RAMEM: ROLE-ALIGNED HIDDEN-STATE MEMORY FOR MULTI-AGENT SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM)-based multi-agent systems (MAS) accumulate experience as their roles collaborate, and memory is the mechanism that turns this experience into guidance for later tasks. However, existing memory pipelines select and reuse it at mismatched granularities: a trajectory can be globally relevant yet hold little that the role acting next can reuse, and compressing its mixed-role content into a fixed latent budget lets irrelevant or conflicting material crowd out what that role needs. To address this, we propose RAMEM, a role-aligned hidden-state memory. Instead of storing text, RAMEM archives the final-layer hidden states of each role's responses in successful trajectories, and it selects experience by both the task and the target role. A role-conditioned composer then compresses the selected states into a fixed-length latent memory while the agent backbone stays frozen. We train the composer in two stages, first aligning it to a text-memory teacher and then optimizing it for task reward with Hidden Memory Advantage Policy Optimization (HMAPO). Across six benchmarks and three MAS architectures, one unseen during memory construction, RAMEM attains the best average score in every architecture, 3.1 to 5.9 points above LatentMem.

1 INTRODUCTION

Large language model (LLM)-based multi-agent systems (MAS) solve problems through specialized roles that propose solutions, execute actions, critique intermediate results, and consolidate outcomes (Wu et al., 2024; Li et al., 2023; Hong et al., 2024; Qian et al., 2024; 2025; Han et al., 2024). Each collaboration leaves a trajectory of role-specific contributions, communication, and outcomes, and agents that accumulate such trajectories across tasks can reuse them as experience (Park et al., 2023; Wang et al., 2024; Zhang et al., 2025). Reusing a collaborative trajectory, however, requires deciding not only *which* past episode is relevant but also *which parts* of it should inform each role.

To reuse experience, agent memory has been built at three levels (Hu et al., 2025). Token-level memories keep it as text, such as observations and reflections (Park et al., 2023), verbal feedback (Shinn et al., 2023), executable skills (Wang et al., 2024), or structured records (Zhong et al., 2024; Zhao et al., 2024), and this text stays readable but lengthens the prompt as it grows. Parametric memories instead fold experience into agent weights (Chen et al., 2023; Yin et al., 2024), avoiding that cost but entangling it with prior knowledge. Latent memories sit between the two, compressing histories into a few continuous vectors the agent attends to directly (Ge et al., 2024; Zhang et al., 2026; Zhou & Sang, 2026). Closest to our setting, LatentMem (Fu et al., 2026) applies this to MAS, giving each role a compact, role-aware memory from retrieved trajectories and role profiles. Its retrieval, however, still ranks whole trajectories by source-task similarity, never checking whether the target role contributed reusable experience to them.

As a result, the composer receives the contributions of every role, even though only one role will use the memory. Consider the critic in Figure 1, which is reviewing an evaluation plan for an AI product. Task similarity retrieves a trajectory on the same task that interleaves planning, evidence collection, critique, and summarization, yet the critic may need only one observation from it, such as the missing baselines. Compressed into a fixed latent budget together with everything else, that observation is easily diluted. RAMEM instead composes the critic's memory from the critic's own past responses, so the whole budget goes to experience the critic can reuse.

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

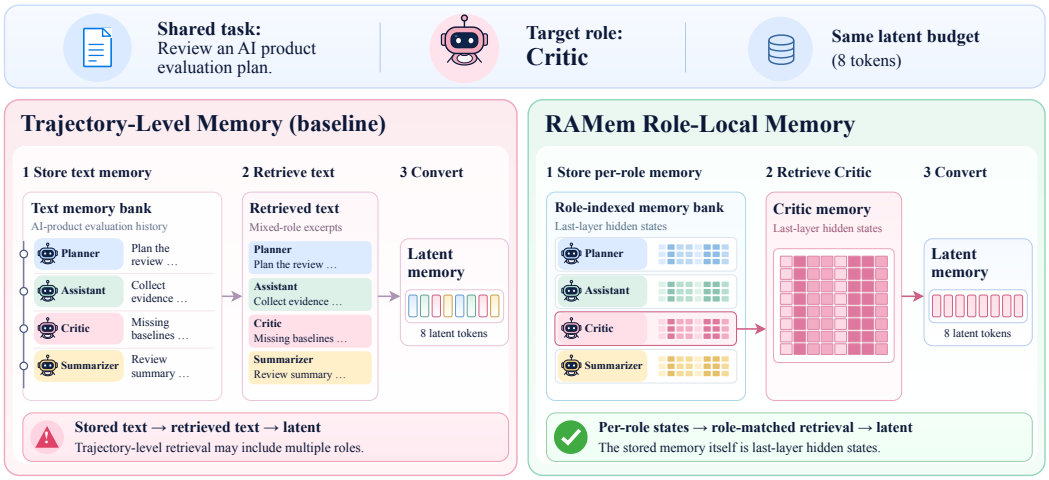

Figure 1: **Trajectory-level text memory and role-local hidden-state memory.** For the same task, Critic role, and latent budget, the baseline compresses retrieved text from multiple roles. RAMEM stores the final-layer hidden states of each role's responses and composes the selected Critic states into latent memory.

In RAMEM, these responses enter memory as the final-layer hidden states of the agent backbone that generates them, not as text. Each state is computed under the full source context, including the system prompt and earlier turns that the role record omits, so it reflects how the source agent conditioned its response on task and role; the hidden states supply this precomputed, whereas text would require re-reading that context.

To assemble a role's memory, RAMEM ranks candidate trajectories not only by how closely their source tasks match the current one but also by how closely the target role's responses in them do, and a role-conditioned composer compresses that role's hidden states from the selected trajectories into a fixed-length latent memory for the agent. The agent backbone stays frozen throughout; we train only the composer, first aligning it to a text-memory teacher and then optimizing the composed memory for task reward with Hidden Memory Advantage Policy Optimization (HMAPO).

We evaluate RAMEM on six benchmarks spanning single-hop and multi-hop question answering, code generation, and planning, and on three MAS architectures, one unseen during memory construction, with memory updated online. It attains the best average score in all three architectures, 3.1 to 5.9 points above LatentMem, while ablating either the hidden-state input or role-aligned retrieval lowers that average.

The contributions are:

- An external hidden-state experience memory for multi-agent systems, which, to our knowledge, is the first to store the agent backbone's final-layer hidden states for each role's responses and reuse them across tasks and MAS architectures, selected by the functional role that will use them. The same responses are kept as role records for retrieval.
- A role-aligned retrieval mechanism that scores candidates by both task similarity and alignment with the target role's recorded responses, favoring trajectories in which that role contributed reusable experience.
- Hidden Memory Advantage Policy Optimization (HMAPO), which trains the hidden-state composer with task-level rewards while keeping the agent backbone frozen, applies each trajectory's advantage to every memory-conditioned agent invocation in it, and anchors the composer to its aligned initialization through adaptive KL regularization.

2 Related Work

Multi-agent memory and retrieval. Memory lets LLM-based multi-agent systems, which coordinate specialized roles through structured communication (Wu et al., 2024; Li et al., 2023; Hong

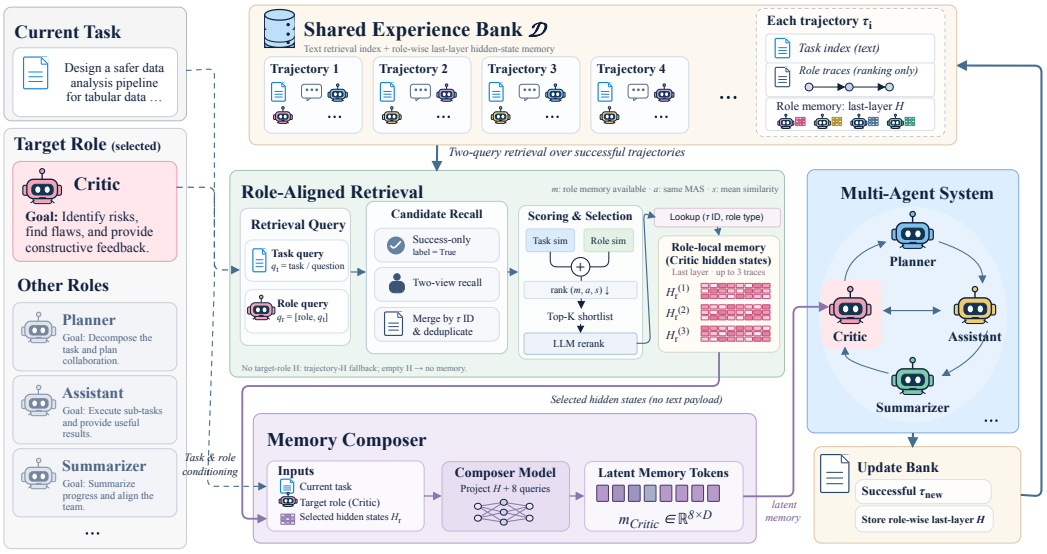

Figure 2: **RAMEM retrieval, composition, and writeback.** Task and role queries select successful trajectories using textual evidence. The selected trajectory identifiers locate the target role's hidden states, which the composer compresses into latent memory. Dashed arrows supply task and role conditioning; successful episodes contribute new hidden states to the stores.

et al., 2024; Qian et al., 2025), carry experience across tasks. Many inherit single-agent designs that keep experience as observations and reflections, verbal feedback, or executable skills (Park et al., 2023; Shinn et al., 2023; Wang et al., 2024), while G-Memory organizes multi-agent interaction history into hierarchical graphs (Zhang et al., 2025). Role-aware memories go further, dividing experience across agents or subtasks, maintaining hierarchical memories, or adapting retrieval granularity to the querying role (Yuen et al., 2025; Han et al., 2026; Huang et al., 2026). All still hold experience as text, whereas RAMEM keeps each role's experience as the hidden states of its own responses and selects it by both task and role similarity.

Latent representations in agent systems. A separate line moves agent computation from words to continuous representations, as in latent reasoning and context compression (Hao et al., 2025; Xu et al., 2025a; Ge et al., 2024; Zhou & Sang, 2026). Latent memories bring this to experience: MemGen (Zhang et al., 2026) weaves latent memory from the agent's current reasoning state and LatentMem (Fu et al., 2026) composes it from retrieved trajectory text, both generating it on demand rather than storing it. Other work stores hidden states, but in a parametric module (Lei et al., 2026) or in a latent store for a single agent (Feng et al., 2026; Zhao et al., 2026). LatentMAS (Zou et al., 2025), RecursiveMAS (Zou et al., 2026), and L^2-VMAS (Yu et al., 2026) share latent states among agents, and translation methods align them across models (Yang & Eshraghian, 2025; Liu et al., 2026), but only within one task. None keeps the hidden states a role produced on one task for reuse on later ones, as RAMEM does across tasks and MAS architectures.

3 RAMEM

As Figure 2 shows, RAMEM organizes memory around the role that will reuse it and keeps two linked stores. The shared experience bank $\mathcal{D}$ holds complete successful trajectories, indexed by source-task embeddings, and role memory $\mathcal{B}$ holds the hidden states of each role's responses in those trajectories. Given task x and role r, task and role queries retrieve candidate trajectories from $\mathcal{D}$; similarity scoring followed by an LLM reranker selects which to reuse. A lookup by trajectory identifier and functional role then loads the target role's hidden states from $\mathcal{B}$, and the composer compresses them into a fixed-length latent memory for the target agent. Successful episodes add new trajectories and hidden states to both stores.

3.1 HIDDEN-STATE EXPERIENCE MEMORY

The shared experience bank $\mathcal{D}$ stores complete successful trajectories with their source tasks, role-labeled messages, communication structures, and outcomes, indexed by source-task embeddings. Role memory $\mathcal{B}$ stores the experience that RAMEM reuses: the agent backbone's final-layer hidden states for each role's responses in these trajectories, with agent names normalized to functional roles so that agents playing the same role in different communication topologies share experience. Since hidden states cannot be searched by text similarity, $\mathcal{B}$ also keeps each role's responses as a role record with their direct upstream messages, e.g., a critic's feedback with the answer it evaluates.

Formally, let x_τ denote the source task of trajectory τ, y_v the response of an agent invocation v, and $\mathcal{I}_{\tau,r}$ the invocations of role r in τ whose hidden states were captured, in temporal order. For each $v \in \mathcal{I}_{\tau,r}$, we store the hidden-state block $H_v = [h_{v,1}^\top; \ldots; h_{v,T_v}^\top] \in \mathbb{R}^{T_v \times d_h}$, whose t-th row is the agent backbone's final-layer state after response token t under the source prompt and the preceding response tokens. The role record is

$$\begin{aligned} c_v &= \mathrm{concat}\left([y_u]_{u \in \mathrm{Pred}_\tau(v)}\right), \\ e_{\tau,r} &= \left(x_\tau,\, [(c_v, y_v)]_{v \in \mathcal{I}_{\tau,r}}\right), \end{aligned} \tag{1}$$

where $\mathrm{Pred}_\tau(v)$ denotes the direct predecessors of v in the communication graph, concatenated in message order. The key $(\mathrm{id}(\tau), r)$ links the trajectory, its role record, and the hidden-state blocks $\{H_v\}_{v \in \mathcal{I}_{\tau,r}}$.

3.2 ROLE-ALIGNED RETRIEVAL

Task similarity alone does not show whether the target role contributed reusable experience: a trajectory on a closely matching task may contain no response from that role, or only responses unrelated to what it must do now. We therefore score each candidate by how well both its source task and the target role's recorded responses match the current task, in the spirit of query–memory alignment (Hu et al., 2026), and then rerank complete trajectories.

Candidate retrieval. Given task x and role r, let q denote the agent's current task context, which in multi-turn tasks includes the interaction so far, and q_r the same text prefixed with the normalized functional role. Both query the source-task index of successful trajectories in $\mathcal{D}$, and we take the union of their results,

$$\mathcal{C}_{x,r} = \mathcal{N}_b(q; \mathcal{D}) \cup \mathcal{N}_b(q_r; \mathcal{D}), \tag{2}$$

where $\mathcal{N}_b$ returns up to b nearest trajectories.

