot: Soft chain-of-thought for efficient reasoning with llms. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 23336–23351.
- Zhangchen Xu, Yang Liu, Yueqin Yin, Mingyuan Zhou, and Radha Poovendran. 2025b. Kodcode: A diverse, challenging, and verifiable synthetic dataset for coding. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 6980–7008.
- Xiaofang Yang, Lijun Li, Heng Zhou, Tong Zhu, Xiaoye Qu, Yuchen Fan, Qianshan Wei, Rui Ye, Li Kang, Yiran Qin, and 1 others. 2026a. Toward efficient agents: Memory, tool learning, and planning. *arXiv preprint arXiv:2601.14192*.
- Yingxuan Yang, Huacan Chai, Shuai Shao, Yuanyi Song, Siyuan Qi, Renting Rui, and Weinan Zhang. 2026b. Agentnet: Decentralized evolutionary coordination for llm-based multi-agent systems. *Advances in Neural Information Processing Systems*, 38:107309–107336.
- Yongjin Yang, Euiin Yi, Jongwoo Ko, Kimin Lee, Zhijing Jin, and Se-Young Yun. 2025. Revisiting multi-agent debate as test-time scaling: A systematic study of conditional effectiveness. *arXiv preprint arXiv:2505.22960*.

- Rui Ye, Xiangrui Liu, Qimin Wu, Xianghe Pang, Zhenfei Yin, Lei Bai, and Siheng Chen. 2025. X-mas: Towards building multi-agent systems with heterogeneous llms. *arXiv preprint arXiv:2505.16997*.
- Zhangyue Yin, Qiushi Sun, Cheng Chang, Qipeng Guo, Junqi Dai, Xuan-Jing Huang, and Xipeng Qiu. 2023. Exchange-of-thought: Enhancing large language model capabilities through cross-model communication. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 15135–15153.
- Qiyang Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, and 1 others. 2026. Dapo: An open-source llm reinforcement learning system at scale. *Advances in Neural Information Processing Systems*, 38:113222–113244.
- Yanwei Yue, Guibin Zhang, Boyang Liu, Guancheng Wan, Kun Wang, Dawei Cheng, and Yiyan Qi. 2025. Masrouter: Learning to route llms for multi-agent systems. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15549–15572.
- Guibin Zhang, Muxin Fu, Kun Wang, Frank Wan, Miao Yu, and Shuicheng Yan. 2025a. G-memory: Tracing hierarchical memory for multi-agent systems. *Advances in Neural Information Processing Systems*, 38:12988–13018.
- Guibin Zhang, Muxin Fu, and Shuicheng Yan. 2026a. Memgen: Weaving generative latent memory for self-evolving agents. In *International Conference on Learning Representations*, volume 2026, pages 22555–22588.
- Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. 2025b. Multi-agent architecture search via agentic supernet. *arXiv preprint arXiv:2502.04180*.
- Guibin Zhang, Haotian Ren, Chong Zhan, Zhenhong Zhou, Junhao Wang, He Zhu, Wangchunshu Zhou, and Shuicheng Yan. 2025c. Memevolve: Meta-evolution of agent memory systems. *arXiv preprint arXiv:2512.18746*.
- Guibin Zhang, Yanwei Yue, Zhixun Li, Sukwon Yun, Guancheng Wan, Kun Wang, Dawei Cheng, Jeffrey Yu, and Tianlong Chen. 2025d. Cut the crap: An economical communication pipeline for llm-based multi-agent systems. In *International Conference on Learning Representations*, volume 2025, pages 75389–75428.
- Guibin Zhang, Yanwei Yue, Xiangguo Sun, Guancheng Wan, Miao Yu, Junfeng Fang, Kun Wang, Tianlong Chen, and Dawei Cheng. 2024a. G-designer: Architecting multi-agent communication topologies via graph neural networks. *arXiv preprint arXiv:2410.11782*.
- Kaiyan Zhang, Kai Tian, Runze Liu, Sihang Zeng, Xuekai Zhu, Guoli Jia, Yuchen Fan, Xingtai Lv, Yuxin Zuo, Che Jiang, and 1 others. 2026b. Marti: A framework for multi-agent llm systems reinforced training and inference. In *International Conference on Learning Representations*, volume 2026, pages 138790–138813.
- Kaiyan Zhang, Yuxin Zuo, Bingxiang He, Youbang Sun, Runze Liu, Che Jiang, Yuchen Fan, Kai Tian, Guoli Jia, Pengfei Li, and 1 others. 2025e. A survey of reinforcement learning for large reasoning models. *arXiv preprint arXiv:2509.08827*.
- Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, and 1 others. 2025f. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*.
- Yiqun Zhang, Xiaocui Yang, Shi Feng, Daling Wang, Yifei Zhang, and Kaisong Song. 2026c. Can llms beat humans in debating? a dynamic multi-agent framework for competitive debate. In *ICASSP 2026-2026 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 19377–19381. IEEE.
- Yusen Zhang, Ruoxi Sun, Yanfei Chen, Tomas Pfister, Rui Zhang, and Sercan Arik. 2024b. Chain of agents: Large language models collaborating on long-context tasks. *Advances in Neural Information Processing Systems*, 37:132208–132237.
- Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19632–19642.
- Han Zhou, Xingchen Wan, Ruoxi Sun, Hamid Palangi, Shariq Iqbal, Ivan Vulić, Anna Korhonen, and Sercan Arik. 2026. Multi-agent design: Optimizing agents with better prompts and topologies. In *International Conference on Learning Representations*, volume 2026, pages 15844–15872.
- Heng Zhou, Hejia Geng, Xiangyuan Xue, Li Kang, Yiran Qin, Zhiyong Wang, Zhenfei Yin, and Lei Bai. 2025. Reso: A reward-driven self-organizing llm-based multi-agent system for reasoning tasks. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 15990–16009.
- He Zhu, Tianrui Qin, King Zhu, Heyuan Huang, Yeyi Guan, Jinxiang Xia, Hanhao Li, Yi Yao, Ningning Wang, Pai Liu, and 1 others. 2025. Oagents: An empirical study of building effective agents. In *EMNLP (Findings)*, pages 13354–13369.
- Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. 2024. Gptswarm: Language agents as optimizable graphs. In *Forty-first International Conference on Machine Learning*.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widayarsi, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, and 1 others. 2025. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. In *International Conference on Learning Representations*, volume 2025, pages 66602–66656.

Jiaru Zou, Ruizhong Qiu, Gaotang Li, Xiyuan Yang, Katherine Tieu, Pan Lu, Ke Shen, Hanghang Tong, Yejin Choi, Jingrui He, and 1 others. 2025. Latent collaboration in multi-agent systems. *arXiv preprint arXiv:2511.20639*.

A More Related Work

A.1 LLM-Based Multi-Agent Systems

LLM-based MAS assign agents complementary roles and communication patterns for tasks (Guo et al., 2024; Li et al., 2024; Tran et al., 2025). Early frameworks use fixed workflows (Hong et al., 2024; Li et al., 2023; Wu et al., 2024; Du et al., 2024; Liang et al., 2024; Zhang et al., 2024b; Wang et al., 2025a), while recent methods adapt organizations or routing policies to the task (Zhuge et al., 2024; Zhang et al., 2024a; Yang et al., 2026b; Zhou et al., 2025, 2026; Zhang et al., 2025b; Yue et al., 2025; Yang et al., 2026a). These studies mainly optimize the agent workflow; **LatentMem** is orthogonal, targeting the memory layer so existing MAS reuse experience without redesigning their collaboration structure.