Task–role alignment. Let $m_{\tau,r} \in \{0,1\}$ indicate whether τ has a nonempty role record $e_{\tau,r}$ for role r, and let $a_\tau \in \{0,1\}$ indicate whether τ comes from the target architecture. For $m_{\tau,r} = 1$, the role view $\bar{e}_{\tau,r} = \mathrm{Serialize}(e_{\tau,r})$ writes the source task, the functional role, and the context–response pairs of $e_{\tau,r}$ as one string. Let f be the sentence encoder with unit-normalized outputs and $\lambda \in [0,1]$ a weight between task and role similarity. We define scores for every candidate as

$$\begin{aligned} s_{\mathrm{task}}(x,\tau) &= f(q)^\top f(x_\tau), \\ s_{\mathrm{role}}(x,r,\tau) &= \begin{cases} f(q_r)^\top f(\bar{e}_{\tau,r}), & m_{\tau,r} = 1, \\ 0, & m_{\tau,r} = 0, \end{cases} \\ s(x,r,\tau) &= (1-\lambda)\, s_{\mathrm{task}}(x,\tau) + \lambda\, s_{\mathrm{role}}(x,r,\tau). \end{aligned} \tag{3}$$

We sort $\mathcal{C}_{x,r}$ by the descending lexicographic key

$$\left(m_{\tau,r},\ a_\tau,\ s(x,r,\tau)\right)$$

and keep the first n_{s} candidates as the shortlist, so role presence and architecture match take priority over similarity.

Trajectory-level reranking. An LLM reranker scores each shortlisted trajectory using its complete interaction history and the current task, and the K highest-scoring trajectories form $\mathcal{T}_{x,r}$. Their identifiers and the target functional role locate the hidden-state blocks $\{H_v\}_{v \in \mathcal{I}_{\tau,r},\, \tau \in \mathcal{T}_{x,r}}$, of which we keep the first three per trajectory and list them in trajectory and temporal order as $\mathcal{H}_{x,r} = (H_1, \ldots, H_J)$, with J the total number of blocks.

3.3 ROLE-CONDITIONED HIDDEN-STATE COMPOSITION

The composer turns the retrieved blocks $\mathcal{H}_{x,r}$ into memory the target agent can use: it must read hidden states rather than text, condition on the current task and role, and fit what it reads into a fixed number of latent tokens. It is a causal language model with LoRA adapters, learned query vectors, an output projection, and a trainable input projection that admits archived hidden states. Task and role instructions and block delimiters enter through its token embeddings, and each hidden-state block enters through the input projection. We initialize it from the trained LatentMem composer (Fu et al., 2026).

Input alignment. Because source hidden states and composer input embeddings occupy different representation spaces (Zou et al., 2026), we map each source state with an affine projection $g_\omega(H) = HW_h^\top + \mathbf{1}b_h^\top$, where $W_h \in \mathbb{R}^{d_c \times d_h}$, $b_h \in \mathbb{R}^{d_c}$, and d_c is the composer width. Let E_c denote the composer's token embedding layer, $w_{x,r}^{\text{task}}$ the task prefix, w_r^{role} the role instruction, and δ_j the delimiter preceding block H_j. The packed input is

$$U_{x,r} = \left[E_c(w_{x,r}^{\text{task}});\, E_c(\delta_1);\, g_\omega(H_1);\, \ldots;\, E_c(\delta_J);\, g_\omega(H_J);\, E_c(w_r^{\text{role}})\right]. \tag{4}$$

Fixed-budget composition. We append L learned query vectors $\mathbf{Q} \in \mathbb{R}^{L \times d_c}$ to $U_{x,r}$; through causal attention, their positions read the preceding task, role instruction, and projected hidden states. With F_ϕ the composer backbone returning final-layer states, the composed latent memory is

$$\begin{aligned} \mathbf{Z}_{x,r} &= \text{Tail}_L(F_\phi([U_{x,r}; \mathbf{Q}]))\,, \\ \mathbf{M}_{x,r} &= P_\psi(\mathbf{Z}_{x,r}) \in \mathbb{R}^{L \times d_a}, \end{aligned} \tag{5}$$

where $[\cdot;\cdot]$ denotes sequence concatenation, Tail_L selects the L query positions, and P_ψ maps them into the agent's d_a-dimensional input space. The composer is shared across roles, and all selected blocks are compressed into the same L latent tokens. Let p_r be the agent's current prompt, including task instructions and the ongoing interaction, with token embeddings $E_r(p_r) \in \mathbb{R}^{n_r \times d_a}$. The agent generates its response from the memory-augmented input

$$\widetilde{\mathbf{E}}_{x,r} = [E_r(p_r); \mathbf{M}_{x,r}] \in \mathbb{R}^{(n_r+L) \times d_a}. \tag{6}$$

3.4 TRAINING THE HIDDEN MEMORY INTERFACE

Training has two stages, both with the agent backbone frozen. Let π_Θ denote the agent backbone's response distribution conditioned on memory produced with interface parameters Θ, which comprise g_ω, the composer's trainable adapters, $\mathbf{Q}$, and P_ψ.

Teacher-guided interface alignment. For each invocation of the target role in a successful trajectory, with response y, the student composes memory from the archived hidden-state blocks as in Eq. (4). The teacher is the LatentMem composer in its original text mode: it reads the text of the same source responses in place of each $g_\omega(H_j)$, with the same task prefix, delimiters, and role instruction. Matching the agent backbone's response distributions under the two memories thus aligns the projected hidden states with the text embeddings they replace. On the target response positions, including EOS, we minimize

$$\mathcal{L}_{\text{align}} = \mathbb{E}_{(p_r,y)} \frac{1}{|y|} \sum_{t=1}^{|y|} \left[-\log \pi_\Theta(y_t \mid p_r, y_{<t}) + \beta\, D_{\text{KL}}(\pi_{\text{text},t} \,\|\, \pi_{\Theta,t})\right], \tag{7}$$

where $\pi_{\text{text},t}$ is the response distribution under the teacher's memory and β weights distillation against response supervision. This stage updates only g_ω; all inherited composer weights stay fixed.

Task-reward optimization with HMAPO. Alignment teaches the composer to read hidden states as it reads text, but not which memory helps the team succeed. Because the agent backbone is frozen, task reward can reach the composer only through the memory it produces, and because one outcome judges every role in a trajectory, that outcome must supervise each memory-conditioned invocation in it. Building on GRPO (Shao et al., 2024) and the dynamic sampling of DAPO (Yu et al., 2025), we propose HMAPO for this setting. Starting from the aligned interface, it merges the inherited LoRA

adapters into the composer backbone and trains new LoRA (Hu et al., 2022) adapters together with both projections and the query vectors. The aligned interface also serves as a frozen reference π_{Θ_0}, and a verifier scores each complete multi-agent trajectory τ_i:

$$R_i = \begin{cases} \mathbb{I}[\text{final answer passes the correctness check}], & \text{QA}, \\ R_{\text{all},i} + \eta(1 - R_{\text{all},i})u_i, & \text{code}, \end{cases} \tag{8}$$

where $R_{\text{all},i}$ indicates full-test success, u_i is the fraction of individual tests passed, and $0 \le \eta < 1$ weights the partial credit that distinguishes progress among failed code solutions. For each task, we sample G current-policy trajectories and compute a leave-one-out advantage from their rewards,

$$A_i = R_i - \frac{1}{G-1} \sum_{j \neq i} R_j, \tag{9}$$

without standard-deviation normalization. Every invocation in τ_i whose response depends on the memory interface applies A_i to all of its response tokens, so the final outcome supervises memory use across the participating roles. A group whose rewards are all identical yields no policy gradient, and we call the remaining groups effective. As in DAPO, when a device holds fewer effective groups than its target, HMAPO samples additional tasks for a bounded number of rounds, weighting them by difficulty strata estimated from observed outcomes.

Let $\mathcal{Y}_{\text{pres}}$ contain the response tokens of trainable invocations in effective groups and $\mathcal{Y}_{\text{pol}} \subseteq \mathcal{Y}_{\text{pres}}$ those with nonzero advantage, with t indexing response tokens across the invocations of trajectory i. Write $\ell_{i,t}(\Theta) = \log \pi_\Theta(y_{i,t} \mid \xi_{i,t})$ for the token log-likelihood given the agent prompt and response prefix $\xi_{i,t}$, $\rho_{i,t} = \exp(\ell_{i,t}(\Theta) - \ell_{i,t}(\Theta_{\text{old}}))$ for the ratio to the rollout policy, and $\Delta_{i,t} = \ell_{i,t}(\Theta_0) - \ell_{i,t}(\Theta)$ for the log-ratio to the reference. We minimize

$$\begin{aligned} \mathcal{L}_{\text{HMAPO}} = &-\frac{1}{N_{\text{pol}}} \sum_{(i,t) \in \mathcal{Y}_{\text{pol}}} \min(\rho_{i,t} A_i,\ \text{clip}(\rho_{i,t}, 1-\epsilon, 1+\epsilon) A_i) \\ &+ \frac{\mu}{N_{\text{pres}}} \sum_{(i,t) \in \mathcal{Y}_{\text{pres}}} \left[\exp(\Delta_{i,t}) - \Delta_{i,t} - 1\right]. \end{aligned} \tag{10}$$

Here N_{pol} and N_{pres} count tokens in $\mathcal{Y}_{\text{pol}}$ and $\mathcal{Y}_{\text{pres}}$, ϵ is the clipping threshold, and the reference term is a nonnegative sampled KL estimate that anchors the policy to the aligned initialization. Each fresh rollout batch is optimized in a single pass, so $\rho_{i,t} = 1$ when the gradient is evaluated and the gradient of the policy term reduces to an advantage-weighted log-likelihood gradient. The coefficient μ adapts to the mean reference divergence $\widehat{D}_k$ at update k and a target $D^\star$:

$$\widetilde{\mu}_{k+1} = \begin{cases} \kappa \mu_k, & \widehat{D}_k > \kappa D^\star, \\ \mu_k / \kappa, & \widehat{D}_k < D^\star / \kappa, \\ \mu_k, & \text{otherwise}, \end{cases} \qquad \mu_{k+1} = \text{clip}(\widetilde{\mu}_{k+1}, \mu_{\min}, \mu_{\max}), \tag{11}$$

where $\kappa > 1$ controls the adjustment and $[\mu_{\min}, \mu_{\max}]$ bounds its strength.

4 Experiments

4.1 Experimental Setup

Benchmarks and metrics. We evaluate on six benchmarks covering single-hop and multi-hop question answering, code generation, and planning. TriviaQA (Joshi et al., 2017), PopQA (Mallen et al., 2023), and KodCode (Xu et al., 2025b) are source-domain tasks from which the initial memory is built. HotpotQA (Yang et al., 2018), BigCodeBench (Zhuo et al., 2025), and PDDL (Ma et al., 2024) are transfer tasks. We report accuracy for QA, test-suite accuracy for KodCode, pass@1 against the official Docker test suites for BigCodeBench, and for PDDL the episode reward, which is the maximum fraction of goal literals satisfied during execution, averaged over episodes.

Architectures and baselines. We instantiate AutoGen (Wu et al., 2024), MacNet (Qian et al., 2025), and CAMEL (Li et al., 2023) with two, five, and four agents, all running `Qwen3-4B-Instruct-2507` (Qwen Team, 2025). AutoGen and MacNet contributed trajectories to the initial memory. CAMEL did not and serves as an unseen architecture. Within each

Table 1: **Main results.** Superscripts show absolute changes from No memory within the same architecture. Avg. is the six-benchmark mean. Bold marks the best result, including ties.