A.2 Positioning of Our Work

Table 3 compares our approach with previous works across six dimensions. The leftmost column separates MAS-memory baselines from latent-method baselines.

Category	Method	Latent interface	Persistent memory	Self-evolving	Hetero. MAS	Role aware	Trajectory retrieval
MAS memory	G-Memory (Zhang et al., 2025a)	×	✓	✓	✓	✓	✓
	AgentKB (Tang et al., 2025)	×	✓	✓	×	×	✓
	LegoMem (Han et al., 2025)	×	✓	×	✓	×	×
Latent method	MemGen (Zhang et al., 2026a)	✓	×	✓	×	×	×
	SoftCoT (Xu et al., 2025a)	✓	×	×	×	×	×
	LatentMAS (Zou et al., 2025)	✓	×	×	✓	×	×
Ours	LatentMem	✓	✓	✓	✓	✓	✓

Table 3: Checklist comparison of representative MAS-memory and latent-method baselines. A check mark indicates that the method explicitly targets the property.

We define these dimensions as follows:

- **Latent interface:** the method injects or manipulates continuous hidden states rather than only textual prompts.

- **Persistent memory:** the method maintains reusable information beyond a single reasoning pass.
- **Self-evolving:** memory can be updated from new experience or feedback.
- **Hetero. MAS:** the method explicitly supports functionally heterogeneous multi-agent systems.
- **Role aware:** memory is customized by agent role or profile.
- **Trajectory retrieval:** historical interaction trajectories are retrieved before producing the memory signal.

B Experimental Details

B.1 Dataset Descriptions

In this section, we describe the datasets used in our experiments:

- **TriviaQA** (Joshi et al., 2017) is a large-scale question answering dataset comprising challenging, naturally occurring trivia questions paired with evidence documents, designed to evaluate reading comprehension and information retrieval over long contexts.
- **KodCode** (Xu et al., 2025b) is a fully synthetic coding dataset with diverse tasks, verifiable solutions, and tests, supporting both supervised fine-tuning and reinforcement learning.
- **StrategyQA** (Geva et al., 2021) is a QA benchmark of yes/no questions requiring multi-step reasoning, with annotated reasoning steps and supporting evidence to evaluate implicit reasoning.
- **PopQA** (Mallen et al., 2023) is a large-scale open-domain question answering dataset of entity-centric QA pairs generated from Wikidata knowledge tuples with annotated subject, relation, object, and Wikipedia page view metadata, designed to support factual QA evaluation.
- **BigCodeBench** (Zhuo et al., 2025) is a programming benchmark dataset of diverse Python coding tasks with structured prompts, canonical solutions, and unit tests, designed

to evaluate and compare code generation capabilities of large language models.

- **PDDL** (Silver et al., 2024) is a game dataset from AgentBoard (Chang et al., 2024), comprising a variety of strategic games where agents use PDDL expressions to complete complex tasks.

B.2 Baseline Setup

In this section, we provide detailed descriptions of each baseline used in our comparison:

- **Voyager**: The Voyager memory originates from the Voyager agent (Wang et al., 2023), where an embodied agent continuously explores the Minecraft environment and generates new artifacts, with memory acting as the central driver of its development. Since Voyager’s memory is originally designed for a single-agent setup, we extend it to the multi-agent setting by retrieving agent-specific histories based on each agent’s visible dialogue context. Other single-agent memory architectures are adapted similarly.
- **Generative**: This memory baseline follows Generative agent (Park et al., 2023) and incorporates both raw observational memory and high-level reflective memory. The reflective component encodes abstract insights produced by the agent via reflection, offering a more organized and conceptual representation of its experiences.
- **JoyAgent**: Adapted from JoyAgent (Liu et al., 2025), this hierarchical memory comprises working, semantic, and procedural layers. Working memory maintains the execution context, semantic memory distills task trajectories and lessons for retrieval, and procedural memory encodes dynamic system prompts for strategic control. This structure ensures long-horizon continuity and adaptive behavior in single-agent task execution.
- **MetaGPT**: This memory design is derived from MetaGPT (Hong et al., 2024), incorporating a shared message pool that enables agents to publish and subscribe to structured information throughout the software development lifecycle. This system facilitates inside-trial coordination by allowing agents to retrieve historical context for self-correction,

while also supporting cross-trial improvement through a long-term memory layer that stores feedback to refine agent constraints over time.

- **ChatDev**: This memory design is adapted from ChatDev (Qian et al., 2024), focusing solely on inside-trial memory—information stored internally during the resolution of a single task by multiple agents.
- **OAgents**: Based on OAgents framework (Zhu et al., 2025), this design uses a hierarchical memory system: short-term for real-time tasks and long-term for fused historical insights. It employs memory summarization and vectorized retrieval to manage information density and extract relevant context. This multi-layered approach ensures long-horizon reasoning while minimizing redundancy during complex tasks.
- **G-Memory**: G-Memory (Zhang et al., 2025a) operates through a structured lifecycle of retrieval, traversal, and updating to manage the extensive interaction histories of multi-agent systems. When a new query arrives, the system first performs a coarse-grained similarity search on the Query Graph to identify relevant historical tasks, followed by a bi-directional traversal: an upward traversal to the Insight Graph to extract high-level strategic guidance and a downward traversal to the Interaction Graph to retrieve condensed, fine-grained collaboration trajectories. These filtered memories are then customized for each agent based on their specific roles to provide actionable context without overwhelming the context window. Once the task is executed, G-Memory performs an agentic update where new collaborative logs are stored, successful or failed experiences are distilled into new nodes in the Insight Graph, and semantic edges are reinforced across the three-tier hierarchy to facilitate continuous self-evolution.

B.3 Multi-agent System Setup

In this section, we detail the setups of our four adopted MAS frameworks, AutoGen, MacNet, CAMEL and DyLAN.

B.3.1 AutoGen

AutoGen (Wu et al., 2024) is a widely adopted framework for orchestrating multi-agent collaboration. In this work, we adopt its A2 setting, which

consists of a Retrieval-augmented User Proxy agent and a Retrieval-augmented Assistant agent, both extended from AutoGen’s built-in agents. The User Proxy agent incorporates a vector database retriever to provide relevant external context, while the Assistant agent utilizes the retrieved information to generate accurate responses. This design enables interactive retrieval and supports effective knowledge grounding for both question answering and code generation tasks.

B.3.2 MacNet

MacNet (Qian et al., 2025) is a representative work that explores decentralized and scalable multi-agent systems. Its key feature lies in the absence of a central agent; instead, it introduces edge agents, which are invoked between agent interactions to provide actionable instructions to the next agent based on the previous agent’s outputs. In our implementation, we adopt the random graph topology from MacNet, shown to be robust across diverse scenarios, and employ a total of five agents, including the edge agents.