Method	Source-domain tasks			Transfer tasks			Avg.
	TriviaQA Acc.	KodCode Acc.	PopQA Acc.	HotpotQA Acc.	BigCodeBench pass@1	PDDL Reward	
AutoGen							SEEN MAS
No memory	60.31	68.40	38.78	36.60	39.21	16.39	43.28
MetaGPT	60.35$^{+0.04}$	70.05$^{+1.65}$	33.80$^{-4.98}$	31.60$^{-5.00}$	39.21$^{0.00}$	11.95$^{-4.44}$	41.16$^{-2.12}$
Generative	59.65$^{-0.66}$	70.90$^{+2.50}$	40.37$^{+1.59}$	31.00$^{-5.60}$	39.04$^{-0.17}$	13.94$^{-2.45}$	42.48$^{-0.80}$
Voyager	57.50$^{-2.81}$	68.95$^{+0.55}$	33.56$^{-5.22}$	32.00$^{-4.60}$	39.39$^{+0.18}$	13.62$^{-2.77}$	40.84$^{-2.44}$
G-Memory	60.56$^{+0.25}$	71.40$^{+3.00}$	42.67$^{+3.89}$	31.00$^{-5.60}$	40.09$^{+0.88}$	17.06$^{+0.67}$	43.80$^{+0.52}$
OAgents	59.85$^{-0.46}$	70.80$^{+2.40}$	40.70$^{+1.92}$	31.60$^{-5.00}$	39.12$^{-0.09}$	16.70$^{+0.31}$	43.13$^{-0.15}$
LatentMem	76.51$^{+16.20}$	76.80$^{+8.40}$	52.70$^{+13.92}$	**40.40**$^{+3.80}$	41.40$^{+2.19}$	23.49$^{+7.10}$	51.88$^{+8.60}$
RAMem	**80.52**$^{+20.21}$	**79.25**$^{+10.85}$	**56.63**$^{+17.85}$	40.20$^{+3.60}$	**42.54**$^{+3.33}$	**30.87**$^{+14.48}$	**55.00**$^{+11.72}$
MacNet							SEEN MAS
No memory	53.77	70.40	24.89	18.20	42.81	20.73	38.47
MetaGPT	61.16$^{+7.39}$	71.50$^{+1.10}$	33.98$^{+9.09}$	25.00$^{+6.80}$	40.53$^{-2.28}$	17.81$^{-2.92}$	41.66$^{+3.19}$
Generative	59.89$^{+6.12}$	71.15$^{+0.75}$	43.39$^{+18.50}$	25.20$^{+7.00}$	39.65$^{-3.16}$	16.81$^{-3.92}$	42.68$^{+4.21}$
Voyager	58.19$^{+4.42}$	69.80$^{-0.60}$	35.38$^{+10.49}$	24.80$^{+6.60}$	38.95$^{-3.86}$	14.75$^{-5.98}$	40.31$^{+1.84}$
G-Memory	62.43$^{+8.66}$	72.50$^{+2.10}$	43.88$^{+18.99}$	26.20$^{+8.00}$	40.09$^{-2.72}$	21.82$^{+1.09}$	44.49$^{+6.02}$
OAgents	60.63$^{+6.86}$	71.30$^{+0.90}$	41.90$^{+17.01}$	26.20$^{+8.00}$	40.70$^{-2.11}$	22.83$^{+2.10}$	43.93$^{+5.46}$
LatentMem	65.98$^{+12.21}$	**78.90**$^{+8.50}$	44.14$^{+19.25}$	35.60$^{+17.40}$	41.84$^{-0.97}$	25.13$^{+4.40}$	48.60$^{+10.13}$
RAMem	**79.45**$^{+25.68}$	78.05$^{+7.65}$	**54.62**$^{+29.73}$	**40.60**$^{+22.40}$	**43.95**$^{+1.14}$	**30.23**$^{+9.50}$	**54.48**$^{+16.01}$
CAMEL							UNSEEN MAS
No memory	56.96	70.70	32.38	18.00	42.46	22.10	40.43
MetaGPT	59.06$^{+2.10}$	69.90$^{-0.80}$	37.47$^{+5.09}$	26.60$^{+8.60}$	38.07$^{-4.39}$	22.55$^{+0.45}$	42.27$^{+1.84}$
Generative	57.63$^{+0.67}$	70.65$^{-0.05}$	35.32$^{+2.94}$	26.80$^{+8.80}$	38.60$^{-3.86}$	17.29$^{-4.81}$	41.05$^{+0.62}$
Voyager	56.57$^{-0.39}$	69.85$^{-0.85}$	36.25$^{+3.87}$	27.40$^{+9.40}$	39.39$^{-3.07}$	23.65$^{+1.55}$	42.18$^{+1.75}$
G-Memory	59.20$^{+2.24}$	70.40$^{-0.30}$	38.04$^{+5.66}$	26.40$^{+8.40}$	37.54$^{-4.92}$	24.56$^{+2.46}$	42.69$^{+2.26}$
OAgents	58.33$^{+1.37}$	71.40$^{+0.70}$	31.99$^{-0.39}$	26.00$^{+8.00}$	37.89$^{-4.57}$	13.89$^{-8.21}$	39.92$^{-0.51}$
LatentMem	68.74$^{+11.78}$	77.75$^{+7.05}$	47.23$^{+14.85}$	38.00$^{+20.00}$	**42.63**$^{+0.17}$	28.12$^{+6.02}$	50.41$^{+9.98}$
RAMem	**79.94**$^{+22.98}$	**79.75**$^{+9.05}$	**54.66**$^{+22.28}$	**40.00**$^{+22.00}$	42.46$^{0.00}$	**30.56**$^{+8.46}$	**54.56**$^{+14.13}$

architecture we compare RAMEM with LatentMem (Fu et al., 2026) and with the memory mechanisms of MetaGPT (Hong et al., 2024), Generative (Park et al., 2023), Voyager (Wang et al., 2024), G-Memory (Zhang et al., 2025), and OAgents (Zhu et al., 2025), plus a no-memory baseline on the same workflow. Appendix B describes each benchmark, baseline, and architecture.

Implementation. The initial shared experience bank $\mathcal{D}$ is built from successful AutoGen and MacNet trajectories on the source training splits, and role memory $\mathcal{B}$ holds the hidden states and role records of these trajectories, whose statistics Appendix B.4 reports. Source tasks are indexed with `all-MiniLM-L6-v2` (Reimers & Gurevych, 2019). The two query views are merged, candidates are ordered with $\lambda = 0.5$ in Eq. (3), and an LLM reranker scores the $n_s = 3$ shortlisted trajectories and keeps $K = 1$. The composer is initialized from the trained LatentMem composer with a Qwen3-4B-Instruct-2507 backbone (Qwen Team, 2025), trained first with teacher-guided interface alignment and then with HMAPO, and encodes the selected hidden states into $L = 8$ latent tokens. Appendix B.1 gives the data splits and training hyperparameters for both stages.

4.2 MAIN RESULTS

Results across architectures and benchmarks. Table 1 shows that text-based memories help unevenly. The best of them, G-Memory (Zhang et al., 2025) on MacNet, raises the six-benchmark average by 6.02 points, while on AutoGen four of the five fall below No memory, with PDDL rewards dropping by up to 4.44 points. Compressing trajectories into latent memory is more reliable: LatentMem raises the average by 8.6 to 10.1 points in every architecture. RAMEM goes further and attains the best average in all three architectures, 3.1 to 5.9 points above LatentMem.

Table 2: **Component ablation on three source-domain benchmarks and the PDDL transfer task.** *Full trajectory* is LatentMem. Superscripts show absolute changes from it within the same architecture. Bold marks the best result, including ties.

Configuration	Components			Source-domain			Transfer	Avg.
	Role input	Hidden states	Aligned retrieval	TriviaQA	KodCode	PopQA	PDDL	
AutoGen								SEEN MAS
Full trajectory	–	–	–	76.51	76.80	52.70	23.49	57.38
Role record	✓	–	–	74.19$^{-2.32}$	78.80$^{+2.00}$	52.58$^{-0.12}$	23.72$^{+0.23}$	57.32$^{-0.06}$
Role record + retrieval	✓	–	✓	77.92$^{+1.41}$	77.75$^{+0.95}$	54.82$^{+2.12}$	28.58$^{+5.09}$	59.77$^{+2.39}$
Role hidden states	✓	✓	–	**80.57**$^{+4.06}$	76.85$^{+0.05}$	56.56$^{+3.86}$	30.53$^{+7.04}$	61.13$^{+3.75}$
RAMEM	✓	✓	✓	80.52$^{+4.01}$	**79.25**$^{+2.45}$	**56.63**$^{+3.93}$	**30.87**$^{+7.38}$	**61.82**$^{+4.44}$
MacNet								SEEN MAS
Full trajectory	–	–	–	65.98	**78.90**	44.14	25.13	53.54
Role record	✓	–	–	72.23$^{+6.25}$	77.55$^{-1.35}$	51.45$^{+7.31}$	19.63$^{-5.50}$	55.22$^{+1.68}$
Role record + retrieval	✓	–	✓	73.63$^{+7.65}$	78.45$^{-0.45}$	51.71$^{+7.57}$	26.69$^{+1.56}$	57.62$^{+4.08}$
Role hidden states	✓	✓	–	78.98$^{+13.00}$	77.45$^{-1.45}$	54.59$^{+10.45}$	29.71$^{+4.58}$	60.18$^{+6.64}$
RAMEM	✓	✓	✓	**79.45**$^{+13.47}$	78.05$^{-0.85}$	**54.62**$^{+10.48}$	**30.23**$^{+5.10}$	**60.59**$^{+7.05}$
CAMEL								UNSEEN MAS
Full trajectory	–	–	–	68.74	77.75	47.23	28.12	55.46
Role record	✓	–	–	72.24$^{+3.50}$	79.15$^{+1.40}$	49.95$^{+2.72}$	20.17$^{-7.95}$	55.38$^{-0.08}$
Role record + retrieval	✓	–	✓	75.00$^{+6.26}$	77.15$^{-0.60}$	52.36$^{+5.13}$	25.69$^{-2.43}$	57.55$^{+2.09}$
Role hidden states	✓	✓	–	66.51$^{-2.23}$	73.50$^{-4.25}$	50.16$^{+2.93}$	28.01$^{-0.11}$	54.55$^{-0.91}$
RAMEM	✓	✓	✓	**79.94**$^{+11.20}$	**79.75**$^{+2.00}$	**54.66**$^{+7.43}$	**30.56**$^{+2.44}$	**61.23**$^{+5.77}$

Transfer tasks and an unseen architecture. Transfer tasks, whose trajectories enter memory only through online writeback, show the same contrast. Text memories raise BigCodeBench and PDDL scores by at most 2.5 points and often lower them, and their effect on HotpotQA depends on the architecture, lowering accuracy on AutoGen by 4.6 to 5.6 points but raising it by 6.6 to 9.4 on MacNet and CAMEL, whose No memory scores are near 18. RAMEM is stable across architectures, reaching 40.0 to 40.6 HotpotQA accuracy in all three. It also carries over to the unseen architecture: CAMEL contributed nothing to the initial memory, yet RAMEM raises its average by 14.13 points over No memory and 4.15 over LatentMem, starting from experience of other architectures through role normalization and adding its own successful episodes through writeback.

4.3 Ablation and Analysis

Component ablation. To separate the contribution of each design choice, we start from LatentMem (Fu et al., 2026), whose composer reads full trajectories as text, and add the components of RAMEM one at a time: restricting the input to the target role's responses, reading those responses as hidden states, and selecting trajectories with role-aligned retrieval. Table 2 reports the five resulting configurations under the same memory, latent length, and writeback. Role records alone have mixed effects and lower PDDL reward on MacNet and CAMEL. Adding role-aligned retrieval to role records raises the average by 2.2 to 2.5 points in every architecture, and replacing role text with hidden states raises it by 3.8 to 5.0 points on the seen architectures. Combining both gives the best average in all three architectures, 4.4 to 7.1 points above LatentMem, and retrieval contributes most on the unseen CAMEL, 6.7 points over hidden states alone. For training, Table 5 in Appendix C shows that HMAPO improves every entry over teacher-guided alignment alone, and Table 6 shows that RAMEM stays 4.0 to 7.3 points above LatentMem's text composer trained with the same HMAPO. Table 7 removes the retrieval components one at a time.

Latent memory visualization. Since RAMEM composes each role's memory from that role's own responses, we check whether different roles indeed receive different memories, projecting the composer outputs for every agent invocation on 300 tasks under LatentMem and RAMEM. Figure 3 shows t-SNE projections for CAMEL and MacNet. Under LatentMem, CAMEL's four roles overlap and MacNet's roles are distributed across disconnected clusters. Under RAMEM, the roles separate

more clearly, indicating that each role receives its own memory rather than a shared summary of the same trajectory.

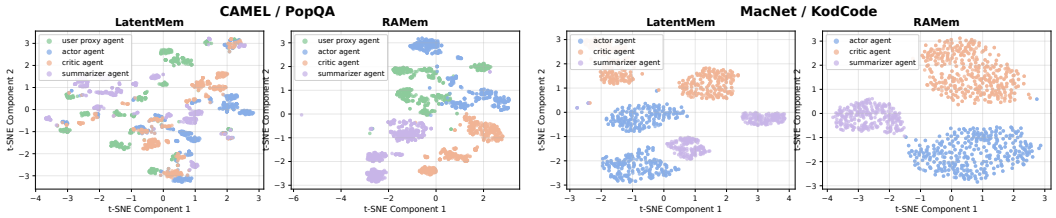

Figure 3: **Role structure of composed latent memories.** Each point is one agent invocation, colored by role; both halves of a panel share one projection.

Sensitivity analysis. Unless stated otherwise, the analyses below use one source-domain benchmark per architecture, and we call each combination a pairing: AutoGen with TriviaQA, CAMEL with PopQA, and MacNet with KodCode. Figure 4 varies the task–role weight λ in Eq. (3) on the first 300 tasks of each pairing, holding the query views and the role and architecture preferences fixed. At $\lambda = 0$, task similarity determines the ordering; at $\lambda = 1$, role-view similarity determines it. Performance is largely insensitive to λ: the endpoints are 1.7 to 2.7 points below the best intermediate value, and $\lambda = 0.5$ stays within 1.3 points of the best setting in every pairing. We therefore use it without per-architecture tuning.

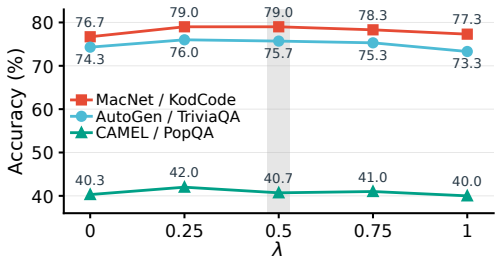

Figure 4: **Sensitivity to the task–role weight λ.** Role and architecture preferences stay fixed; the band marks $\lambda = 0.5$.

Latent composition vs. text injection. Composing latent memory takes a trained composer, so we check whether it pays off compared with giving the agent the retrieved responses directly. Holding retrieval fixed on 300 tasks per pairing, we compare RAMEM's latent composition, in which the composer reads the hidden states of the retrieved role responses, with text injection, which appends the serialized role records of the same responses to the agent prompt. Table 3 reports accuracy and the input and output tokens per agent invocation. Latent composition improves accuracy by 8 to 12 points in all three pairings. Replacing the text record with eight latent tokens shortens the prompt by 3 to 51%, and each invocation generates 16 to 71% fewer output tokens. Appendix D gives qualitative cases, including ones where the retrieved experience transfers a procedure, such as verifying before answering, rather than an answer.

Table 3: **Latent composition versus text injection.** Accuracy (%) and tokens per agent invocation on 300 tasks per pairing with retrieval held fixed. Tokens are averaged over all agent invocations. Δ columns give the change from text to latent.