B.3.3 CAMEL

CAMEL (Li et al., 2023) is a role-playing-based communication agent framework designed to guide agents to autonomously collaborate on complex tasks with minimal human intervention via inception prompting. The framework introduces a task specifier agent that concretizes vague ideas and achieves task objectives through multi-turn, instruction-following dialogues. In our implementation, we employ four agents: three participating in the debate and one dedicated to summarization.

B.3.4 DyLAN

DyLAN (Liu et al., 2023) is a debate-style framework similar to LLM-Debate, but incorporates a more efficient agent-wise early stopping mechanism during multi-turn interactions. DyLAN utilizes an agent selection algorithm based on an unsupervised metric, namely the Agent Importance Score, which identifies the most contributive agents through a preliminary trial tailored to the specific task. In our implementation of DyLAN, four agents engage in the debate, while an additional ranker agent evaluates their relative importance.

B.4 Training Details

B.4.1 Evaluation of Generalization

To evaluate the generalization capabilities of different methods, we consider two dimensions: (1) whether the method can generalize to domains unseen during training, and (2) whether it can generalize to multi-agent systems unseen during training. For the in-domain and out-of-domain evaluation, we use TriviaQA, KodCode, StrategyQA, and PopQA as in-domain datasets, and BigCodeBench and PDDL as out-of-domain datasets. For MAS generalization, AutoGen and MacNet are considered as MAS encountered during training, while CAMEL and DyLAN are treated as MAS not seen during training.

B.4.2 Training Trajectories Collection and Usage

For all baselines and LatentMem, we first collect data on all in-domain datasets using the training splits and in-distribution MAS. Specifically, we gather training data by running AutoGen and MacNet on TriviaQA, KodCode, StrategyQA, and PopQA. In total, we obtain 40,580 trajectories. Training the memory composer on these trajectories takes approximately 15 hours on 2 NVIDIA H200 GPUs, and this is a one-time optimization cost. For non-parametric baselines, such as MetaGPT (Hong et al., 2024) and G-Memory (Zhang et al., 2025a), the training data is processed and stored directly in their memory modules. For parametric baselines, such as MARTI (Zhang et al., 2026b), the training data is used to optimize model parameters. For LatentMem, we first follow the same procedure as non-parametric baselines: the MAS performs standard reasoning on the training data, and the resulting trajectories are stored in the experience bank (without using the memory composer at this stage). We then train the memory composer via LMPO on the same training data, enabling it to effectively transform retrieved trajectories from the experience bank into latent memories that can assist MAS reasoning.

B.4.3 Sequential Agent Gradient Accumulation

We employ a memory-efficient multi-agent optimization strategy called Sequential Agent Gradient Accumulation (SAGA). Unlike jointly computing the gradients of all agents within a single forward-backward pass, which incurs activation

memory overhead that scales linearly with the number of agents, SAGA decomposes each MAS trajectory into agent-specific message batches and sequentially performs per-agent gradient computation through gradient accumulation. Concretely, each accumulation step processes only one agent trajectory, while the accumulated update preserves the same effective optimization objective as joint optimization across agents. This design reduces the peak activation memory complexity with respect to the number of agents from $O(N)$ to $O(1)$, enabling scalable training of large multi-agent systems without changing the underlying optimization target.

B.4.4 Parameter Configurations

Here, we provide full details of the parameter settings used to implement **LatentMem**.

In practice, the memory composer is instantiated as a pretrained LLM (e.g., Qwen3-4B-Instruct-2507 or Llama-3.1-8B-Instruct) and optimized with LoRA adapters (Hu et al., 2021). Prior to training, we collect diverse datasets and raw MAS trajectories on AutoGen and MacNet using the official training splits of in-domain datasets (TriviaQA, KodCode, StrategyQA, PopQA). Training then proceeds following the same protocol applied to all baseline methods. Detailed hyperparameter configurations are reported in Table 4.

Table 4: The detailed parameter settings when implementing **LatentMem**.

Parameter	Setting
Token limit for prompts	10240
Token limit for responses	4096
Training temperature	1.0
Evaluation temperature	0.0
Discount factor	1.0
Clipping epsilon	0.2
Weight of KL penalty	0.0
Number of training epochs	1
Number of prompt reuse	1
Macro training batch size	32
Micro training batch size	8
Macro rollout batch size	32
Micro rollout batch size	8
Optimizer name	AdamW
Learning rate	1e-5
Warmup ratio	0.1
Lora r	16
Lora alpha	32
Lora target modules	[q_proj, v_proj]
Lora dropout	0.1
Lora bias	None
Lora task type	CAUSAL_LM
Gradient norm	1.0
Gradient clipping	True
Gradient checkpoint	True
Flash Attention	True
Mixed precision	True
Enable vLLM	True
Enable DeepSpeed	True

C Extra Results

C.1 Cost Analysis

Figure 8 provides additional comparisons of token and time costs across various benchmarks and MAS frameworks with different memory architectures. Overall, **LatentMem** leverages its latent memory design to achieve the fastest inference and lowest token usage among all memory baselines, while consistently delivering the most substantial improvements.

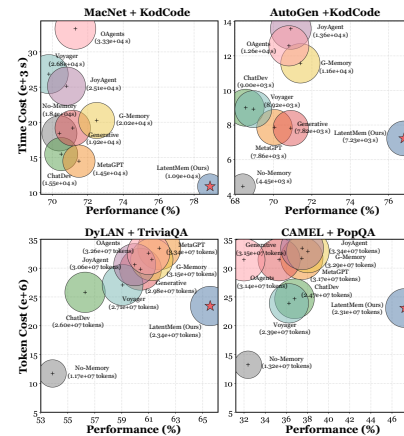


Figure 8: Time and token consumption of LatentMem. The top row reports performance versus time, and the bottom row reports performance versus token cost. Circle area reflects relative resource consumption, and “A + B” denotes MAS framework A evaluated on dataset B.

Method	TriviaQA	KodCode	StrategyQA	PopQA	PDDL
LatentMem (SFT)	60.82	74.55	61.92	45.80	10.34
LatentMem (LMPO)	76.51	76.80	65.48	52.70	23.49

Table 5: Comparison of SFT and LMPO for training the memory composer. TriviaQA, KodCode, StrategyQA, and PopQA are held-in, while PDDL is held-out.

Method	Retriever	TriviaQA	KodCode	PDDL
Generative	MiniLM-L6-v2	59.65	70.90	13.94
Generative	Qwen3-Embedding-8B	61.20	71.45	14.33
LatentMem	MiniLM-L6-v2	76.51	76.80	23.49
LatentMem	Qwen3-Embedding-8B	77.36	77.20	23.81

Table 6: Retriever comparison under AutoGen with Qwen3-4B. All methods use $K = 1$; MiniLM denotes all-MiniLM-L6-v2.

C.2 Role-aware Memory Analysis

This section outlines the approach used to examine the geometric organization of latent memories (illustrated in Figures 4 and 9) and to discover semantically meaningful clusters. The workflow consists of two main steps: first, deriving a fixed-size vector representation for each memory sequence, and second, applying dimensionality reduction and clustering techniques.

Consider $\{m_1, m_2, \dots, m_B\}$ as a collection of B latent memory sequences obtained from **LatentMem**’s inference, where each sequence $m_i = (m_{i,1}, \dots, m_{i,L'}) \in \mathbb{R}^{L' \times D}$ contains L' token embeddings of dimensionality D .