	Accuracy		Input tokens			Output tokens		
Pairing	Text	Latent (Δ)	Text	Latent	Δ	Text	Latent	Δ
AutoGen / TriviaQA	64.3	**72.7** (+8.4)	612	**596**	−3%	58	**17**	−71%
CAMEL / PopQA	32.0	**40.3** (+8.3)	880	**501**	−43%	54	**27**	−50%
MacNet / KodCode	70.0	**82.0** (+12.0)	1100	**540**	−51%	139	**117**	−16%

5 CONCLUSION

In this work, we present RAMEM, a role-aligned hidden-state memory for multi-agent systems. By archiving the agent backbone's hidden states for each role's responses, selecting them with task and role evidence, and composing the target role's states into a fixed latent budget, RAMEM gives each role memory built from its own experience. Across six benchmarks and three MAS architectures, including an unseen one, it attains the best average in every architecture, suggesting that multi-agent memory benefits from being organized around the role that will reuse it.

AI USE STATEMENT

LLM assistants were used to draft and polish the text of this paper, to search for related work and check citation records, and to help design the ablation protocol and write the evaluation and plotting scripts. The authors reviewed and edited all generated content and take responsibility for it.

ETHICS STATEMENT

This work uses publicly available benchmarks and models and involves no human subjects or personal data. Stored trajectories are generated by the agents on these benchmarks, and only episodes that passed the benchmark check enter memory, which limits the propagation of earlier mistakes. The work follows the ICLR Code of Ethics.

REPRODUCIBILITY STATEMENT

The training and evaluation configurations are documented in Section 4.1 and Appendix B, including training hyperparameters, benchmark settings, and evaluation metrics. Our code, configuration files, and evaluation scripts are included in the supplementary material.

REFERENCES

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. FireAct: Toward language agent fine-tuning. *arXiv preprint arXiv:2310.05915*, 2023. doi: 10.48550/arXiv.2310.05915. URL https://arxiv.org/abs/2310.05915.

Tao Feng, Chongrui Ye, Tianyang Luo, Jingjun Xu, Xueqiang Xu, Haozhen Zhang, Ge Liu, and Jiaxuan You. ElasticMem: Latent memory as a learnable resource for LLM agents. *arXiv preprint arXiv:2605.30690*, 2026. doi: 10.48550/arXiv.2605.30690. URL https://arxiv.org/abs/2605.30690v1.

Muxin Fu, Xiangyuan Xue, Yafu Li, Zefeng He, Siyuan Huang, Xiaoye Qu, Yu Cheng, and Yang Yang. LatentMem: Customizing latent memory for multi-agent systems. *arXiv preprint arXiv:2602.03036*, 2026. doi: 10.48550/arXiv.2602.03036. URL https://arxiv.org/abs/2602.03036v2.

Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. Precise zero-shot dense retrieval without relevance labels. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1762–1777. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.acl-long.99. URL https://aclanthology.org/2023.acl-long.99/.

Tao Ge, Jing Hu, Lei Wang, Xun Wang, Si-Qing Chen, and Furu Wei. In-context autoencoder for context compression in a large language model. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/0b276510ec2d3f6613a8b60c41ff0438-Abstract-Conference.html.

Dongge Han, Camille Couturier, Daniel Madrigal Diaz, Xuchao Zhang, Victor Rühle, and Saravan Rajmohan. LEGOMem: Modular procedural memory for multi-agent LLM systems for workflow automation. In *Proceedings of the 25th International Conference on Autonomous Agents and Multiagent Systems*, 2026. doi: 10.65109/vlua1303. URL https://doi.org/10.65109/vlua1303.

Shanshan Han, Qifan Zhang, Weizhao Jin, and Zhaozhuo Xu. LLM multi-agent systems: Challenges and open problems. *arXiv preprint arXiv:2402.03578*, 2024. doi: 10.48550/arXiv.2402.03578. URL https://arxiv.org/abs/2402.03578v3.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=Itxz7S4Ip3.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VtmBAGCN7o.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Xuxian Hu, Zhu Teng, Wei Zhang, Ming He, and Jianping Fan. QueryLink: Leveraging query-memory alignment for long-term reasoning in LLM agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 15608–15621, San Diego, California, United States, 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.findings-acl.765. URL https://aclanthology.org/2026.findings-acl.765/.

Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, Senjie Jin, Jiejun Tan, Yanbin Yin, Jiongnan Liu, Zeyu Zhang, Zhongxiang Sun, Yutao Zhu, Hao Sun, Boci Peng, Zhenrong Cheng, Xuanbo Fan, Jiaxin Guo, Xinlei Yu, Zhenhong Zhou, Zewen Hu, Jiahao Huo, Junhao Wang, Yuwei Niu, Yu Wang, Zhenfei Yin, Xiaobin Hu, Yue Liao, Qiankun Li, Kun Wang, Wangchunshu Zhou, Yixin Liu, Dawei Cheng, Qi Zhang, Tao Gui, Shirui Pan, Yan Zhang, Philip Torr, Zhicheng Dou, Ji-Rong Wen, Xuanjing Huang, Yu-Gang Jiang, and Shuicheng Yan. Memory in the age of AI agents. *arXiv preprint arXiv:2512.13564*, 2025. doi: 10.48550/arXiv.2512.13564. URL https://arxiv.org/abs/2512.13564v2.

Weiquan Huang, Zixuan Wang, Hehai Lin, Sudong Wang, Bo Xu, Qian Li, Beier Zhu, Linyi Yang, and Chengwei Qin. AMA: Adaptive memory via multi-agent collaboration. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 3099–3120, San Diego, California, United States, 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.findings-acl.152. URL https://aclanthology.org/2026.findings-acl.152/.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. 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)*, pp. 1601–1611, Vancouver, Canada, 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1147. URL https://aclanthology.org/P17-1147/.

Haodong Lei, Junming Liu, Yirong Chen, Pinlong Cai, Botian Shi, Ding Wang, and Hongsong Wang. TransMem: Transforming hidden states into memory for large language models. *arXiv preprint arXiv:2607.29032*, 2026. doi: 10.48550/arXiv.2607.29032. URL https://arxiv.org/abs/2607.29032v1.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative agents for "mind" exploration of large language model society. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/a3621ee907def47c1b952ade25c67698-Abstract-Conference.html.

Xiaoze Liu, Ruowang Zhang, Weichen Yu, Siheng Xiong, Liu He, Feijie Wu, Hoin Jung, Matt Fredrikson, Xiaoqian Wang, and Jing Gao. The vision wormhole: Latent-space communication in heterogeneous multi-agent systems. *arXiv preprint arXiv:2602.15382*, 2026. doi: 10.48550/arXiv.2602.15382. URL https://arxiv.org/abs/2602.15382v2.

Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. AgentBoard: An analytical evaluation board of multi-turn LLM agents. In *Advances in Neural Information Processing Systems*, volume 37, pp. 74325–74362, 2024. doi: 10.52202/079017-2365. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/877b40688e330a0e2a3fc24084208dfa-Abstract-Datasets_and_Benchmarks_Track.html.

Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. 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)*, pp. 9802–9822. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.acl-long.546. URL https://aclanthology.org/2023.acl-long.546/.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*. Association for Computing Machinery, 2023. doi: 10.1145/3586183.3606763. URL https://doi.org/10.1145/3586183.3606763.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. ChatDev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15174–15186. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.810. URL https://aclanthology.org/2024.acl-long.810/.

Chen Qian, Zihao Xie, YiFei Wang, Wei Liu, Kunlun Zhu, Hanchen Xia, Yufan Dang, Zhuoyun Du, Weize Chen, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Scaling large language model-based multi-agent collaboration. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/66a026c0d17040889b50f0dfa650e5e0-Abstract-Conference.html.

Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Rakibul Hasan Rajib, Mengxin Zheng, and Qian Lou. Learning what to retain: Gated-memory routing for efficient collaboration in multi-agent LLM systems. *arXiv preprint arXiv:2609.00237*, 2026. doi: 10.48550/arXiv.2609.00237. URL https://arxiv.org/abs/2609.00237v2.

Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 3982–3992. Association for Computational Linguistics, 2019. doi: 10.18653/v1/D19-1410. URL https://aclanthology.org/D19-1410/.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. doi: 10.48550/arXiv.2402.03300. URL https://arxiv.org/abs/2402.03300.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-0377. URL https://papers.nips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024. URL https://openreview.net/forum?id=ehfRiF0R3a.

Liang Wang, Nan Yang, and Furu Wei. Query2doc: Query expansion with large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 9414–9423. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.585. URL https://aclanthology.org/2023.emnlp-main.585/.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and

Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversations. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=BAakY1hNKS.

Yige Xu, Xu Guo, Zhiwei Zeng, and Chunyan Miao. SoftCoT: Soft chain-of-thought for efficient reasoning with LLMs. *arXiv preprint arXiv:2502.12134*, 2025a. doi: 10.48550/arXiv.2502.12134. URL https://arxiv.org/abs/2502.12134v2.

Zhangchen Xu, Yang Liu, Yueqin Yin, Mingyuan Zhou, and Radha Poovendran. KodCode: A diverse, challenging, and verifiable synthetic dataset for coding. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 6980–7008, Vienna, Austria, 2025b. Association for Computational Linguistics. doi: 10.18653/v1/2025.findings-acl.365. URL https://aclanthology.org/2025.findings-acl.365/.

Fu-Chun Yang and Jason Eshraghian. Direct semantic communication between large language models via vector translation. *arXiv preprint arXiv:2511.03945*, 2025. doi: 10.48550/arXiv.2511.03945. URL https://arxiv.org/abs/2511.03945v1.

Wei Yang, Bryce Kan, Shixuan Li, Li Li, Yuehan Qin, Jiate Li, Paul Bogdan, and Jesse Thomason. RaMem: Contextual reinstatement for long-term agentic memory. *arXiv preprint arXiv:2606.22844*, 2026. doi: 10.48550/arXiv.2606.22844. URL https://arxiv.org/abs/2606.22844.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380, 2018. URL https://aclanthology.org/D18-1259/.

Da Yin, Faeze Brahman, Abhilasha Ravichander, Khyathi Chandu, Kai-Wei Chang, Yejin Choi, and Bill Yuchen Lin. Agent Lumos: Unified and modular training for open-source language agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12380–12403. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.670. URL https://aclanthology.org/2024.acl-long.670/.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, et al. DAPO: An open-source LLM reinforcement learning system at scale. In *Advances in Neural Information Processing Systems*, volume 38, pp. 113222–113244, 2025. doi: 10.52202/085713-3775. URL https://papers.nips.cc/paper_files/paper/2025/hash/a4277440d50f1f15d2cb4c14f7e0c0d2-Abstract-Conference.html.

Xinlei Yu, Chengming Xu, Zhangquan Chen, Bo Yin, Cheng Yang, Yongbo He, Yihao Hu, Jiangning Zhang, Cheng Tan, Xiaobin Hu, and Shuicheng Yan. Dual latent memory for visual multi-agent system. *arXiv preprint arXiv:2602.00471*, 2026. doi: 10.48550/arXiv.2602.00471. URL https://arxiv.org/abs/2602.00471.

Sizhe Yuen, Francisco Gomez Medina, Ting Su, Yali Du, and Adam J. Sobey. Intrinsic memory agents: Heterogeneous multi-agent LLM systems through structured contextual memory. *arXiv preprint arXiv:2508.08997*, 2025. doi: 10.48550/arXiv.2508.08997. URL https://arxiv.org/abs/2508.08997v2.

Guibin Zhang, Muxin Fu, Kun Wang, Guancheng Wan, Miao Yu, and Shuicheng Yan. G-Memory: Tracing hierarchical memory for multi-agent systems. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://papers.nips.cc/paper_files/paper/2025/hash/136a45cd9b841bf785625709a19c6508-Abstract-Conference.html.

Guibin Zhang, Muxin Fu, and Shuicheng Yan. MemGen: Weaving generative latent memory for self-evolving agents. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/26407588dfb08e48c459c074ab6adb7d-Abstract-Conference.html.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024. doi: 10.1609/aaai.v38i17.29936. URL https://ojs.aaai.org/index.php/AAAI/article/view/29936.

Chengfeng Zhao, Yuqiao Tan, Shizhu He, Yequan Wang, Jun Zhao, and Kang Liu. Neural procedural memory: Empowering LLM agents with implicit activation steering. *arXiv preprint arXiv:2606.29824*, 2026. URL https://arxiv.org/abs/2606.29824.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. MemoryBank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19724–19731, 2024. doi: 10.1609/aaai.v38i17.29946. URL https://ojs.aaai.org/index.php/AAAI/article/view/29946.

Zhengze Zhou and Hejian Sang. LatentPress: Context compression beyond text and vision. *arXiv preprint arXiv:2609.01507*, 2026. doi: 10.48550/arXiv.2609.01507. URL https://arxiv.org/abs/2609.01507v2.

He Zhu, Tianrui Qin, King Zhu, Heyuan Huang, Yeyi Guan, Jinxiang Xia, Yi Yao, Hanhao Li, Ningning Wang, Pai Liu, Tianhao Peng, Xin Gui, Xiaowan Li, Yuhui Liu, Yuchen Eleanor Jiang, Jun Wang, Changwang Zhang, Xiangru Tang, Ge Zhang, Jian Yang, Minghao Liu, Xitong Gao, Jiaheng Liu, and Wangchunshu Zhou. OAgents: An empirical study of building effective agents. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 13354–13369, Suzhou, China, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.findings-emnlp.720. URL https://aclanthology.org/2025.findings-emnlp.720/.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, Simon Brunner, Chen Gong, James Hoang, Armel Zebaze, Xiaoheng Hong, Wen-Ding Li, Jean Kaddour, Ming Xu, Zhihan Zhang, Prateek Yadav, Naman Jain, Alex Gu, Zhoujun Cheng, Jiawei Liu, Qian Liu, Zijian Wang, Binyuan Hui, Niklas Muennighoff, David Lo, Daniel Fried, Xiaoning Du, Harm de Vries, and Leandro Von Werra. BigCodeBench: Benchmarking code generation with diverse function calls and complex instructions. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/a6a90bcc2aa470c3871b2d39a67d26e8-Abstract-Conference.html.