To generate a single, compact representation for each sequence, we compute its mean embedding $\bar{m}_i \in \mathbb{R}^D$ as follows:

$$\bar{m}_i = \frac{1}{L'} \sum_{l=1}^{L'} m_{i,l}. \quad (14)$$

The resulting set of high-dimensional vectors, $\{\bar{m}_1, \bar{m}_2, \dots, \bar{m}_B\}$, is then used as input for downstream analysis.

For visualization purposes, we apply t-SNE (Maaten and Hinton, 2008) to map the vectors $\{\bar{m}_1, \bar{m}_2, \dots, \bar{m}_B\}$ into a two-dimensional space. Formally, this transformation is defined as

$$y_i \in \mathbb{R}^2 = f_{\text{t-SNE}}(\bar{m}_i). \quad (15)$$

These 2D embeddings are subsequently employed to produce the scatter plots.

C.3 Sensitivity Analysis

As a supplement to Section 5.6, we conduct a sensitivity analysis of **LatentMem** with respect to the

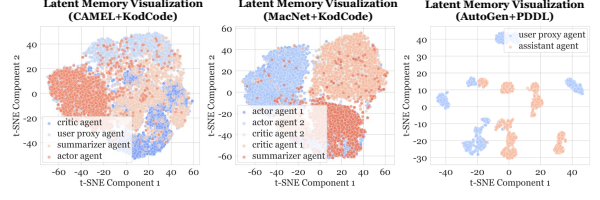


Figure 9: t-SNE visualization of latent memories generated by **LatentMem** across different datasets and MAS frameworks.

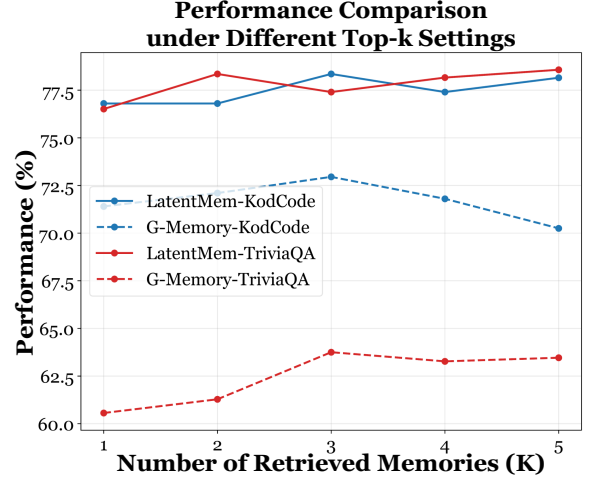


Figure 10: Comparison between **LatentMem** and G-Memory under varying top- K settings on KodCode and TriviaQA with AutoGen. While **LatentMem** consistently benefits from more trajectories as K increases, G-Memory’s performance declines when $K > 3$.

top- K raw trajectories retrieved from the experience bank, as shown in Figure 10. It can be observed that G-Memory initially benefits from increasing K , but its performance starts to decline when $K > 3$; for example, on KodCode, accuracy drops from 72.95% at $K = 3$ to 70.25% at $K = 5$. In contrast, **LatentMem** continues to improve as K increases, highlighting its ability to leverage fixed-length, role-aware latent memory to distill useful information even from redundant trajectories.

C.4 Prefix-Tuning Control

To examine whether the gains of **LatentMem** mainly come from learned prompt-like latent tokens, we compare against a role-conditioned prefix-tuning control under the AutoGen setting with the Qwen3-4B-Instruct-2507 backbone. As shown in Table 8, role-conditioned prefix-tuning improves over a simple static memory interface but remains substantially below **LatentMem**, suggesting that retrieval-conditioned latent memory composition is more effective than learning fixed

Table 7: Performance comparison with diverse memory frameworks on six benchmarks. The underlying LLM backbone is Llama-3.1-8B-Instruct. We highlight the **best** and **second best** results. TriviaQA, KodCode, StrategyQA, and PopQA are in-domain benchmarks, while BigCodeBench and PDDL are out-of-domain; AutoGen and MacNet are in-distribution MAS frameworks, whereas CAMEL and DyLAN are unseen frameworks. Additional details are provided in Appendix B.4.

	Method	Held-in				Held-out		Average	
		TriviaQA	KodCode	StrategyQA	PopQA	BigCodeBench	PDDL		
Held-in	AutoGen	No-memory	72.03 $\uparrow$ 0.00	47.45 $\uparrow$ 0.00	61.50 $\uparrow$ 0.00	41.20 $\uparrow$ 0.00	80.96 $\uparrow$ 0.00	14.03 $\uparrow$ 0.00	52.86 $\uparrow$ 0.00
		MetaGPT	74.10 $\uparrow$ 2.07	50.00 $\uparrow$ 2.55	58.73 $\downarrow$ 2.77	47.41 $\uparrow$ 6.21	82.11 $\uparrow$ 1.15	25.61 $\uparrow$ 11.58	56.33 $\uparrow$ 3.47
		Generative	73.77 $\uparrow$ 1.74	49.40 $\uparrow$ 1.95	63.97 $\uparrow$ 2.47	46.00 $\uparrow$ 4.80	82.11 $\uparrow$ 1.15	20.63 $\uparrow$ 6.60	55.98 $\uparrow$ 3.12
		Voyager	74.53 $\uparrow$ 2.50	46.45 $\downarrow$ 1.00	61.28 $\downarrow$ 0.22	46.75 $\uparrow$ 5.55	81.49 $\uparrow$ 0.53	21.65 $\uparrow$ 7.62	55.36 $\uparrow$ 2.50
		G-Memory	74.60 $\uparrow$ 2.57	50.20 $\uparrow$ 2.75	64.20 $\uparrow$ 2.70	47.24 $\uparrow$ 6.04	82.67 $\uparrow$ 1.71	24.31 $\uparrow$ 10.28	57.20 $\uparrow$ 4.34
		OAgent	73.12 $\uparrow$ 1.09	49.55 $\uparrow$ 2.10	65.94 $\uparrow$ 4.44	47.25 $\uparrow$ 6.05	82.11 $\uparrow$ 1.15	16.99 $\uparrow$ 2.96	55.83 $\uparrow$ 2.97
		LatentMem	74.92 $\uparrow$ 2.89	65.90 $\uparrow$ 18.45	67.89 $\uparrow$ 6.39	49.40 $\uparrow$ 8.20	83.84 $\uparrow$ 2.88	28.96 $\uparrow$ 14.93	61.82 $\uparrow$ 8.96
	MacNet	No-memory	71.62 $\uparrow$ 0.00	48.50 $\uparrow$ 0.00	62.94 $\uparrow$ 0.00	40.59 $\uparrow$ 0.00	79.28 $\uparrow$ 0.00	17.33 $\uparrow$ 0.00	53.38 $\uparrow$ 0.00
		MetaGPT	72.57 $\uparrow$ 0.95	50.30 $\uparrow$ 1.80	62.66 $\downarrow$ 0.28	48.42 $\uparrow$ 7.83	81.28 $\uparrow$ 2.00	21.47 $\uparrow$ 4.14	56.12 $\uparrow$ 2.74
		Generative	72.83 $\uparrow$ 1.21	49.85 $\uparrow$ 1.35	60.80 $\downarrow$ 2.14	47.32 $\uparrow$ 6.73	81.11 $\uparrow$ 1.83	24.80 $\uparrow$ 7.47	56.12 $\uparrow$ 2.74
		Voyager	72.17 $\uparrow$ 0.55	44.80 $\downarrow$ 3.70	61.35 $\downarrow$ 1.59	47.11 $\uparrow$ 6.52	81.67 $\uparrow$ 2.39	18.97 $\uparrow$ 1.64	54.35 $\uparrow$ 0.97
		G-Memory	71.79 $\uparrow$ 0.17	49.20 $\uparrow$ 0.70	62.23 $\downarrow$ 0.71	48.96 $\uparrow$ 8.37	81.46 $\uparrow$ 2.18	19.10 $\uparrow$ 1.77	55.46 $\uparrow$ 2.08
		OAgent	72.20 $\uparrow$ 0.58	48.50 $\uparrow$ 0.00	63.76 $\downarrow$ 0.82	48.30 $\uparrow$ 7.71	82.28 $\uparrow$ 3.00	16.71 $\downarrow$ 0.62	55.29 $\uparrow$ 1.91
		LatentMem	74.45 $\uparrow$ 2.83	65.50 $\uparrow$ 17.00	66.42 $\uparrow$ 3.48	50.16 $\uparrow$ 9.57	83.84 $\uparrow$ 4.56	25.61 $\uparrow$ 8.28	61.00 $\uparrow$ 7.62
Held-out	CAMEL	No-memory	72.86 $\uparrow$ 0.00	48.95 $\uparrow$ 0.00	60.25 $\uparrow$ 0.00	40.44 $\uparrow$ 0.00	80.28 $\uparrow$ 0.00	17.45 $\uparrow$ 0.00	53.37 $\uparrow$ 0.00
		MetaGPT	72.99 $\uparrow$ 0.13	51.70 $\uparrow$ 2.75	62.66 $\uparrow$ 2.41	47.68 $\uparrow$ 7.24	81.93 $\uparrow$ 1.65	23.24 $\uparrow$ 5.79	56.70 $\uparrow$ 3.33
		Generative	72.36 $\downarrow$ 0.50	49.80 $\uparrow$ 0.85	60.26 $\uparrow$ 0.01	49.09 $\uparrow$ 8.65	80.28 $\uparrow$ 0.00	23.97 $\uparrow$ 6.52	55.96 $\uparrow$ 2.59
		Voyager	72.57 $\downarrow$ 0.29	48.95 $\uparrow$ 0.00	61.57 $\uparrow$ 1.32	48.68 $\uparrow$ 8.24	81.75 $\uparrow$ 1.47	15.79 $\downarrow$ 1.66	54.88 $\uparrow$ 1.51
		G-Memory	73.69 $\uparrow$ 0.83	49.50 $\uparrow$ 0.55	61.14 $\uparrow$ 0.89	48.80 $\uparrow$ 8.36	81.93 $\uparrow$ 1.65	20.69 $\uparrow$ 3.24	55.96 $\uparrow$ 2.59
		OAgent	73.52 $\uparrow$ 0.66	49.90 $\uparrow$ 0.95	61.48 $\uparrow$ 1.23	47.90 $\uparrow$ 7.46	81.49 $\uparrow$ 1.21	17.12 $\downarrow$ 0.33	55.23 $\uparrow$ 1.86
		LatentMem	74.00 $\uparrow$ 1.14	63.85 $\uparrow$ 14.90	65.28 $\uparrow$ 5.03	49.90 $\uparrow$ 9.46	82.75 $\uparrow$ 2.47	24.46 $\uparrow$ 7.01	60.04 $\uparrow$ 6.67
	DyLAN	No-memory	72.62 $\uparrow$ 0.00	47.55 $\uparrow$ 0.00	60.67 $\uparrow$ 0.00	40.50 $\uparrow$ 0.00	80.26 $\uparrow$ 0.00	15.20 $\uparrow$ 0.00	52.80 $\uparrow$ 0.00
		MetaGPT	73.87 $\uparrow$ 1.25	49.85 $\uparrow$ 2.30	62.88 $\uparrow$ 2.21	47.92 $\uparrow$ 7.42	81.28 $\uparrow$ 1.02	18.45 $\uparrow$ 3.25	55.71 $\uparrow$ 2.91
		Generative	73.22 $\uparrow$ 0.60	47.10 $\downarrow$ 0.45	64.85 $\uparrow$ 4.18	48.55 $\uparrow$ 8.05	82.19 $\uparrow$ 1.93	17.69 $\uparrow$ 2.49	55.60 $\uparrow$ 2.80
		Voyager	73.69 $\uparrow$ 1.07	44.80 $\downarrow$ 2.75	61.35 $\uparrow$ 0.68	48.88 $\uparrow$ 8.38	81.58 $\uparrow$ 1.32	14.33 $\downarrow$ 0.87	54.10 $\uparrow$ 1.30
		G-Memory	74.00 $\uparrow$ 1.38	48.80 $\uparrow$ 1.25	61.35 $\uparrow$ 0.68	47.32 $\uparrow$ 6.82	82.28 $\uparrow$ 2.02	19.64 $\uparrow$ 4.44	55.56 $\uparrow$ 2.76
		OAgent	73.49 $\uparrow$ 0.87	48.25 $\uparrow$ 0.70	63.25 $\uparrow$ 2.58	47.90 $\uparrow$ 7.40	81.30 $\uparrow$ 1.04	16.75 $\uparrow$ 1.55	55.16 $\uparrow$ 2.36
		LatentMem	74.10 $\uparrow$ 1.48	64.25 $\uparrow$ 16.70	66.83 $\uparrow$ 6.16	49.34 $\uparrow$ 8.84	83.25 $\uparrow$ 2.99	22.62 $\uparrow$ 7.42	60.06 $\uparrow$ 7.26

Table 8: Comparison with a role-conditioned prefix-tuning control on AutoGen.

Method	KodCode	PopQA
Role-conditioned prefix-tuning	70.15	42.78
LatentMem	76.80	52.70

role-conditioned prefixes.

C.5 Case Study

Figure 6 illustrates how **LatentMem**, by providing role-aware memory, can help correct common error patterns in MAS. For example, when solving the PDDL task: “Satisfy the following conditions: ball1 is at room b, ball2 is at room a, ball3 is at room b, ball4 is at room b, ball5 is at room a, ball6 is at room a”, the vanilla MacNet repeatedly moves ball2 between room a and room b while ignoring the task’s termination conditions, resulting in a loop, which is called *Step Repetition* error. MacNet with OAgents, although able to retrieve similar historical trajectories, is influenced by excessively long histories and fails to account for differences between the current task and the re-

trieved trajectories. This leads to blindly following the retrieved steps without satisfying the actual task requirements, which is called *Disobey Task Specification* error. Moreover, both methods suffer from significant *Reasoning-Action Mismatch*, where the agent’s planned reasoning steps are correct, but its executed actions deviate from the intended plan.