Jiaru Zou, Ruizhong Qiu, Gaotang Li, Xiyuan Yang, Katherine Tieu, Pan Lu, Ke Shen, Hanghang Tong, Yejin Choi, Jingrui He, James Zou, Mengdi Wang, and Ling Yang. Latent Collaboration in Multi-Agent Systems. *arXiv preprint arXiv:2511.20639*, 2025. doi: 10.48550/arXiv.2511.20639. URL https://arxiv.org/abs/2511.20639v4.

Jiaru Zou, Rui Pan, Ruizhong Qiu, Pan Lu, Shizhe Diao, Jindong Jiang, Hanghang Tong, Tong Zhang, Markus J. Buehler, Jingrui He, and James Zou. Recursive Multi-Agent Systems. *arXiv preprint arXiv:2604.25917*, 2026. doi: 10.48550/arXiv.2604.25917. URL https://arxiv.org/abs/2604.25917v2.

A Additional Related Work

Retrieval and query–memory alignment. Deciding which memory to reuse is itself a retrieval problem. Dense retrievers compare queries and memories in a shared embedding space (Reimers & Gurevych, 2019), and query expansion methods such as HyDE (Gao et al., 2023) and Query2Doc (Wang et al., 2023) narrow the gap between the two by moving queries toward document space. Agent memories further shape what is stored and when it is used: ExpeL (Zhao et al., 2024) distills higher-level insights from trajectories, Gated-Memory Routing (Rajib et al., 2026) learns gates that decide which intermediate execution memory to write and retrieve, and the contextual-reinstatement memory RaMem (Yang et al., 2026) anchors memory fragments to their episodic conditions. Closest to our retrieval, QueryLink (Hu et al., 2026) aligns queries and memories in a shared multi-grained space. RAMEM adopts this query–memory alignment view and adds the target role as a second signal: whereas LatentMem (Fu et al., 2026) ranks trajectories by task similarity alone, RAMEM selects complete trajectories with both task and role evidence and then loads the target role's archived hidden states from them.

B Experimental Details

Table 4: **Training hyperparameters.** Alignment operates on supervised invocations; HMAPO samples groups of complete trajectories. Learning rates refer to the indicated parameter groups.

Setting	Alignment	HMAPO
Optimizer	AdamW	AdamW
Learning-rate schedule	Cosine	Cosine
Warmup fraction	0.1	0.1
Weight decay	0.01	0.01
Maximum gradient norm	1.0	1.0
Epochs	2	1
Training steps	2,180	488
Input-projection learning rate	10^{-4}	10^{-5}
Adapter / output-projection / query learning rate	Not updated	10^{-5}
Supervised batch size per device	1 invocation	—
Gradient accumulation (alignment)	8 steps	—
Effective rollout groups per device	—	1
Trajectories per group G	—	8
Invocations per gradient microbatch	—	2
Updates per rollout batch	—	1
Memory length L	8	8
Hidden-state dimension	2,560	2,560
Composer context limit	8,192	8,192
Agent-backbone context limit	8,192	14,336

B.1 Training Configuration

Initialization and trainable parameters. The composer is initialized from the trained LatentMem composer, whose backbone is `Qwen3-4B-Instruct-2507` (Qwen Team, 2025); its LoRA adapters, query vectors, and output projection are inherited, and only the hidden-state input projection is new. During teacher-guided interface alignment, only this input projection is trained. The inherited composer weights and the agent backbone remain frozen, and the same LatentMem composer reading the source responses as text serves as the teacher. HMAPO continues from the aligned input projection, merges the inherited LoRA adapters into the backbone, and trains new composer LoRA adapters, both projections, and the query vectors.

Data and optimization. Training uses experience from AutoGen and MacNet on KodCode, TriviaQA, and PopQA. Teacher-guided interface alignment uses 69,732 training invocations and 2,014 validation invocations. Its objective combines response cross-entropy with teacher-to-student KL divergence, using distillation weight $\beta = 1$ and temperature $T = 1$. The teacher truncates its text

input to 2,048 tokens. The HMAPO task bank contains 3,909 training and 156 development tasks. Table 4 lists the optimization settings.

HMAPO rollout and regularization. HMAPO keeps the retrieved hidden-state references of each task and role fixed and recomposes the latent memory with the current interface parameters. It runs on eight GPUs and samples responses with temperature 1.0, top-$p = 1.0$, and at most 4,096 new tokens per invocation. Code rewards use partial-test weight $\eta = 0.2$. The policy clipping threshold is $\epsilon = 0.2$. The reference-KL coefficient starts at $\mu = 0.02$, targets a mean KL of 0.03, and adapts by a factor of 1.5 within $[0.002, 0.5]$. Difficulty-stratified sampling assigns weights 0.5, 0.3, 0.1, and 0.1 to mixed-outcome, unseen, consistently failed, and consistently solved tasks, respectively, with up to two resampling rounds for effective groups. Composer LoRA (Hu et al., 2022) adapters use rank 16, scaling parameter 32, zero dropout, and the `q_proj` and `v_proj` modules. HMAPO updates these adapters, the input and output projections, and the query vectors while keeping the composer's base weights and the agent backbone frozen.

B.2 BENCHMARKS

We evaluate on six benchmarks spanning single-hop and multi-hop question answering, code generation, and planning. TriviaQA, PopQA, and KodCode are the source domains: the initial memory is built from trajectories on their training splits. HotpotQA, BigCodeBench, and PDDL are transfer tasks whose trajectories enter memory only through online writeback during evaluation. All scores are on a 0–100 scale.

- **TriviaQA** (Joshi et al., 2017) consists of trivia questions authored by enthusiasts, with answers given as sets of accepted aliases. We report accuracy against these aliases.
- **PopQA** (Mallen et al., 2023) is an entity-centric open-domain question answering set generated from Wikidata triples and spanning a long tail of entity popularity, which makes it sensitive to whether a system can recall facts about rarely mentioned subjects. We report accuracy.
- **KodCode** (Xu et al., 2025b) is a synthetic coding benchmark in which every task is paired with a verified reference solution and a unit-test suite. A task is counted as solved when the submitted function passes every test.
- **HotpotQA** (Yang et al., 2018) contains Wikipedia-based multi-hop questions whose answers require combining evidence from two documents. We use the fullwiki setting, in which supporting documents must be retrieved from all of Wikipedia, evaluate on 500 questions randomly sampled from the development set, and report accuracy.
- **BigCodeBench** (Zhuo et al., 2025) contains Python tasks whose solutions must call functions from a wide range of libraries under detailed natural-language instructions. Candidates are executed against the official test suites in Docker and we report task-level pass@1.
- **PDDL** (Ma et al., 2024) presents classical planning domains in which the agent must produce a sequence of PDDL actions that reaches a goal state. We take the environment from AgentBoard, whose progress-rate metric we use as the episode reward. The reward of an episode is the largest fraction of goal literals satisfied at any step of execution and we report the mean over episodes.

B.3 BASELINES

Every baseline runs the same multi-agent workflow and the same backbone as RAMEM and differs only in the memory attached to the agents. Memories designed for a single agent are attached to each agent separately, indexed by that agent's own visible context. On TriviaQA, KodCode, PopQA, and PDDL, we follow the evaluation protocol of LatentMem. BigCodeBench and HotpotQA are run for all methods with our configuration.

- **No memory** runs the workflow without access to any past episode and is the reference for the superscripts in Table 1.

- **MetaGPT** (Hong et al., 2024) organizes agents around a shared message pool: within a task, agents publish structured messages that other roles subscribe to, and across tasks a long-term store retains feedback that later runs can consult.
- **Generative** (Park et al., 2023) keeps a memory stream of observations and periodically synthesizes higher-level reflections from them. Retrieval ranks entries by recency, importance, and relevance to the current situation.
- **Voyager** (Wang et al., 2024) maintains a library of skills stored as executable programs, each indexed by a natural-language description of what it does. Skills whose descriptions are similar to the current task are retrieved and offered to the agent.
- **G-Memory** (Zhang et al., 2025) stores multi-agent history in three linked graphs: a query graph of past tasks, an insight graph of distilled lessons, and an interaction graph of condensed trajectories. A new task is matched in the query graph and the traversal then collects both the insights above it and the interactions below it, which are customized for each role before use.
- **OAgents** (Zhu et al., 2025) separates a short-term memory holding the current task context from a long-term memory of summarized past runs. Long-term entries are embedded and retrieved by similarity, allowing the agent to see a compact summary rather than a raw history.
- **LatentMem** (Fu et al., 2026) retrieves whole trajectories from an experience bank by task similarity and passes them, together with the agent's role profile, through a composer trained with Latent Memory Policy Optimization to produce fixed-length latent tokens.

B.4 Multi-Agent Architectures

All agents in all architectures run `Qwen3-4B-Instruct-2507` (Qwen Team, 2025). The architectures differ in how many agents take part, which functional roles they play, and how messages flow between them.

- **AutoGen** (Wu et al., 2024) is instantiated with two agents, a user proxy and an assistant, that exchange messages in a fixed two-party conversation until the proxy accepts an answer.
- **MacNet** (Qian et al., 2025) arranges five agents on a directed acyclic graph and passes each agent's output along the graph's edges to its successors. The roles are actor, critic, and summarizer, and the same functional role can appear at more than one node.
- **CAMEL** (Li et al., 2023) runs four agents in a role-playing dialogue: a user proxy that states and refines the task, an actor and a critic that carry out and review the work over multiple turns, and a summarizer that produces the final answer.

Memory construction and the unseen architecture. The initial shared experience bank is collected from successful AutoGen and MacNet runs on the training splits of the three source benchmarks, giving 17,216 trajectories. CAMEL contributes no trajectories to this memory and is evaluated as an unseen architecture. Because agent names are normalized to functional roles before records are built, a CAMEL agent can still retrieve experience written by other architectures: with no CAMEL trajectories in the initial memory to prefer, its user proxy draws on AutoGen trajectories, and its actor, critic, and summarizer draw on MacNet trajectories where the same functional roles occur.

C Additional Results

Composition of the experience bank. The shared experience bank $\mathcal{D}$ initially contains 17,216 successful trajectories, 9,755 from AutoGen and 7,461 from MacNet, and Figure 5 shows how they are distributed over the source benchmarks.

Training-stage ablation. Table 5 compares teacher-guided interface alignment with the complete training procedure after adding HMAPO. Both stages use task-similarity retrieval without role alignment, matching the *Role hidden states* configuration in Table 2; the agent backbone and hidden-

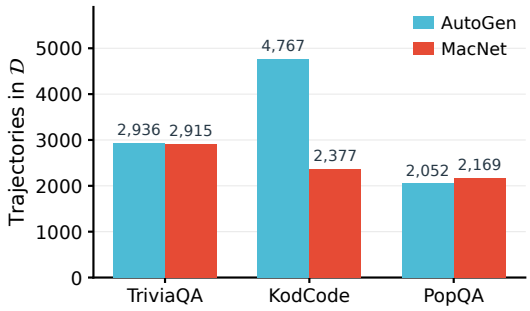

Figure 5: **Composition of the shared experience bank.** Successful trajectories in $\mathcal{D}$ by architecture and source benchmark.

memory interface are otherwise unchanged. HMAPO improves every entry in both architectures: TriviaQA and PopQA by 4.3 to 11.3 points, KodCode by 0.6 to 1.4, and PDDL reward by 0.6 to 0.8.

Table 5: **Training-stage ablation.** "Teacher-guided alignment" uses response supervision together with teacher distillation; "+ HMAPO" continues from that interface with the second-stage GRPO variant. QA and code entries are accuracy (percent); PDDL entries are average reward multiplied by 100.

MAS	Training	TriviaQA	KodCode	PopQA	PDDL
AutoGen	Teacher-guided alignment	74.49	76.25	52.26	29.70
	+ HMAPO	80.57	76.85	56.56	30.53
MacNet	Teacher-guided alignment	67.70	76.05	47.90	29.10
	+ HMAPO	78.98	77.45	54.59	29.71

Reward optimization on the text composer. To test whether the gains come from HMAPO rather than from hidden-state input, we also train LatentMem's text composer with HMAPO, keeping its task-similarity retrieval of complete trajectories. Table 6 compares it with LatentMem, the *Role hidden states* configuration of Table 2, and RAMEM. Because LatentMem's composer was already trained with LMPO (Fu et al., 2026) on the same source datasets, HMAPO changes its average by only -1.0 to $+0.4$ points. RAMEM remains 4.0 to 7.3 points above the HMAPO-trained text composer in every architecture, and hidden states without role-aligned retrieval already exceed it by 3.4 and 6.9 points on AutoGen and MacNet.

Table 6: **HMAPO on the text composer.** *LatentMem + HMAPO* trains LatentMem's text composer with HMAPO; *Role hidden states* and RAMEM are the configurations of Table 2. Avg. is the mean over the four benchmarks; bold marks the best result within each architecture.

MAS	Configuration	TriviaQA	KodCode	PopQA	PDDL	Avg.
AutoGen	LatentMem	76.51	76.80	52.70	23.49	57.38
	LatentMem + HMAPO	76.45	77.15	52.57	24.96	57.78
	Role hidden states	**80.57**	76.85	56.56	30.53	61.13
	RAMEM	80.52	**79.25**	**56.63**	**30.87**	**61.82**
MacNet	LatentMem	65.98	78.90	44.14	25.13	53.54
	LatentMem + HMAPO	65.47	**79.55**	45.41	22.82	53.31
	Role hidden states	78.98	77.45	54.59	29.71	60.18
	RAMEM	**79.45**	78.05	**54.62**	**30.23**	**60.59**
CAMEL	LatentMem	68.74	77.75	47.23	28.12	55.46
	LatentMem + HMAPO	68.40	**79.75**	46.91	22.83	54.47
	Role hidden states	66.51	73.50	50.16	28.01	54.55
	RAMEM	**79.94**	**79.75**	**54.66**	**30.56**	**61.23**

Number of retrieved trajectories. Figure 6 varies the number of retrieved trajectories K on the first 300 tasks of the three pairings, the same tasks as in Figure 4, enlarging the shortlist when $K > n_s$. Increasing K does not yield consistent improvements across the three pairings. AutoGen is unchanged, CAMEL loses 1.3 to 1.7 points beyond $K = 1$, and MacNet gains 2.3 points at $K = 2$ before falling slightly below its $K = 1$ level at $K = 8$. Each added trajectory contributes more hidden-state blocks of the target role, but the composer still emits eight latent tokens, so the blocks share a fixed budget and the reranker's single best trajectory likely already carries what the role can use. We therefore keep $K = 1$ throughout.

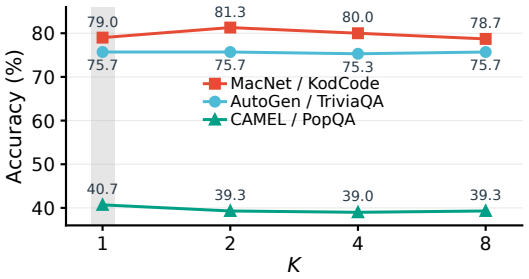

Figure 6: **Sensitivity to the number of retrieved trajectories.** The band marks the default.

Retrieval components. Table 7 removes the components of role-aligned retrieval from RAMEM one at a time on CAMEL with TriviaQA, using 300 randomly sampled tasks. Ranking candidates by similarity alone, without first preferring trajectories that contain the target role and come from the target architecture, lowers accuracy by 3.00 points. Removing the LLM reranker lowers it by 1.66 points, and retrieving with the task query alone, without the role-prefixed query, lowers it by 1.33 points. As Table 2 shows, removing role alignment altogether costs 13.4 points, so the components partly compensate for one another when only one is absent.

Table 7: **Retrieval ablation on CAMEL with TriviaQA.** Each row removes one component from RAMEM; Δ is the change from full retrieval.

Retrieval	Accuracy	Δ
RAMEM (full retrieval)	**80.33**	–
w/o LLM reranker	78.67	−1.66
w/o role and architecture priority	77.33	−3.00
w/o role-prefixed query	79.00	−1.33

Number of latent tokens. Figure 7 varies the number of latent tokens L on 300 randomly sampled tasks of AutoGen with KodCode and MacNet with TriviaQA. Reducing L from 8 to 4 lowers accuracy by 16.34 and 2.67 points, and raising it to 16 increases accuracy by 1.66 and 5.00 points. We keep $L = 8$, the latent length of LatentMem.

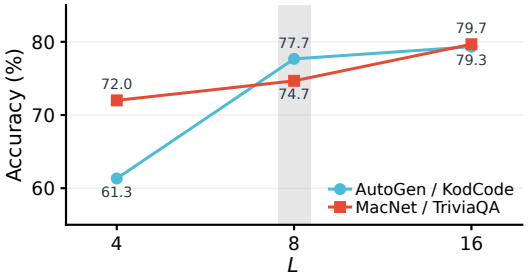

Figure 7: **Sensitivity to the number of latent tokens.** The band marks the default.

Online writeback. During evaluation, the successful episodes of each batch enter memory after the batch finishes and are retrievable from the next batch on. Their hidden states are captured during decoding, so a written-back block lacks the state after the final token, which the archive built by replaying responses includes. Figure 8 disables this writeback on CAMEL, which contributes no trajectories to the initial memory, using the last 300 tasks of TriviaQA and PopQA, where the memory accumulated during evaluation is largest. Without writeback, RAMEM draws only on experience from AutoGen and MacNet and still improves over No memory by 3.7 and 3.0 points. Writeback, which makes CAMEL's own successful episodes retrievable, adds a further 3.7 and 9.3 points.

Latent projections on further pairings. Figure 9 extends the comparison in Figure 3 to AutoGen on TriviaQA and MacNet on PopQA, using the same projection procedure. In AutoGen, LatentMem interleaves the assistant and user proxy, while RAMEM produces more distinct local role regions

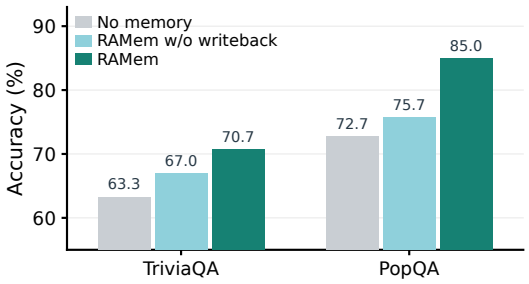

Figure 8: **Online writeback on CAMEL.** Accuracy on the last 300 tasks of each benchmark.

with overlap remaining. On MacNet/PopQA, RAMEM organizes the actor, critic, and summarizer into broader role-specific regions compared with the dispersed clusters under LatentMem.

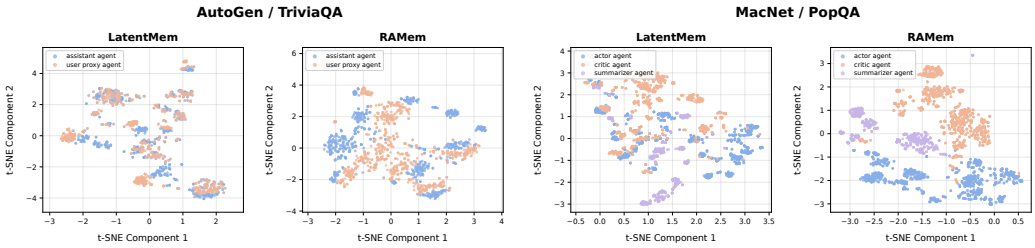

Figure 9: **Role structure on additional architecture–benchmark pairings.** AutoGen on TriviaQA (left) and MacNet on PopQA (right). Each point is one agent invocation, colored by role. Both halves of each pairing share one projection.

Latent projections under other perplexities. Figure 3 uses t-SNE with perplexity 30. Figure 10 repeats the projection for the two multi-role architectures with perplexity 10 and 50, computed as in the main text: the composer outputs of both pipelines on the same 300 tasks are standardized, reduced to 50 principal components, and embedded jointly so the two method panels in each cell share one projection. The qualitative pattern is the same at both settings. At both perplexities LatentMem leaves CAMEL's roles interleaved and splits each MacNet role into several clusters, while under RAMEM the roles occupy largely separate regions.

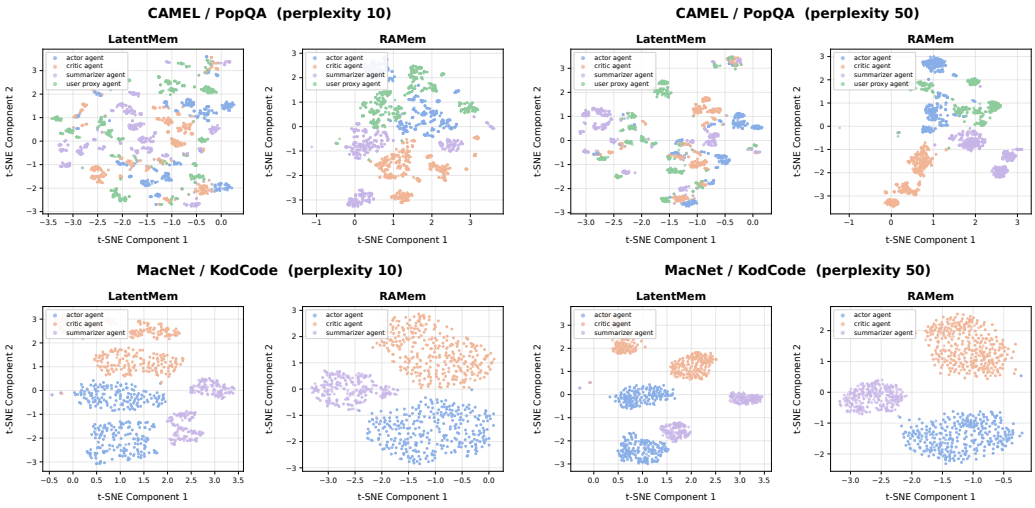

Figure 10: **Latent projections at different perplexities.** Rows show CAMEL on PopQA (top) and MacNet on KodCode (bottom). Columns use perplexity 10 (left) and 50 (right). Each cell compares LatentMem and RAMEM in a shared projection, with agent invocations colored by role.

Results with a Qwen3-1.7B agent backbone. To test whether RAMEM carries over to another backbone, we repeat the evaluation with `Qwen3-1.7B` as the backbone of every agent, keeping the memory construction and training data of the main experiments. The composer is also initialized from `Qwen3-1.7B` instead of the LatentMem composer, so the first training stage updates the entire composer rather than only the input projection; it still encodes the selected hidden states into eight latent tokens. The text-memory baselines run the same workflows with the composer disabled and retrieve only textual memories. Table 8 reports the results. With this backbone, text memories change the five-benchmark average by at most 1.12 points relative to No memory, whereas RAMEM attains the best average in every architecture, 9.84 to 16.84 points above No memory. Its largest gain is on PopQA, where it reaches 34.6 to 38.8 accuracy while the text baselines reach at most 4.4 and No memory 1.0 to 1.2.

Table 8: **Results with a Qwen3-1.7B agent backbone.** All methods are evaluated on the first 500 tasks of each benchmark. Superscripts show absolute changes from No memory within the same architecture. Avg. is the five-benchmark mean. Bold marks the best result, including ties.

Method	Source-domain tasks			Transfer tasks		Avg.
	TriviaQA Acc.	KodCode Acc.	PopQA Acc.	HotpotQA Acc.	BigCodeBench pass@1	
AutoGen						SEEN MAS
No memory	35.8	50.8	1.0	12.6	21.4	24.32
MetaGPT	37.8$^{+2.0}$	45.2$^{-5.6}$	4.4$^{+3.4}$	11.2$^{-1.4}$	20.0$^{-1.4}$	23.72$^{-0.60}$
Generative	35.8$^{0.0}$	50.8$^{0.0}$	1.0$^{0.0}$	12.2$^{-0.4}$	21.4$^{0.0}$	24.24$^{-0.08}$
Voyager	38.4$^{+2.6}$	46.4$^{-4.4}$	4.4$^{+3.4}$	11.8$^{-0.8}$	17.2$^{-4.2}$	23.64$^{-0.68}$
G-Memory	36.6$^{+0.8}$	47.2$^{-3.6}$	3.6$^{+2.6}$	11.0$^{-1.6}$	20.4$^{-1.0}$	23.76$^{-0.56}$
OAgents	38.4$^{+2.6}$	45.6$^{-5.2}$	4.4$^{+3.4}$	11.0$^{-1.6}$	19.8$^{-1.6}$	23.84$^{-0.48}$
RAMEM	**66.4**$^{+30.6}$	**56.8**$^{+6.0}$	**35.4**$^{+34.4}$	**19.4**$^{+6.8}$	**27.8**$^{+6.4}$	**41.16**$^{+16.84}$
MacNet						SEEN MAS
No memory	36.0	54.0	1.0	11.4	26.2	25.72
MetaGPT	36.4$^{+0.4}$	54.4$^{+0.4}$	4.2$^{+3.2}$	10.4$^{-1.0}$	22.4$^{-3.8}$	25.56$^{-0.16}$
Generative	35.8$^{-0.2}$	**59.0**$^{+5.0}$	1.0$^{0.0}$	11.8$^{+0.4}$	26.0$^{-0.2}$	26.72$^{+1.00}$
Voyager	35.6$^{-0.4}$	53.0$^{-1.0}$	4.4$^{+3.4}$	10.4$^{-1.0}$	19.6$^{-6.6}$	24.60$^{-1.12}$
G-Memory	35.2$^{-0.8}$	55.0$^{+1.0}$	3.6$^{+2.6}$	10.0$^{-1.4}$	20.8$^{-5.4}$	24.92$^{-0.80}$
OAgents	36.2$^{+0.2}$	54.2$^{+0.2}$	4.2$^{+3.2}$	10.4$^{-1.0}$	22.6$^{-3.6}$	25.52$^{-0.20}$
RAMEM	**43.2**$^{+7.2}$	58.0$^{+4.0}$	**38.8**$^{+37.8}$	**12.6**$^{+1.2}$	**26.6**$^{+0.4}$	**35.84**$^{+10.12}$
CAMEL						UNSEEN MAS
No memory	34.2	56.4	1.2	12.4	25.2	25.88
MetaGPT	35.8$^{+1.6}$	55.8$^{-0.6}$	4.2$^{+3.0}$	11.0$^{-1.4}$	23.2$^{-2.0}$	26.00$^{+0.12}$
Generative	34.2$^{0.0}$	57.4$^{+1.0}$	1.2$^{0.0}$	12.4$^{0.0}$	25.2$^{0.0}$	26.08$^{+0.20}$
Voyager	37.0$^{+2.8}$	56.6$^{+0.2}$	4.4$^{+3.2}$	11.6$^{-0.8}$	23.8$^{-1.4}$	26.68$^{+0.80}$
G-Memory	34.4$^{+0.2}$	56.0$^{-0.4}$	4.0$^{+2.8}$	10.2$^{-2.2}$	23.0$^{-2.2}$	25.52$^{-0.36}$
OAgents	36.4$^{+2.2}$	56.6$^{+0.2}$	4.2$^{+3.0}$	10.0$^{-2.4}$	22.8$^{-2.4}$	26.00$^{+0.12}$
RAMEM	**43.2**$^{+9.0}$	**59.4**$^{+3.0}$	**34.6**$^{+33.4}$	**15.0**$^{+2.6}$	**26.4**$^{+1.2}$	**35.72**$^{+9.84}$

Effect of memory on agent behavior. The two kinds of memory also change how the agents act. On PopQA, for example, the gap tracks whether the agents search for evidence on the current question. A textual memory retrieved for a 1.7B agent is often a long successful execution for a neighboring question, together with its search documents, and it contains a gold answer for only 3.4% to 3.8% of questions. The 1.7B agent nonetheless tends to treat it as sufficient context and searches on only 14.2% to 28.4% of questions. RAMEM does not insert the trajectory into the prompt. Its composer compresses the role's hidden states into eight latent tokens, and with this memory the agents search on 93.4% to 99.6% of questions, obtaining observations that contain a gold answer for 43.2% to 47.2% of them. These results suggest that the composer conveys procedural guidance rather than answer content: the composed memory steers the 1.7B agents to search the current question, consistent with Cases 2 and 3 in Appendix D, where the retrieved trajectories share a procedure with the current task rather than its answer.