In contrast, **LatentMem** provides high-level, role-aware latent memory to each agent at every reasoning step. By leveraging personalized high-level historical experience, each agent naturally reinforces its role compliance, thereby enabling the actor-critic style coordination mechanism in MacNet to function effectively. Even when a short-term error occurs in the trajectory (e.g., generating an incorrect action “pick ball5 rooma right”), the MAS can immediately self-correct in the next reasoning step using a “check valid answer” procedure, ultimately completing the task successfully.

D Prompt Templates

In this section, we provide the prompt templates used in our framework. Note that these templates

may vary slightly across different task domains. As an example, we present the prompt template of CAMEL and AutoGen for code tasks.

CAMEL's Prompt Template for Strategy Agent

You are a strategy-generation agent. Your task is to read a given coding problem and provide a **detailed implementation strategy**, but **do not write any code**.

- # Objectives
- Understand the problem requirements.
 - Describe the algorithm, data structures, and step-by-step approach.
 - Ensure the strategy is clear enough for a developer to implement directly.
- # Output Guidelines
- Focus on logic and process; avoid including actual code or irrelevant explanations.
 - You should keep your response concise, no more than 3 sentences.

CAMEL's Prompt Template for Code Agent

You are a Code Implementation agent. You will be provided with a problem and an analysis of that problem from a user agent. Your task is to produce complete and correct code implementations based on coding problems.

- # Objectives
- Write clear, well-structured, and correct Python code.
 - Do not include any explanations or comments outside the code.
- # Output Guidelines
- Wrap the entire Python code inside a code block using triple backticks:
- ```
```python
# your code here
```
```

### CAMEL's Prompt Template for Test Agent

You are a code evaluator. Your task is to review the current coding problem and the code written by the actor agent for that problem.

- If the code is correct, reply only with: "Agree".
- If the code has issues, give brief and concise feedback only (Keep your response short and within 3 sentences).

### CAMEL's Prompt Template for Summarizer Agent

You are a summarization and final-code-generation agent. Your task is to read the previous actor code implementations and the corresponding critic improvement suggestions, and then produce the final, corrected, and consolidated code solution for the current task.

- # Objectives
- Carefully examine the actor's code solutions.
  - Incorporate the critic's improvement suggestions when necessary.
  - Produce a clean, complete, and correct final code implementation.
  - Do not include explanations, comments, or any text outside the code block.

- # Output Format
- Wrap the entire final Python code inside triple backticks:
- ```
```python
final code here
```
```

AutoGen's Prompt Template for Assistant Agent

You are a strategy-generation agent. Your task is to read a given coding problem and provide a **detailed implementation strategy**, but **do not write any code**.

- # Objectives
- Understand the problem requirements.
 - Describe the algorithm, data structures, and step-by-step approach.
 - Ensure the strategy is clear enough for a developer to implement directly.

- # Output Guidelines
- Focus on logic and process; avoid including actual code or irrelevant explanations.
 - You should keep your response concise, no more than 3 sentences.

AutoGen's Prompt Template for User Proxy Agent

You are a Code Implementation agent. Your task is to read the implementation strategy provided by the Assistant agent and produce **complete, executable Python code** that follows the strategy exactly.

- # Objectives
- Implement the solution according to the detailed strategy from the Assistant.
 - Write clear, well-structured, and correct Python code.
 - Make sure the code covers all steps and handles edge cases mentioned in the strategy.
 - Do not include any explanations or comments outside the code.

- # Output Guidelines
- Wrap the entire Python code inside a code block using triple backticks:
- ```
```python
# your code here
```
```

## E Example

In this section, we present example trajectories generated by LatentMem-enhanced AutoGen on Knowledge QA (TriviaQA) and Game (PDDL) tasks to demonstrate the effectiveness of LatentMem.

### Example trajectory on PDDL

```
<|im_start|>user
Successful Examples (Reference Cases)
Below are some examples of similar tasks that were
successfully completed.
Please use these as references to guide your thinking
and approach to the current task:

<|im_start|>user
<observation>Ball1 is a ball. Ball1 is at rooma. Ball2
is a ball. Ball2 is at rooma. Ball3 is a ball.
Ball3 is at rooma. Left is a gripper. Left is
free. Right is a gripper. Right is free. Robby
is at rooma. Room rooma Room roomb</observation>
><|im_end|>
<|im_start|>assistant
<action>Pick up ball1 at rooma with arm right.</action>
><|im_end|>
<|im_start|>user
<observation>Ball1 is a ball. Ball1 is carrying right.
Ball2 is a ball. Ball2 is at rooma. Ball3 is a
ball. Ball3 is at rooma. Left is a gripper. Left
is free. Right is a gripper. Robby is at rooma.
Room rooma Room roomb</observation><|im_end|>
<|im_start|>assistant
<action>Pick up ball2 at rooma with arm left.</action>
><|im_end|>
<|im_start|>user
<observation>Ball1 is a ball. Ball1 is carrying right.
Ball2 is a ball. Ball2 is carrying left. Ball3
is a ball. Ball3 is at rooma. Left is a gripper.
Right is a gripper. Robby is at rooma. Room
rooma Room roomb</observation><|im_end|>
<|im_start|>assistant
<action>Move from rooma to roomb</action><|im_end|>
<|im_start|>user
<observation>Ball1 is a ball. Ball1 is carrying right.
Ball2 is a ball. Ball2 is carrying left. Ball3
is a ball. Ball3 is at rooma. Left is a gripper.
Right is a gripper. Robby is at roomb. Room
rooma Room roomb</observation><|im_end|>
<|im_start|>assistant
<action>drop ball1 at rooma with arm right</action><|
im_end|>
<|im_start|>user
<observation>The action is not valid and therefore
takes no effect. Please remember to satisfy the
restriction of actions. You can also check valid
actions.</observation><|im_end|>
<|im_start|>assistant
<action>check valid actions</action><|im_end|>
<|im_start|>user
<observation>Valid actions are: drop ball1 roomb right
, drop ball2 roomb left, Move from roomb to
rooma. , check valid actions</observation><|
im_end|>
<|im_start|>assistant
<action>drop ball1 roomb right</action><|im_end|>
<|im_start|>user
<observation>Ball1 is a ball. Ball1 is at roomb. Ball2
is a ball. Ball2 is carrying left. Ball3 is a
ball. Ball3 is at rooma. Left is a gripper.
Right is a gripper. Right is free. Robby is at
roomb. Room rooma Room roomb</observation><|
im_end|>
<|im_start|>assistant
<action>drop ball2 roomb left</action><|im_end|>
<|im_start|>user
<observation>Ball1 is a ball. Ball1 is at roomb. Ball2
is a ball. Ball2 is at roomb. Ball3 is a ball.
Ball3 is at rooma. Left is a gripper. Left is
free. Right is a gripper. Right is free. Robby
```

```
is at roomb. Room rooma Room roomb</observation>
><|im_end|>
<|im_start|>assistant
<action>move from roomb to rooma</action><|im_end|>
<|im_start|>user
<observation>Ball1 is a ball. Ball1 is at roomb. Ball2
is a ball. Ball2 is at roomb. Ball3 is a ball.
Ball3 is at rooma. Left is a gripper. Left is
free. Right is a gripper. Right is free. Robby
is at rooma. Room rooma Room roomb</observation>
><|im_end|>
<|im_start|>assistant
<action>Pick up ball3 at rooma with arm right.</action>
><|im_end|>
<|im_start|>user
<observation>Ball1 is a ball. Ball1 is at roomb. Ball2
is a ball. Ball2 is at roomb. Ball3 is a ball.
Ball3 is carrying right. Left is a gripper. Left
is free. Right is a gripper. Robby is at rooma.
Room rooma Room roomb</observation><|im_end|>
<|im_start|>assistant
<action>Move from rooma to roomb</action><|im_end|>
<|im_start|>user
<observation>Ball1 is a ball. Ball1 is at roomb. Ball2
is a ball. Ball2 is at roomb. Ball3 is a ball.
Ball3 is carrying right. Ball4 is a ball. Ball4
is at rooma. Left is a gripper. Left is free.
Right is a gripper. Robby is at roomb. Room
rooma Room roomb</observation><|im_end|>
<|im_start|>assistant
<action>drop ball3 rooms right</action><|im_end|>
<|im_start|>user
<observation>Ball1 is a ball. Ball1 is at roomb. Ball2
is a ball. Ball2 is at roomb. Ball3 is a ball.
Ball3 is at roomb. Ball4 is a ball. Ball4 is at
rooma. Left is a gripper. Left is free. Right is
a gripper. Right is free. Robby is at roomb.
Room rooma Room roomb. The task is completed.</
observation><|im_end|>