Composer size. Table 9 replaces the Qwen3-4B composer backbone with Qwen3-1.7B, keeping the Qwen3-4B agent backbone and all memory stores. The composer is initialized from

`Qwen3-1.7B`, so the first training stage updates the entire composer; the second stage is unchanged. With the smaller composer, RAMEM keeps 74% and 68% of its gain over No memory on AutoGen and MacNet, where its average matches or slightly exceeds that of LatentMem with a 4B composer, and 51% on the unseen CAMEL.

Table 9: **Composer size.** RAMEM with a Qwen3-4B or Qwen3-1.7B composer backbone; the agent backbone is Qwen3-4B in all rows, and LatentMem uses a 4B composer. Superscripts show absolute changes from No memory within the same architecture. Avg. is the six-benchmark mean. Bold marks the best result, including ties.

Method	Source-domain tasks			Transfer tasks			Avg.
	TriviaQA Acc.	KodCode Acc.	PopQA Acc.	HotpotQA Acc.	BigCodeBench pass@1	PDDL Reward	
AutoGen							SEEN MAS
No memory	60.31	68.40	38.78	36.60	39.21	16.39	43.28
LatentMem	76.51 +16.20	76.80 +8.40	52.70 +13.92	**40.40** +3.80	41.40 +2.19	23.49 +7.10	51.88 +8.60
RAMEM 4B	**80.52** +20.21	**79.25** +10.85	**56.63** +17.85	40.20 +3.60	**42.54** +3.33	**30.87** +14.48	**55.00** +11.72
RAMEM 1.7B	77.20 +16.89	77.20 +8.80	53.60 +14.82	33.60 −3.00	39.90 +0.69	30.09 +13.70	51.93 +8.65
MacNet							SEEN MAS
No memory	53.77	70.40	24.89	18.20	42.81	20.73	38.47
LatentMem	65.98 +12.21	**78.90** +8.50	44.14 +19.25	35.60 +17.40	41.84 −0.97	25.13 +4.40	48.60 +10.13
RAMEM 4B	**79.45** +25.68	78.05 +7.65	**54.62** +29.73	**40.60** +22.40	**43.95** +1.14	**30.23** +9.50	**54.48** +16.01
RAMEM 1.7B	66.90 +13.13	77.50 +7.10	44.60 +19.71	33.40 +15.20	43.80 +0.99	30.00 +9.27	49.37 +10.90
CAMEL							UNSEEN MAS
No memory	56.96	70.70	32.38	18.00	42.46	22.10	40.43
LatentMem	68.74 +11.78	77.75 +7.05	47.23 +14.85	38.00 +20.00	42.63 +0.17	28.12 +6.02	50.41 +9.98
RAMEM 4B	**79.94** +22.98	**79.75** +9.05	**54.66** +22.28	**40.00** +22.00	42.46 0.00	**30.56** +8.46	**54.56** +14.13
RAMEM 1.7B	65.60 +8.64	76.80 +6.10	45.20 +12.82	28.00 +10.00	**43.00** +0.54	27.39 +5.29	47.66 +7.23

D QUALITATIVE CASES

Each case compares LatentMem and RAMEM on the same task and architecture. Archived responses are printed as text for readability; RAMEM's composer reads their hidden states, whereas LatentMem's composer reads the text of the whole retrieved trajectory.

Case 1. Coding task with the same retrieved trajectory

Task (MacNet / KodCode, task 250).

```
Given two strings s1 and s2, find the minimum number of steps
required to make s1 and s2 anagrams of each other. In each step,
you can replace any character in s1 with any other character. [...]
Example 2: s1 = "xabcdefx", s2 = "zabcdefg", Output: 2.
Complete minStepsToAnagram(s1, s2). Expected time O(N), auxiliary
space O(1). Both strings consist of lowercase English letters.
```

LatentMem.

```
Retrieved trajectory: "Anagram Verification" (is_anagram_optimized),
nearest by task text, reranker 10/10. The text of the whole trajectory
is composed for every role.
Actor 1 output:
def minStepsToAnagram(s1: str, s2: str) -> int:
    freq1 = {}
    freq2 = {}
    for char in s1:
        freq1[char] = freq1.get(char, 0) + 1
    for char in s2:
        freq2[char] = freq2.get(char, 0) + 1
    total_steps = 0
    all_chars = set(freq1.keys()) | set(freq2.keys())
```

Case 1. Coding task with the same retrieved trajectory (continued)

```
    for char in all_chars:
        count1 = freq1.get(char, 0)
        count2 = freq2.get(char, 0)
        diff = abs(count1 - count2)
        total_steps += diff // 2
    return total_steps
Critic 1: The solution is correct in logic: it computes the minimum
    steps by comparing character frequencies and summing half the
    absolute differences. However, it violates the expected auxiliary
    space constraint of O(1), as it uses dictionaries that can grow
    with the number of unique characters (up to 26, which is
    constant) so space is actually O(1). The time complexity is O(N),
    which meets the requirement. Agree.
Summarizer keeps the per-character halving. Outcome: 2/8 tests;
("abc", "def") returns 0, expected 3.

RAMem.
Selected trajectory: the same "Anagram Verification" (task 0.76, role
0.57, reranker 10/10). Only the hidden states of the actor's archived
response are composed. The archived response (shown as text):
def is_anagram_optimized(s: str, t: str) -> bool:
    if len(s) != len(t):
        return False
    freq = [0] * 26
    for char in s:
        freq[ord(char) - ord('a')] += 1
    for char in t:
        freq[ord(char) - ord('a')] -= 1
        if freq[ord(char) - ord('a')] < 0:
            return False
    return all(count == 0 for count in freq)
Actor 1 output:
def minStepsToAnagram(s1, s2):
    freq1 = [0] * 26
    freq2 = [0] * 26
    for char in s1:
        freq1[ord(char) - ord('a')] += 1
    for char in s2:
        freq2[ord(char) - ord('a')] += 1
    steps = 0
    for i in range(26):
        steps += abs(freq1[i] - freq2[i])
    return steps // 2
Critic 1: The code is correct and efficiently solves the problem. It
    counts character frequencies, computes the total mismatch, and
    divides by 2 since each step fixes two mismatches. The time
    complexity is O(N) and space is O(1), meeting the requirements.
    Agree.
Summarizer keeps the total halving. Outcome: 8/8 tests.
```

Case 2. Cross-architecture reuse on an ambiguous title (unseen architecture)

```
Task (CAMEL / PopQA, task 28).
Who was the director of The Clock?   Gold: Vincente Minnelli
"The Clock" names both a 1945 film (Vincente Minnelli) and a 2010
video installation (Christian Marclay); the gold answer is the film.
CAMEL contributed no trajectories or records to the initial memory.
All retrieved experience comes from AutoGen (user proxy) and MacNet
(actor, critic, summarizer) collaborations.
```

Case 2. Cross-architecture reuse on an ambiguous title (unseen architecture) (continued)

LatentMem.
Round 1 retrieves a trajectory rated 2/10 and every role emits a search. Round 2 retrieves one rated 10/10.
Critic (round 1): The actor agent's response is incomplete and unhelpful. It only contains a search query rather than an actual answer. The director of *The Clock* (2010) is Christopher Nolan. The response fails to provide the correct information.
Summarizer (round 2): <answer> Christian Marclay</answer>
Outcome: wrong (the 2010 video installation).

RAMem.
Each role composes the hidden states of its own archived responses, retrieved from other architectures. User proxy: "Who was the director of Time After Time?" from an AutoGen trajectory (reranker 2/10). Actor and summarizer: "Who was the director of The Room?" from a MacNet trajectory. Critic: "Who was the director of Time After Time?".
The retrieved tasks concern other films and do not contain the answer. What they share is a procedure: resolve a title to a specific dated work.
User-proxy archived response (shown as text):
<search>director of Time After Time</search>
[step=1] <answer> The director of Time After Time (1979 film) is Nicholas Meyer. </answer>
Round 1: every role emits <search> director of The Clock </search>.
Round 2, all four roles: <answer> The director of *The Clock* (1945 film) was Vincente Minnelli. </answer>
Outcome: correct. The answer names the dated work, as the user-proxy response does, while LatentMem's summarizer resolves the title to the 2010 installation.

Case 3. Verifying before answering

Task (AutoGen / TriviaQA, task 30).
David Lloyd George was British Prime Minister during the reign of which monarch? Gold: George V

LatentMem.
Round 1 (reranker 5/10): both roles search. Round 2 (reranker 8/10):
Assistant: <search>David Lloyd George Prime Minister during which monarch's reign</search>
User proxy: <answer>Queen Victoria</answer>
Outcome: wrong.

RAMem.
Round 1 source task (both roles): "Who was the first British Prime Minister?" (reranker 4/10). Rounds 2 and 3: "Who was Britain's first ever Labour prime minister in 1924?" (reranker 8 and 9/10). The retrieved facts are unrelated to the monarch; the archived assistant response instead checks a candidate answer against the date given in the question.
Assistant archived response, rounds 2 and 3 (shown as text):
The question asks who was Britain's first ever Labour Prime Minister in 1924.

Clement Attlee became the Prime Minister of the United Kingdom in 1945, not 1924. In 1924, the first Labour Party leader to serve as Prime Minister was Ramsay MacDonald. He became the first Labour Prime Minister of the United Kingdom after leading the Labour Party to victory in the 1923 general election, though he was not officially appointed until 1924. [...]

Case 3. Verifying before answering (continued)

```
Round 1: both roles search.
Round 2, assistant: <search>David Lloyd George Prime Minister during
which monarch's reign</search>
Round 2, user proxy: <search>David Lloyd George Prime Minister during
which monarch's reign</search>
Round 3, assistant: <answer>King George V</answer>
Round 3, user proxy: <answer>King George V</answer>
Outcome: correct. In round 2 both roles search again, whereas
LatentMem's user proxy answers from its own recall.
```

Case 4. Exact role match on a related but different procedure

Task (MacNet / KodCode, task 209).

```
You are given an array `heights` representing the heights of students
in a class, where `heights[i]` is the height of the `i-th` student.
Implement the function `maxVisibleStudents(height)` that returns the
maximum number of students visible from the front of the class. A
student at index `i` is visible if there is no student taller than
them before position `i`. The function should output the number of
students who [...]
```

LatentMem.

```
Retrieved trajectory rated 5/10.
Actor 1 output:
def maxVisibleStudents(heights):
    if not heights:
        return 0

    visible = 0
    max_height = 0

    for height in heights:
        if height > max_height:
            visible += 1
            max_height = height

    return visible
Both critics: Agree. Outcome: 7/7 tests.
```

RAMem.

```
Selected trajectory: "You are given an array of integers `heights`
that represents the height of individuals standing in a line [...]"
(exact role match, task 0.65, role 0.43, reranker 3/10).
Actor archived response (shown as text):
def max_individuals(heights, k):
    ""Returns the maximum number of individuals that can be placed in
        the line
    such that no two consecutive individuals have a height difference
        greater than k.""
    if not heights:
        return 0

    heights.sort()
    count = 1  # First individual can always be placed

    for i in range(1, len(heights)):
        if heights[i] - heights[i-1] <= k: [...]
Actor 1 output:
def maxVisibleStudents(heights):
    if not heights:
```

Case 4. Exact role match on a related but different procedure (continued)

```
        return 0

    visible = 0
    n = len(heights)

    # Use a stack to keep track of visible students
    stack = []

    for height in heights:
        # While stack is not empty and current height is greater than
        # the height at the top of stack
        # Pop the stack as the current student will be visible and
        # block the previous one
        while stack and height > stack[-1]:
            stack.pop()
        stack.append(height)

    return [...]
Critic 1: The code is correct and efficiently solves the problem using
a stack to simulate visibility from the front. It ensures that only
students who are taller than all previous students are counted, which
m [...]
Outcome: 2/7 tests (only the empty and single-student cases pass).
```

E PROMPT TEMPLATES

This appendix lists the prompts used at each stage. Agent prompts are inherited from LatentMem's implementation of the three architectures and are identical for all memory methods. Each agent receives a system prompt and a user turn, and the user turn carries a `Retrieved Content` slot that holds serialized memory text for the text-based baselines. For LatentMem and RAMEM this slot holds the placeholder token `<|Memory_Empty|>`, and the L latent tokens are appended after the chat template instead. Placeholders in braces are filled at run time.