```

Use the above examples and insights as a foundation, and now work on the following task:

You are a robot with a gripper that can move objects between different rooms. Your name is Robby.

```
There are three actions defined in this domain:
- think xxx:
 - Format: `think: I need to move block A first
 because...`
 - Any other format is invalid and will not be
 recognized by the system.
- move <room1> <room2>: This action allows the robot
 to move from one room to another. The action has
 a single precondition, which is that the robot
 is currently in a room. The effect of this
 action is to move the robot to another room and
 to remove the fact that it is in the original
 room.
- pick <obj> <room> <gripper>: This action allows the
 robot to pick up an object using the gripper.
 The action has three preconditions: (1) the
 object is located in a room (2) the robot is
 currently in the same room and (3) the gripper
 is free (i.e., not holding any object). The
 effect of this action is to update the state of
 the world to show that the robot is carrying the
 object using the gripper, the object is no
 longer in the room, and the gripper is no longer
 free.
- drop <obj> <room> <gripper>: This action allows the
 robot to drop an object that it is carrying. The
 action has two preconditions: (1) the robot is
 currently carrying the object using the gripper,
 and (2) the robot is currently in a room. The
 effect of this action is to update the state of
 the world to show that the robot is no longer
 carrying the object using the gripper, the
 object is now located in the room, and the
 gripper is now free.
```

## NOTE:

- You should strictly follow these actions, no any other actions are allowed!

- Before you respond, you must ensure that your output actions strictly comply with the rules described above!

Here is your initial observation: Ball1 is a ball.  
Ball1 is at rooma. Ball2 is a ball. Ball2 is at rooma. Ball3 is a ball. Ball3 is at roomb. Ball4 is a ball. Ball4 is at roomb. Left is a gripper. Left is free. Right is a gripper. Right is free. Robby is at rooma. Room rooma Room roomb

**\*\*Here is your task:** The goal is to satisfy the following conditions: ball1 is at roomb. , ball2 is at roomb. , ball3 is at rooma. , ball4 is at rooma.<|im\_end|>

<|im\_start|>assistant  
<action>pick ball1 rooma right</action><|im\_end|>  
<|im\_start|>user  
Ball1 is a ball. Ball1 is carrying right. Ball2 is a ball. Ball2 is at rooma. Ball3 is a ball. Ball3 is at roomb. Ball4 is a ball. Ball4 is at roomb. Left is a gripper. Left is free. Right is a gripper. Robby is at rooma. Room rooma Room roomb<|im\_end|>  
<|im\_start|>assistant  
<action>move from rooma to roomb</action><|im\_end|>  
<|im\_start|>user  
Ball1 is a ball. Ball1 is carrying right. Ball2 is a ball. Ball2 is at rooma. Ball3 is a ball. Ball3 is at roomb. Ball4 is a ball. Ball4 is at roomb. Left is a gripper. Left is free. Right is a gripper. Robby is at roomb. Room rooma Room roomb<|im\_end|>  
<|im\_start|>assistant  
<action>drop ball1 roomb right</action><|im\_end|>  
<|im\_start|>user  
Ball1 is a ball. Ball1 is at roomb. Ball2 is a ball. Ball2 is at rooma. Ball3 is a ball. Ball3 is at roomb. Ball4 is a ball. Ball4 is at roomb. Left is a gripper. Left is free. Right is a gripper. Right is free. Robby is at roomb. Room rooma Room roomb<|im\_end|>  
<|im\_start|>assistant  
<action>pick ball2 rooma left</action><|im\_end|>  
<|im\_start|>user  
The action is not valid and therefore takes no effect.  
You should use `check valid actions.` command to get some clues!<|im\_end|>  
<|im\_start|>assistant  
<action>check valid actions</action><|im\_end|>  
<|im\_start|>user  
Valid actions are: Pick up ball1 at roomb with arm left. , Pick up ball3 at roomb with arm left. , Move from roomb to rooma. , Pick up ball3 at roomb with arm right. , Pick up ball4 at roomb with arm left. , Pick up ball1 at roomb with arm right. , Pick up ball4 at roomb with arm right. , check valid actions<|im\_end|>  
<|im\_start|>assistant  
<action>pick ball3 roomb left</action><|im\_end|>  
<|im\_start|>user  
Ball1 is a ball. Ball1 is at roomb. Ball2 is a ball. Ball2 is at rooma. Ball3 is a ball. Ball3 is carrying left. Ball4 is a ball. Ball4 is at roomb. Left is a gripper. Right is a gripper. Right is free. Robby is at roomb. Room rooma Room roomb<|im\_end|>  
<|im\_start|>assistant  
<action>move from roomb to rooma</action><|im\_end|>  
<|im\_start|>user  
Ball1 is a ball. Ball1 is at roomb. Ball2 is a ball. Ball2 is at rooma. Ball3 is a ball. Ball3 is carrying left. Ball4 is a ball. Ball4 is at roomb. Left is a gripper. Right is a gripper. Right is free. Robby is at rooma. Room rooma Room roomb<|im\_end|>  
<|im\_start|>assistant  
<action>drop ball3 rooma left</action><|im\_end|>  
<|im\_start|>user  
Ball1 is a ball. Ball1 is at roomb. Ball2 is a ball. Ball2 is at rooma. Ball3 is a ball. Ball3 is at rooma. Ball4 is a ball. Ball4 is at roomb. Left is a gripper. Left is free. Right is a gripper. Right is free. Robby is at rooma. Room rooma Room roomb<|im\_end|>  
<|im\_start|>assistant