E.1 AGENT PROMPTS

AutoGen, question answering: Assistant

```
Answer the given question. You must conduct reasoning inside <think>
and </think> first every time you get new information.
After reasoning, if you find you lack some knowledge, you can call a
search engine by <search> query </search>.
You can search as many times as your want.
If you find no further external knowledge needed, you can directly
provide the answer inside <answer> and </answer>, without detailed
illustrations. For example, <answer> Beijing </answer>.

--- user turn ---
Below is the retrieved memory content and the current task. Please
provide your answer.

# Retrieved Content
{memory_content}

# Current Task
{task_description}
```

AutoGen, question answering: User Proxy

```
You will be given a question and an assistant's answer for that
question. Follow the procedure below and produce outputs accordingly:

Answer the given question. You must conduct reasoning inside <think>
and </think> first every time you get new information.
After reasoning, if you find you lack some knowledge, you can call a
search engine by <search> query </search>.
You can search as many times as your want.
If you find no further external knowledge needed, you can directly
provide the answer inside <answer> and </answer>, without detailed
illustrations. For example, <answer> Beijing </answer>.

Please consider the assistant agent's answer and provide your own
answer.

--- user turn ---
Below is the relevant content retrieved from memory, the answer
provided by the assistant, and the current task. Please provide your
answer.

# Retrieved Content
{memory_content}

# Assistant Answer
{assistant_output}

# Current Task
{task_description}
```

MacNet, question answering: Actor

```
Answer the given question. You must conduct reasoning inside <think>
and </think> first every time you get new information.
After reasoning, if you find you lack some knowledge, you can call a
search engine by <search> query </search>.
You can search as many times as your want.
If you find no further external knowledge needed, you can directly
provide the answer inside <answer> and </answer>, without detailed
illustrations. For example, <answer> Beijing </answer>.

--- user turn ---
Below is the relevant content retrieved from memory and the current
task requirements. Please provide your answer.

# Retrieved Content
{memory_content}

# Current Task
{task_description}
```

MacNet, question answering: Critic

```
You are a strategy evaluator. Your task is to review the response
provided by the actor agent for the current problem.

- If you believe the actor agent's response is correct and has no
  issues, reply only with: "Agree".
- If you believe the actor agent's response has issues, provide brief
```

MacNet, question answering: Critic (continued)

```
  and concise feedback only (keep your response short and within 3
  sentences).

--- user turn ---
Below are the relevant contents retrieved from memory, the response
given by the actor agent, and the requirements of the current task.
Please provide your review.

# Retrieved Content
{memory_content}

# Actor Output
{actor_output}

# Current Task
{task_description}
```

MacNet, question answering: Summarizer

```
You will be given a question, the responses produced by actor agents,
and the corresponding feedback from critic agents for that question.
Follow the procedure below and produce outputs accordingly:

Answer the given question. You must conduct reasoning inside <think>
and </think> first every time you get new information.
After reasoning, if you find you lack some knowledge, you can call a
search engine by <search> query </search>.
You can search as many times as your want.
If you find no further external knowledge needed, you can directly
provide the answer inside <answer> and </answer>, without detailed
illustrations. For example, <answer> Beijing </answer>.

--- user turn ---
# Retrieved Content
{memory_content}

# Actor1 Code and Critic1 Feedback
{feedback_page1}

# Actor2 Code and Critic2 Feedback
{feedback_page2}

# Current Task
{task_description}

--- each feedback page ---
## Actor Code
{actor_output}

## Critic Feedback
{critic_output}
```

CAMEL, question answering: User Proxy

```
Answer the given question. You must conduct reasoning inside <think>
and </think> first every time you get new information.
After reasoning, if you find you lack some knowledge, you can call a
search engine by <search> query </search>.
You can search as many times as your want.
```

CAMEL, question answering: User Proxy (continued)

```
If you find no further external knowledge needed, you can directly
provide the answer inside <answer> and </answer>, without detailed
illustrations. For example, <answer> Beijing </answer>.

--- user turn ---
Below is the retrieved memory content and the current task. Please
provide your answer.

# Retrieved Content
{memory_content}

# Current Task
{task_description}
```

CAMEL, question answering: Actor

```
You will be given a question and an user proxy agent's answer for that
question. Follow the procedure below and produce outputs accordingly:

Answer the given question. You must conduct reasoning inside <think>
and </think> first every time you get new information.
After reasoning, if you find you lack some knowledge, you can call a
search engine by <search> query </search>.
You can search as many times as your want.
If you find no further external knowledge needed, you can directly
provide the answer inside <answer> and </answer>, without detailed
illustrations. For example, <answer> Beijing </answer>.

Please consider the user proxy agent's answer and provide your own
answer.

--- user turn ---
Below is the relevant content retrieved from memory, the answer
provided by the user proxy agent, and the current task. Please provide
your answer.

# Retrieved Content
{memory_content}

# User Proxy Answer
{userproxy_output}

# Current Task
{task_description}
```

CAMEL, question answering: Critic

```
You are a strategy evaluator. Your task is to review the response
provided by the actor agent for the current problem.

- If you believe the actor agent's response is correct and has no
  issues, reply only with: "Agree".
- If you believe the actor agent's response has issues, provide brief
  and concise feedback only (keep your response short and within 3
  sentences).

--- user turn ---
Below are the relevant contents retrieved from memory, the response
given by the actor agent, and the requirements of the current task.
```

CAMEL, question answering: Critic (continued)

```
Please provide your review.

# Retrieved Content
{memory_content}

# Actor Output
{actor_output}

# Current Task
{task_description}
```

CAMEL, question answering: Summarizer

```
You will be given a question, the responses produced by actor agents,
and the corresponding feedback from critic agents for that question.
Follow the procedure below and produce outputs accordingly:

Answer the given question. You must conduct reasoning inside <think>
and </think> first every time you get new information.
After reasoning, if you find you lack some knowledge, you can call a
search engine by <search> query </search>.
You can search as many times as your want.
If you find no further external knowledge needed, you can directly
provide the answer inside <answer> and </answer>, without detailed
illustrations. For example, <answer> Beijing </answer>.

--- user turn ---
# Retrieved Content
{memory_content}

# Actor Output
{actor_output}

# Critic Feedback
{critic_output}

# Current Task
{task_description}
```

AutoGen, code: Assistant and User Proxy

```
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 tao 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.

--- user turn ---
Below is the relevant content retrieved from memory and the current
task. Please provide a **detailed implementation strategy** for the
```

AutoGen, code: Assistant and User Proxy (continued)

```
task.

# Retrieved Content
{memory_content}

# Current Task
{task_description}

=== User Proxy ===
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
```

--- user turn ---
Below is the relevant content retrieved from memory, the
implementation strategy provided by the assistant, and the current
task requirements. Please complete the code implementation.

Retrieved Content
{memory_content}

Implementation Strategy
{assistant_output}

Current Task
{task_description}
```

MacNet, code: Actor, Critic, and Summarizer

```
You are a Code Implementation 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
```

--- user turn ---
```

MacNet, code: Actor, Critic, and Summarizer (continued)

```
Below is the relevant content retrieved from memory and the current
task requirements. Please complete the code implementation.

# Retrieved Content
{memory_content}

# Current Task
{task_description}

=== Critic ===
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).

--- user turn ---
Below are the relevant contents retrieved from memory, the code
implementation provided by the actor agent, and the current task
requirements. Please provide your review.

# Retrieved Content
{memory_content}

# Actor Output
{actor_output}

# Current Task
{task_description}

=== Summarizer ===
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
'''

--- user turn ---
# Retrieved Content
{memory_content}

# Actor1 Code and Critic1 Feedback
{feedback_page1}

# Actor2 Code and Critic2 Feedback
{feedback_page2}

# Current Task
{task_description}
```

CAMEL, code: User Proxy, Actor, Critic, and Summarizer

```
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 tao 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.

--- user turn ---
Below is the relevant content retrieved from memory and the current
task.

# Retrieved Content
{memory_content}

# Current Task
{task_description}

=== Actor ===
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
```

--- user turn ---
Below is the relevant content retrieved from memory, the user agent's
analysis, and the current task requirements. Please complete the code
implementation.

Retrieved Content
{memory_content}

User Agent Analysis
{userproxy_output}

Current Task
{task_description}

=== Critic ===
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".
```

CAMEL, code: User Proxy, Actor, Critic, and Summarizer (continued)

```
- If the code has issues, give brief and concise feedback only(Keep
  your response short and within 3 sentences).

--- user turn ---
Below are the relevant contents retrieved from memory, the code
implementation provided by the actor agent, and the current task
requirements. Please provide your review.

# Retrieved Content
{memory_content}

# Actor Output
{actor_output}

# Current Task
{task_description}

=== Summarizer ===
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
'''

--- user turn ---
# Retrieved Content
{memory_content}

# Actor Code
{actor_output}

# Critic Feedback
{critic_output}

# Current Task
{task_description}
```

PDDL system prompts (all architectures)

```
AutoGen Assistant / User Proxy:
You are a problem-solving agent.

MacNet Actor:
You are a problem-solving agent.

MacNet Critic:
You are a strategy evaluator. Your task is to review the response
provided by the actor agent for the current problem.
```

PDDL system prompts (all architectures) (continued)

```
- If you believe the actor agent's response is correct and has no
  issues, reply only with: "Agree".
- If you believe the actor agent's response has issues, provide brief
  and concise feedback only (keep your response short and within 3
  sentences).

MacNet Summarizer:
You are a summarizer agent.
You will be given a question, the responses produced by actor agents,
and the corresponding feedback from critic agents for that question.

CAMEL User Proxy:
You are a problem-solving agent.

CAMEL Actor:
You will be given a question and an user proxy agent's answer for that
question. Please consider the user proxy agent's answer and provide
your own answer.

CAMEL Critic:
You are a strategy evaluator. Your task is to review the response
provided by the actor agent for the current problem.

- If you believe the actor agent's response is correct and has no
  issues, reply only with: "Agree".
- If you believe the actor agent's response has issues, provide brief
  and concise feedback only (keep your response short and within 3
  sentences).

CAMEL Summarizer:
You will be given a question, the responses produced by actor agents,
and the corresponding feedback from critic agents for that question.
Please provide your response in accordance with the task requirements.
```

E.2 Role Record Serialization

The template below serializes the role record used for retrieval scoring. The header names the source task and functional role, followed by the role's responses and their direct upstream messages. After trajectory selection, the associated hidden-state blocks provide the composer's experience input. The serialization keeps at most three steps of the role, the first, middle, and last, whereas the composer reads the role's first three hidden-state blocks in temporal order.

Serialized role record (retrieval scoring)

```
## Private Memory from successful trajectory {i}
[role_trace]
Source task:
{source_task}

Successful pre-response local trace for role={role_type}:

[step={step_index}]
node={node_id}
Role response:
{role_response}
Direct upstream role outputs:
- {upstream_node_id}: {upstream_response}

[step=...]  (at most three steps: first, middle, and last)
```

E.3 COMPOSER PROMPT

The templates below show the textual scaffold for composition. In RAMEM, the experience fields are supplied as projected hidden-state blocks with structural delimiters, packed with the task and role instruction as in Eq. (4). The second template includes the role's earlier output as additional context. Learned query vectors follow the packed input; their final states are projected into the agent's input space.

Composer prompt

```
# Current Task
{current_task}

# Examples of previous successful tasks related to this task
{text_memory}

You are acting as a {role}. Using no more than {k} tokens, extract the
most relevant information from the memory above
that will help you accomplish the current task effectively.
```

Composer prompt with the role's own earlier output

```
# Current Task
{current_task}

# Examples of previous successful tasks related to this task
{text_memory}

# {role} response in each successful task above
{agent_message}

You are acting as a {role}. Using no more than {k} tokens, extract the
most relevant information from the memory above
that will help you accomplish the current task effectively.
```

F ALGORITHM

Algorithm 1 gives the two procedures of RAMEM: building role memory $\mathcal{B}$ from the shared experience bank $\mathcal{D}$, and serving one agent invocation. Construction captures the agent backbone's final-layer hidden states for each role invocation and keeps the text of the same responses alongside them, since similarity search operates on text rather than on the states directly. The serving procedure follows Section 3 step by step: retrieval uses the role records and the complete trajectories to decide which hidden states to load, and composition reads the states themselves. Retrieval uses the source-task index of $\mathcal{D}$ with query views q and q_r, and $\mathcal{T}_{x,r}$ holds the K trajectories kept after reranking, with $K = 1$ in all reported experiments. As described in Section 4.1, the writeback step before the return runs during evaluation on every benchmark and is disabled during training; a successful episode enters both stores and becomes retrievable by later tasks.

Algorithm 1 RAMEM: role memory construction and memory-augmented invocation

Require: Shared experience bank $\mathcal{D}$; frozen composer F_ϕ, input projection g_ω, query vectors $\mathbf{Q}$, and output projection P_ψ; latent length L; weight λ; shortlist size n_s; number of kept trajectories K

Role memory construction

$\mathcal{B} \leftarrow \varnothing$
for each successful trajectory $\tau \in \mathcal{D}$ **do**
  for each functional role r with at least one response in τ **do**
    for each invocation $v \in \mathcal{I}_{\tau,r}$ **do**
      $H_v \leftarrow [h_{v,1}^\top; \ldots; h_{v,T_v}^\top]$, the agent backbone's final-layer states after each response token of v
    end for
    Keep the same responses as the role record $e_{\tau,r}$, with the source task and context–response pairs, Eq. (1)
    Store $\{H_v\}_{v \in \mathcal{I}_{\tau,r}}$ with $e_{\tau,r}$ in $\mathcal{B}$ under $(\mathrm{id}(\tau), r)$
  end for
end for

Invocation of role r on task x

Form the task query q and the role-conditioned query q_r
$\mathcal{C}_{x,r} \leftarrow$ union of the nearest trajectories for both queries, Eq. (2)
Compute the role-alignment score $s(x, r, \tau)$ for every candidate, Eq. (3)
Sort $\mathcal{C}_{x,r}$ by $(m_{\tau,r}, a_\tau, s(x, r, \tau))$ and keep the first n_s; $\mathcal{T}_{x,r} \leftarrow$ the K trajectories with the highest LLM reranker scores
$\mathcal{H}_{x,r} \leftarrow (H_1, \ldots, H_J)$ looked up in $\mathcal{B}$ by the identifiers in $\mathcal{T}_{x,r}$ and role r
$U_{x,r} \leftarrow$ task and role embeddings packed with $g_\omega(H_1), \ldots, g_\omega(H_J)$, Eq. (4)
$\mathbf{M}_{x,r} \leftarrow P_\psi\big(\mathrm{Tail}_L(F_\phi([U_{x,r}; \mathbf{Q}]))\big)$, Eq. (5); append $\mathbf{M}_{x,r}$ to the agent's prompt embeddings, Eq. (6)
Generate the agent's response from the extended input
if writeback is enabled and the episode succeeds **then** add the trajectory to $\mathcal{D}$ and its hidden-state blocks and role records to $\mathcal{B}$
return the agent's response