<action>pick ball4 roomb right</action><|im\_end|>  
<|im\_start|>user  
The action is not valid and therefore takes no effect.  
You should use `check valid actions.` command to get some clues!<|im\_end|>  
<|im\_start|>assistant  
<action>check valid actions</action><|im\_end|>  
<|im\_start|>user  
Valid actions are: Pick up ball2 at rooma with arm right. , Pick up ball3 at rooma with arm right. , Pick up ball3 at rooma with arm left. , Pick up ball2 at rooma with arm left. , Move from rooma to roomb. , check valid actions<|im\_end|>  
<|im\_start|>assistant  
<action>pick ball2 rooma right</action><|im\_end|>  
<|im\_start|>user  
Ball1 is a ball. Ball1 is at roomb. Ball2 is a ball. Ball2 is carrying right. Ball3 is a ball. Ball3 is at rooma. Ball4 is a ball. Ball4 is at roomb. Left is a gripper. Left is free. Right is a gripper. Robby is at rooma. Room rooma Room roomb<|im\_end|>  
<|im\_start|>assistant  
<action>move from rooma to roomb</action><|im\_end|>  
<|im\_start|>user  
Ball1 is a ball. Ball1 is at roomb. Ball2 is a ball. Ball2 is carrying right. Ball3 is a ball. Ball3 is at rooma. Ball4 is a ball. Ball4 is at roomb. Left is a gripper. Left is free. Right is a gripper. Robby is at roomb. Room rooma Room roomb<|im\_end|>  
<|im\_start|>assistant  
<action>drop ball2 roomb right</action><|im\_end|>  
<|im\_start|>user  
Ball1 is a ball. Ball1 is at roomb. Ball2 is a ball. Ball2 is at roomb. Ball3 is a ball. Ball3 is at rooma. Ball4 is a ball. Ball4 is at roomb. Left is a gripper. Left is free. Right is a gripper. Right is free. Robby is at roomb. Room rooma Room roomb<|im\_end|>  
<|im\_start|>assistant  
<action>pick ball4 roomb right</action><|im\_end|>  
<|im\_start|>user  
Ball1 is a ball. Ball1 is at roomb. Ball2 is a ball. Ball2 is at roomb. Ball3 is a ball. Ball3 is at rooma. Ball4 is a ball. Ball4 is carrying right. Left is a gripper. Left is free. Right is a gripper. Robby is at roomb. Room rooma Room roomb<|im\_end|>  
<|im\_start|>assistant  
<action>move from roomb to rooma</action><|im\_end|>  
<|im\_start|>user  
Ball1 is a ball. Ball1 is at roomb. Ball2 is a ball. Ball2 is at roomb. Ball3 is a ball. Ball3 is at rooma. Ball4 is a ball. Ball4 is carrying right. Left is a gripper. Left is free. Right is a gripper. Robby is at roomb. Room rooma Room roomb<|im\_end|>  
<|im\_start|>assistant  
<action>drop ball4 rooma right</action><|im\_end|>

## Example trajectory on TriviaQA

<|im\_start|>user  
In 2006, who was the first football player in World Cup history to win the "Best Young Player Award"?<|im\_end|>  
<|im\_start|>assistant  
<search> first football player to win Best Young Player Award at World Cup 2006 </search><|im\_end|>  
<|im\_start|>user  
Doc 1(Title: "FIFA World Cup awards") specific award for their position, they are still eligible for the Golden Ball as well, as when Oliver Kahn was awarded in 2002. In the event of a tie, the Golden Glove Award goes to the goalkeeper who progressed furthest in the competition. The next tiebreakers are saves made, then minutes played. Although the Golden Glove Award was first awarded in 1994, every All-Star Team in World Cups prior to 1998 except 1990 included only one

goalkeeper. The ""Best Young Player"" award was awarded for the first time at the 2006 World Cup in Germany and given to Germany's Lukas

Doc 2(Title: "2006 Ballon d'O") Netherlands, New Zealand, Norway, Poland, Portugal, Republic of Ireland, Romania, Russia, San Marino, Scotland, Serbia, Slovakia, Slovenia, Spain, Sweden, Switzerland, Turkey, Ukraine and Wales. Each picked a first (5pts), second (4pts), third (3 pts), fourth (2pts) and fifth choice (1pt). Cannavaro was the fifth Italian player to win the award after Omar Sivori (1961), Gianni Rivera (1969), Paolo Rossi (1982) and Roberto Baggio (1993). Gianluigi Buffon (Italy) was the top ranked goalkeeper in the list, in second place; Ronaldinho (Brazil) was the top ranked midfielder, in fourth place, while Thierry Henry (France) was the top-ranked forward, in third place. The following

Doc 3(Title: "2006 FIFA World Cup") goals were scored by 110 players, with four of them credited as own goals. FIFA's Technical Study Group (TSG) also granted a Man of the Match award to one player in each match. Italy's Andrea Pirlo won the most Man of the Match awards, with three in total. Miroslav Klose, Agustin Delgado, Arjen Robben, Ze Roberto, Alexander Frei, Michael Ballack, and Patrick Vieira each received two awards. The All-star team is a squad consisting of the 23 most impressive players at the 2006 World Cup, as selected by FIFA's Technical Study Group. The team was chosen from a shortlist of

<|im\_end|>

<|im\_start|>assistant

<search>first football player to win Best Young Player Award at 2006 World Cup</search><|im\_end|>

<|im\_start|>user

Doc 1(Title: "FIFA World Cup awards") specific award for their position, they are still eligible for the Golden Ball as well, as when Oliver Kahn was awarded in 2002. In the event of a tie, the Golden Glove Award goes to the goalkeeper who progressed furthest in the competition. The next tiebreakers are saves made, then minutes played. Although the Golden Glove Award was first awarded in 1994, every All-Star Team in World Cups prior to 1998 except 1990 included only one goalkeeper. The ""Best Young Player"" award was awarded for the first time at the 2006 World Cup in Germany and given to Germany's Lukas

Doc 2(Title: "2006 Ballon d'Or") Netherlands, New Zealand, Norway, Poland, Portugal, Republic of Ireland, Romania, Russia, San Marino, Scotland, Serbia, Slovakia, Slovenia, Spain, Sweden, Switzerland, Turkey, Ukraine and Wales. Each picked a first (5pts), second (4pts), third (3 pts), fourth (2pts) and fifth choice (1pt). Cannavaro was the fifth Italian player to win the award after Omar Sivori (1961), Gianni Rivera (1969), Paolo Rossi (1982) and Roberto Baggio (1993). Gianluigi Buffon (Italy) was the top ranked goalkeeper in the list, in second place; Ronaldinho (Brazil) was the top ranked midfielder, in fourth place, while Thierry Henry (France) was the top-ranked forward, in third place. The following

Doc 3(Title: "2006 FIFA World Cup") goals were scored by 110 players, with four of them credited as own goals. FIFA's Technical Study Group (TSG) also granted a Man of the Match award to one player in each match. Italy's Andrea Pirlo won the most Man of the Match awards, with three in total. Miroslav Klose, Agustin Delgado, Arjen Robben, Ze Roberto, Alexander Frei, Michael Ballack, and Patrick Vieira each received two awards. The All-star team is a squad consisting of the 23 most impressive players at the 2006 World Cup, as selected by FIFA's Technical Study Group. The team was chosen from a shortlist of

<|im\_end|>

<|im\_start|>assistant

<answer>Lukas Podolski</answer><|im\_end|>

## F AI Assistant Usage

AI assistants were used only for minor language polishing and coding assistance, without affecting the scientific content of the paper.